

SESSION

NOVEL ALGORITHMIS AND APPLICATIONS + METHODOLOGIES

Chair(s)

TBA

Optimization of out-of-core data preparation methods identifying runtime determining factors

Tamás Schrádi¹, Ákos Dudás² and Sándor Juhász³

Department of Automation and Applied Informatics, Budapest University of Technology and Economics, Budapest, Hungary

¹Tamas.Schradi@aut.bme.hu

²Akos.Dudas@aut.bme.hu

³Sandor.Juhasz@aut.bme.hu

Abstract: In the data preparation phase of a data mining task the raw, fine granulated data has to be transformed according to analytical aims into a more compact form in order to represent data at a higher abstraction level suitable for machine processing and human understanding as well. Vast datasets require sophisticated, out-of-core methods, which are prepared to handle these datasets using external storages during their execution. In this paper we investigate different pre-processing approaches to overcome the limitation of the size of the main memory from theoretical and practical points of view. We propose possible alternatives for different processing scenarios. Both of the proposed out-of-core algorithms are capable of processing datasets which are by orders of magnitude larger than the main memory; all this is done in a fault-tolerant way and even on an average PC.

Keywords: out-of-core methods; data preparation; Periodic Partial Result Merging, K-way Merge based Technique; performance analysis

1 Introduction and motivation

Data mining, an interdisciplinary field of science has the aim of revealing the knowledge from datasets, which are not suitable for human understanding due their high cardinality and high dimensionality. A data mining tasks consists of different steps, one of which is data preparation. After determining the objective of the whole data mining process, in the data preparation phase the raw dataset has to be transformed according to analytical aims into a higher abstraction level form. The transformed data can be cleaned, filtered, separated and then an actual data mining method (e.g. classification, clustering, time series analysis, etc) can be performed on the prepared dataset.

The transformation of the dataset usually cannot be avoided in data mining tasks in transaction based systems. In these system the primary goal, and the organizing scheme of the whole dataset, is to fulfill the requests of users. It is also typical that some meta-information related to the requests are usually stored in form of log files. Such log files contains in raw form of raw data different behaviors, frequent patterns, which are the typical targets of data mining based knowledge revealing. To serve the large number of user requests as soon as possible the logging in such systems must happen in a quick way. In everyday workflow there is no time for a long logging operation, thus the logs typically store elementary data. Beside

the significant time constraint for logging we do not necessarily know in advance what kind of analytical task will be executed, thus what kind of data representation to use. Consequently there is no guarantee that a transformation between two high abstraction level datasets can be done. Because of these arguments the logging typically happens at a low abstraction level.

The log requires a pre-processing which can consume significant portion of the whole data mining process; data preparation is one of the most time-consuming tasks regarding the different phases of a data mining task [1]. The acceleration of the whole data mining analysis can be achieved with the speed-up and optimization of data preparation step. The bigger the dataset is the more emphasized the efficient data preparation task should be.

In Web analytics the log analysis is a frequently applied technique, when datasets based on server side logs are analyzed. The principle of storage at low abstraction level presented in the previous paragraph is reasonable in the case of log file analysis as well. The log files store the elementary data belonging to the interactions of the users (e.g. the timestamp of the request or the type of the browser, which initiated the user request, operating system, etc). With cookie-based techniques the users can be tracked and a profile can be derived, which describes the behavior of the user [2][3]. It is not the specific profiles that are important for content providers, but the typical groups of users, which is valuable information for them. To identify the typical user groups we can apply clustering, which has the aim of detecting the typical groups, but before this, the raw dataset must be transformed into an analyzable form.

Our research is motivated by a real life project dealing with Web log mining. This paper focuses on the performance analysis and on factors which determine the execution time of out-of-core methods. The profiles about the temporal behavior of users are derived from a dataset containing more than 6 billion records about the temporal behavior of users. Based on the user identifier and on the timestamp of the request a complex structure has to be derived, a temporal profile created by aggregating the clicks belonging to the same user. Such a huge cardinality calls for out-of-core methods in order to process the raw dataset. The fact, that a log belonging only to a single month contains more than 6 billion records supports the

need for out-of-core processing for analysis over a longer period of time. The proposed algorithms, the Periodic Partial Result Merging and K-way Merge based Technique are scalable, out-of-core methods, capable of process datasets several times bigger than the main system memory.

The organization of this paper is as follows: Section 2 presents the properties of methods dealing with large datasets and the state-of-the-art scientific approaches in this field. Section 3 introduces two novel out-of-core approaches, their execution time analysis compared to execution complexities of other approaches and the execution time determining factors of algorithms. Section 4 shows the results of algorithms measured on real datasets, while the last section summarizes our work and presents possible further work.

2 Related work

Although the amount of available memory has been significantly increasing during the past decades, handling large datasets is a still challenging problem in environments limited by. In this paper a dataset is regarded large, if it does not fit in the main memory at the same time. During the evolution of computers this size is continuously varying. In computer-related literature several approaches were published how to handle such vast datasets.

Creating a representative subset of the original dataset based on statistical sampling is a frequently applied method to solve the problem of limited memory. With simple random sampling or a more sophisticated sampling technique (e.g. stratified sampling) a dataset can be generated which has very similar statistical properties to the original dataset. Thus, according to our assumption the processed datasets will share similar statistical properties as the result of the original dataset. But from a sample-based dataset only a partial result can be generated; this fact makes this method inappropriate for problems requiring results based on the whole dataset (e.g. aggregation).

Another approach to overcome the limited memory issue is the compression of the dataset. This technique is based on the idea, that the redundancy of the dataset makes it unmanageable large. According to information theory, there is a lower bound of compression, thus in general there is no guarantee that the compressed dataset will in fact fit in the main memory. The compression of a dataset can be done using specific data structures which can have other favorable properties as well, like in [4][5][6]. Another issue related to the compression is whether such external data structures can be designed that have a similar I/O performance as the uncompressed structures [7]. A compressed, external data structure, with competitive I/O performance could further accelerate the data preparation step.

If we have to process large datasets by orders of magnitude larger than the size of main memory out-of-core methods can be a well-scalable solution. Out-of-core methods make the processing even in a memory limited environment possible by the usage of secondary storages (e.g. hard disks). These methods follow a partitioning principle, which means that the

original dataset is processed in smaller blocks, resulting partial processed sets, from which the global result can be obtained. Due to the high cardinality of dataset the principle is applicable so that the partial results are stored on a secondary storage, freeing the main memory for processing another block.

In out-of-core literature we can find several techniques for generation of global result from the partial results: the global result of some problems can be generated as the union of the partial result sets, as presented in [8][9][10]. For other problems a merging can be the applicable technique to determine the global dataset from partial results [11][12]. In other cases a more complex algorithm has to be performed to derive the global result [13].

We have approaches which solve the memory limited issues using secondary storages. In this paper we discuss the performance analysis of these methods and an essential factor of the out-of-core methods can be observed even in conceptual phase. If we take a look at the up-to-date computer architecture a crucial runtime determining factor can be pointed out: accessing a secondary storage lasts more than accessing the same data being actually held in main memory. This factor influences the efficiency of processing, resulting that in an out-of-core method the number of I/O instructions should be kept at a minimum level.

This requirement of minimal I/O instructions is essential from another point of view as well: the raw datasets are generated in an automated way, continuously, thus it is needed to avoid the extrusion of the raw data. This could be done by assuring that the procedural steps have linear time complexity, which in general cannot be satisfied. But if we keep the I/O instructions at the lowest level the performance of the processing will be still efficient to avoid data extrusion.

Based on the two previous points the core-efficiency of the out-of-core methods depends on whether they read only constant times the input dataset or not.

Before applying an out-of-core method the block size has to be chosen: the successively, equal-sized partitioning is a trivial, but working method [11][12][14]. But according to [9] a sophisticated partitioning approach can have performance increasing effect. The carefully chosen size is an important performance determining factor, because with it the consumed memory can be controlled. An eager, in-memory algorithm will be presented in this paper to demonstrate the undesirable behavior of the processing when the main memory reaches its physical bounds.

3 Out-of-core processing methods

In this paragraph five different processing approaches are presented, together with their runtime complexity analysis and the resulting factors. First we discuss an in-memory algorithm, which have a major shortcoming assuming that processed data will fit in the main memory at the same time. Two different extension ways will be presented to overcome the problem of limited memory: dataset modification (sampling, compression)

and out-of-core methods (Periodic Partial Result Merging and a K-way based Merging).

3.1 An eager method

In order to illustrate the weaknesses of an in-memory algorithm an eager processing method is presented in this paper. This method is not only important because demonstrates the incapability of the eager approach, but its out-of-core extensions are the base of the other algorithms. Although the name of the algorithm suggests that this approach is not a sophisticated one, still some facts have to be taken into consideration to build an efficient core approach. During the aggregation we have to find one specific element among millions of others, thus a very fast searchable container is needed. This container will be held in the main memory and this will be refreshed periodically. A well-scaled hash-based container makes the search fast ($O(1)$). The eager method works as follows: at every procedural step the hash-based container is updated and when the processing is done, the hash-based container from the main memory is saved to a persistent storage (hard disk).

Assuming that our dataset contains n records, partitioned equally-sized (a block contains m records), the number of blocks created with partitioning is $s = \lceil \frac{n}{m} \rceil$. A further assumption is that from an x sized dataset $\bar{\alpha}x$ will be the size of the resulting dataset, where $\bar{\alpha} \ll 1$. Investigating the disk complexity of the eager method, we have to read all s pieces of blocks from hard disk and write the aggregated dataset back to it, expressed by the following formula:

$$c_{disk} = \sum_{i=1}^s m + \bar{\alpha}n = (1 + \bar{\alpha})n \quad (1.1)$$

Similarly, the procedural complexity can be calculated based on the following expression

$$c_{proc} = \sum_{i=1}^s f(m) = g(m)n, \quad (1.2)$$

where $g(m)$ is a monotonically increasing function, expressing the time of processing of an m sized block. As the formulas suggests this is a fast method, but its applicability is limited due to immense memory need of the algorithm, bounded only by the size of the processed dataset.

3.2 Sampling

In some cases sampling could be a possible solution for a large dataset. This algorithm samples the block and picks up an m' sized dataset, so that the whole sample will fit in the main memory and then the eager method can be done on it. Assuming a γ sampling ratio ($\gamma < 1$) the disk and processing complexity of the algorithm can be expressed as follows.

$$c_{disk} = \sum_{i=1}^s m' + \bar{\alpha} \cdot \gamma \cdot n = (1 + \bar{\alpha})\gamma \cdot n \quad (2.1)$$

$$c_{proc} = \sum_{i=1}^s f(m') = \gamma \cdot g(m')n \quad (2.2)$$

The incompleteness of the resulting dataset makes this algorithm inappropriate for our aggregation problem, although the complexities are reduced.

3.3 Compression

Compressing the resulting datasets could mean another approach to handle a vast dataset: here the processing is done according to eager method, but every partial result is compressed in the memory. Assuming a compression ratio of η , ($\eta < 1$) the disk and processing complexity of this approach is as follows

$$c_{disk} = \sum_{i=1}^s m + \bar{\alpha} \cdot n' = (1 + \bar{\alpha} \cdot \eta)n \quad (3.1)$$

$$c_{proc} = \sum_{i=1}^s f(m) = g'(m)n, \quad (3.2)$$

where $g'(m)$ denotes the time needed to process an m -sized dataset and to compress it. This approach has the same problem like the eager method, namely the memory constraint, because there is no guarantee that the compressed dataset will fit in the main memory. Furthermore the processing algorithm should be able to handle compressed data structures, which lead to additional runtime.

3.4 Periodic Partial Result Merging

The Periodic Partial Result Merging algorithm processes only smaller sized datasets at the same time in the main memory, creating a local result on the storage [15][16]. After processing the first dataset the result is saved to the storage, while in other steps the actual partial result is merged with the existing one from the disk. In this approach the local results are propagated during the phases of processing, and they are merged after finishing the processing of a block. After the last merging the resulting dataset will be the global result. The processing of a data block is done according to the eager method, but here a block fits in the main memory. An essential step in whole processing is the merging phase. In order to elaborate an efficient working version of the algorithm, an ordering is needed, defined on processed datasets, this ensuring a minimal additional execution time, caused by merging.

Having the same assumption regarding the cardinalities, the disk complexity and the processing complexity can be expressed as follows:

$$c_{disk} = \sum_{i=1}^s m(1 + \alpha_i(2i - 1)) \approx n + \frac{\bar{\alpha}}{m}n^2 \quad (4.1)$$

$$c_{proc} = \sum_{i=1}^s (f(m) + i \cdot m \cdot \alpha_i \cdot \beta) \quad (4.2)$$

$$\approx \left(\frac{f(m)}{m} + \frac{\bar{\alpha} \cdot \beta}{2} \right) n + \frac{\bar{\alpha} \cdot \beta}{2m} n^2,$$

where β is a factor making the execution time of processing and merging operations commensurable and $f(m)$ denotes the processing cost of a m -sized block.

Both of the complexities are quadratics in the number of elements to be processed. As mentioned in a previous paragraph it is essential to keep the number of I/O instruction linear in number of elements. This can be approximated making the coefficient of the quadratic member closer to zero than to one, thus decreasing the undesirable effect of this member. The average compression factor of the dataset ($O(1)$) is a fixed property, which usually satisfies our prerequisite being a small positive number, close to 0. In the denominator of the coefficient we have the size of a block; increasing the size of the block the value of the fraction will decrease, making its value even closer to zero. Based on this observation the bigger the size of a block is, the more efficient will the algorithm be, but due to memory constraint we cannot increase the size of the memory arbitrarily. By choosing a big dataset size the effect of the quadratic member can be reduced, in the processing complexity as well.

3.5 K-Way Merge based Technique

The K-way Merge based Technique follows the partitioning principle too, but besides propagating the results at every procedural step, separates the processing from merging, being to different phases of the processing. As first step the algorithm processes all the partitioned blocks, according to eager method. It is essential to remark that the blocks are partitioned so, that the processing can be done in the main memory. After processing a block the resulting dataset is saved to the persistent storage. When all the blocks are processed, the merging phase will be done on partial results saved to disk, containing processed elements. This means a k -way merge among the elements, resulting at the root of the merging tree the global result.

Having the same assumptions with cardinalities, the disk and runtime complexity of this algorithm can be expressed as follows:

$$c_{disk} = s \cdot m + 2 \sum_{i=1}^{\lceil \log_k s \rceil - 1} \left[\frac{s}{k^i} \right] \cdot \bar{\alpha} \cdot m \cdot k^i + \bar{\alpha} m k^{\lceil \log_k s \rceil}$$

$$c_{disk} \approx 2\bar{\alpha} \cdot n \cdot \log_k n + (1 + \bar{\alpha}(1 - 2 \log_k m)) n \quad (5.1)$$

$$c_{proc} = s \cdot f(m) + \sum_{i=1}^{\lceil \log_k s \rceil} \left[\frac{s}{k^i} \right] \bar{\alpha} \cdot m \cdot k^i \cdot h \left(\left[\frac{s}{k^i} \right] \bar{\alpha} \cdot m \cdot k^i \right)$$

$$c_{proc} \approx \bar{\alpha} \cdot n \cdot \log_k n + \left(\frac{f(m)}{m} - \bar{\alpha} \cdot \log_k m \right) n \quad (5.2)$$

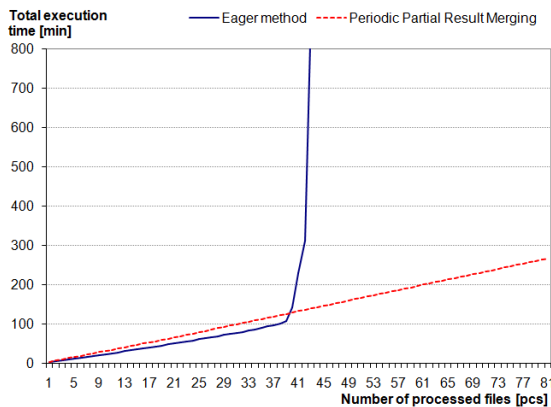
A basic runtime determining factor is the number of levels in the merging tree; so many times will the slow I/O instructions be performed. Reducing these levels, i.e. merging more files at the same time can increase run-time, but here another factor should be taken into consideration as well. Namely, reading from several files during a merge, causes performance decrease, due to continuous positioning on the hard disk. This factor is not expressed in formulas, but the experimental results support it. Both of the complexities can be approximated with $O(n \cdot \log_k n)$ complexity.

3.6 Hybrid approach

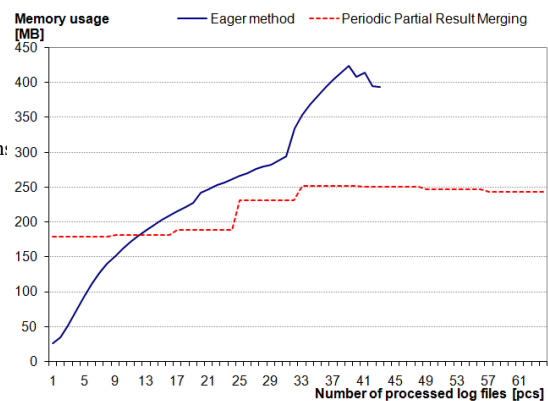
The determination of block size is a crucial step in PPRM, thus we propose that amount of processed data at the same time to be determined based on memory usage. It is an adaptive version of PPRM because the number of blocks processed at one procedural step is determined by the used memory amount.

Beside this optimization we propose an algorithm consisting of two steps: first according to the adaptive version of PPRM we aggregate log records until we get a given number of partial result. This number is the optimal number of ways in k -way based merging, because the resulting aggregated chunks are processed with the k -way merge as the next step. The value of k in the merge phase is equal with the number of remaining chunks after the processing with PPRM.

With this approach we overcome the high hard disk demand of the K-Way Merge based Technique, but we gain in performance with second step, due its lower complexity.



ifferent version:



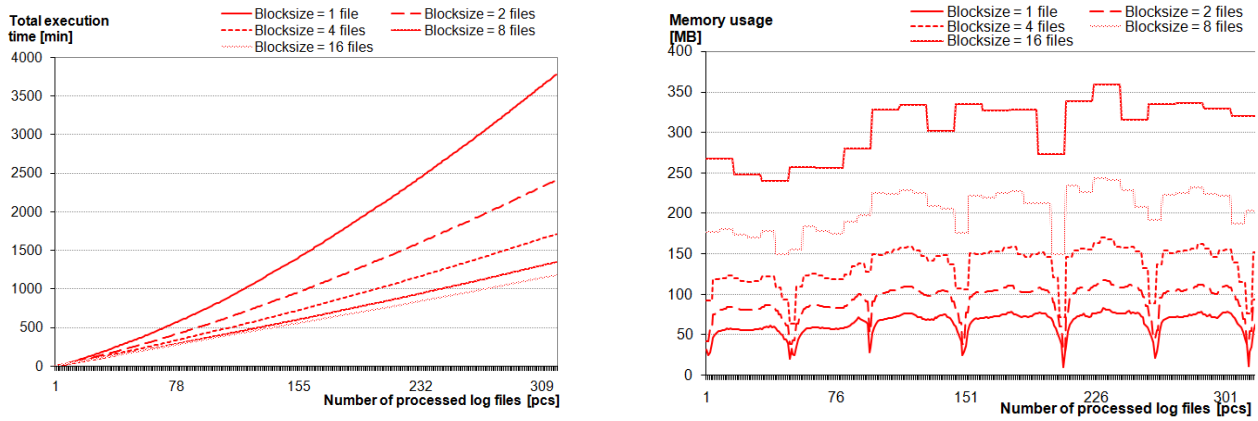


Figure 2. Performance analysis of the K-Way Merge Based Technique

TABLE I. PROPERTIES OF DIFFERENT APPROACHES, HANDLING LARGE DATASETS

	<i>Eager method</i>	<i>Sampling</i>	<i>Compression</i>	<i>PPRM</i>	<i>K-way Merge based Technique</i>
<i>I/O cost</i>	$O(n)$	$O(\gamma \cdot n)$	$O(n)$	$O(n^2)$	$O(n \log(n))$
<i>Proc. cost</i>	$O(n)$	$O(n)$	$O(n)$ +C/D	$O(n^2)$	$O(n \log(n))$
<i>Result type</i>	Compl.	Partial	Compl.	Compl.	Compl.
<i>Limits</i>	Memory limits		Memory limits	Merging is applicable to generate the result	Merging is applicable to generate the result

4 Experimental results

The experiments were carried out in two different environments: the first hardware configuration (CPU: AMD Athlon at 2800 MHz, memory: 512 MB, operating system: Windows XP Professional) represented a memory limited environment in order to show the weaknesses of the in-

memory approach, while the second configuration (CPU: Intel Pentium 4 at 3200 MHz, memory: 4 GB, operating system: Microsoft Windows Server 2003) meant a real test environment. The algorithms, presented in this paper were implemented in C++.

The graphs based on tests executed on the first configuration are presented in Figure 1; the other figures are created from the results of tests carried out in the second environment.

Figure 1 proves the necessity of out-of-core processing. In this test we used the memory limited configuration to show the incapability of the eager method to process effectively in a memory limited environment. The figure on the left side shows the runtime of different methods. It is clearly that the curve corresponding to the runtime of the eager method, when the physical memory reaches its limit and according to virtual memory handling paging occurs, becomes unmanageable. The right side of Figure 1 presents the memory consumption of the algorithms. The curve of the eager method supports that the memory need of this method is limited only by the size of processed dataset, which is bigger than the size of the main memory.

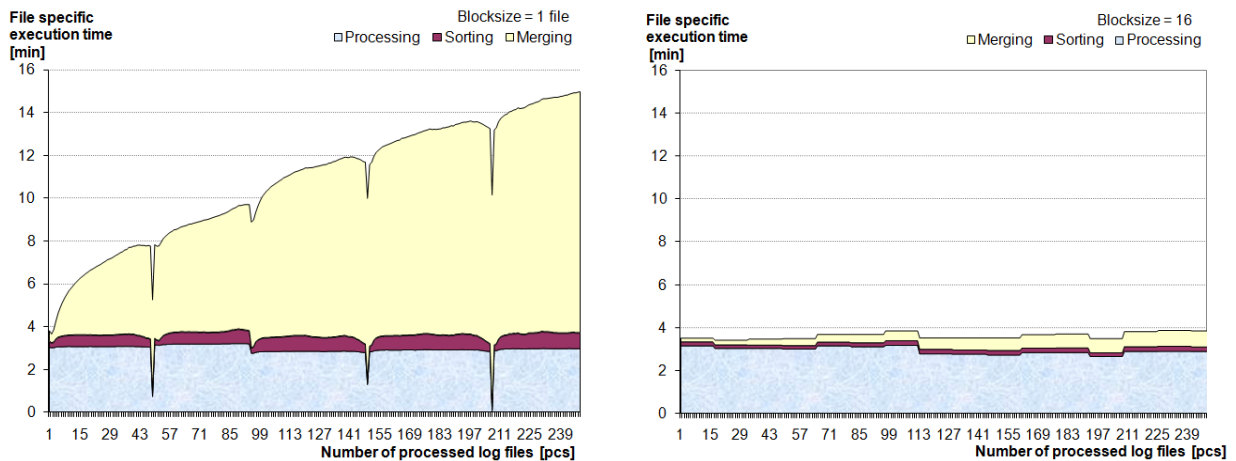


Figure 3. The block-specific execution time components of Periodic Partial Result Merging

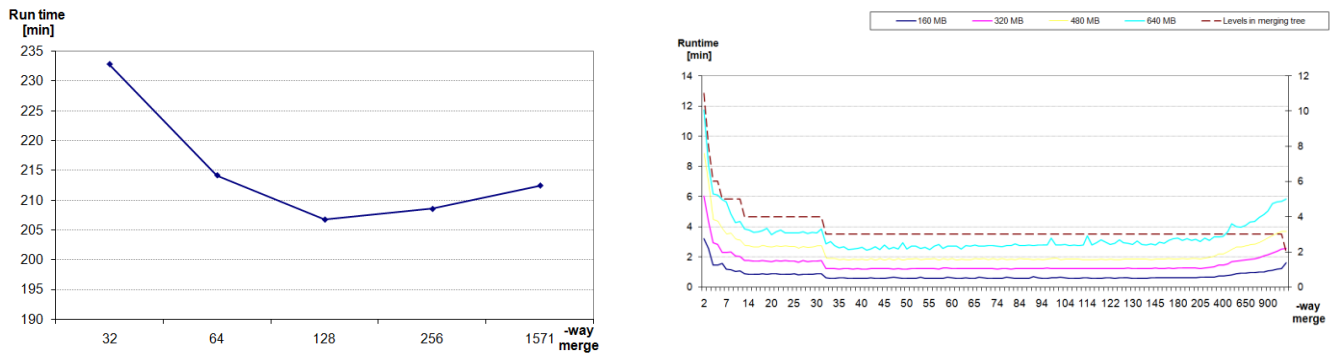


Figure 4. The performance analysis of K-way Merge Based Processing (left side) and a comparative graph of execution times of K-way Merge based Technique with different datasizes

Figure 2 shows the dependency on I/O instructions of the Periodic Partial Result Merging: the graph on left side sketches the runtime of different versions. The more files are processed at the same time in the main memory, this means the number of I/O instructions are reduced, the faster the processing is. The runtime is inversely proportional to the size of dataset processed at the same time in the main memory. But there are other factors which influences the runtime of this out-of-core method. The graph on the right side shows the used memory of algorithms: in tests the different block sizes fitted in the main memory. But the memory consumption graph shows an oscillating characteristic and in general there is no assurance that at every procedural step the processed dataset will fit the main memory.

Figure 3 shows the time consumed by every procedural step on every file. The left side graph shows the case of a smaller block size, while the right side graph for a larger block size. The processing step per block at both of the approaches lasts nearly the same time, due to the well-scaled, hash based container. In this implementation the ordering was done using a sort on the identifier of the user. The third component, the merging step makes the significant difference between the two processing approaches, supporting our theoretical consideration of possible optimization.

Figure 4 shows the runtime plots of different sized datasets together with the number of levels in the merging tree. For smaller k values the processing follows the changes of level in merging tree (the broken line). This is in accordance with the theoretical performance analysis, that the number of levels is a basic execution time determining factor. For bigger values of k there is an increase in run time, although the number of levels is not increasing. A bigger k value means that a merging is done on more files, meaning a continuous positioning of the head of hard disk, increasing the execution time. According to Figure 4 the optimal k values for the merging is somewhere around 32. Here another fact has to be taken into consideration, namely that the dataset of the test execution was an artificial dataset, created from the real dataset dissolving the overlapping between the records.

The right side of Figure 4 shows the run time performance of K-way Merge Based Technique, measured on real dataset.

Based on this graph the optimal value for the ways of merging is around 128, a higher value then on the artificial dataset. The overlapping factor is the cause of performance increase although the merging tree has equal levels, but handling more blocks together avoids the multiple propagation of a record during the merging. Merging from too much files causes increase in execution time due to the continuous positioning on the hard disk. The figure on the right side shows a summarizing graph with the execution time of total processing using different out-of-core approaches.

5 Conclusion and further work

In this paper we have investigated the different factors determining the performance of out-of-core methods and we have proposed two novel out-of-core approaches. The Periodic Partial Result Merging with continuous propagation and merge of partial results on disk overcomes the difficulties of limited memory. The K-way Merge based Technique processes smaller blocks of data saving the result on external storage and then processes the partial results according to a k -way merge. The PPRM method requires more memory to process the data efficiently, while K-way Merge based Technique has a bigger external storage demand.

As state-of-the-art hardware technology experience suggested it the number of I/O instructions has a major effect on the runtime of out-of-core methods; also supported by our empirical results. Thus to optimize an out-of-core algorithm it is necessary to keep the number of I/O instructions at a minimal level.

As the performance analysis section of different algorithms showed it, the complexity of out-of-core approaches is higher than linear (quadratic in the case of Periodic Partial Result Merging, $O(n \log_k n)$ in case of the K-way Merge based Technique), but these complexities can be reduced by choosing a large block size. On the other hand, as the experimental results showed it, the memory consumption of the process has to be controlled during the whole processing to avoid unmanageable runtime. Beside the large processing block the compression factor has its reducing influence on

disk- and runtime complexity, but this is a given factor of the dataset (typically closer to 0 than to 1).

Because of the large sizes to be processed the runtime of these methods can be better expressed in hours than in minutes, thus a fault-tolerance is a favorable, demanded property of the out-of-core approaches. By their nature both of the algorithms contain a fault-tolerance possibility.

In K-Way Merge based Technique, based only on the formula, the more way we use in merging the faster the processing will be. In this method the number of levels in merging tree should be kept at a minimal level, but other influencing factor is the number of files processed together at the same time.

The eager method presented in this paper supports that if the presumption of fitting the main memory does not hold, out-of-core methods mean an applicable and scalable approach for processing. With Periodic Partial Result Merging or with K-Way Merge based Technique large datasets can be processed efficiently, in a fault-tolerant way, even on an average PC.

The analysis of the parallel version of methods can be a further work, based on [17].

6 Acknowledgment

This work is connected to the scientific program of the "Development of quality-oriented and cooperative R+D+I strategy and functional model at BUTE" project. This project is supported by the New Hungary Development Plan (Project ID: TAMOP-4.2.1/B-09/1/KMR-2010-0002).

7 References

- [1] Poll: Data preparation part in data mining projects. Data Mining Community's Top Resource. Oct. 2003. 23 Apr. 2009 <http://www.kdnuggets.com/polls/2003/data_preparation.htm>.
- [2] Benczúr, A. A., Csalogány, K., Lukács A., Rácz, B., Sidló, Cs., Uher, M. and Végh, L., Architecture for mining massive web logs with experiments, In Proceedings of the HUBUSKA Open Workshop on Generic Issues of Knowledge Technologies
- [3] Iváncsy, R. and Juhász, S. 2007, Analysis of Web User Identification Methods, Proc. of IV. International Conference on Computer, Electrical, and System Science, and Engineering, CESSE 2007, Venice, Italy, pp. 70-76.
- [4] M. Y. Eltabakh, W.-K. Hon, R. Shah, W. Aref, and J. S. Vitter, "The SBCTree: An index for run-length compressed sequences," in Proceedings of the International Conference on Extending Database Technology, Nantes, France: Springer-Verlag, March 2008.
- [5] P. Ferragina and R. Grossi, "The String B-tree: A new data structure for string search in external memory and its applications," Journal of the ACM, vol. 46, pp. 236–280, March 1999.
- [6] P. Ferragina, R. Grossi, A. Gupta, R. Shah, and J. S. Vitter, "On searching compressed string collections cache-obliviously," in Proceedings of the ACM Conference on Principles of Database Systems, Vancouver: ACM Press, June 2008.
- [7] Vitter, J. S. Algorithms and Data Structures for External Memory. Boston, MA: Now, 2008. Print.
- [8] Grahne, G. and Zhu, J. 2004, Mining frequent itemsets from secondary memory, ICDM '04. Fourth IEEE International Conference on Data Mining, pp. 91-98.
- [9] Nguyen Nhu, S. and Orłowska, M. E. 2005, Improvements in the data partitioning approach for frequent itemsets mining, 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD-05), pp. 625-633.
- [10] Nguyen Nhu, S., Orłowska, M. E. 2006, A further study in the data partitioning approach for frequent itemsets mining, ADC '06 Proceedings of the 17th Australasian Database Conference, pp. 31-37.
- [11] Lin, J. and Dunham, M. H 1998., Mining association rules: Anti-skew algorithms, In 14th Intl. Conf. on Data Engineering, pp. 486-493.
- [12] Savasere, A., Omiecinski, E. and Navathe, S. 1995, An efficient algorithm for mining association rules in large databases, VLDB '95: Proceedings of the 21th International Conference on Very Large Data Bases, pp. 432-444
- [13] Tang, P., Ning, L., and Wu, N. 2005. Domain and data partitioning for parallel mining of frequent closed itemsets. In Proceedings of the 43rd Annual Southeast Regional Conference - Volume 1 (Kennesaw, Georgia, March 18 - 20, 2005). ACM-SE 43. ACM, New York, NY, 250-255. DOI= <http://doi.acm.org/10.1145/1167350.11674230>
- [14] Lucchese, S C. and Perego, O. R. 2006, Mining frequent closed itemsets out-of-core, 6th SIAM International Conference on Data Mining, pp. 419-429.
- [15] Juhász S., Iváncsy R., Out-of-core Data Handling with Periodic Partial Result Merging, IADIS International Conference on Data Mining, Algarve, Portugal, 18-20 June, 2009.
- [16] Juhász S., Iváncsy R., Approaches for Efficient Handling of Large Datasets, IADIS International Conference on Data Mining, Algarve, Portugal, 18-20 June, 2009
- [17] R. Dementiev and P. Sanders, "Asynchronous parallel disk sorting," in Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, pp. 138–148, ACM Press, 2003

Effect of cache lines in array-based hashing algorithms

Ákos Dudás¹, Sándor Kolombán², and Tamás Schrádi³

Department of Automation and Applied Informatics,
Budapest University of Technology and Economics,
1117 Budapest, Magyar Tudósok körútja 2. QB-207 Hungary
{¹akos.dudas,²kolomban.sandor,³schradi.tamas}@aut.bme.hu

Abstract—Hashing algorithms and their efficiency is modeled with their expected probe lengths. This value measures the number of algorithmic steps required to find the position of an item inside the table. This metric, however, has an implicit assumption that all of these steps have a uniform cost. In this paper we show that this is not true on modern computers, and that caches and especially cache lines have a great impact on the performance and effectiveness of hashing algorithms that use array-based structures. Spatial locality of memory accesses plays a major role in the effectiveness of an algorithm. We show a novel model of evaluating hashing schemes; this model is based on the number of cache misses the algorithms suffer. This new approach is shown to model the real performance of hash tables more precisely than previous methods. For developing this model the sketch of proof of the expected probe length of linear probing is included.

Keywords: hash table; hashing; linear probing; cache-aware; performance

1. Introduction

Hashing algorithms are a popular choice in a great variety of applications for fast storage and retrieval of items. Over the years there have been many hashing algorithms presented in the literature. These algorithms are usually compared based on the expected probe lengths [1], [2], [3], [4], that is, the number of steps the algorithm has to take before an item can be inserted into the table. (This is equivalent to the number of steps it takes to find an item. Both will be referred to as probe length throughout this paper.)

It has been our observation [5], and also the suggestion of others [4], [6], that the expected probe length does not model the true performance correctly. Measuring the wall-clock execution times of the algorithms and using the expected probe length-based raking of hashing schemes we can arrive at two contradicting conclusions; in this paper we propose a solution that unifies the expected probe length-based comparison and the physical capabilities of the hardware, resulting in a more precise efficiency estimation.

The fact is, that the true performance of array-based hashing schemes is effected by the physical hardware it is executed on. The expected probe length-based efficiency

analysis has the implicit assumption that every probe in the probe sequence has the same cost; this is not necessarily true though. Modern CPUs have fast (but small) integrated caches that mask the latency of the main system memory. Accessing data in these caches is by one or two orders of magnitude faster than reading from the main memory. These caches speed up temporally, and which is more relevant for us, spatially local memory reads.

Algorithms that exploit these caches are called cache friendly [7]. What we propose in this paper is basically a new efficiency model that rewards cache friendly algorithms. We focus our attention to hashing schemes that use arrays; the basic idea however it generally applicable, and not just to hash tables but to other data intensive algorithms as well.

The rest of the paper is organized as follows. Section 2 presents the related literature of hash tables and their performance and complexity. The expected probe lengths of two hashing schemes are presented in Section 3 followed by the comparison of the hash tables based in the expected probe lengths and wall-clock execution times. To resolve the contradictory results new efficiency model is presented in Section 4. We conclude in Section 5 with the summary of our results.

2. Related works

Hash tables [8] store and retrieve items identified with a unique key. A hash table is basically a fixed-size table, where the position of an item is determined by a hash function. If the position calculated by the hash function is already occupied, a collision occurs. This collision can be resolved by storing the items externally (e.g. using a linked list), the approach of bucket hash tables; or a secondary hash function can be applied which calculates a new address for the item. Repeating this process until a free slot is found the algorithm traverses a probe path. The shorter this probe path is, the faster the hash table is.

In linear probing [8] the secondary hash function calculates the next address by adding a constant (usually one) to the previous address. This, of course, is not a true hash function. However, this “laziness” is what makes linear probing cache friendly [7].

There is a theoretical hashing scheme, which produces perfectly random address generation. The idea is that there

is a uniformly chosen permutation from the space of permutations over the possible addresses for each element. The initial hash function returns the first element of the permutation, the secondary hash function iterates through the permutation. This is called uniform hashing [9]. It has been shown, that uniform hashing is optimal for all open-address hashing schemes in its expected probe length. It was proven that double hashing, a practical realization of a hash scheme [4], is asymptotically equivalent to uniform hashing when the number of available addresses is large [10], [11]. It is generally thought that these sophisticated methods are superior to linear probing.

Black et al. [6], however, has shown that linear probing can have better performance than double hashing. Moreover, by tuning the parameters of double hashing to make it approximate linear probing, its performance increased as well.

Heileman and Luo [4] also conducted similar examinations and they ended up with another conclusion. According to their results, the cache friendliness of linear probing cannot compensate the disadvantage of the longer probe sequences in case of realistic data sets. They also suggested that the relation between the size of the data and the cache size is what lies behind the seemingly contradictory results. In Section 4 we will confirm their hypothesis.

3. Expected probe length based comparison

The expected probe length is used to measure the efficiency of hash tables. The formula is known for uniform hashing; in this section we show the sketch of calculating it for linear probing.

3.1 The expected probe length of uniform hashing

The expected probe length for uniform hashing [8] in an α -filled table is

$$\mathbb{E}(L_{uni}^\alpha) = -\frac{\ln(1-\alpha)}{\alpha} \quad (1)$$

where \mathbb{E} denotes the expected value, L_{uni} is the expected number of steps needed to find a uniformly chosen element in a hash table built with uniform hashing and α is the load factor of the hash table (i.e. the ratio between the number of elements in the table and the number of all the slots).

3.2 The expected probe length of linear probing

In order to calculate the expected probe length (i.e. number of steps in takes to insert an item), first we need to understand how a single item is inserted into an α -filled hash table using linear probing. The expected probe length in a given state of the hash table is the average of probe lengths needed to insert the elements of the tables.

We describe a table configuration using the following two notions in this paper. A *cluster* is a group of adjoint occupied slots. A *closed cluster* is formed from an empty slot and the cluster that precedes it. If, for a certain empty slot there is no preceding cluster (i.e. the empty slot is preceded by another empty slot), then this empty slot forms a closed cluster by itself. Figure 1 gives a graphical representation of these notions. Closed clusters cover the whole table, while clusters obviously do not.

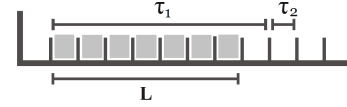


Fig. 1: A cluster (L) and two closed clusters (τ_1 and τ_2).

During insertion the probe length depends on the length of the closed cluster in which its initial address hits, since the insertion has to iterate over the items in this closed cluster. Suppose that the initial address for the new element is part of a closed cluster with length τ . The expected number of steps needed to insert the element is $\frac{\tau+1}{2}$, since the initial address is considered to be uniformly distributed over the whole table, and consequently, it is uniform restricted to the given closed cluster as well. If we knew the probability of the event that the initial address falls into a closed cluster with length τ then we would be able to calculate the expected probe length required to insert a new element.

The followings are just the sketch of how we can calculate the expected probe length for linear probing. It is out of the scope of this paper to show every step; this is merely the general idea.

Suppose there are X_i elements initially hashed to the i -th slot, $i = 1, \dots, N$. Suppose that a closed cluster starts at the j -th position. In this case the number of elements in this closed cluster is at least X_j , and the j -th slot can hold only one. So $X_j - 1$ elements will be held in the rest of the closed cluster. The $j + 1$ -th slot adds X_{j+1} elements to the cluster, but it will hold one as well, so the number of elements in the cluster after the $j + 1$ -th slot is $X_j - 1 + X_{j+1} - 1$, and so on. When this sum reaches 0, then the cluster is closed.

We can define the stochastic process S the following way: $S_0 = 1$, $S_{i+1} = S_i + X_{j+i} - 1$. The stopping time $\tau^{(k)}$ (2) is a good approximation of the distribution of the length of the closed clusters:

$$\tau_0^{(k)} = \inf(i : S_i = 0 | S_0 = k) \quad (2)$$

The explicit formula for the distribution of the length of closed clusters can be found as

$$p(k) = \mathbb{P}(\tau = k) = \frac{(\alpha k)^{k-1}}{k!} e^{-\alpha k} \quad (3)$$

We also need the expected value of this distribution, which is

$$\mathbb{E}(\tau) = \frac{1}{1-\alpha} \quad (4)$$

The expected number of steps to find a uniformly chosen element in the table is the average of the steps needed to insert them. There are M elements in the table. When inserting the i -th element, there were $i - 1$ already inserted. In other words, the i -th element was inserted into an $\alpha' = \frac{i-1}{N}$ -filled hash-table. The average of these step counts gives us

$$\mathbb{E}(L_{lin}^\alpha) = 1 + \frac{1}{2} \frac{\alpha}{1 - \alpha} \quad (5)$$

where $\mathbb{E}(L_{lin}^\alpha)$ denotes the expected probe length for linear probing in an α -filled table.

3.3 Evaluation using probe lengths and wall-clock execution times

Figure 2 shows the expected probe length for linear probing and uniform hashing for various load factors. It is obvious that uniform hashing has a smaller expected probe length. Based on this fact, one could say that linear probing is to be neglected while choosing hashing algorithms for practical purposes.

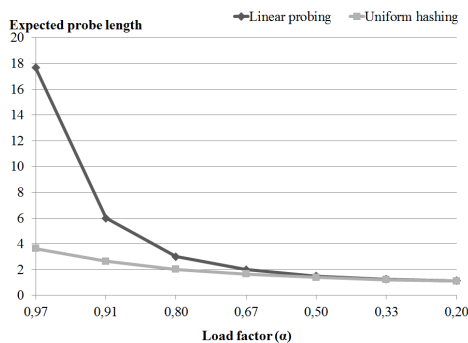


Fig. 2: The expected probe length of linear probing and uniform hashing for different load factors.

Our experimental results, on the other hand, show different results. Figure 3 plots the measured wall-clock execution times of building a table using linear probing and double hashing. Linear probing has shorter lookup execution than double hashing. This is the exact opposite of the previous result.

To resolve this contradiction a new complexity function is required, one that approximates the true performance of hash tables. The problem with the probe length based ranking is that it assumes that every probe has the same cost; instead, the characteristics of the execution environment have to be considered and integrated into the cost function.

4. Cache-line aware algorithm complexity

This section presents a simple model of memory hierarchy which is then incorporated into the cost function of the steps of the probe sequence.

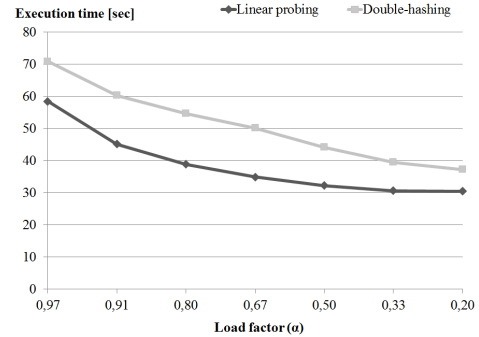


Fig. 3: The measured wall-clock execution times of linear probing and uniform hashing for different load factors.

4.1 Caches

Open hash tables span over a large block of allocated memory. This block is split into slots, which are identified by a number (memory address; i.e. indexes of the array). The address of neighboring slots are sequential, therefore in linear probing, after slot i is visited, whose address is j , the next slot, $i + 1$, will have the address $j + 1$. This is not true for uniform hashing; the addresses of the probe sequence will be scattered across the table. Let us explain why this is important.

In current computer systems CPUs have caches, which are fast access, but limit space storages integrated into the CPU or onto the same die. The cache stores a small fraction of the data that is stored in the system memory, therefore a small portion of the hash table also resides in the cache. Whenever the CPU requests data, it is first checked in the cache. If found, the main system memory is not queried as the cache returns the data. However, if the data is not in the cache (this event is called a cache miss), the data is loaded from the main system memory; this operation is by one or two orders of magnitude slower than reading from the cache.

Another important factor is cache lines. The memory is partitioned into small blocks, called cache lines. Whenever data is loaded into the cache, an entire cache line is loaded; the one the requested data is inside. This means, that with a single memory access it is not just the requested data that is loaded, but some neighboring addresses are read as well - at no additional cost. If the next read is in this very same cache line, it will be served by the low latency cache.

If accesses to the memory is temporarily or spatially local, the cache speeds up the algorithm by not requiring the system to read data from the system memory. When an algorithm exploits these effects, it is called cache friendly. It allows the algorithm to have lots of data requests at fraction of the cost.

4.2 Cache-line based memory model

The cost difference between accessing data from the main memory and from the cache is often neglected in

performance models. Let physical memory requests have a cost of one. Altering the usual memory model, where every access has a uniform cost, we propose a new model. In our model the blocks of the memory are grouped into lines of equal lengths. These lines correspond to the cache lines of the real system. The characteristics of these lines is that they are read as one.

In this memory model, the true performance of an application is determined not by the total number of memory accesses, but by expected number of read lines. In case of hashing algorithms this is equal to the number of probed lines, which the probe sequence accesses. In other words, the number of produced cache misses is the determining factor.

Suppose that the parameters of the memory architecture and the hash table are such that an integer number of hash table slots fit in a cache line. This parameter will be denoted with B . Figure 4 shows a scenario where three hash table slots fit into a single cache line ($B = 3$). Items with the same color are hashed to the same position by the primary hash function.

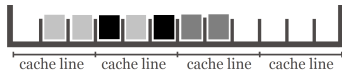


Fig. 4: The segmentation of the memory into cache lines of length $B = 3$. Items with the same color have the same initial address.

As an example, the second item from the left can be found with cost of one if linear probing is used, since the first checked slot and the final position are both in the same cache line, and the probe sequence examines no slots outside of this line.

4.3 Cache-aware cost function for uniform hashing and linear probing

Given a relatively large hash tables that does not fit into the cache (usual size of caches is 2-4-8 MB) but instead is stored in the main system memory, the CPU cache has considerable impact. A typical hash table entry consists of a unique integer id and a data pointer. This means that the number of hash slots that fit in a cache line (B) is about 2-8 entries (assuming a cache line is 64 bytes) and the number of lines that can be stored in the cache memory is negligible compared to the number of lines that are covered by the hash table.

For uniform hashing the probe sequence is a random permutation of the positions in the table. Thus, it is a fair approximation that all probes fall into one of the not cached lines. This means that every probe has a cost of one when uniform hashing is used. In other words every probe produces a cache miss. This can be formalized as

$$\mathbb{E}(C_{uni}^\alpha) = \mathbb{E}(L_{uni}^\alpha) = -\frac{\ln(1-\alpha)}{\alpha} \quad (6)$$

where C_{uni}^α is the cost of a probe sequence that finds a random element in an α -filled table built with uniform hashing. The value is independent of B .

To verify this formula, Figure 5 shows the calculated values against our measured values.

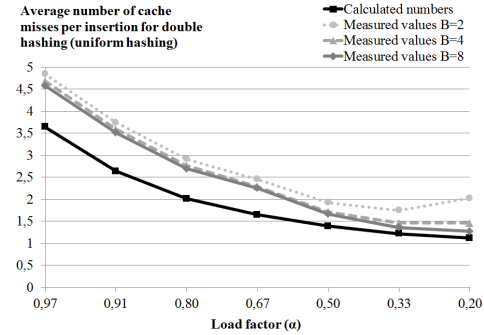


Fig. 5: The theoretical and experimental number of cache misses for double hashing (uniform hashing).

In case of linear probing the first address of a probe sequence will not be among the list of cached addresses. This means that the first probe will request a new line to be read from the memory. But the following probes have a high probability of being served from the cache, since the neighboring slots of the first probed one were cached when the first probe was performed. If the initial address is given then each step will produce a cache miss with probability of zero or one, depending on whether it is in the same cache line, or in the next one, respectively. Since the initial address of a probe sequence is uniform over the slots of the line it falls into, each of the remaining $L_{lin}^\alpha - 1$ steps produce a cache miss with probability $\frac{1}{B}$. Thus, we can say that

$$\mathbb{E}(C_{lin}^\alpha) = 1 + \frac{1}{B}(\mathbb{E}(L_{lin}^\alpha) - 1) = 1 + \frac{1}{B} \frac{1}{2} \frac{\alpha}{1-\alpha} \quad (7)$$

where C_{lin}^α is the cost of the probe sequence that finds a uniformly chosen element in an α -filled table built with linear probing.

To verify this formula as well, Figure 6 plots the measured and the calculated values for various B s.

4.4 Cache-aware cost model and true performance

Finally, let us compare the true performance of the hash tables, measured in terms of wall-clock execution time, with the proposed cost model.

Figure 7 shows how the expected number of produced cache misses look like (left), and the what are the corresponding measured execution times (right). The number of slots that fit in a cache line is $B = 2$, $B = 4$ and $B = 8$.

It is clear that for any given value of B there is a load factor α_B under which linear probing has lower cost and over which uniform hashing algorithms are better. From

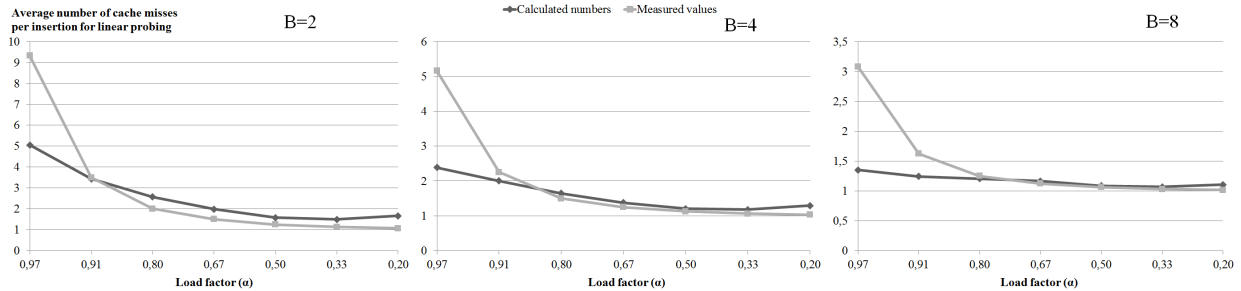


Fig. 6: The theoretical and experimental number of cache misses for linear probing.

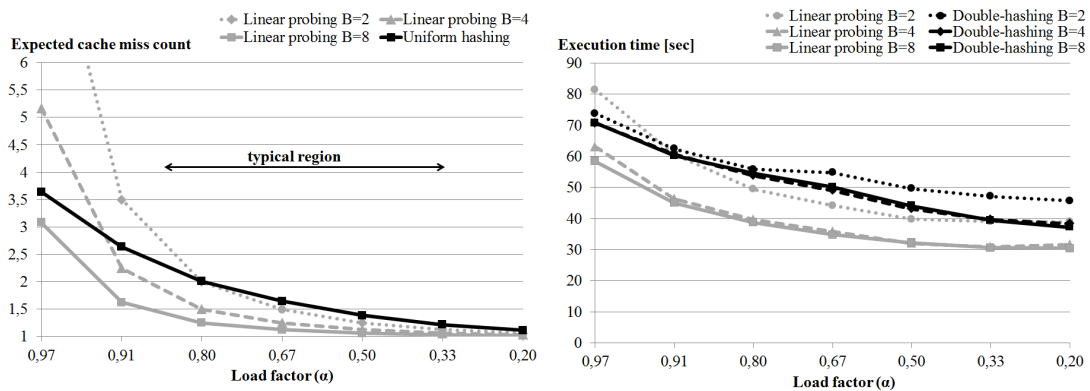


Fig. 7: The expected number of cache misses for linear probing and uniform hashing for different load factors for $B = 2, 4, 8$.

equations (6) and (7) this load factor can be obtained. In the typical operation region of $\alpha \in [0.3 \ 0.8]$, linear probing has lower expected cache miss count even when $B = 2$. This is confirmed by the execution times as well.

In general, when choosing a hashing algorithm, one should consider the parameters of the hash table and memory architecture, namely parameter B should be determined and the operational load factor should be decided.

Generally, our conclusion is that the simple algorithmic step count based ranking of algorithms, especially for algorithms that intensively use memory, is not sufficient. The physical capabilities of the machine that executed the algorithms should be taken into consideration, and with integrating the memory model into the cost function, a better efficiency comparison can be derived.

5. Conclusion

Hashing algorithms are usually ranked by their expected probe lengths. It has been our observation, and also published in the literature, that this is not always true. Based on previous works we know that in case of open-address hashing the performance of the algorithm is greatly affected by its memory characteristics.

We have shown that the expected probe path based efficiency comparison is not fair for linear probing, which is generally thought of as an inferior choice of hashing

scheme. Under real-life circumstances, however, it is able to outperform more sophisticated hash tables, such as double hashing.

Incorporating the effect of cache lines into the cost function of hashing algorithms we have presented a novel model of evaluation. This approach models the true performance of these hash tables more precisely.

Acknowledgment

This project is supported by the New Hungary Development Plan (Project ID: TÁMOP-4.2.1/B-09/1/KMR-2010-0002) and by the fund of the Hungarian Academy of Sciences for control research, the Hungarian National Research Fund (grant number T68370).

The authors express their thanks to Sándor Juhász for his help as scientific advisor.

References

- [1] G. H. Gonnet, "Expected Length of the Longest Probe Sequence in Hash Code Searching," *Journal of the ACM*, vol. 28, no. 2, pp. 289–304, Apr. 1981.
- [2] M. V. Ramakrishna, "Hashing practice: analysis of hashing and universal hashing," *ACM SIGMOD Record*, vol. 17, no. 3, pp. 191–199, Jun. 1988.
- [3] A. Pagh, R. Pagh, and M. Ruzic, "Linear probing with constant independence," *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing - STOC '07*, p. 318, 2007.

- [4] G. L. Heileman and W. Luo, "How Caching Affects Hashing," in *Proc. 7th ALENEX*, 2005, pp. 141–154.
- [5] S. Juhász and A. Dudás, "Adapting Hash Table Design to Real-life Datasets," in *Proc. of the IADIS European Conference on Informatics 2009, part of the IADIS Multiconference of Computer Science and Information systems 2009*, Algarve, Portugal, 2009, pp. 3–10.
- [6] J. R. Black, C. U. Martel, and H. Qi, "Graph and Hashing Algorithms for Modern Architectures: Design and Performance," pp. 37–48, 1998.
- [7] A. Binstock, "Hashing Rehashed," *Dr. Dobb's Journal*, vol. 4, no. 2, 1996.
- [8] D. E. Knuth, *The art of computer programming, Vol 3*. Addison-Wesley, Nov. 1973.
- [9] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *J. Comput. System Sci.*, vol. 18, no. 2, pp. 143–154, May 1979.
- [10] L. J. Guibas, "The Analysis of Hashing Techniques That Exhibit k-ary Clustering," *Journal of the ACM*, vol. 25, no. 4, pp. 544–555, Oct. 1978.
- [11] M. L. Fredman, J. Komlós, and E. Szemerédi, "Storing a Sparse Table with $O(1)$ Worst Case Access Time," *Journal of the ACM*, vol. 31, no. 3, pp. 538–544, Jun. 1984.

Performance and Quality of Random Number Generators

V. du Preez, M.G.B. Johnson, A. Leist and K.A. Hawick

¹Computer Science, Massey University

²Albany, North Shore 102-904, Auckland, New Zealand

Abstract—Random number generation continues to be a critical component in much of computational science and the tradeoff between quality and computational performance is a key issue for many numerical simulations. We review the performance and statistical quality of some well known algorithms for generating pseudo random numbers. Graphical Processing Units (GPUs) are a powerful platform for accelerating computational performance of simulations and random numbers can be generated directly within GPU code or from hosting CPU code. We consider an alternative approach using high quality and genuinely “random” deviates generated using a Quantum device and we report on how such a PCI bus device can be linked to a CPU program. We discuss computational performance and statistical quality tradeoffs of this architectural model for Monte Carlo simulations such as the Ising system.

Keywords: quantum random number generation; GPU; CUDA.

1. Introduction

The fast generation of good quality random numbers [1]–[6] is a long-standing challenge [7], [8]. Random numbers are needed for many applications, but are used in very large quantities in computer simulations that employ the Monte-Carlo algorithm [9], [10]. It is neither trivial nor computationally cheap to generate large sets of pseudo-random numbers that have the right statistical “randomness” needed to perform an unbiased calculation. Until recently it has not been practical to use random number generation hardware that was economically priced and suitably unbiased. Instead, pseudo random numbers that were generated from a suitable deterministic algorithm were employed. A great deal has been written in the literature about such algorithms, but for the most part there are many very good ones that are “random enough” and are at least uncorrelated with the application so that they suffice. One important area of use has been the numerical investigation of phase transitions and critical phenomena. In such work the Monte Carlo algorithm is used to sample appropriate points in a physical model space to simulate the actual dynamical behaviour of a model and identify the location of critical points – abrupt changes - that result when a model parameter such as temperature changes by a small amount.

This work is very demanding and a certain degree of caution is perceived in the reported literature as researchers go to great lengths to be sure their simulations



Fig. 1: Quantis RNG Card for PCI Bus, showing four independent quantum generator devices.

are not overly biased by random numbers with unfortunate statistical properties.

Pseudo-random number generators are often formulated in terms of mathematical recurrence relations [11] whereby repeated application of a transformation will project a number to another in an apparently random or decorrelated sequence - at least to the extent that any patterns discernible in the resulting sequence are on a scale that is irrelevant to the application using them. Any random or pseudo random number generator delivers a sequence of random deviates - either floating point uniform deviates or integers or sometimes just plain bits. An application will use or consume deviates from such a sequence as it runs.

There are still some philosophically deep questions concerning what it really means for a sequence of deviates to be truly random. It is widely believed however that some quantum physical processes yield deviates that are as random as we can ever make them. Such devices are presently available as special purpose cards or external drivers that connect via a bus-based hardware interface such as PCIe or USB. We investigate the use of the ID Quantique “Quantis” device below in section 4. Intel and other CPU manufacturers [12] are now actively considering provision of random number generation hardware directly on the chip. This closeness to the arithmetic and logic hardware means that these devices will produce very fast deviates, and the expectation is that the thermal noise and quantum processes involved are sufficiently well understood at a statistical level to ensure that these sources are also unbiased.

Our paper is structured as follows: In Section 2 we review some key issues for random number generation. In Section 3 we briefly review the Ising model and associated Monte Carlo analysis algorithms as demanding consumers

of random deviates. In Section 4 we describe some of the pseudo random number generator algorithms and implementation strategies we have explored. We present some performance and statistical test results for both algorithmically generated and quantum device generated sequences in Section 5. We discuss some of the implications for future generation simulation work and offer some conclusions and ideas for further study in Section 6.

2. Generation Algorithm Issues

Generally speaking there are two main criteria that are considered when choosing a pseudo-random number generator. The first is the period of the generated sequence. Ideally this should be so long as to never repeat during the life-cycle of the application. Modern generators – as we discuss here – usually have periods that are very long and that when run on current computer clock speeds have repeat times comparable with the lifetime of the known universe. In this sense the period is not often a direct concern.

A few deviates generated to make a game program behaviour “interesting” to a player does not require a generator with a challengingly long repeat length. However, Monte Carlo calculations that may take weeks or months of supercomputer resources must have generators with very long period lengths. In the last 20-30 years of steadily increasing supercomputer performance, there has been continued interest in ever longer period generator algorithms. This often ties in with the need for more bits used in the generator. The 16-bit integer based generators of the late 1970s, were superseded by 24-bit (floating-point) algorithms such as the Marsaglia lagged-Fibonacci algorithm [6], by the 64-bit integer based Mersenne-Twistor and in very recent times by 128-bit algorithms [13] and even longer for cryptographically strong random number generation [14].

The second criteria is more subtle however and has definitely been a known concern with some algorithms. This issue concerns just how actually random or uncorrelated the deviates in the sequence are – in the context of the needs of the driving application. There are some widely used statistical tests [15] that are now in wide circulation and which represent the research communities best wisdom on what is “random enough.” We discuss these in section 4.

Applications usually need either random integers with a flat uniform probability of obtaining all values within a set range, or uniform floating point deviates in the range $[0, 1)$, again with a uniform probability distribution across the range. Generally if one has a generator that produces either of these, one can construct deviates of more sophisticated distributions with suitable transformation algorithms [16], [17].

The apparatus for implementing pseudo-random number generators usually give rise to raw deviates in one of those two forms - uniform integers or uniform floating point number and one can find ways of transforming one to the other. In the case of floating point deviates

one can simply multiply by N to obtain integers on the $[0, N)$ range, and in the case of integers known to be in that range, one can divide by N . Different processing hardware will carry these operations out with different speeds depending on clock speeds and floating point standards. If one has a random source of uncorrelated b -bits [18], [19] one can readily obtain (unsigned) integers [20]. in a suitable range of $[0, 2^b)$ or $[0, 2^b - 1]$. One can then divide accordingly to obtain floating point uniform deviates. The reverse operation is not so simple however [18]. Most processors use the IEEE floating point standard bit representation for 32-bit or 64 bit precision. These specify sign bit, exponent and mantissa from which it is not trivial to obtain evenly unbiased random bits without some arithmetic that must necessarily waste some of the original 32 or 64.

This gives rise to another important criteria for random number generators - ideally they should be well engineered in terms of having plug-compatible software programming interfaces. This means that a code can be tested and implemented using any number of different generator algorithms with little code change required. A pragmatic implementer therefore finds it is often better to have a generator that produces unbiased integers or raw bits from which an unbiased unsigned integer can be constructed. It is often then easier to make a family of suitable software interfaces to supply all the sorts of deviates that are needed by applications from the one root algorithm.

In this present paper, we discuss Monte Carlo algorithmic uses of random numbers where we need a fast supply of good deviates. Another application is for cryptographic use, where usually the need for extreme speed is less, but the need for very high randomness - to the point of uncrackability is extreme [21].

For some applications it is actually desirable to have pseudo-random number generator that is repeatable – from the same starting conditions. Seeding generators is of course an interesting issue in itself and this problem is often exacerbated when using a parallel computer. While a generator algorithm may have a known very long period, often one has to run the generator many times or on parallel processors and the choice of seeds matters to avoid accidentally correlating the sub-sequences generated by each instance [22], [23]. Parallel computing applications such as parallel and supercomputer implementations of Monte Carlo simulations have been a target for many special purpose implementations of pseudo random number generators. Work has been done on: array processors [24]; vector computer architectures [25]; transputers using parallel Occam [26]; and more recently on specialist processors such as the Cell [27] or on Graphics Processing Units (GPUs) [28]–[30].

Techniques for generating seeds vary. When debugging an application it can be very helpful to be able to specify the same seed and ensure identical results. Once in production mode seeds might be generated by an algorithm based on precise time and dates or from special purpose

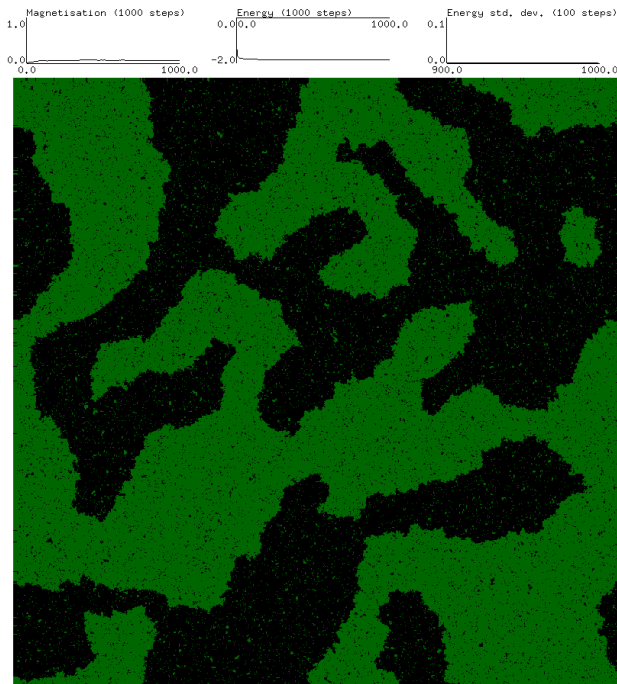


Fig. 2: A 1024×1024 Ising model simulation with temperature $T = 2.0$ after 1000 simulation steps.

hardware. Many operating systems will now support a hardware source via for example `/dev/random` on Unix based systems. This may supply bits from thermal noise or other sources. Such deviates are unfortunately not necessarily statistically unbiased nor necessarily particularly fast - but they certainly suffice for seeding a proven pseudo random algorithm that does have the required qualities.

Another approach which has only recently become economically feasible and which may become more widespread soon [31], is to have a hardware source of genuinely random numbers - that are drawn from some quantum physical phenomena [32] that is as random as we can imagine given our current understanding of the universe, and which therefore do not require a starting seed. Figure 1 shows a special purpose device, produced by Quantis, that generates around 16Mbits/s that are – as we have determined and discussed below – of superb quality.

3. Ising Model Applications

Monte Carlo simulations use random sampling to approximate results when it is infeasible or impossible to compute the exact result for a physical or mathematical system [33]. The Ising model [34]–[36] uses such a method to calculate the critical point of metal alloy phase transitions. The numbers in these systems need to be as close to truly random as possible to avoid bias in the results which may result in incorrect conclusions

Simulations of the Ising model typically start with a random “hot” system. The system is then quenched to a specific temperature. If this temperature is below a critical

“cold” temperature, then the system undergoes a phase transition where like spin values begin to clump together, creating order in the initially random system. The Ising model has just two possible spin values, “up” and “down”, but can be extended to the Q -state Potts model [37] that uses Q spin values. A system quenched to a temperature very close to the critical temperature shows clusters of like-like spins on all possible length scales. Figure 2 illustrates a 2-dimensional Ising model simulation.

A number of different Monte-Carlo update algorithms for the Ising model have been proposed over time [38]–[41]. The Metropolis algorithm [38], which was later generalised by Hastings [42], has formed the basis for Monte-Carlo statistical mechanics [43], [44] and has been used widely for Ising model simulations [45]–[48]. It is a Markov chain Monte-Carlo (MCMC) method, where the transitions from one state to the next only depend on the current state and not on the past. Using the Metropolis update algorithm for the Ising model simulation, at each discrete time step, a new system configuration is chosen at random by picking a spin to “hit” and flipping its value. If the energy E of the proposed configuration is lower than or equal to the current energy, $\Delta E \leq 0$, then the move to the new configuration is always accepted. Otherwise, the new configuration is accepted with probability $\exp(-\Delta E/k_B T)$, where T is the temperature and k_B is the Boltzmann constant. The current configuration is retained if the move is rejected.

The spins in the Ising model interact with their nearest neighbours according to an energy function or Hamiltonian of the form: $H = -\sum_{\langle i,j \rangle} J_{ij} S_i S_j$, where $S_i = \pm 1$, $i = 1, 2, \dots, N$ sites, and J_{ij} is $|J| = 1/k_B T$ is the ferromagnetic coupling over neighbouring sites i and j on the network.

The Ising model and other Monte Carlo algorithms can be used themselves as demanding tests of the quality of random numbers, based on comparisons with known results [7].

4. Implementation & Timing

Common methodologies utilise computer CPUs to produce pseudo-random numbers using bitwise operations and mathematical operations to suitably randomise a number. The Mersenne-Twistor [49] is a common generator algorithm to produce high quality numbers, whereas the linear congruential algorithm, which is used in Unix `rand`, is a common and well known low quality example. Producing truly random numbers is impossible when using an algorithm running on a computer, this is the realm of the hardware random number generators (RNGs).

The algorithmic tradeoff space covers very high-quality generator algorithms such as the Mersenne-Twistor that are significantly slower than those very-fast but lower-quality algorithms such as linear congruential generators. In between these extreme cases it is often possible to improve low-quality generator algorithms by adding lag tables and shuffle tables to further randomise or decorrelate

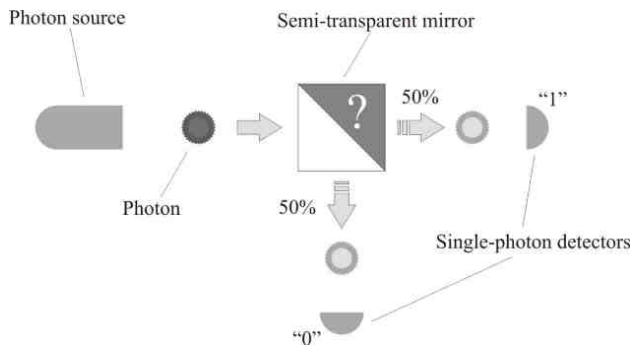


Fig. 3: Description of the method for producing a random bit in the Quantis device.

the sequences of random deviates and indeed to combine several independent algorithmic sources together.

4.1 Quantis Random Number Generator

The quantum random number generator we assess in this paper is the Quantis PCI quantum random number generator produced by ID QUANTIQUE SA [32]. This generator uses a photon emitter directed at a semi transparent mirror, which lets the photons through with a theoretical probability of 50% as shown in Figure 3. Each generator allows for a constant stream of random bits of up to 4 MBits/s. The PCI device contains 4 separate generators, bringing the theoretical maximum random stream to 16MBits/s or ≈ 500 deviates per millisecond.

The Quantis card supports both Windows and various flavours of Linux. For our testing we used Ubuntu Linux with the standard Quantis driver installation. The drivers API facility provides various methods for retrieving different data types. The most low level of these is the `QuantisRead` method:

```
int QuantisRead(QuantisDeviceType deviceType,
                unsigned int deviceNumber,
                void* buffer, size_t size);
```

This generates `size` bytes of random numbers into the variable `buffer`, where `size` is constrained to `QUANTIS_MAX_READ_SIZE`. To get more than this we must loop until the desired size has been reached. Alternately we can use:

```
int QuantisReadInt(QuantisDeviceType deviceType,
                  unsigned int deviceNumber,
                  int* value);
```

To get a signed integer value from the device. This method is much slower at reading multiple numbers than reading raw bytes as we show in section 5. To overcome this problem, we use `QuantisRead` in a multi-threaded environment where one thread is caching blocks of random bytes while the consumer thread uses them. This method may still not be sufficient for algorithms such as the Monte Carlo, but will significantly reduce the time over using `QuantisReadInt`.

5. Performance & Quality

For most scientific purposes it is sufficient to say that they need to be sufficiently uncorrelated that when used

for a Monte Carlo simulation or other application the deviate quality does not lead to an observable bias [50]. Or put more simply – that the random number generator does not lead the applications programmer to the wrong answer. Various statistical tests [8], both at a straightforward level [51], checking for visual planar correlations [52] planes and other approaches such as the spacing test, scatter-plots, that detect obvious patterns or simple statistics are possible, as well as very specific application related tests that are highly sensitive to correlations.

To evaluate the raw performance of generators we test four different popular pseudo-random number generators: Mersenne Twister (MT), Ran described in the book Numerical Recipes (Ran) [53], the standard Unix `rand` and Marsaglia's lagged-Fibonacci Generator (LFG). These generators were tested for randomness using the birthday spacings test found in the diehard testing suite for random numbers, with the values $N = 2^{32}$, $M = 2^{12}$ and $\lambda = 4$. This configuration is advised in [54]. Supplementary tests were also performed with the standard diehard test suite [55] and these confirm the below findings.

Algorithm	Birthday Spacings Pass/Fail
Ran	✓
LFG	✓
MT	✓
Quantis (to CPU)	✓
Unix Rand	X

Table 1: Results of Birthday Spacings test of different RNG algorithms. Tick and Cross indicate pass and fail respectively

Table 1 shows that all except the Unix `rand` random numbers pass the birthday spacings test. This is in line with common knowledge about the periods of these generators [1].

Applications of specific random number generators are dependent on the speed in which the numbers can be attained by the client, where client refers to a central processing unit, graphics processing unit, etc. In random number intensive applications, such as the Monte Carlo algorithms in Ising/ Potts models, computation time is negligible compared to the fetch time for random numbers. Whereas, in cryptography the computation time significantly outweighs the fetch time for the random numbers, which allows slow generators to hide their speed by caching numbers for fast use by other threads.

To test the speed of the algorithms we generate ten million uniform floating point numbers and find the number of deviates per millisecond on an Intel Core 2 Duo at 2.1 GHz using the four algorithms that passed the birthday spacing test. The CPU algorithms only utilise one of the cores available on the CPU. We have also implemented a CUDA GPU version of the lagged-Fibonacci generator [30] and report the performance measured on an NVidia GeForce GTX 580.

Table 2 shows that the results for all of the CPU pseudo-random number generators are comparable in

Algorithm	Performance Deviates Per Millisecond
Ran	24085
LFG	13367
MT	22795
Quantis (Single Thread)	61
Quantis (Multi Thread)	111
CUDA(LFG)	$1.28e10^7$

Table 2: Performance of different RNG algorithms.

speed, with the Ran algorithm producing the most at 24085 deviates per millisecond. This is more than two orders of magnitude faster than the single threaded Quantis generator at about 61 32-bit deviates per millisecond. The lagged-Fibonacci generator on the CUDA GPU is another 2-3 orders of magnitude faster than the CPU algorithms.

6. Discussion & Conclusions

Section 5 shows that all but the Unix rand pseudo-random algorithms pass the Extended Birthday spacing and Die Hard tests that we have implemented. These are well known algorithms and the results are common knowledge [3], hence it is unsurprising that the widely used Unix rand failed. This is further proof that this function should not be used. It has been suggested [3] that lagged-Fibonacci generators may erroneously fail the Birthday spacings test, but this does not appear to be the case for our implementation, which passes the test.

Although these pseudo-random number generators pass most common and also more stringent tests implemented in this paper, this does not guarantee their true randomness in the face of tests yet to be advised. Using physical phenomena, such as photon emitters like the one used in this paper or Intel's on chip temperature variation random source, allows us to guarantee that the number is completely random and free from any bias. Although, the question remains how to test these hardware random sources and can we engineer a test that identifies only a truly random number?

Performance of the generators was as expected [30], with the CUDA GPU LFG algorithm producing $1.28e10^7$ random deviates per millisecond. The single threaded Quantis card algorithm produces only 61 32-bit deviates per millisecond and 111 deviates for the multi-threaded implementation. This is much slower than the theoretical maximum of 500 32-bit deviates from the 16Mbits/s stream of random bits [32]. We attribute this latency to the fetch time from the card over the PCI bus and the conversion time to the specified data type. The speed-up attained by introducing multiple threads is significant as this allows us to hide the time lost in the conversion process and by fetching the maximum number of bytes at each API call we minimise any latency that is associated in calling the Quantis card via the PCI bus. For Monte Carlo algorithms even the CPU pseudo random algorithms are the bottleneck in the simulation, hence the Quantis card is much too slow for these. A good compromise is to use the numbers produced by the Quantis card to seed

a good pseudo-random number generator, thus ensuring that the seeds are statistically independent.

If Intel succeeds in creating a truly random number generator producing 2.4 billion random bits per second [31], then this will significantly increase the reasons for using a hardware random source for random heavy algorithms. Until that point, long period pseudo-random number generators will continue to be the best choice for Monte Carlo algorithms. However, for low random frequency algorithms that depend on high quality random numbers, such as generation of cryptographic keys, current hardware generators are an excellent choice.

We have found that when used in the correct situation the Quantis card is an invaluable resource to computer simulations. However, random number generation is very much an application specific field and we have shown that, when compared to conventional pseudo-random generators, the time it takes to produce a single random deviate with the Quantis card is several orders of magnitude slower. Furthermore, the generation with the Quantis card is inherently serial and does not benefit from parallelisation on either the CPU or GPU. However, we have discussed how this latency may be hidden when the program does not require random numbers often by using a separate thread that fetches the numbers from the Quantis device and prepares them for the main process to use as needed. Another method we have discussed is using the Quantis device to produce truly random seeds for a high-quality pseudo-random number generator.

Graphics processing units offer a performance increase of about 2-3 orders of magnitude over the tested sequential CPU implementations. They have been shown [56] to be a powerful accelerator for Monte Carlo simulations that heavily depend on random numbers. However, developing high-performance code for GPUs is significantly more complex and time consuming than it is to write a sequential or even multi-threaded CPU implementation.

In summary, the field of computer generated random number algorithms is one of "horses for courses" - there is no single best algorithm that will satisfy all requirements. Before starting any project using Monte Carlo algorithms and for which the quality of the random numbers matters, it is therefore of great worth to carefully consider which algorithm to use.

References

- [1] P. L'Ecuyer, "Software for uniform random number generation: distinguishing the good and the bad," in *Proc. 2001 Winter Simulation Conference*, vol. 2, 2001, pp. 95-105.
- [2] R. P. Brent, "A fast vectorized implementation of wallace's normal random number generator," Australian National University, Tech. Rep., 1997.
- [3] G. Marsaglia, "Random Number generation," florida Preprint, 1983.
- [4] —, "A Current view of random number generators," in *Computer science and statistics: 16th symposium on the interface*, Atlanta, Mar 1984, keynote address.
- [5] G. Marsaglia and L.-H. Tsay, "Matrices and the Structure of Random Number Sequences," *Linear algebra and its applications*, vol. 67, pp. 147-156, 1985.
- [6] G. Marsaglia, A. Zaman, and W. W. Tsang, "Toward a universal random number generator," *Statistics and Probability Letters*, vol. 9, no. 1, pp. 35-39, January 1987, florida State preprint.

- [7] P. D. Coddington, "Analysis of random number generators using monte carlo simulation," *J. Mod. Physics C*, vol. 5, no. 3, pp. 547–560, 1994.
- [8] S. Cuccaro, M. Mascagni, and D. Pryor, "Techniques for testing the quality of parallel pseudo-random number generators," in *Proc. of the 7th SIAM Conf. on Parallel Processing for Scientific Computing*, Philadelphia, USA: SIAM, 1995, pp. 279–284.
- [9] K. Binder, Ed., *Monte Carlo Methods in Statistical Physics*, 2nd ed., ser. Topics in Current Physics. Springer-Verlag, 1986, number 7.
- [10] —, *Applications of the Monte Carlo Method in Statistical Physics*, ser. Topics in Current Physics. Springer-Verlag, 1987.
- [11] R. Johnsonbaugh, *Discrete Mathematics*, 5th ed. Prentice Hall, 2001, no. ISBN 0-13-089008-1.
- [12] S. Srinivasan, S. Mathew, R. Ramnarayanan, F. Sheikh, M. Anders, H. Kaul, V. Erraguntla, R. Krishnamurthy, and G. Taylor, "2.4ghz 7mw all-digital pvt-variation tolerant true random number generator in 45nm cmos," in *VLSI Circuits (VLSIC), 2010 IEEE Symposium on*, june 2010, pp. 203–204.
- [13] L.-Y. Deng and H. Xu, "A system of high-dimensional, efficient, long-cycle and portable uniform random number generators," *ACM TOMACS*, vol. 14, no. 4, pp. 299–309, 2003.
- [14] B. Schneier, *Applied Cryptography*, 2nd ed. Wiley, 1996, ISBN 0-471-11709-9.
- [15] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," U.S. National Institute of Standards and Technology, Tech. Rep. NIST Special Publication 800-22,, April 2010.
- [16] W. Hormann, "The transformed rejection method for generating poisson random variables," *Insurance: Mathematics and Economics*, vol. 12, no. 1, pp. 39–45, February 1993.
- [17] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes in C*. Cambridge University Press, 1988, ch. 7, pp. 204–241, random Numbers.
- [18] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, 1996, ch. 5 - Pseudorandom bits and sequences, pp. 169–190.
- [19] E. Barker and J. Kelsey, "Recommendation for random number generation using deterministic random bit generators (revised)," U.S. National Institute of Standards and Technology, Tech. Rep. NIST Special Publication 800-90, March 2007.
- [20] P. Coddington, J. Mathew, and K. Hawick, "Interfaces and implementations of random number generators for java grande applications," in *Proc. High Performance Computing and Networks (HPCN), Europe 99, Amsterdam*, April 1999.
- [21] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM J. Comput.*, vol. 13, pp. 850–864, November 1984. [Online]. Available: <http://portal.acm.org/citation.cfm?id=2054.2068>
- [22] P. Coddington and A. Newall, "Japara - a java parallel random number generator library for high-performance computing," The University of Adelaide, Tech. Rep. DHPC-144, 2004.
- [23] A. Newell, "Parallel random number generators in java," Master's thesis, Computer Science, Adelaide University, 2003.
- [24] K. A. Smith, S. F. Reddaway, and D. M. Scott, "Very high performance pseudo-random number generation on DAP," *Comp. Phys. Comms.*, vol. 37, pp. 239–244, 1985.
- [25] R. P. Brent, "Uniform random number generators for supercomputers," in *Proc. 5th Australian Supercomputer Conference*, Melbourne, Australia, 1992.
- [26] W. Paul, D. W. Heermann, and R. C. Desai, "Implementation of a random number generator in OCCAM," Dec 1989, mainz preprint 87/47, 749.
- [27] D. A. Bader, A. Chandramowlishwaran, and V. Agarwal, "On the design of fast pseudo-random number generators for the cell broadband engine and an application to risk analysis," in *Proc. 37th IEEE Int. Conf on Parallel Processing*, 2008, pp. 520–527.
- [28] M. Giles, "Notes on CUDA implementation of random number genertors," January 2009, oxford University.
- [29] W. Langdon, "A Fast High Quality Pseudo Random Number Generator for nVidia CUDA," in *Proc. ACM GECCO'09*, 2009.
- [30] K. Hawick, A. Leist, D. Playne, and M. Johnson, "Speed and Portability issues for Random Number Generation on Graphical Processing Units with CUDA and other Processing Accelerators," in *Proc. Australasian Computer Science Conference (ACSC 2011)*, 2011.
- [31] G. Anthes, "The quest for randomness," *Comm. ACM*, vol. 54, no. 4, pp. 13–15, April 2011.
- [32] ID Quantique White Paper, "Random Number Generation Using Quantum Physics," ID Quantique SA, Switzerland, Tech. Rep. Version 3.0, April 2010, QUANTIS.
- [33] D. P. Landau and K. Binder, *A Guide to Monte-Carlo Simulations in Statistical Physics*. Cambridge University Press, 2009, vol. 3rd edition.
- [34] M. Niss, "History of the Lenz-Ising Model 1920-1950: From Ferromagnetic to Cooperative Phenomena," *Arch. Hist. Exact Sci.*, vol. 59, pp. 267–318, 2005.
- [35] E. Ising, "Beitrag zur Theorie des Ferromagnetismus," *Zeitschrift fuer Physik*, vol. 31, p. 253258, 1925.
- [36] L. Onsager, "Crystal Statistics I. Two-Dimensional Model with an Order-Disorder Transition," *Phys.Rev.*, vol. 65, no. 3, pp. 117–149, Feb 1944.
- [37] R. B. Potts, "Some generalised order-disorder transformations," *Proc. Roy. Soc.*, pp. 106–109, 1951, received July.
- [38] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *J. Chem. Phys.*, vol. 21, no. 6, pp. 1087–1092, Jun 1953.
- [39] R. H. Swendsen and J.-S. Wang, "Nonuniversal critical dynamics in Monte-Carlo simulations," *Phys. Rev. Lett.*, vol. 58, no. 2, pp. 86–88, Jan 1987.
- [40] U. Wolff, "Comparison Between Cluster Monte Carlo Algorithms in the Ising Model," *Physics Letters B*, vol. 228, no. 3, pp. 379–382, September 1989.
- [41] E. Marinari and G. Parisi, "Simulated Tempering - a New Monte-Carlo Scheme," *Europhysics Letters*, vol. 19, no. 6, pp. 451–458, July 1992.
- [42] W. Hastings, "Monte-Carlo Sampling Methods Using Markov Chains And Their Applications," *Biometrika*, vol. 57, no. 1, pp. 97–107, 1970.
- [43] W. Jorgensen, "Perspective on "Equation of state calculations by fast computing machines","" *Theoretical Chemistry Accounts*, vol. 103, no. 3-4, pp. 225–227, February 2000.
- [44] S. Chib and E. Greenberg, "Understanding the Metropolis-Hastings Algorithm," *American Statistician*, vol. 49, no. 4, pp. 327–335, November 1995.
- [45] A. Linke, D. W. Heermann, P. Altevogt, and M. Siegert, "Large-scale simulation of the two-dimensional kinetic Ising model," *Physica A: Statistical Mechanics and its Applications*, vol. 222, no. 1-4, pp. 205–209, December 1995.
- [46] K. Okano, L. Schülke, K. Yamagishi, and B. Zheng, "Universality and scaling in short-time critical dynamics," *Nuclear Physics B*, vol. 485, no. 3, pp. 727–746, February 1997.
- [47] U. Fulco, L. Lucena, and G. Viswanathan, "Efficient search of critical points in Ising-like systems," *Physica A: Statistical Mechanics and its Applications*, vol. 264, no. 1-2, pp. 171–179, February 1999.
- [48] F. Lima and D. Stauffer, "Ising model simulation in directed lattices and networks," *Physica A: Statistical Mechanics and its Applications*, vol. 359, no. 1, pp. 423–429, January 2006.
- [49] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-diminsionally equidistributed uniform pseudorandom number generator," *ACM Transactions on Modeling and Computer Simulation*, vol. 8 No 1., pp. 3–30, 1998.
- [50] D. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, 3rd ed. Addison-Wesley, 1997, vol. 2.
- [51] P. D. Coddington and S. H. Ko, "Techniques for empirical testing of parallel random number generators," in *Proc. International Conference on Supercomputing (ICS'98)*, 1998, dHPC-025.
- [52] G. Marsaglia, "Random numbers fall mainly in the planes," *Proc. Natl. Acad. Sci.*, vol. 61, no. 1, pp. 25–28, 1968.
- [53] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes - The Art of Scientific Computing*, 3rd ed. Cambridge, 2007, ISBN 978-0-521-88407-5.
- [54] G. Marsaglia and W. W. Tsang, "Some Difficult-to-pass Tests of Randomness," *Journal of Statistical Software*, vol. 7, no. 3, pp. 1–9, January 2002.
- [55] G. Marsaglia, "Diehard Battery Of Tests Of Randomness," <http://www.stat.fsu.edu/pub/diehard/>, 1995.
- [56] K. Hawick, A. Leist, and D. Playne, "Regular Lattice and Small-World Spin Model Simulations using CUDA and GPUs," *Int. J. Parallel Prog.*, vol. 39, no. CSTN-093, pp. 183–201, 2011.

Crafting a Lightweight Search Engine

Feng-Jen Yang

Department of Mathematics and Information Sciences

University of North Texas at Dallas

Dallas, TX 75241

Abstract – *Web-based search is commonly perceived as a desirable functionality of web sites. Although there are several open source search engines that can be tailored and embedded in a web site for free, those engines tend to be general and may not be easy to adapt them into an efficient search engine within a particular application domain. In this paper, I presented an easy but practical alternative that can be followed by readers as a template to build their own lightweight search engines with only a few programming.*

Keywords: *Lightweight Search Engine, Web-Based Query.*

1 Introduction

As more and more information is digitized and made available on the web, nowadays, online search is popularly used and commonly perceived as an essential functionality of a web site. Although there are several open source search engines, such as ASPSeek [1], Lucene [2], Namazu [3], mnoGoSearch [4], and WebGlimpse [5], available for web programmers to customize and fit into their own business operations, most of those free search engines are originated from general purposes and may not be easy to be reformed into special purpose search engines [6, 7]. Besides, embedding a big volume component into a small scope of operational web site may compromise the search performance due to the big consumption of CUP time.

Another downside of being an open source search engine follower is the limitation on free technical support. Although some voluntary participants are willing to share their know-how through mailing list or online conferencing, most of the advanced consultants are still by payment. To this end, embedding an open source engine into a special purposed web site may not be the best choice despite the openness of its source code. With these concerns in mind, this hands-on project is crafted to provide an easy alternative for those in need of a web-based search engine but embedding an open source engine is not a practical practice. Instead of spending a great deal of time

striving to understand a huge open source search engine and then tailor it to fit into a specific domain. We can actually perform a very similar functionality by crafting a lightweight search engine from the scratch that requires only a few programming.

As an illustrative implementation of this lightweight search engine, I hypothetically confined the search domain into online books lookup and have the search engine perform partial-matched searches instead of exact-matched search to allow some fuzziness during the compare operations. For the purpose of quick prototyping, I am using only three technologies that most of the programmers are familiar with, namely the Microsoft Access, the Active Server Page (ASP) and the HyperText Markup Language (HTML). To ensure that most of the readers can follow and try out this implementation, the rest of this paper is written in an instructional and stepwise manner.

1.1 The Coherence of Search Operations

An effective search engine is not only efficient in the lookup for items but, more importantly, also able to cope with human's partial, fuzzy, or incomplete memory about the keywords and other search criteria. Personally, I perceived this as the *coherence* between general users and the search operations. This expectation can be met by allowing users to perform searches based on their partial or fuzzy memory about the data they are looking for. Technically, this can be achieved by allowing partial and incomplete keywords to be used as search criteria [8]. Indeed, expecting users to spell out complete and correct keywords for the items they are looking for is neither necessary nor practical in real life, since most of the human memory can only be retained for a short term. Fuzziness and uncertainty caused by human's short term memory should be considered and incorporated into the design and implementation of search operations.

1.2 The Replacement for Web Crawler

Unlike a typical search engine that is counting on a web crawler to glean and index information

automatically from the entire World Wide Web. For smaller domains of search, it is not necessary to be overwhelmed by world-wide information. Instead, it could be more efficient, if the crawler is replaced by a supportive database which can be maintained in a regular manner of database design and administration. In this project, I adopted Microsoft Access as the platform for creating and maintaining the database that is running at the backend to support the web-based searches.

2. The Supportive Database

In this demonstrative implementation, a relational database is created and executed at the backend to support the web-based query operations. For simplification, the data contents are minimized on purpose to have only one table that can be created by using the integrated development environment of Microsoft Access in the following steps:

1. Start from a blank database and use the table design wizard to create a table named *Book* with the following schema, in which the *No* field is chosen to be the primary key:

Field Name	Data Type
No	Number
Title	Text
Author	Text

2. Within the same wizard, set the *No* field with the following properties:

General	Lookup
Field Size	Integer
Format	
Decimal Places	0
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	Yes
Indexed	Yes (No Duplicates)
Smart Tags	
Text Align	General

3. Within the same wizard, set the *Title* field with the following properties:

General	Lookup
Field Size	60
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	Yes
Allow Zero Length	No
Indexed	No
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Smart Tags	

4. Within the same wizard, set the *Author* field with the following properties:

General	Lookup
Field Size	60
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	Yes
Allow Zero Length	No
Indexed	No
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Smart Tags	

5. Open the *Book* table and enter the following hypothetical data:

No	Title	Author
1	Introduction to Computer Science	Albert Yang
2	The Theory of Computation	Rose Wang
3	Introduction to Computer Networks	Nancy Chen
4	Advanced Operating Systems	Nancy Chen
5	Introduction to Formal Lanauages	Albert Yang
6	Advanced Database Management	Rose Wang

6. Name the database as *BookDB.accdb* and save it to the folder at:

`C:\inetpub\wwwroot\search1`

Although the above database is very simplified, it does reflect the stereotype of data collections for the purpose of online search. The number of columns as well as the number of tables can be expanded as needed. The whole database can also be further refined by performing a certain level of normalizations on the schema.

3. The User Interface

Since the supportive database is hidden from general users but running at the backend, the search engine must provide a friendly frontend operational interface that allows users to specify their search criteria for their target data. The engine can then go on to look for those items that are partially matched to these criteria. In the context of text-based search, the search criteria are usually represented by a combination of keywords entered from users. To work with human's fuzzy and uncertain memory about their intended data, this search engine relaxes the restriction on search

¹ To use a Windows PC as the web server, the *Internet Information Server (IIS)* component of Windows must be installed. After installing IIS, the *inetpub* folder and its subfolder *wwwroot* are created automatically. The programs have to create the *search* folder as a subfolder of *wwwroot* and save the database at this location.

criteria from a combination of exact keywords to a combination of patterns, i.e., a combination of partial keywords.

The user interface of this query engine is shown in Figure 1. Below the user prompt, there is an input form in which two *textboxes* are arranged to take search criteria from users and two *command buttons* are used to either *submit* the search or *reset* the form. The source code of this interface is listed in Figure 2. This HTML file is named as *index.htm* and saved at the same location where the backend database is located, i.e., *C:\inetpub\wwwroot\search*.

4 The Pattern Matched Search

Since human memory tends to be fuzzy, uncertain, or even partial. A real coherent search engine should be friendly enough to take these memory-caused factors into consideration [9, 10, 11]. To ensure a good alignment between the user's fuzzy memory and the engine's actual search operations, the following SQL

syntax is applied to perform queries based on pattern-matched comparison in which the *LIKE* operator is used in the *WHERE* clause to search for a specified *pattern* in the given *column*. The "%" character is used to define *wildcards* both before and after the pattern:

```
SELECT *
FROM table
WHERE column LIKE "%pattern%"
```

In this manner, a search criterion can be formed by incorporating both certain and uncertain memories from a user. As a result, only those rows of data containing the designated pattern in the given column are extracted. On the other hand, what are before and after the designated pattern in the given column are not concerned.

The source code of these search operations is listed in Figure 3. This ASP program is name as *search.asp* and saved at the same location where the backend database and frontend user interface are located, i.e., *C:\inetpub\wwwroot\search*.

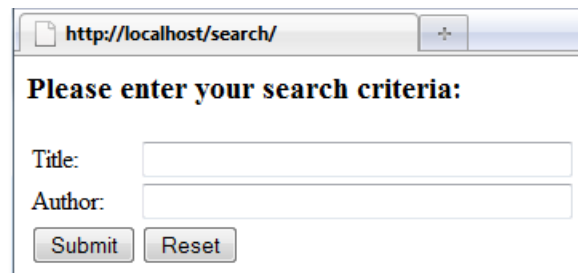


Figure 1. The User Interface

```
<html>
<h3>Please enter your search criteria:</h3>
<form action="search.asp" method="post" name="fromCriteria">
  <table width="200">
    <tr>
      <td>Title:</td>
      <td><input name="txtTitle" type="text" size="60" maxlength="60"></td>
    </tr>
    <tr>
      <td>Author:</td>
      <td><input name="txtAuthor" type="text" size="60" maxlength="60"></td>
    </tr>
    <tr>
      <td><input type="submit" value="Submit"></td>
      <td><input type="reset" value="Reset"></td>
    </tr>
  </table>
</form>
</html>
```

Figure 2. The HTML Code of User Interface


```

<html>

  <%

    dim stream    'the input stream
    dim conn      'the connection object
    dim results   'the query result
    dim no, title, author 'the table data in a row

    'create the input stream
    stream = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" & Server.MapPath("BookDB.accdb")

    'attach the input stream to the database
    set conn = server.createobject("adodb.connection")
    conn.open stream

    'get search criteria (keywords) from user
    keyTitle = request.form("txtTitle")
    keyAuthor = request.form("txtAuthor")

    'run the query and keep the result
    sql = "select * from Book where Title like '%" & keyTitle & "%' And Author like '%" & keyAuthor &
    "%'"
    set results = conn.execute(sql)

  %>

  <h3>The Search Result:</h3>

  <table border="1">

    <tr><th>No</th><th>Title</th><th>Author</th></tr>

    <%

      'display the results in a table
      while (not results.eof)

        'get table data
        no = results("No")
        title = results("Title")
        author = results("Author")

        'write a row
        response.write("<tr><td>" & no & "</td><td>" & title & "</td><td>" & author & "</td></tr>")

        'go on to the next row
        results.movenext

      wend

      'close the database
      conn.close
      Set conn = nothing
      set results = nothing

    %>

  </table>

</html>

```

Figure 3. The ASP Code of Search Operations

5. Sample Executions

After complete afore mentioned implementations, this search engine is ready to operate at the follow web address:

http://localhost/search

It can be operated in a manner similar to the operations of most web-based search operations. Some sample executions are illustrated as follows:

1. To look for all books having “intro” in the *Title* filed and having “an” in the *Author* field:

http://localhost/search/

Please enter your search criteria:

Title:

Author:

After clicking the *submit* button, the search results are shown as follows:

http://localhost/search/search.asp

The Search Result:

No	Title	Author
1	Introduction to Computer Science	Albert Yang
3	Introduction to Computer Networks	Nancy Chen
5	Introduction to Formal Lanauages	Albert Yang

2. To look for all books having “rose” in the *Author* filed:

http://localhost/search/

Please enter your search criteria:

Title:

Author:

After clicking the *submit* button, the search results are shown as follows:

http://localhost/search/search.asp

The Search Result:

No	Title	Author
2	The Theory of Computation	Rose Wang
6	Advanced Database Management	Rose Wang

3. To look for all books having “adv” in the *Title* filed:

http://localhost/search/

Please enter your search criteria:

Title:

Author:

After clicking the *submit* button, the search results are shown as follows:

http://localhost/search/search.asp

The Search Result:

No	Title	Author
4	Advanced Operating Systems	Nancy Chen
6	Advanced Database Management	Rose Wang

4. The entire database can be browsed by leaving both criteria blank:

http://localhost/search/

Please enter your search criteria:

Title:

Author:

After clicking the *submit* button, the search results are shown as follows:

http://localhost/search/search.asp

The Search Result:

No	Title	Author
1	Introduction to Computer Science	Albert Yang
2	The Theory of Computation	Rose Wang
3	Introduction to Computer Networks	Nancy Chen
4	Advanced Operating Systems	Nancy Chen
5	Introduction to Formal Lanauages	Albert Yang
6	Advanced Database Management	Rose Wang

6 Conclusion

The modernization of communication technologies has turned the web into a global platform where people can work across time and space. This digital revolution is making web-based search operations more vitally demanded than ever. As a result, an effective search engine is no longer just a facilitating feature, but, more realistically, it has become a core function.

In some way, the effectiveness of web search operations is empowered by the swiftness of response time as well as the easiness of operations. Although web programmers can tailor and embed open source search engines into their search domains, the customization from general purposes into special purposes may be time consuming and not a practical choice.

This project demonstrated an easier alternative that most of the readers can follow to craft their own lightweight search engines to suit their own search domains. With this self-defined search engine, users can flexibly turn their fuzzy memories about the intended data into combinational fuzzy search criteria. By relaxing or tensing the fuzziness of search, users can easily extend up or narrow down the search results until the target data is located. This approach is practical, flexible and, more importantly, it requires only a few programming.

7 References

- [1] The ASPSeek open source search engine is available at <http://www.aspseek.org>.
- [2] The Lucene open source search engine is available at <http://lucene.apache.org>.
- [3] The Namazu open source search engine is available at <http://www.namazu.org>.
- [4] The mnoGoSearch open source search engine is available at <http://www.mnogosearch.org>.
- [5] The WebGlimpse open source search engine is available at <http://webglimpse.net>.
- [6] Micarelli, A., Gasparetti, F., Sciarrone, F. and Gauch S. 2007. Personalized Search on The World Wide Web. Brusilovsky, P., Kobsa, A. and Nejdl, W. (Eds.), *The Adaptive Web: Methods and Strategies of Web Personalization*, Heidelberg, Germany: Springer, pp. 195-230.
- [7] Burke, R. 2007. Hybrid Web Recommender Systems. Brusilovsky, P., Kobsa, A. and Nejdl, W. (Eds.), *The Adaptive Web: Methods and Strategies of Web Personalization*, Heidelberg, Germany: Springer, pp. 377-408.
- [8] Zou, L., Chen, L. and Özsu, M. 2009. Distance-Join: Pattern Match Query in a Large Graph Database, *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB09)*, Lyon, France, pp. 886-897.
- [9] Jonas S Karlsson, J. and Kersten, M. 2000. O-Storage: A Self Organizing Multi-Attribute Storage Technique for Very Large Main Memories, *Proceedings of the Australasian Database Conference (ADC '00)*, City, State, pp. 57-64.
- [10] Ford, N., Miller, D. and Moss, N. 2005. Web Search Strategies and Human Individual Differences: Cognitive and Demographic Factors, Internet attitudes, and approaches. *Journal of the American Society for Information Science and Technology*, Volume 56, Issue 7, pp. 741 - 756.
- [11] Ford, N., Miller, D. and Moss, N. 2005. Web Search Strategies and Human Individual Differences: A Combined Analysis. *Journal of the American Society for Information Science and Technology*, Volume 56, Issue 7, pp. 757 - 764.

A 4 out of n Secret Sharing Scheme in Visual Cryptography without Expansion

Ying-Yu Chen

Department of Computer Science and Information
Engineering, National Chi Nan University
Puli, Nantou 54561, Taiwan, R.O.C.
e-mail: s99321514@ncnu.edu.tw

Justie Su-Tzu Juan*

Department of Computer Science and Information
Engineering, National Chi Nan University
Puli, Nantou 54561, Taiwan, R.O.C.
e-mail: jsjuan@ncnu.edu.tw

Abstract – *Visual cryptography (VC, for short) encrypts the secret image into n shares (transparency). We cannot see any information from any one share, and decrypt the original image by stacking all of the shares. Now, we extend it to the k out of n secret sharing scheme. (k, n) threshold secret sharing scheme encrypts as the same way and decrypts the original image by stacking at least k shares. If one stacks less than k shares, he (or she) cannot recognize the secret image. In this paper, we construct a new scheme for $(4, n)$ threshold secret sharing encrypt in VC by using a method of combination and the size of the share is as small as the original image. That is, there is no expanded need while some of the previous scheme need.*

Keywords-*visual cryptography; secret sharing scheme; security; share.*

I. INTRODUCTION

Visual cryptography (VC, for short) and the (k, n) -threshold secret sharing scheme were proposed by Naor and Shamir in 1995 [9]. Visual cryptography means the secret image is turned into n shares that combined with black and white pixel, and the decrypting by stacking the shares together to reveal the secret image. So we can decrypt the secret image by human's eye without using computer. The (k, n) -threshold secret sharing scheme means a dealer sends a share to each of the n users. In the condition of k is small or equal than n , the fewer than k users stack their shares together, they cannot see any information from the image. But at least k users stack their shares together, they will find out the secret from the image.

In most of the VC scheme [2, 4, 7, 8, 9, 10, 12], the pixel of each share will expand. The more value of n is, the more value of the expansion will be. However, the size of the share is larger than the original image. In 1995, Naor and Shamir [9] proposed some (k, n) -threshold secret sharing in VC for three kinds of condition. First, some of their schemes are the efficient solutions for $(2, n)$ and $(3, n)$. Second, they propose a general k out of k scheme.

Third, they propose a general k out of n scheme when k is small or equal than n . But the pixel of each share will expand in their later two methods. For convenience, the third method is called NS scheme in this paper.

In 2008, Fang et al. propose a new algorithm (called FLL scheme in this paper) [6]. They solve the problem in expansion. In FLL scheme, the authors use Hilbert-curve [1] and two queues to present a VC scheme. The shares they generate are as small as the input image S , so the pixel of each share won't expand. But since Fang et al. use Naor and Shamir's scheme to design their scheme, the more number of the expansion in Naor and Shamir's scheme is, the image we decrypted will be more unclear in FLL scheme.

Another subject has been considered these years, *progressive visual secret sharing* (PVSS, for shout) scheme [3, 5]. In 2011, Hou et al. propose a new algorithm for $(2, n)$ threshold PVSS scheme [3].

In above researches, no matter the shares will be expansion or not, it cannot reveal the secret image by stacking less than k shares and it can reveal the secret image by stacking at least k shares for $k \geq 4$. Because the expansion in the most of $(4, n)$ -threshold secret sharing scheme in VC is quite large and the image we decrypted will be not clear in FLL scheme for $(4, n)$ -threshold secret sharing scheme in VC. Hence, we propose a new scheme to improve it. We use the theory of combination to construct the scheme. Actually, our scheme is also a $(4, n)$ threshold PVSS scheme. The detail of our scheme is presented on next section. Some experiment results are given in section III, and the conclusion is stated in section IV.

II. THE PROPOSED SCHEME

We will show our $(4, n)$ -threshold secret sharing scheme in VC as follows. For convenience, let

*Corresponding author.

$$C_j^n = \begin{cases} \frac{n!}{(n-j)!j!} & , \text{if } 0 \leq j \leq n; \\ 0 & , \text{otherwise.} \end{cases}$$

Definition 1. An $n \times m$ 0-1 matrix $M(n, j)$ is called *totally symmetric* if each column has the same weight, say j , and m equal to the number of C_j^n and every column vectors are difference to each others, where the *weight* of a column vector means the sum of each entry in this column vector.

Definition 2. Given an $n \times m_1$ matrix A and an $n \times m_2$ matrix B , we define:

1. $[A||B]$ be an $n \times (m_1 + m_2)$ matrix that obtained by concatenating A and B ;
2. $[a \times A || b \times B]$, for any two positive integer a and b , be an $n \times (a \times m_1 + b \times m_2)$ matrix that obtained by concatenating A for a times and B for b times.

Lemma 1. Give an $n \times m$ totally symmetric matrix $A = M(n, j)$, For $i = 1, 2, \dots, n$, let $f_i(A)$ represent the Hamming weight of the row vector that is the result of applying “or” operation for any i rows in A . Then $f_i(A) = f_i(M(n, j)) = C_j^n - C_j^{n-i}$.

Proof. The number of column in $M(n, j)$ is equal to C_j^n . When we choose any i rows, because if any one column vector has all zeros in these i rows, the result entry for applying “or” operation for these row vectors still will be zero. In the other way, if any one column vector has all zeros in these i rows, there must has j ones show on the other $n - i$ rows. Hence, the number of those kind column vectors is C_j^{n-i} . So $f_i(M(n, j))$ equals to the number of all columns subtract the number of columns that has all zeros in these i rows (and has j ones in the other $n - i$ rows). Hence, $f_i(M(n, j)) = C_j^n - C_j^{n-i}$. \square

We use an example to demonstrate Lemma 1:

Example 1. Let $A = M(4, 2) =$

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

Then we have $f_1(A) = 3, f_2(A) = 5, f_3(A) = 6$, and $f_4(A) = 6$. \square

Lemma 2. $f_i([A||B]) = f_i(A) + f_i(B)$ for any two totally

symmetric matrices A and B .

Proof. A matrix $[A||B]$ is obtained by concatenating matrices A and B . Let A denote an $n \times m_1$ matrix, B denote an $n \times m_2$ matrix, and $A = M(n, j_A), B = M(n, j_B)$. Then $f_i([A||B]) = f_i([M(n, j_A)||M(n, j_B)])$. Because do Hamming weight for the resulting row vector of applying “or” operation for some i row of $[A||B]$ is equal to that $[A||B]$ be divided into two parts A, B and do the same thing to these two parts, then add those two results together. According to the above reason, $f_i([M(n, j_A)||M(n, j_B)]) = f_i(M(n, j_A)) + f_i(M(n, j_B))$. That is, $f_i([A||B]) = f_i(A) + f_i(B)$. \square

Let $[A_1||A_2||\dots||A_k] = [\dots[[A_1||A_2]||A_3]|\dots||A_k]$ for any matrices A_1, A_2, \dots, A_k , with the size of A_i is $n \times m_i$ for $i = 1, 2, \dots, k$. We have the following corollary.

Corollary 1. For any k totally symmetric matrices A_1, A_2, \dots, A_k , where $A_i = M(n, j_i)$ for some $0 \leq j_i \leq n$ for any $1 \leq i \leq k$. Let $[A_1||A_2||\dots||A_k] = B$. Then $f_i(B) = f_i(A_1) + f_i(A_2) + \dots + f_i(A_k)$.

Definition 3. The *light transmission rate* $\mathfrak{I}(S) = w / p = 1 - (b / p)$, where w means the number of the white pixel in the image S , p means the number of the all pixel in the image S , b means the number of the black pixel in the image S .

Definition 4. Given totally symmetric matrices A_1, A_2, \dots, A_k . For any $1 \leq i \leq k, A_i = M(n, j_i)$ for some $0 \leq j_i \leq n$. Let $[A_1||A_2||\dots||A_k] = B$ and B is an $n \times m$ matrix. Define $\mathfrak{I}(B, k) = 1 - (f_k(B) / m)$, where $k \leq n$.

To construct the $(4, n)$ secret sharing scheme, we will construct two matrices, C_0 and C_1 , which will be used for constructing n shares later. Since we will choose any one column vector of C_0 (or C_1 , respectively) randomly when construct the pixels of n shares according to a white pixel of secret image S (or a black pixel of secret image S , respectively), we have to follow two conditions:

- $\mathfrak{I}(C_0, k) = \mathfrak{I}(C_1, k)$ for $1 \leq k \leq 3$.
- $\mathfrak{I}(C_0, k) > \mathfrak{I}(C_1, k)$ for $k \geq 4$.

The first rule ensures C_0 and C_1 have the same light transmission rate when k is between one and three, so it won't see any information when we stack less than 4 shares. The second ensure C_0 and C_1 have the different light transmission rate when k is larger than three, and the light transmission rate for the white pixel of secret image S is greater than that for the black pixel of secret image S , so it can reveal the

secret image. Hence we can finger out the secret when stacking at least 4 shares. For the following algorithm, let $m = n^2 - 2n = (n-3)C_1^n + C_{n-1}^n = C_2^n + (n-2)C_n^n + \frac{n^2-5n+6}{2}C_0^n$.

(4, n) scheme algorithm:

Input: A binary secret image S with size $w \times h$ and the value of n .

Output: n shares R_1, R_2, \dots, R_n , each with size $w \times h$.

1. Let $C_0 = [M(n, 2) \parallel (n-3) \times M(n, n) \parallel ((n^2 - 5n + 6) / 2) \times M(n, 0)]$ and $C_1 = [(n-3) \times M(n, 1) \parallel M(n, n-1)]$, the size of C_0 and C_1 are both $n \times m$.
2. for $(1 \leq i \leq h; 1 \leq j \leq w)$
 $t = \text{random}(1..m);$
 for $(1 \leq k \leq n)$
 if $(S(i, j) == 0)$
 $R_k(i, j) = C_0(k, t);$
 else
 $R_k(i, j) = C_1(k, t);$

Theorem 1. In the proposed scheme, we stack at least four shares can reveal the secret, and stack one, two or three shares cannot.

Proof. We use the light transmission rate to prove that we cannot recognize the secret if we stack less than four shares, but we can see the image if at least four shares stack together. We divided into five cases. The first three cases are to prove that if we stack less than four shares, the light transmission rate for the white and black pixel of the stacked image are in the same. The last two cases prove the light transmission rate for the white pixel of the stacked image is larger than the light transmission rate for the black pixel of the stacked image when at least four shares stack together. Therefore we can obey the $(4, n)$ -threshold secret sharing scheme. For $A = C_0$ or C_1 , consider the proposed algorithm, the definition of $f_i(A)$ and $\mathcal{I} = 1 - (b/p)$, we have $\mathcal{I} = 1 - (f_i(A) / m) = \mathcal{I}(A, k)$. Note that Lemma 1 will be used in the following proof.

Case 1. For any one share

$$\begin{aligned} \mathcal{I}(C_0, 1) &= 1 - \frac{f_1\left(\left[M(n, 2) \parallel (n-3) \times M(n, n) \parallel \left(\frac{n^2-5n+6}{2}\right) \times M(n, 0)\right]\right)}{m} \\ &= 1 - \frac{f_1(M(n, 2)) + (n-3) \times f_1(M(n, n)) + \left(\frac{n^2-5n+6}{2}\right) \times f_1(M(n, 0))}{n^2 - 2n} \\ &= \frac{n^2 - 4n + 4}{n^2 - 2n}, \end{aligned}$$

$$\begin{aligned} \mathcal{I}(C_1, 1) &= 1 - \frac{f_1\left(\left[(n-3) \times M(n, 1) \parallel M(n, n-1)\right]\right)}{m} \\ &= 1 - \frac{(n-3) \times f_1(M(n, 1)) + f_1(M(n, n-1))}{n^2 - 2n} = \frac{n^2 - 4n + 4}{n^2 - 2n}. \end{aligned}$$

So $\mathcal{I}(C_0, 1) = \mathcal{I}(C_1, 1)$ when one get one share, and one cannot see any information.

Case 2. Stack any two shares

$$\begin{aligned} \mathcal{I}(C_0, 2) &= 1 - \frac{f_2\left(\left[M(n, 2) \parallel (n-3) \times M(n, n) \parallel \left(\frac{n^2-5n+6}{2}\right) \times M(n, 0)\right]\right)}{m} \\ &= 1 - \frac{f_2(M(n, 2)) + (n-3) \times f_2(M(n, n)) + \left(\frac{n^2-5n+6}{2}\right) \times f_2(M(n, 0))}{n^2 - 2n} \\ &= \frac{n^2 - 5n + 6}{n^2 - 2n}, \\ \mathcal{I}(C_1, 2) &= 1 - \frac{f_2\left(\left[(n-3) \times M(n, 1) \parallel M(n, n-1)\right]\right)}{m} \\ &= 1 - \frac{(n-3) \times f_2(M(n, 1)) + f_2(M(n, n-1))}{(n-3) \times C_1^n + C_{n-1}^n} = \frac{n^2 - 5n + 6}{n^2 - 2n}. \end{aligned}$$

So $\mathcal{I}(C_0, 2) = \mathcal{I}(C_1, 2)$. That means when stacking any two shares, we cannot see any information.

Case 3. Stack any three shares

$$\begin{aligned} \mathcal{I}(C_0, 3) &= 1 - \frac{f_3\left(\left[M(n, 2) \parallel (n-3) \times M(n, n) \parallel \left(\frac{n^2-5n+6}{2}\right) \times M(n, 0)\right]\right)}{m} \\ &= 1 - \frac{f_3(M(n, 2)) + (n-3) \times f_3(M(n, n)) + \left(\frac{n^2-5n+6}{2}\right) \times f_3(M(n, 0))}{n^2 - 2n} \\ &= \frac{n^2 - 6n + 9}{n^2 - 2n}, \\ \mathcal{I}(C_1, 3) &= 1 - \frac{f_3\left(\left[(n-3) \times M(n, 1) \parallel M(n, n-1)\right]\right)}{m} \\ &= 1 - \frac{(n-3) \times f_3(M(n, 1)) + f_3(M(n, n-1))}{n^2 - 2n} = \frac{n^2 - 6n + 9}{n^2 - 2n}. \end{aligned}$$

Again, $\mathcal{I}(C_0, 3) = \mathcal{I}(C_1, 3)$ when stacking any three shares, so we cannot see any information.

Case 4. Stack any four shares

$$\begin{aligned} \mathcal{I}(C_0, 4) &= 1 - \frac{f_4\left(\left[M(n, 2) \parallel (n-3) \times M(n, n) \parallel \left(\frac{n^2-5n+6}{2}\right) \times M(n, 0)\right]\right)}{m} \\ &= 1 - \frac{f_4(M(n, 2)) + (n-3) \times f_4(M(n, n)) + \left(\frac{n^2-5n+6}{2}\right) \times f_4(M(n, 0))}{n^2 - 2n} \\ &= \frac{n^2 - 7n + 13}{n^2 - 2n}, \\ \mathcal{I}(C_1, 4) &= \end{aligned}$$

$$= 1 - \frac{f_4([(n-3) \times M(n,1) || M(n,n-1)])}{m}$$

$$= 1 - \frac{(n-3) \times f_4(M(n,1)) + f_4(M(n,n-1))}{n^2 - 2n} = \frac{n^2 - 7n + 12}{n^2 - 2n}.$$

Hence, $\mathcal{I}(C_0, 4) > \mathcal{I}(C_1, 4)$, so we could recognize the secret from the image.

Case 5. Stack any t shares, $t > 4$.

$$\mathcal{I}(C_0, t)$$

$$= 1 - \frac{f_t\left(\left[M(n,2) || (n-3) \times M(n,n) || \left(\frac{n^2-5n+6}{2}\right) \times M(n,0)\right]\right)}{m}$$

$$= 1 - \frac{f_t(M(n,2)) + (n-3) \times f_t(M(n,n)) + \left(\frac{n^2-5n+6}{2}\right) \times f_t(M(n,0))}{n^2 - 2n},$$

$$\mathcal{I}(C_1, t)$$

$$= 1 - \frac{f_t([(n-3) \times M(n,1) || M(n,n-1)])}{m}$$

$$= 1 - \frac{(n-3) \times f_t(M(n,1)) + f_t(M(n,n-1))}{n^2 - 2n}.$$

Because $\mathcal{I}(A, t) = 1 - (f_t(A) / m)$ and the value of m in C_0 and C_1 are the same, we could only use $f_t(A)$ to compare $\mathcal{I}(C_0, t)$ and $\mathcal{I}(C_1, t)$. If $f_t(C_0) < f_t(C_1)$ then $\mathcal{I}(C_0, t) > \mathcal{I}(C_1, t)$. Since

$$f_t(C_0)$$

$$= f_t(M(n,2)) + (n-3) \times f_t(M(n,n)) + \left(\frac{n^2-5n+6}{2}\right) \times f_t(M(n,0))$$

$$= C_2^n - C_2^{n-t} + (n-3)C_n^n$$

$$= \{(n^2 - n) - (n-t)^2 + (n-t)\} / 2 + n - 3$$

$$= nt + n - t(t+1) / 2 - 3$$

$$< nt + n - 3t$$

$$= (n-3)(n - (n-t)) + (n-0)$$

$$= (n-3)(C_1^n - C_1^{n-t}) + (C_{n-1}^n - C_{n-1}^{n-t})$$

$$= (n-3) \times f_t(M(n, 1)) + f_t(M(n, n-1))$$

$$= f_t(C_1).$$

Where $-t(t+1) / 2 - 3 < -3t$ is hold because $t(t-5) / 2 + 3 > 0$ for any $t \geq 5$. Hence, $\mathcal{I}(C_0, t) > \mathcal{I}(C_1, t)$ when $t > 4$, and we can recognize the secret from the stacked image. \square

III. EXPERIMENTAL RESULT

We use (4, 5)-threshold secret sharing scheme as an example. The input n is 4, the C_0 and C_1 that the algorithm generated are:

$$C_0 = [M(n, 2) || (n-3) \times M(n, n) || ((n^2 - 5n + 6) / 2) \times$$

$$M(n, 0)] = [M(5, 2) || 2 \times M(5, 5) || 3 \times M(5, 0)] =$$

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix},$$

$$C_1 = [(n-3) \times M(n, 1) || M(n, n-1)] = [2 \times M(5, 1) ||$$

$$M(5, 4)] =$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

The experiment results show in Figure 1. We use an image (NCNU) to be the secret image as Figure 1.(a) and we encrypt the secret image into 5 shares as Figure 1.(b), (c), (d), (e), and (f). We show the result of stacking share 1 and share 2 in Figure 1.(g) and the result of stacking any other two shares are almost the same, so we just show one of combined image that stacks any two shares. Therefore, we cannot see any information when stacking any two shares. We show the result of stacking share 1, share 2 and share 3 in Figure 1.(h) and the result of stacking any other three shares are almost the same, so we just show one of combined image that stacks any three shares. Again, we cannot see any information when stacking any three shares. So it can not reveal any information when stack less than any four shares. We show the result of stacking share 1, share 2, share 3 and share 4 in Figure 1.(i) and the result of stacking any other four shares are almost the same, so we just show one of combined image that stacks any four shares. By watching the Figure 1.(i), we can see the secret image slightly. Then we show the result of stacking all the shares together in Figure 1.(j) and we can see the secret image clearly. So it reveals the secret when stack at least any four shares and if more and more share stacks together, the secret image will reveal more and more clearly.

IV. CONCLUSIONS

For general $n \geq 4$, we had shown a (4, n)-threshold secret sharing scheme in VC. We construct the scheme by using a method of combination. In [9], the authors defined α which means the relative different in weight between C_0 and C_1 of stacking k shares. It would like α to be as large as possible, so the image will be clearer. The value of

α in (4, 5)-threshold secret sharing scheme of NS scheme is approach to $1 / 4000$. Because Fang et al. use NS scheme to design their scheme, α in FFL scheme is in the same as α in NS scheme. In our scheme, α is equal to $3 / 15 - 2 / 15 = 1 / 15$ in the (4,

5)-threshold secret sharing scheme. Also, for any (4, n)-threshold secret sharing scheme in VC for $n \geq 6$, our proposed scheme has better performance than NS scheme [9] and FLL scheme [6] in the value of α .

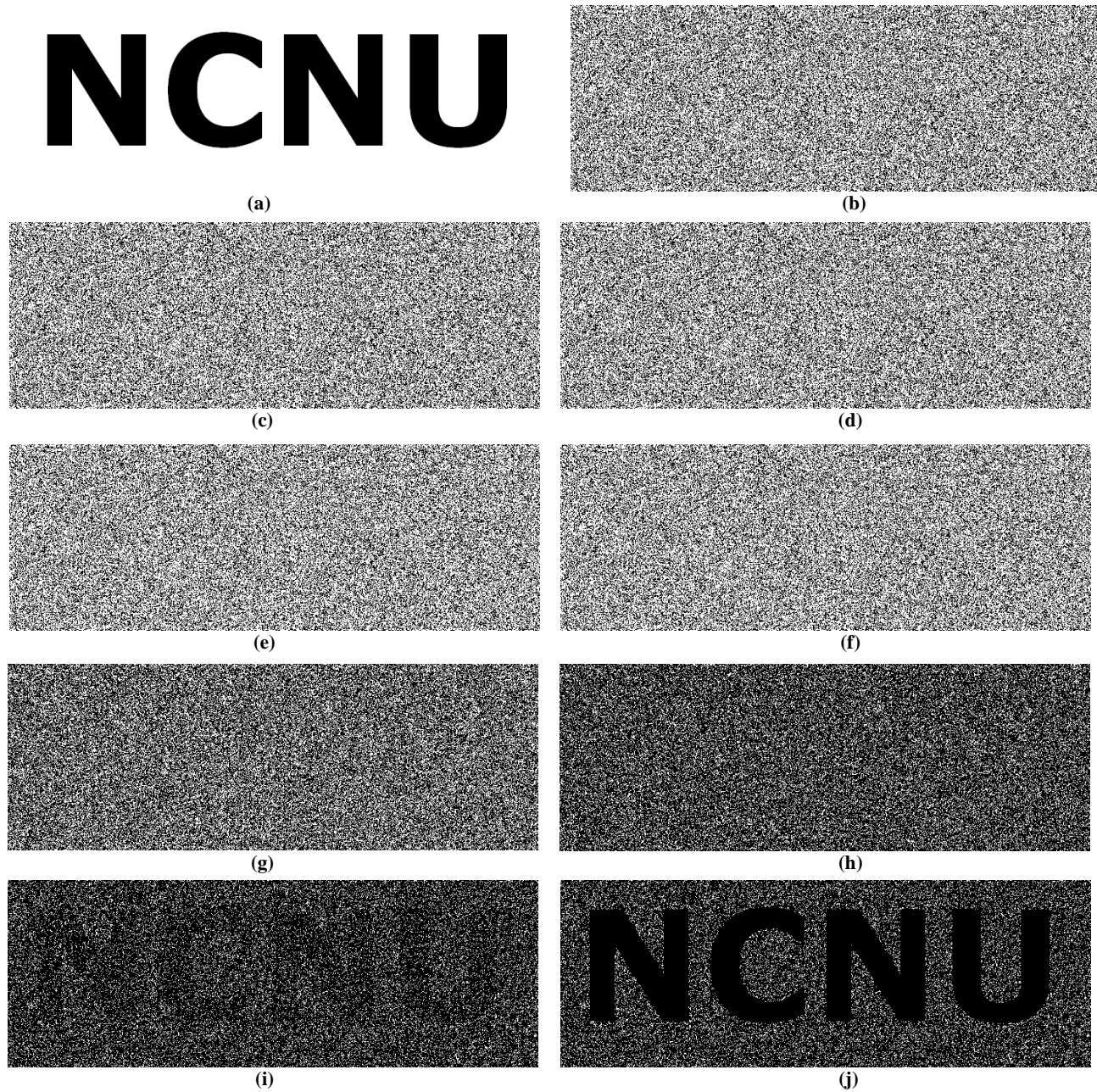


Figure 1. Experimental result of (4, 5)

- (a) Secret image S , (b) Share 1, (c) Share 2, (d) Share 3, (e) Share 4, (f) Share 5, (g) The result of stacking share 1 and share 2, (h) The result of stacking share 1, share 2 and share 3, (i) The result of stacking share 1, share 2, share 3 and share 4, (j) The result of stacking all the shares.

Besides, there is no expansion in our scheme which is smaller than the NS scheme [9] and we also generate shares by randomly without using Hilbert-curve while [6] need.

Overall, our proposed scheme reveals the secret image when stacking at least four shares and the secret image will be clearer if stacking more and more shares together. That is, our scheme is a $(4, n)$ threshold PVSS scheme. The advantages of our proposed scheme are that it has no pixel expanded, and has the larger α that the stacked image will be clearer. The future work is to generate the proposed scheme to be a general k out of n secret sharing scheme in VC.

ACKNOWLEDGMENT

This work was partially supported by National Science Council of Taiwan, ROC under Grant No. NSC99-2628-E-2-60-011-.

REFERENCES

- [1] D. Hilbert, "Über die stetige Abbildung einer Linie auf ein Flächenstück," *Mathematische Annalen*, vol. 38, pp. 459—460, 1891.
- [2] C. Blundo, A. De Santis and D. R. Stinson. "On The Contrast in Visual cryptography schemes," *Journal Of Cryptology*, vol. 12, pp. 261—289, 1999.
- [3] Young-Chang Hou, Zen-Yu Quan, "Progressive Visual Cryptography with Unexpanded Shares," *IEEE Transactions on Circuits and Systems for Video Technology*, accepted 2011.
- [4] P. A. Eisen, D.R. Stinson, "Threshold visual cryptography schemes with specified whiteness levels of reconstructed pixels," *Des. Codes Cryptogr.* 25, pp. 15—61, 2002.
- [5] W. P. Fang, J. C. Lin, "Progressive viewing and sharing of sensitive images," *Pattern Recognition and Image Analysis*, Vol.16, no. 4, pp. 638—642, 2006
- [6] Wen-Pinn Fang, Sen-Jen Lin, Ja-Chen Lin, "Visual cryptography (VC) with non-expanded shadow images: a hilbert-curve approach," *Proceeding on IEEE International Conference on Intelligence and Security Informatics (ISI2008)*. Grand Formosa Regent Hotel, Taipei, Taiwan, pp. 271—272, 2008.
- [7] N. Linial and N. Nisan, "Approximate inclusion-exclusion," *Combinatorica* 10, pp. 349—365, 1990.
- [8] C. C. Lin and W. H. Tsai, "Secret multimedia information sharing with data hiding capability by simple logic operations," *Pattern Recognition and Image Analysis*, vol. 14(4), pp. 594—600, 2004.
- [9] M. Naor and A. Shamir, "Visual cryptography," *Eurocrypt'94*, *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, vol. 950, pp. 1—12, 1995.
- [10] T. Hofmeister, M. Krause and H. U. Simon, "Contrast-Optimal k out of n Secret Sharing Schemes in Visual Cryptography," *Theory of Computer Science*, vol. 240, pp. 471—485, 2000.
- [11] S.-J. Shyu, "Image encryption by multiple random grids," *Pattern Recognition*, vol. 42, no. 7, pp. 1582—1596, 2009.
- [12] C. N. Yang, "New Visual secret sharing schemes using probabilistic method," *Pattern Recognition Letters*, vol. 25(4), pp.481—495, 2004.

Presenting the Test Cell Algorithm for Solving Sudoku Puzzles

Tom Kigezi

Department of Electrical and Computer Engineering, Makerere University, Kampala, Uganda

Abstract— *Sudoku, the logic based combinatorial number-placement puzzle has gained worldwide fame among mathematicians and scientists alike in the field of Computational Game Theory. Notably, a vast majority of computer-based algorithms available for solving these puzzles try to mimic human logic in their implementation, making them liable to errors from puzzle inconsistencies. This paper presents a straightforward computer-based algorithm for solving Sudoku puzzles, hereafter called the Test Cell algorithm. It utilizes a highly efficient backtracking technique that is easy to implement in most available computer programming languages. Test runs of the algorithm, implemented using the C++ computer programming language, have been successful for essentially all valid puzzle difficulty categories.*

Keywords— Sudoku puzzle, Sudoku algorithms, Easy Cell method, Test Cell

1. Introduction

Introduced in Japan in 1984 and made world popular by British newspaper *London Times* in 2005[1]-[2], the Sudoku puzzle has become the passion of many people the world over in the past few years. Sudoku is a logic-based, combinatorial number-placement puzzle whose objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 sub-grids that compose the grid (also called "boxes", "blocks", "regions", or "sub-squares") contains all of the digits from 1 to 9. The puzzle is a partially completed grid and does not necessarily have a single legitimate solution.

Ironically, despite being a game of numbers, Sudoku doesn't demand an iota of mathematics of its solvers. In fact, no operation—including addition or multiplication helps in completing a grid, which in theory could be filled with any set of nine different symbols. Nevertheless, Sudoku presents mathematicians and computer scientists a host of challenging issues viz. the choice of the number of Sudoku grids that can be constructed, the minimal number of starting clues that yield a unique solution, and, whether Sudoku belongs to the N-P complete class of problems[2]-[3].

Unlike manually solving the puzzle with hand-developed algorithms or natural methods[4], computer-based algorithms offer quick solutions to Sudoku puzzles, even with rigorous backtracking - a systematic form of trial and error in which partial solutions are proposed and slightly modified if proved wrong. Challenging puzzles however

tend to require multiple hand-developed solving strategies that may not be easily integrated into a single logically sound and efficient computer-based algorithm. This is a fundamental challenge in writing Sudoku puzzle solving programs that the Test Cell algorithm overcomes.

2. The Test Cell Algorithm

The Test Cell algorithm developed for solving Sudoku puzzles does not mimic human logic or hand-developed algorithms. It is a straightforward, easy to implement, consistent and efficient computer-based algorithm that employs Test Cells - special testing cells determined by a specific criterion, that guide to the ultimate puzzle solution. While absent in typical easy puzzles (as a solution is reached without a need to identify them), Test Cells begin to appear as the difficulty of the puzzle is raised. Though only tested on 9X9 Sudoku grids, the Test Cell algorithm can ideally be scaled to other puzzle dimensions.

2.1 Algorithm Terminology

The Test Cell algorithm is better explained by a preliminary description of all terms pertaining to it in an intended logical order. These are:

Lists- An empty cell has a set of numbers that cannot occupy it (Impossibility List) and consequently, a corresponding set of those that can occupy it (Possibility List). These lists are developed for every empty cell according to the Sudoku game rules.

Invalid Puzzle Configuration- A puzzle configuration in which there is at least one empty cell whose Possibility List is empty. This shall be the *only* test for puzzle validity during the course of solving.

Dead End- If an invalid puzzle configuration is reached during the course of solving a Sudoku puzzle, then we have hit a "Dead End".

Easy Cell- An empty cell whose Possibility List contains only one number and as such the cell can only be filled with that one available number.

Solution- A fully filled Sudoku puzzle according to the Sudoku game rules. For a 9X9 grid, the sum of all numbers in each row, column and region is 45, making the sum of all numbers occupying the grid 405.

Easy Cell method- A method of achieving a solution to the puzzle by only filling any Easy Cells if available until either a solution is reached or there are no more Easy Cells in the puzzle, in which case the puzzle is not yet fully solved. This method is normally sufficient to completely

	7						1	
				4			3	
	9		5				8	2
9	4			8				1
5								
	1			6	4			7
6	3	4	2		7		5	
						9	2	6
	5				6			

Figure 1: Puzzle that may be solved by the Easy Cell method alone

solve easy puzzles. Figure 1 shows a typical puzzle that only requires the Easy Cell method for its solution.

Test Cell- A special empty cell (as determined by the later described procedure) whose Possibility List contains just two numbers, only one of which is the correct one for insertion. It's important to note that not every cell with only two numbers in its Possibility List is a Test Cell. Test Cells are indeed special because of the procedure used to identify them as will be described later. Puzzles typically have a number of Test Cells. The sequence in which they are discovered and correctly filled may or may not lead to solution variations. Therefore, the Test Cell algorithm ideally has the capability to produce all the possible solutions for a given puzzle.

2.2 Determination and Manipulation of Test Cells

In determination of a Test Cell, the following procedure applies to and is only performed for all empty cells with two numbers in their Possibility Lists, in a puzzle whose configuration is such that all Easy Cells have been filled with the Easy Cell method:

- i. Let the puzzle be in configuration X
- ii. Fill/insert the empty cell with one of the numbers in its Possibility List and proceed with solving the puzzle using the Easy Cell method, while registering the number of newly generated and filled Easy Cells.
- iii. Reset the Puzzle to configuration X
- iv. Fill/insert the empty cell with the other number in its Possibility list and proceed with solving the puzzle using the Easy Cell method, while registering the number of newly generated and filled Easy Cells.
- v. Reset the Puzzle to configuration X

Following the above procedure, a Test Cell is identified as the one with a number in the Possibility List which registered the most number of newly generated Easy Cells. For this particular cell, this number is called the *alpha number* while the other number is called the *beta number*.

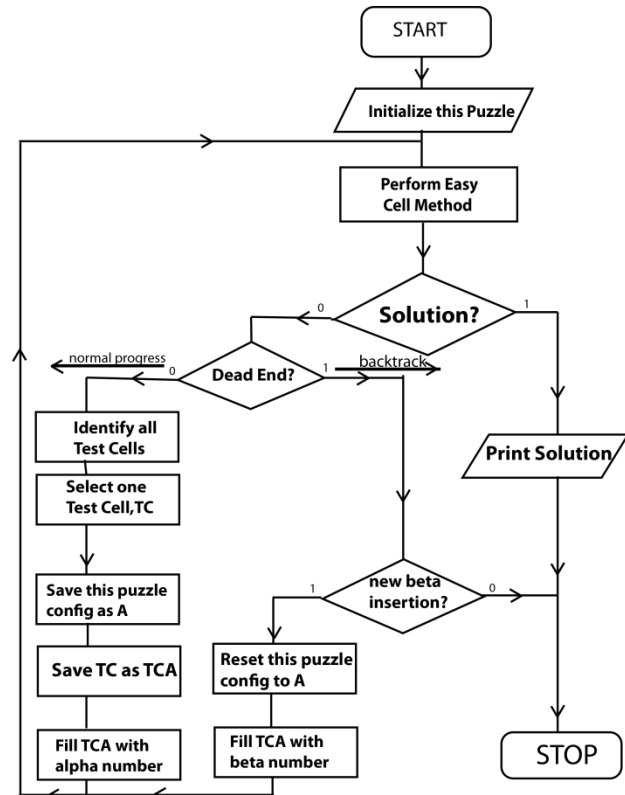


Figure 2: Test Cell algorithm flow chart

Upon realisation of a Test Cell, the alpha number is inserted *first*, as logic would dictate. If the alpha insertion leads to a Dead End, only then is the beta insertion made and considered the correct insertion. This is illustrated by the Test Cell algorithm flow chart shown in Figure 2.

The “new beta insertion” check in the flow chart aims to determine if the beta insertion to be made is following an alpha insertion, as is required. If not, then both the alpha and beta insertions led to a Dead End, a situation that contradicts the Test Cell theory, causing the algorithm to terminate without a solution to the puzzle. A solution to this impasse is the extended Test Cell algorithm which adds an easy extension to the algorithm within the confines of the Test Cell theory.

Simply put, to implement the algorithm, the Easy Cell method is first performed. If it doesn't yield a solution, and this is not a Dead End, identify a Test Cell and make its alpha insertion. If this leads to a Dead End in the next single run, then the puzzle is reset appropriately and the Test Cell's beta insertion is made instead. This must lead to a normal progress in the next run, these steps repeating themselves until a solution is reached. Figure 3 shows a puzzle that can be solved with the Test Cell algorithm.

							9	1
							4	
4	5				6	7		
	7				5			6
3			7		1			
6			3					
	3		8		4		6	
8	6	1			7			2
2		7					8	

Figure 3: A moderately challenging puzzle that can be solved with the Test Cell algorithm.

2.3 Extension to the Test Cell Algorithm

For a guaranteed solution to more difficult Sudoku puzzles, the Test Cell algorithm adopts an easy extension with the use of “BackPuzzle” puzzle configurations. In this arrangement, Test Cells are further classified as *immediate* or *non-immediate* (NI) Test Cells. Immediate Test Cells are the commonest in puzzles falling in the typical Easy, Medium, and Hard categories while non-immediate Test Cells begin to occur in the more difficult puzzle categories.

An immediate Test Cell is one for which insertion of the alpha or beta number leads to a Dead End upon performing the Easy Cell method for the first chosen insertion, and a normal progress (i.e. NOT Dead End) for the other. This is illustrated in Figure 4 and Figure 5 where A and B represent alpha and beta numbers respectively. It is instructive to note that the Test Cell algorithm without the extension assumes only immediate Test Cells for a puzzle.

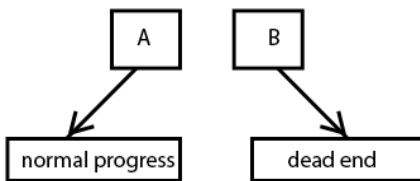


Figure 4: An immediate Test Cell with the alpha number correct and the beta number wrong, after performing the Easy Cell method

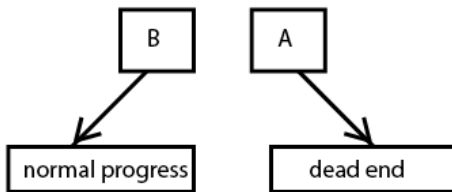


Figure 5: An immediate Test Cell with the alpha number wrong and the beta number correct, after performing the Easy Cell method

A non-immediate (NI) Test Cell on the other hand is one for which either insertion of the alpha or beta number leads to a normal progress upon performing the Easy Cell method, such that either insertion appears to be correct. According to the Test Cell theory however, one of these insertions will eventually produce a Dead End at a later stage if you proceed solving the puzzle with it while the other will not, thereby remaining consistent with the Test Cell definition. This is illustrated in Figure 6 and Figure 7.

The BackPuzzle configuration mentioned earlier is the puzzle configuration at the point when a non-immediate Test Cell is found to exist in the puzzle being solved. BackPuzzle configurations are useful because if both alpha and beta insertions of a Test Cell lead to a Dead End (as is the case when the Test Cell algorithm terminates with no solution to a puzzle), then the current puzzle configuration is reset to the most recent BackPuzzle configuration, and a beta insertion is made in the corresponding non-immediate Test Cell. This is illustrated in the flow chart for the extended Test Cell algorithm shown in Figure 8.

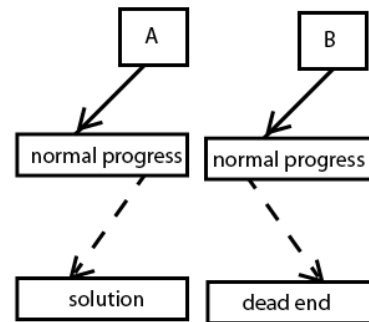


Figure 6: A non-immediate (NI) Test Cell where the alpha number is correct while the beta number is ultimately wrong.

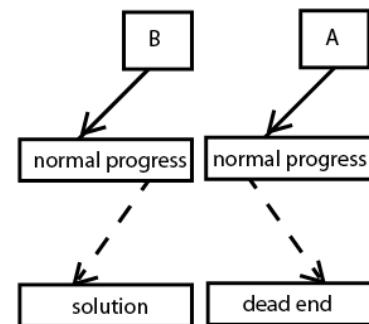


Figure 7: A non-immediate Test Cell where the alpha number is ultimately wrong while the beta number is correct.

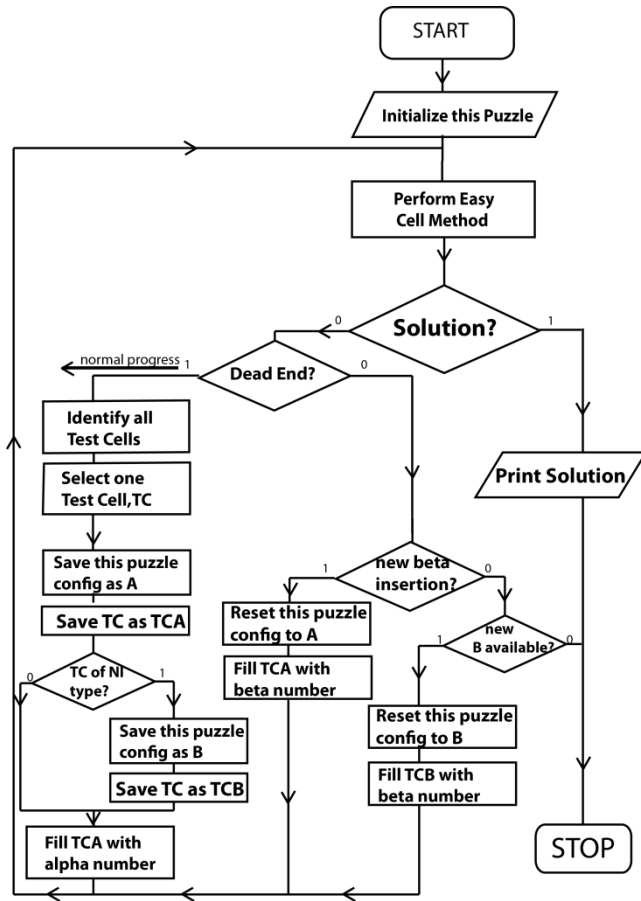


Figure 8: extended Test Cell algorithm flow chart

In a well-written program, the algorithm exhaustively explores all possible hypotheses and finally determines the solution, if one indeed exists. Figure 9 shows a challenging puzzle that can be solved with the extended Test Cell algorithm.

	2			7		8	5	
		6						7
	7		3				1	6
7					8			
	5		9	2	3		7	
			5					3
2	8				1		3	
6						5		
	3	4		5			6	

Figure 9: A challenging puzzle that can be solved with extended Test Cell algorithm

3. Sudoku Puzzle Difficulty Ranking by the Test Cell Algorithm

Sudoku puzzles are often ranked by difficulty. Perhaps surprisingly, the number of givens or "clues" has little or no bearing on a puzzle's difficulty. A puzzle with a minimum number of givens may be very easy to solve, and a Sudoku with more than the average number of givens can still be extremely difficult to solve by hand. Computer solvers can estimate the difficulty for a human to find the solution, based on the complexity of the solving techniques required.

A ranking system by the Test Cell algorithm would suggest that a puzzle's difficulty level for a human player is 'Very-difficult' if it requires the extended Test Cell algorithm to be solved. This relative difficulty further increases with the number of BackPuzzle configurations that are created during the course of solving the puzzle.

4. Conclusion

This paper has presented an alternative computer-based algorithm for solving Sudoku puzzles that utilizes a rather basic technique. Its relative simplicity coupled with its effectiveness merit it as a prime candidate of choice for any device. Tests with the algorithm have been successful with all puzzle categories at <http://www.krazydad.com>, a popular online resource for Sudoku puzzles.

5. References

- [1] Delahaye, Jean-Paul, "The Science Behind Sudoku", *Scientific American* magazine, June 2006.
- [2] J.F. Crook, "A Pencil-and-Paper Algorithm for Solving Sudoku Puzzles" Retrieved December 20, 2010, from the World Wide Web: <http://www.ams.org/notices/200904/rtx090400460p.pdf>
- [3] Kim, Scott, "The Science of Sudoku", 2006
- [4] Mike, Adams, "Sudoku Puzzle Secrets: Learn How to Solve Sudoku Puzzles With Little Effort", 2007

Author

Tom Kigezi is a third-year student of BSc. Electrical Engineering at Makerere University. He is a student researcher with an online laboratories project iLabs@MAK and has a keen interest in robotics, artificial intelligence and automated systems.

Three Heuristic Algorithms for Generation of Random Polygons by a Given Set of Vertices

A. Nourollah¹, L.Mohammadi²

¹ Faculty of Electrical & Computer Engineering of Islamic Azad University, Qazvin, Iran

& Faculty of Electrical & Computer Engineering of Shahid Rajaei Teacher Training University, Tehran, Iran

² Department of Electrical & Computer Engineering of Islamic Azad University, Qazvin, Iran

Abstract - One of the discussed issues in computer graphics and computational geometry is the generation of random polygons. This problem is of considerable importance in the practical evaluation of algorithms that operate on polygons, where it is necessary to check the correctness and to determine the actual CPU-usage of an algorithm experimentally. To this time, no polynomial-time algorithm is known for the random generation of polygons with a uniform distribution. In This paper, three new inventive algorithms are presented for generating random polygons using a set of random vertices using $O(n \log n)$ time.

Keywords: angular scanning, convex hull, random Simple polygon, star-shaped polygon, computer graphics.

1 Introduction

The computer graphics is the science of computing the images which would be led to generate them. One of the important branches of computer graphics is the computational geometry. The computational geometry executes some computing on the geometrical shapes such as polygons. The polygons are appropriate shapes for representing the shapes of real world and every object in the nature would be representable as a set of polygons. For the reason of this issue importance, one of the important discussed issues in computer graphics and computational geometry is the problem of generating random polygons.

The problem of random polygon generation is defined as follows: Given S , a uniformly random polygon on S is a polygon generated with probability $\frac{1}{k}$ if there exist k simple polygon on S in total.

Consider two different pairs of points the two edges defined by these pairs. It is easy to observe that the number of simple polygons containing one of these edges by a dashed line: For the first edge there exactly two simple polygons (Fig 1-a, Fig 1-b) which contain it, whereas for the second edge three simple polygons (Fig 1-c, Fig 1-d, Fig 1-e) containing it exist. Unfortunately no algorithm has been presented yet for the problem of generating random simple polygons with linear time complexity. This motivated researchers to pursue heuristics for generating random simple polygons.

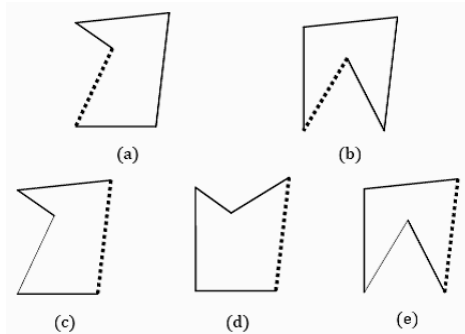


Fig. 1. Two edges belonging to a different number of simple polygons

The generation of random simple polygons has two main areas of application: The empirical verification of an algorithm operating on polygons. The practical testing of its CPU-usage. Too many algorithms have been presented for solving the problem of random polygons and some of them are briefly explained here in this paper [1].

The following sections of this paper has been organized in this way: section 2 has been allocated to related works for generating of random polygons. In section 3, three proposed algorithms would be offered and section 4 investigates the performance of the discussed algorithms and finally in section 5 conclusions would be discussed.

2 The related work

For the reason mentioned in section ago, the random generation of geometric objects has received some attention by researchers: Epstein and Sack presented an $O(n^4)$ algorithm for generating of simple polygons at random [2].

Devroy, Epstein and Sack studied the random generation of intervals and hyperrectangles. They consider the problem of generating a random hyperrectangle in a unit hypercube [3].

Atkinson and Sack studied the uniform generation of forests of restricted height. A k -way tree is either empty or consists of a root node and an ordered sequence of k subtrees. their

algorithms runs in $O(n^3 \cdot h)$, where h denotes the height of the tree[4].

An algorithm running in at most $O(n^2)$ time for the random generation of x -monotone polygons was described by Zhu et al [5]. An interesting approach for the generation of random polygons in the plane (but not on a given set of points) was researched by O'Rourke and Virmany[6]. In the next section three proposed heuristic algorithms are designed for this problem.

3 Proposed algorithms

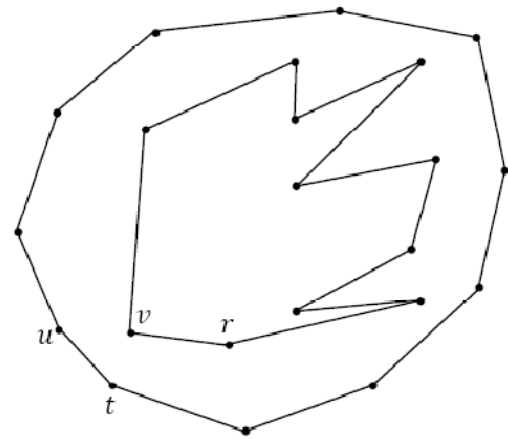
In this section, three proposed algorithms are discussed for generating of random polygons. each algorithm has time complexity $O(n \log n)$.

3.1 Algorithm 1

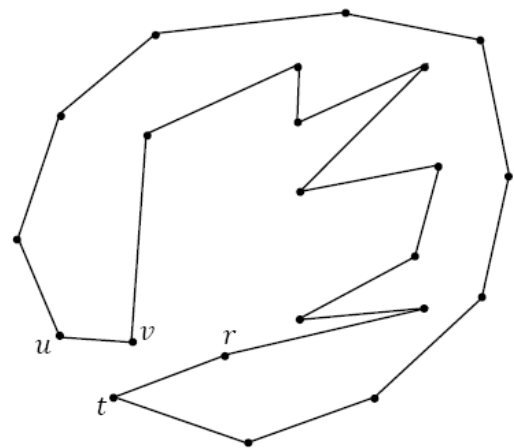
Let S be a set of random vertices and no three points are on the same line. The convex hull, the set of S vertices in the plane ($CH(S)$), of a point set S in the plane is the enclosing convex polygon with smallest area. With this description, this algorithm performs in a way that first $CH(S)$ would be obtained. Let \hat{S} be a set of interior vertices of this convex hull. Then, for the set of those vertices which are inside it will perform in this way, i.e., : first the vertex with the least y -coordinate is found. It is supposed that this vertex is v . The vertices of \hat{S} is sorted according to the polar angle in the counter-clockwise around v . By angular scanning of these vertices around vertex v , visited vertices would be joined together. The last visited vertex is joined to vertex v and in this way a star-shaped polygon would be obtained (that from a point inside it all vertices are visited). Now, from the set of vertices over $CH(S)$, the closest vertex to v is selected and it is supposed to be this vertex u . If the vertex close to v over interior star-shaped polygon in the counter-clockwise is r and the vertex close to vertex u over $CH(S)$ in the counter-clockwise is t (Fig 2-a), so the edges (v,r) , (u,t) with the edges (v,u) , (t,r) are replaced. Thus, a random polygon would be created (Fig 2-b). The procedure of this algorithm is as the following:

- First, $CH(S)$ would be computed over the set of random vertices S .
- The set of interior vertices $CH(S)$ is considered as \hat{S} and the vertex with the least y -coordinate is selected from the set of vertices (vertex v).
- The vertices \hat{S} are visited according to the polar angle in the counter-clockwise and a star-shaped polygon is obtained with angular scanning of vertices around vertex v and joining the sorted vertices.
- The closest vertex to vertex v is selected From the set of vertices over $CH(S)$ (vertex u).
- The vertex close to v and the vertex close to u in the counter-clockwise is called r and t , respectively.

- The edges (v,r) , (u,t) are replaced by the edges (v,u) , (t,r) .



(a)



(b)

Fig. 2. (a) Creating a star-shaped polygon with the points inside the convex hull, (b) creating the random polygon with the replacement of edges (v,r) , (u,t) with the edges (v,u) , (t,r) .

3.2 Algorithm 2

In this algorithm like the previous algorithm, first $CH(S)$ is computed. Let \hat{S} be the vertex set inside of the convex hull. For this set of vertices, the existing convex bottom algorithm would be applied which has $O(n \log n)$ time and performs in this way that first two vertices which has the least and the most x -coordinate is considered. These vertices is joined together using a presumptive line so that the set of vertices would be divided into two upper and lower half (if the set of vertices in the lower half is empty, the presumptive line is considered as the convex hull of two points and the algorithm executes again).

Now, the convex hull of the presumptive line's lower half is computed and the presumptive line is deleted. All remaining vertices would be sorted from left to right in the order of x and the most left and the most right points would be joined to the left and right end vertices of convex hull, respectively. Then, like the above algorithm, vertices v, u, r and t have been considered as it's explain before in the previous algorithm (Fig 3-a) and the edges $(v, r), (u, t)$ is replaced with the edges $(v, u), (t, r)$ (Fig 3-b). the created polygon is a simple random polygon.

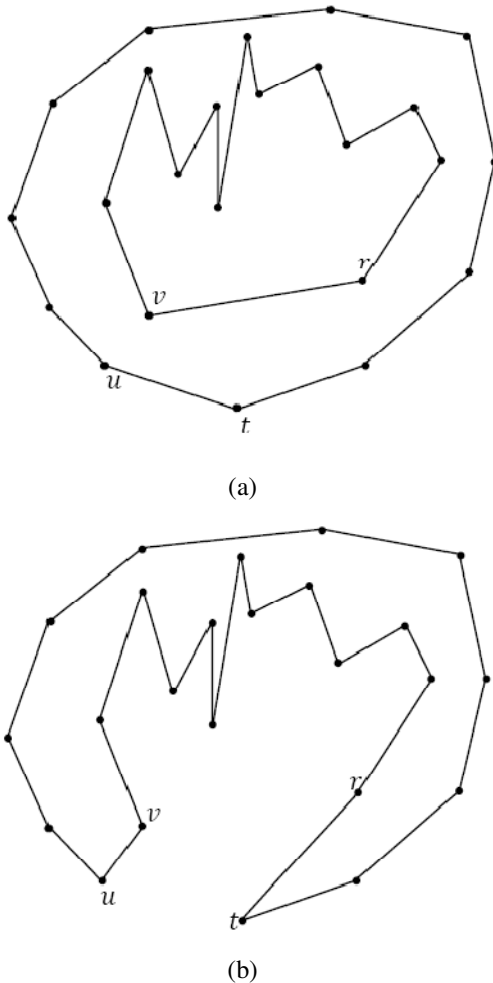


Fig. 3. (a) Creating a polygon with the Convex Bottom algorithm, (b) creating the random polygon by replacing the of edges $(v, r), (u, t)$ with the edges $(v, u), (t, r)$.

The procedure of this algorithm with assumption the existing convex bottom algorithm is as the following:

- First, $CH(S)$ would be computed over the set of random vertices S .
- The set of interior vertices $CH(S)$ is considered as \hat{S} and the vertex with the least y -coordinate is selected from the set of vertices (vertex v).

- The Convex Bottom algorithm is called for those vertexes which are inside of the convex hull (the set \hat{S}).
- The closest vertex to vertex v is selected From the set of vertices over $CH(S)$ (vertex u).
- The vertex close to v and the vertex close to u in the counter-clockwise is called r and t , respectively.
- The edges $(v, r), (u, t)$ is replaced with the edges $(v, u), (t, r)$.

3.3 Algorithm 3

In this algorithm like the two previous algorithms, first $CH(S)$ is computed. Let \hat{S} be the vertex set inside of the convex hull \hat{S} . for this set of vertices, the existing TwoPeasants algorithm (for the presumptive line according to y -coordinate) would be applied which it has time complexity $O(n \log n)$ and performs in this way that first two vertices which has the least and the most x -coordinate is considered. These vertices would be joined together using a presumptive line so that the set of vertices would be divided into two upper and lower half (if the set of vertices is empty, the algorithm executes again) [7]. The next steps are as the following: The upper half vertices would be sorted while they started from the left end point and the lower vertices would be sorted like this way and the end vertices would be joined from both sides and like two previous algorithms the vertices v, u, r and t as it's explained before in section 3-2 have been considered (Fig 4-a) and the edges $(v, r), (u, t)$ is replaced with the edges $(v, u), (t, r)$. The shape which is obtained is a simple random polygon (Fig 4-b). Thus the created polygon in this way is a simple random polygon. The procedure of this algorithm with assumption the existing TwoPeasants algorithm is as the following:

- First, $CH(S)$ would be computed over the set of random vertices S .
- The set of interior vertices $CH(S)$ is considered as \hat{S} and the vertex with the least y -coordinate is selected from the set of vertices (vertex v).
- The TwoPeasants algorithm is called for those vertexes which are inside of the convex hull (the set \hat{S}).
- The closest vertex to vertex v is selected From the set of vertices over $CH(S)$ (vertex u).
- From the set of vertices over $CH(S)$, the closest vertex to vertex v is selected (vertex u).
- The vertex close to v and the vertex close to u in the counter clockwise is called r and t respectively.
- The edges $(v, r), (u, t)$ is replaced with the edges $(v, u), (t, r)$.

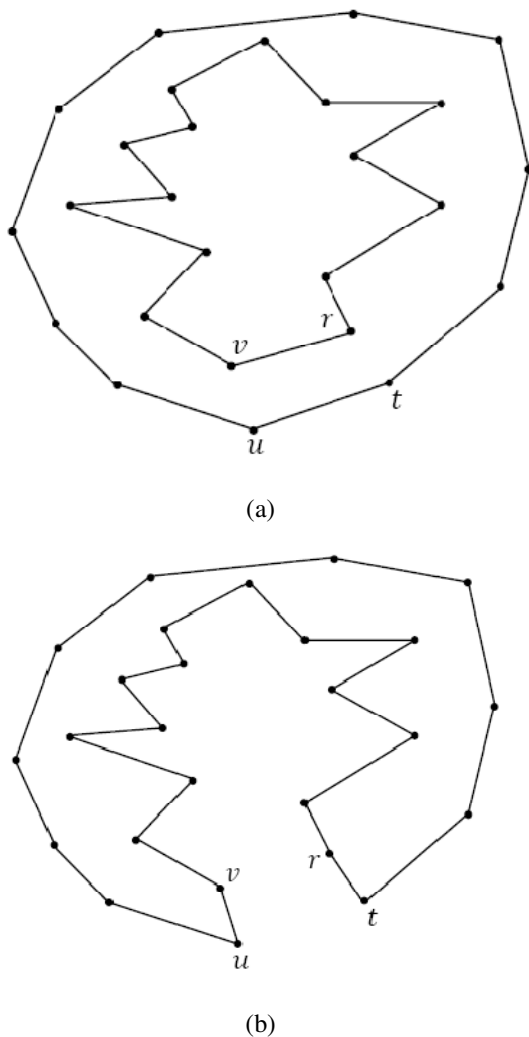


Fig. 4. (a) Creating a polygon with the TwoPeasants algorithm, (b) creating the random polygon by replacing the edges (v, r) , (u, t) with the edges (v, u) , (t, r) .

In the next section, the performance of proposed discussed algorithms would be evaluated.

4 Performance evaluation

In this section, performance and time complexity of the proposed algorithms would be investigated. In the proposed algorithm 1, since the time complexity the computation of $CH(S)$, is $O(n \log n)$, finding a vertex with the least y -coordinate (vertex v), is $O(n)$, and finding the closest vertex to vertex v from the vertices set $CH(S)$ is $O(n)$. Likewise, since the time complexity of sorting of the set of vertices S according to the polar angle around vertex v is $O(n \log n)$ and the replacement the edges (v, r) , (u, t) with the edges (v, u) , (t, r) , would be performed in linear time, therefore the time complexity for this algorithm is $O(n \log n)$.

For the proposed algorithm 2, since time order for computation $CH(S)$, is $O(n \log n)$, and finding a vertex with

the least y -coordinate (vertex v), is $O(n)$, and the time complexity of convex bottom algorithm is $O(n \log n)$, so time complexity for this algorithm is $O(n \log n)$, as well.

In the proposed algorithm 3, time order for computation $CH(S)$, is $O(n \log n)$, finding a vertex with the least y -coordinate (vertex v) is $O(n)$ and time complexity of TwoPeasants algorithm is $O(n \log n)$. Thus, similar to previous algorithms, time complexity of this algorithm is $O(n \log n)$. In the next section, the conclusions would be investigated.

5 Conclusions

In this paper, a review of existing algorithms was done for generating random polygons. Many algorithms have been executed for generating random polygons but no algorithm has been presented yet for the problem of generating simple random polygons in linear time. This problem causes to apply heuristic approaches. In this paper, three inventive algorithms have been proposed for generating simple random polygons which has time complexity of $O(n \log n)$. Since the generation of simple polygon is not possible in less than $O(n \log n)$ time, so these algorithms have optimal order.

6 References

- [1] Auer, T. "Heuristic for generation of polygons"; In: Proc. Canada. Conf. Compute. Geom., 34-44, 1994.
- [2] Epstein, P., Sack, J.-R. "Generating objects at Random"; Master's thesis CS Dept., Carleton university, Ottawa K1S 5B6, Canada, 1992.
- [3] Derroy, L., Epstein, P., Sack J.-R. "on generating random intervals and hyperrectangles"; J.Compute. Graph.Stat., 291-307, 1993.
- [4] Atkinson, M.D., Sack, J.-R. "Uniform generation of forests of restricted height"; Inform. Process. Lett., 323-327, 1994.
- [5] Zhu, C., Sundaram, G., Snoeyink, J., Mitchell, J.S.B. "Generating random polygon with given vertices"; Compute. Geom.: Theory Applic, 1996.
- [6] O'Rourke, J., Virmani, M. "generating random polygons"; Technical report 11. CS Dept. Smith College, MA, USA, 1991.
- [7] <http://www.geometrylab.de/polygon/RandomPolygon.html.en>

Minimum Pseudo-Triangulation Using Convex Hull Layers

F. Taherkhani¹, A. Nouroollah^{1,2}

¹Department of Computer Engineering & IT, Islamic Azad University, Qazvin, Iran

²Department of Electrical & Computer Engineering, Shahid Rajaee Teacher Training University, Tehran, Iran

Abstract - Pseudo-triangulation is regarded as one of the most commonly used problems in computational geometry. In this paper we consider the problem of minimum pseudo-triangulation of a given set of points S in the plane using convex hull layers and we propose two new methods that will lead to the production of minimum pseudo-triangulation. This means that the number of pseudo-triangles created in minimum pseudo-triangulation is exactly $n-2$ pseudo-triangles and the minimum number of edges needed is $2n-3$.

Keywords: pseudo-triangulation, reflex chain, convex hull layers, visibility

1 Introduction

The names pseudo-triangle and pseudo-triangulation were coined by Pocchiola and Vegter in 1993. For polygons, pseudo-triangulations has been already expressed in the computational geometry's literature in the early 1990's, under the name of geodesic triangulations [1]. The geodesic path between two points of a polygon is the shortest path from one to the other in polygon. Pseudo-triangulations of a simple polygon are also called geodesic triangulations, because they arise by inserting non-crossing geodesic paths in polygon.

A pseudo-triangle is a planar polygon with exactly three convex vertices, called corners and three reflex chains of edges join the corners. Let S be a set of n points in general position in the plane. A pseudo-triangulation for S is a partition of the convex hull of S into pseudo-triangles whose vertex set is S [2].

In 2000, Streinu [3] has shown that there are strong links between minimally rigid graphs and minimum pseudo-triangulations. In addition, she proved that the minimum number of edges needed to obtain a pseudo-triangulation is $2n-3$ and thus, by Euler's polyhedron theorem, the number of pseudo-triangles in a minimum pseudo-triangulation is $n-2$, which does not depend on the structure of the point set but only on its size [4]. Every vertex of a minimum pseudo-triangulation is pointed. A vertex is pointed if it has an incident angle greater than π .

Pseudo-triangulations are received considerable attention in computational geometry. This is mainly due to their applications in rigidity theory, robot arm motion planning, visibility, ray-shooting, kinetic collision detection and guarding polygons [5-8].

With respect to the fact that some of the interesting geometric and combinatorial properties of pseudo-triangulations have been recently discovered, but many main open questions still remain [2]. In this paper we consider the problem of minimum pseudo-triangulation of a set S of n points in the plane and we show that the generation of convex hull layers for set points and their pseudo-triangulation, using two new methods, minimizes pseudo-triangulation.

The rest of this paper has been organized as follows: In section 2 some basic definitions are presented. Section 3 describes how to create convex hull layers. In section 4 determine for all vertices in convex hull layers visible vertices and finally in section 5 we propose two new methods of pseudo-triangulation on created convex hull layers to attain minimum pseudo-triangulation.

2 Initial definitions

A simple polygon is called a *convex polygon* when all the internal angles are less than π . According to this definition, the set of points S on a plane is called convex if and only if in exchange for both the points $p, q \in S$, the line segment pq completely lies inside S ($pq \subseteq S$).

The most applicable structure in robotic geometry is *convex hull*. Convex hull of the given points p_0, \dots, p_{n-1} is the smallest convex set on the plane which contains the points.

Let three points $p_1(x_1, y_1)$, $p_2(x_2, y_2)$ and $p_3(x_3, y_3)$ are given in the plane. Hence matrix A is defined as follows:

$$A = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \quad (1)$$

Let $\det(A)$ refers to determinant of matrix A . Three cases can be occurred.

- *Case a:* $\det(A) > 0$, Sequence p_1, p_2, p_3 are counter-clockwise (left turn).
- *Case b:* $\det(A) < 0$, Sequence p_1, p_2, p_3 are clockwise (right turn).
- *Case c:* $A = 0$ implies that the three points p_1, p_2, p_3 are collinear.

Two points p and q on the Euclidean plane are *visible* towards each other if the line segment pq doesn't intersect any other line segments.

Let p_0, \dots, p_{n-1} be the vertices of a simple polygon P which lie in counter-clockwise direction (Fig. 1). We call $\delta(p_i, p_j)$, the shortest path between the two vertices p_i, p_j from the vertices of P . $\delta(p_i, p_j)$ path is called convex chain If we move from vertex p_i towards vertex p_j on the path, the relevant path will be counter-clockwise, otherwise the $\delta(p_i, p_j)$ path is called reflex chain.

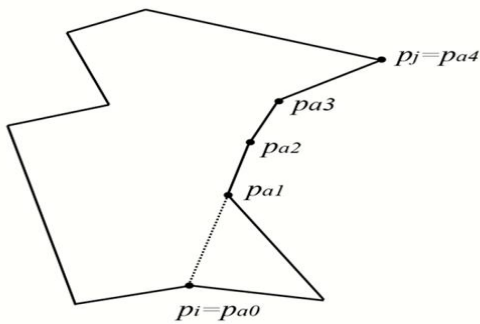


Fig. 1 Convex and reflex chain

3 The generation of convex hull layers

In this section we will consider algorithm of the generation of convex hull. In order to generate convex hull, the coordinates of vertices and the order of their connections are required. There are different algorithms in order to generate convex hull. In this paper the Graham algorithm version B has been used. In order to compute convex hull, first one should find boundary points. In this algorithm, the lowest point is the first starting extreme point.

The set S with n points on a plane is given. According to Graham scan algorithm version B , the following steps are taken:

- *Step 1-* Find the lowest point and call it point p_0 .
- *Step 2-* The remaining points are put in order based on the angle around point p_0 . If two points have the same angle with p_0 , (i.e. they are collinear) then the point which has a larger distance from p_0 is taken

into consideration. We call these points p_1, \dots, p_{n-1} and connecting these points to one another generating a star shaped polygon (Fig. 2-a).

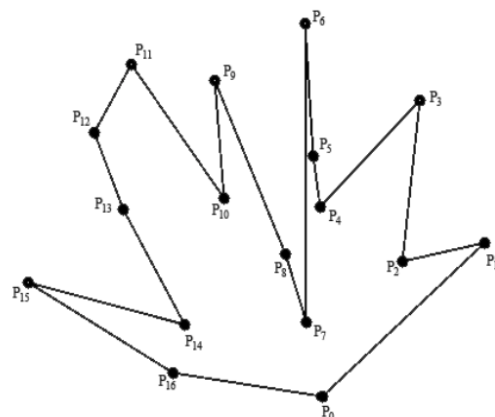
- *Step 3-* Line segment $p_0 p_1$ definitely lies on the convex hull. Thus these two vertices are pushed into a stack so that p_1 lies on the top of the stack. Two top of stack vertices together with the following vertex (p_2) are considered and the clockwise or the counter-clockwise directions of these consecutive three vertices are determined. If the angle is counter-clockwise, the vertex will be pushed into the stack and the next vertex is considered, otherwise the top of stack is popped and similarly the algorithm is continued. Eventually, all the vertices which lie on the stack are the same vertices sorted on the most external convex hull layer. The time complexity of the presented algorithm is $O(n \log n)$ (Fig. 2-b).

Pseudo code of the Graham algorithm version B:

Procedure Graham

```

p0 ← find the point whose y coordinate is minimum
Sort the other points around p0 and call them p1, ..., pn-1
Push (p0)
Push (p1)
for i ← 2 to n-1 do
    while Right (stack [top-1], stack [top], pi) do
        Pop
    Repeat
        Push (pi)
Repeat
    
```



(a)

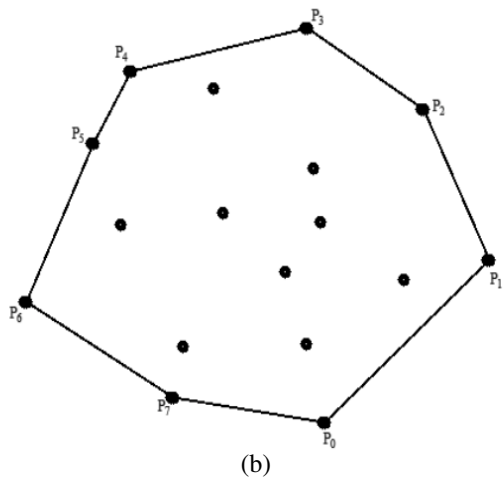


Fig. 2 (a) Star shape polygon (b) the most external convex hull layer

By extracting the convex hull points, the algorithm is repeated on the remaining points, a new convex hull is generated and this action goes on until it comes to less than three points. This means that just one or two points remains, so that in this case for using the two new methods of minimum pseudo-triangulation there exist certain conditions which should be taken into consideration. Hence, the convex hull layers are generated in this manner (Fig. 3).

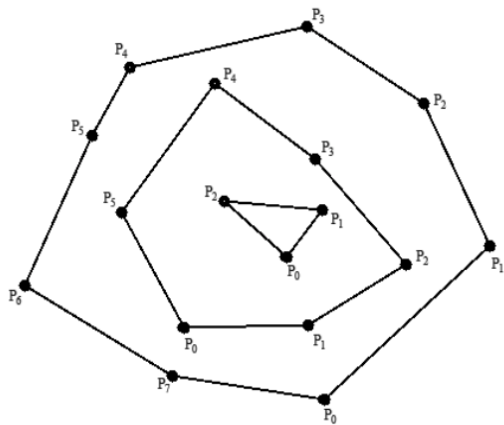


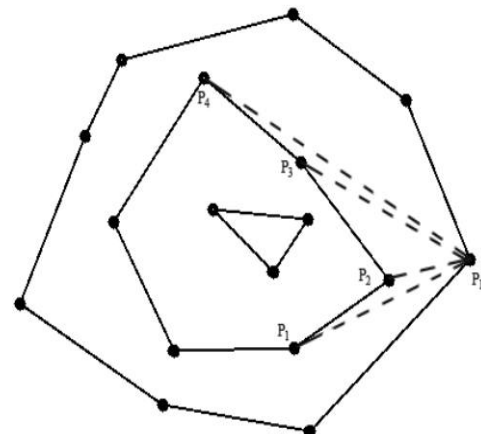
Fig. 3 The convex hull layers

4 Suggested algorithms

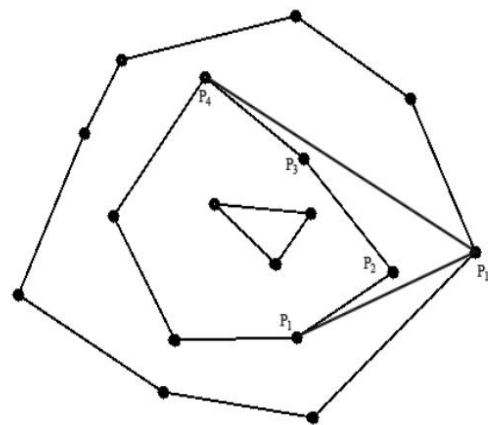
In this section first the visible vertices in the convex hulls should be determined each vertex of the convex hull. If the two adjacent convex hulls are considered as a pitted polygon, the vertices which have the following term are visible for the vertex: the linking line between the vertex and the vertices sorted on the internal convex hull don't lie

outside this pitted polygon (i.e. it shouldn't intersect any sides of the polygon).

Among all the vertices visible for each vertex, the two vertices which have the smallest and the largest angle towards this vertex are regarded as the two farthest visible vertices. Hence, for all the vertices of the external layer, compared to the following layer, there will be two visible vertices. For instance, as it is shown in Fig. 4-a, the vertex p_1 from the most external convex hull meets four vertices p_3, p_2, p_1 and p_4 from the following convex hull so that p_1 and p_4 are chosen as the two farthest visible vertices (Fig. 4-b).



(a)



(b)

Fig. 4 (a) The visible vertices of p_1 from external convex hull. (b) The two farthest visible vertices of p_1 from external convex hull.

As it was expressed all the vertices in the convex hull layer compared to the following layer have two farthest visible vertices. In this paper one of these two vertices should be chosen to generate pseudo-triangulation here two new methods of choosing one of these two visible vertices in order to generate pseudo-triangulation is put forward.

4.1 Method of choosing clockwise visible vertices

In the method of choosing clockwise visible vertices, from the two farthest visible vertices determined, the one is selected which holds a clockwise relation between the relevant vertex from the external convex hull and its two visible vertices. If the number of layers is M , $P_{i,j}$ the vertex of the i th in j th layer, that j shifts from $1, \dots, M$. Thus, the relevant vertex in j th is considered as $P_{i,j}$ and we call the two visible vertices $P_{s,j+1}$ (index of the nearest visible vertex) and $P_{k,j+1}$ (index of the farthest vertex). We consider the rotation from the relevant vertex toward the two visible vertices. If $(P_{i,j}P_{s,j+1}P_{k,j+1})$ is clockwise the line segment is linked between $P_{i,j}$ and $P_{s,j+1}$ in the case that $(P_{i,j}P_{k,j+1}P_{s,j+1})$ is clockwise the line segment is linked between $P_{i,j}$ and $P_{k,j+1}$. Linking these lines every layer, the clockwise visible vertices are produced (Fig. 5).

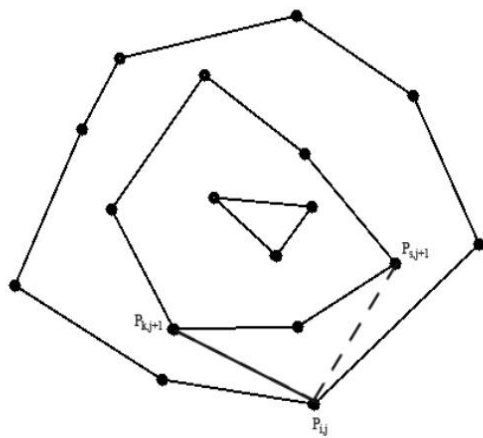


Fig. 5 Select of the clockwise visible vertex for $P_{i,j}$

4.2 Method of choosing counter-clockwise visible vertices

In the method of choosing counter-clockwise visible vertices from the two farthest visible vertices, the one is selected which holds a counter-clockwise relation between the relevant vertex from the external convex hull and its two visible vertices such that we consider the relevant vertex in the i th layer as $P_{i,j}$ and call the two visible vertices $P_{s,j+1}$ and $P_{k,j+1}$.

We consider the rotation from the vertex toward the two visible vertices. If $(P_{i,j}P_{s,j+1}P_{k,j+1})$ is counter-clockwise, the line segment is linked between $P_{i,j}$ and $P_{s,j+1}$ and in the case that $(P_{i,j}P_{k,j+1}P_{s,j+1})$ is counter-clockwise the line segment is linked between $P_{i,j}$ and $P_{k,j+1}$. Linking these lines in every layer, the counter-clockwise visible vertices are produced (Fig. 6).

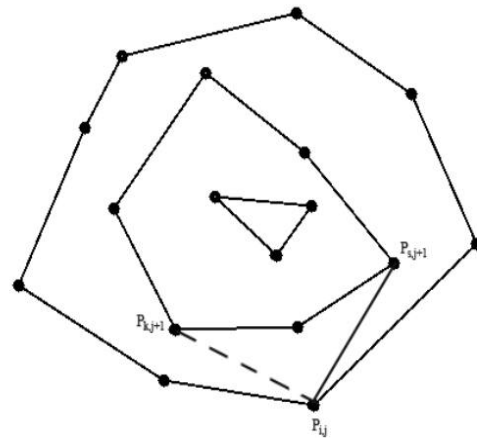
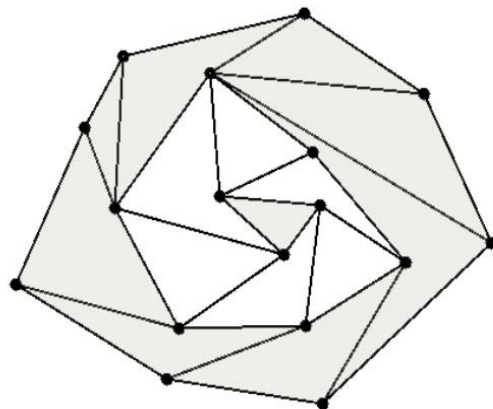


Fig. 6 Select of the counter-clockwise visible vertex for $P_{i,j}$

If we perform one of the two proposed methods or a combination of the two methods (in a way that just one method is used for each layer), the convex layers are pseudo-triangulated according to Fig. 7. In the next section the outcome will be considered.



points: 17 layers: 3 edges: 31 pseudo-triangulations: 15

Fig. 7 Minimum pseudo-triangulation using two new methods pseudo-triangulation

5 Conclusions

In this paper two new methods for pseudo-triangulation of the set of points S were put forward. The trend was in a way that first a layer was generated for the set of points S on the plane of the convex hull and then performing one of the two methods or a combination of them on the layers the act of pseudo-triangulation was done. The surveys showed that the pseudo-triangulation performed were minimum i.e. the number of the produced pseudo-triangles is $n-2$ pseudo-triangle and the number of edges in it is the least possible amount i.e. $2n-3$.

6 References

- [1] G. Rote, F. Santos, and I. Streinu, "Pseudo-Triangulation – a Survey," *Discrete Comput. Geom.* 2007.
- [2] O. Aichholzer, F. Aurenhammer, H. Krasser, and B. Speckmann, "Convexity minimizes pseudo-triangulations," *Computational Geometry* 28, pp.3-10, 2004.
- [3] I. Streinu, "A combinatorial approach to planar non-colliding robot arm motion planning," *In: Proc. 41st Annu.IEEE Sympos. Foundat. Comput.Sci. (FOCS'00)*, pp.443-453, 2000.
- [4] S. Gerdjikov, and A. Wolff, "Decomposing a simple polygon into pseudo-triangles and convex polygons," *Computational Geometry* 41, pp.21-30, 2008.
- [5] M. Pocchiola, and G. Vegter, "Topologically sweeping visibility complexes via pseudo-triangulations," *Discrete Compute.Geom.* 16, pp.419-453, 1996.
- [6] M.T. Goodrich, and R. Tamassia, "Dynamic ray shooting and shortest paths in planar subdivisions via balanced geodesic triangulations," *J. Algorithms* 23 (1), pp.51-73, 1997.
- [7] D.G. Kirkpatrick, J. Snoeyink, and B. Speckmann, "kinetic collision detection for simple polygons," *Internat. J. Comput. Geom. Appl.* 12(1-2), pp. 3-27, 2002.
- [8] B. Speckmann, and C.D. Tóth, "Allocating vertex π -guard in simple polygons via pseudo-triangulations," *Discrete Comput. Geom.* 33 (2), pp.345-364, 2005.

Generating Sunflower Random Polygons on a Set of Vertices

L.Mohammadi², A. Nouroollah¹

¹Department of Electrical & Computer Engineering of Islamic Azad University, Qazvin, Iran

²Faculty of Electrical & Computer Engineering of Islamic Azad University, Qazvin, Iran

& Faculty of Electrical & Computer Engineering of Shahid Rajaei Teacher Training University, Tehran, Iran

Abstract - Generating random polygons problem is important for verification of geometric algorithms. Moreover, this problem has applications in computing and verification of time complexity for computational geometry algorithms such as Art Gallery. Since it is often not possible to get real data, a set of random data is a good alternative. In this paper, a heuristic algorithm is proposed for generating sunflower random polygons using $O(n \log n)$ time.

Keywords: convex hull, sunflower random polygon, visibility

1 Introduction

Computational geometry is a very important research field in computer science in which most computations are performed on known geometrical objects as polygons. Polygons are a convenient representation for many real-world objects; convenient both in that an abstract polygon is often an accurate model of real objects and in that it is easily manipulated computationally. Examples of their applications include representing shapes of individual letters for automatic character recognition, of an obstacle to be avoided in a robot's environment, or a piece of a solid object to be displayed on a graphic screen [7].

The generation of random geometrical objects has received some attention by researchers [2][3][6]. A challenge of these problems is the generation of random simple polygons. Since no polynomial time algorithm is known to solve the problem, researchers either try to use heuristic algorithms which don't have uniformed distribution or restrict the problem to certain classes of polygon such as monotone and star-shaped polygons [1][3][4].

The importance of geometric objects application is the simplicity of testing geometric algorithms. Since a set of data may become both too large and too hard to define for practical purposes, what one might do is to use randomly generated data that has a high probability to cover all the different classes of inputs. Thus, since practical data may not be available for testing, it is natural to test the algorithm on randomly input data.

Polygons are one of the fundamental building blocks in geometric modeling and they are used to present a wide variety of shapes and figures in computer graphics, vision, pattern recognition, robotics and other computational fields.

Some recent applications address uniformed random generation of simple polygons with given vertices, in the sense that a polygon will be generated with probability $\frac{1}{T}$ if there exist a total of T simple polygons with such vertices.

One of the important geometry problems in which polygons play an important rule, is art gallery whose purpose is guarding a polygonal art gallery with the least number of guards (cameras). A well-known kind of art gallery problem is sunflower art gallery. The proposed question is this: What is the smallest number of guards required to protect the Sunflower Art Gallery?

Figur 1 shows a sunflower art gallery which is protected by 4 stationary guards. Some of the guards can not see through walls around corners of art gallery. Every point is visible at least one guard and it would be more economical to protect the gallery with fewer guards, if possible [5].

In this paper a heuristic algorithm is proposed for the generation of random sunflower polygons to estimate such problems.

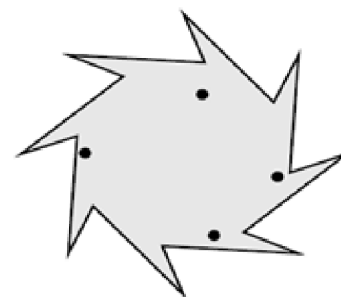


Fig. 1. The sunflower art gallery [5]

The following sections of this paper have been organized in this way: section 2 has been allocated to related works. In section 3 the needed preliminary concepts are stated. In section 4 the proposed algorithm is posed for generating random sunflower polygon and its performance and accuracy are investigated and finally in section 6 the conclusion will be discussed.

2 The related works

Recently, the generation of random geometric objects and specially simple polygons has received some attention by researchers. For example, Epstein studied the uniformly random generation of triangulations [8]. Zhu et al. presented on an algorithm for generating x -monotone polygons on a given set of vertices uniformly at random [3]. A heuristic for the generation of simple polygons was investigated by O'Rourke and Virmani [6]. Auer and Held presented the following heuristic algorithms:

- *Steady Growth*: an incremental algorithm adding one point after the other whose time complexity in the worst case and in the best case is $O(n^2)$ and $O(n \log n)$, respectively.
- *Space Partitioning*: which is a divide and conquer algorithm and it has time complexity of $O(n^2)$.
- *Permute & Reject*: which creates random permutations (polygons) and surveys whether it is corresponding with a simple polygon or not, until a simple polygon is encountered. Its complexity is $O(n \log n)$.
- *2-opt Moves*: which by starting from a completely random polygon and replacing its intersected edges encounters a simple polygon and its complexity is $O(n^4)$ [2].

3 Preliminaries

Let S be a set of random vertices, there exists T simple polygons on S in total, such that every polygon is generated with probability $\frac{1}{T}$. It is supposed that no three points are linear. A simple polygon, is a limited plane by a limited set of line segments that form a simple closed curve. In other words, a simple polygon P on S , is a polygon whose edges don't intersect one another except on vertices S .

A convex polygon, is a simple polygon which for both of vertices x and y from polygon, the line segment \overline{xy} lies inside P or on its border, i.e., $\overline{xy} \subseteq P$. The convex hull of a finite set of points on the plane ($CH(S)$) is the smallest convex polygon P that encloses S . The smallest polygon means that there is no polygon \hat{P} such that $S \subseteq \hat{P} \subset P$ (Figure 2).

Let k be the number of all convex hull layers on the set of points S . Every layer is defined by l_c ($k > 1$ and $1 \leq l_c \leq k$). By the supposition of numbering layers from the most internal to the most external convex hull layers, n_i is supposed to be the number of the most internal convex hull layer, i.e., l_1 .

Vertex x sees vertex y if $\overline{xy} \subseteq P$ and \overline{xy} doesn't lie out of P . In this case y is visible from x or in other words x has visibility toward y . The polar sorting, is the sorting of a set of vertices around a given vertex according to polar angle. The star-shaped polygon, is a polygon which is visible at least

from an interior vertex. In the next section, the proposed heuristic algorithm is posed to generate random sunflower polygon.

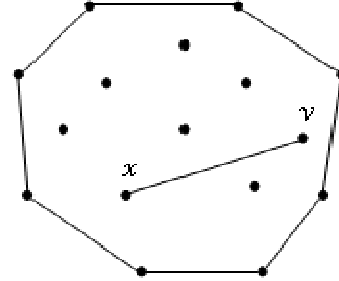


Fig. 2. The convex hull of set of points S

4 The proposed algorithm

In this section, an algorithm is posed to generate the random sunflower polygon with time complexity $O(n \log n)$ in which n is the number of vertices.

To generate a random sunflower polygon on a vertex set it is performed in this way that first, According to Graham greedy algorithm, the assumption $CH(S)$ is obtained. The Graham algorithm for the generation of $CH(S)$ performs in this way:

- First, the vertex with lowest y -coordinate which is the rightmost point, has selected from the point set S and is called p_0 .
- All the remaining points are sorted around p_0 according to polar angle (p_1, \dots, p_{n-1}).
- A stack is constructed and p_0 and p_1 are pushed to it.
- By starting from point p_2 to p_{n-1} for every vertices the following case is investigated:
- p_i is pushed in the stack if it is left of two tops of the stack and is incremented counter i ; otherwise, the top of stack is removed if p_i is right of two tops of the stack (Figure 3).

```

Algorithm: Graham Scan
Find rightmost lowest point; label it  $p_0$ .
Sort all other points angularly about  $p_0$ .
Stack  $S = (p_1, p_0) = (p_t, p_{t-1}); t$  indexes top.
 $i = 2$ 
While  $i < n$  do
  If  $p_i$  strictly left of  $p_{t-1}p_t$  Then
    Push( $p_i, S$ ) and set  $i \leftarrow i + 1$ 
  Else
    Pop( $S$ )
  End If
End While

```

Fig. 3. The Graham scan algorithm

Time complexity of Graham algorithm is $O(n \log n)$. After determining $CH(S)$, while n is opposite of zero, Graham algorithm is recalled for any remaining point set S . Let the

number of points on the most internal assumption convex hull, i.e., l_1 be n_i , the point set S can be partitioned to n_i sections. This partitioning is in this way that by starting from the leftmost point on l_1 and in counter-clockwise direction, every edge l_1 is continued from the second point. Thus, n_i set of points are obtained (Figure 4).

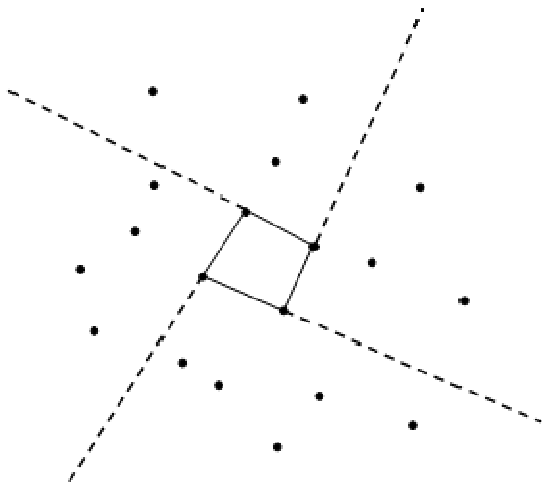


Fig. 4. Partitioning of the set if points S by continuing the second vertex of every edge l_1

After partitioning of the point set S , by starting from the leftmost point or the vertex with least x -coordinates on l_1 on the first part of partitioned vertices, the points of the part until to starting point of the next part, are connected together in this way: points are sorted around point v_j on l_1 ($1 \leq j \leq n_i$) according to polar angle in counter-clockwise direction. Then these points sorted in order are connected together from v_j to v_{j+1} .

Thus, by scanning of all parts and connecting points of every partitioned part based on explained process, all points of set S are connected together and a simple polygon P is resulted which is not necessarily a star-shaped polygon (Figure 5). Procedure of the algorithm is following way:

- while, n is the opposite of zero, all assumption convex hull layers are computed on the set of points S based on Graham algorithm.
- The number of points of assumption convex hull l_1 are counted and are supposed as n_i .
- Every assumption edge $v_j v_{j+1}$ on l_1 is continued from vertex v_{j+1} and the set of points are partitioned to n_i sets.
- For every resulted point set, it is performed in this way: all point of that part have been sorted around v_j based on polar angle and then points of between v_j and v_{j+1} sorted in order are connected together from v_j to v_{j+1} .

- The resulted random simple polygon, is a sunflower polygon. In the special case which $n_i = 1$, it is called star-shaped polygon.

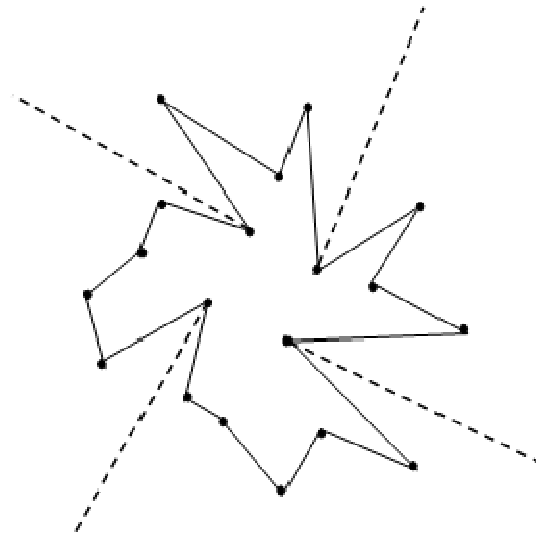


Fig. 5. Generation of a sunflower polygon with connecting the sorted points of any partition together

4.1 Verification of algorithm performance

In this section, the performance of proposed algorithm is investigated by computing its time complexity. In this algorithm, time of obtaining the most external convex hull layer is $O(n \log n)$ based on Graham algorithm which it would be the maximum time needed to generate of convex hull since, it is computed on n points. Partitioning of points S to n_i sets, requires $O(n_i)$ time that is in the best case $O(1)$, i.e., in the case of existing only a resulted part of points partitioning. Finding the leftmost point with least x -coordinates on l_1 with the number of n_i points, is $O(n_i)$. Sorting points of the any part set, in the case the sprawl of points is good, i.e., $\frac{n}{n_i}$ points exist in any part averagely, is $O(\frac{n}{n_i} \log \frac{n}{n_i})$. In the worst case, which exists only one partitioned part, this sorting takes $O(n \log n)$ time. Therefore, time complexity is $O(n \log n)$ in total.

4.2 Determining of visible polygon

In this section, the posed heuristic algorithm is investigated. This algorithm performs properly for any set of given points with the number of assumption convex hull layers $k > 1$ in general case.

To proof the accuracy of this algorithm, it is discuss every set of partitioned points and the visibility of all points in every part from one point on l_1 . Since, for generating this random polygon, all partitioned parts are independent of one another and they haven't subscription together and also as in every

part, all its point set are visible from v_j on l_1 , because of polar sorting of points around v_j , it is verified simply that a random sunflower polygon is inevitably generative by this proposed algorithm on any set of given points with number of assumption convex hull layers $k > 1$.

5 Conclusions

In this paper, a heuristic algorithm with time complexity of $O(n \log n)$ was proposed for the generation of random sunflower polygon. This algorithm can be used to estimate many algorithms and geometric problems such as art gallery. Also it was proved that this algorithm has properly on any set of given points with intricate assumption convex hulls (with the number of layers $k > 1$) and it generates a random sunflower polygon.

6 References

- [1] McDonald, B., Smith, I. "Partial symmetry breaking"; In: P. Van Hentenryck(Ed.), Proc. Of CP'02 LNCS2470, Springer-verlag, 431—445, 2002.
- [2] Auer, T., Held, M. "Heuristics for the Generation of Random Polygons"; Proc. 8th Canadian on Computational Geometry(CCCG'96), 38—41, 1996.
- [3] Zhu, C., Sundaram, G., Soneyink, J., Mitchell, J.S.B. "Generating Random Polygon with Given Vertices"; Comput. Geom. Theory and Appl., Vol 6, Issues 5, 277—290, 1996.
- [4] Sohler, C. "Generating random star-shaped polygons"; In: Proc. 11th Canadian Conference on Computational Geometry(CCCG'99), 174—177, 1999.
- [5] Epstein, P. "Generating Geometric Objects at Random"; Master's thesis, CS Dept., Carleton University Ottawa K1S5B6, Canada, 1992.
- [6] O'rouke, J., Virmani, M. "Generating Random Polygons"; Technical Report 011, CS Dept. Smith Colege, Northhampton, MA 01063 1991.
- [7] Berg, M.D. "Computational Geometry Algorithms and Applications"; 3rd, Published by springer-verlag, 2008.
- [8] Michael, T.S. "How to Guard an Art Gallery and other Discrete Mathematical Adventures"; the Book of the Johns Hopkins University Press, printed in the United Stetes of American, 2009.

The Generation of Pseudo-Triangulated Spiral Polygon Using Convex Hull Layers

F. Taherkhani¹, A. Nourollah^{1,2}

¹Department of Computer Engineering & IT, Islamic Azad University, Qazvin, Iran

²Department of Electrical & Computer Engineering, Shahid Rajaee Teacher Training University, Tehran, Iran

Abstract - The generation of random simple polygon and the pseudo-triangulation of a polygon are regarded as the proposed problems in computational geometry. The production of a random polygon is used in the context of the consideration of the accuracy of algorithms. In this paper, a new algorithm is presented to generate a simple spiral polygon on a set of random points S in the plane using convex hull layers in a way that pseudo-triangulation is also performed on it simultaneously. The new algorithm can be done in $O(n \log n)$ time, so it is considered as one of the optimal algorithms.

Keywords: simple spiral polygon, pseudo-triangulation, convex hull layers, convex and concave chain

1 Introduction

Polygons are suitable shapes to demonstrate the objects of real world and every object in nature is demonstrable as a set of polygons. The generation of random simple polygons has two main areas of application: a) testing the correctness and b) evaluating the CPU-time consumption of algorithms that operate on polygons.

The generation of random geometric objects has received some attention by researchers. Epstein studied the uniformly random generation of polygon triangulation [2]. Polygon pseudo-triangulation is a generalized form of polygon triangulation. The names pseudo-triangle and pseudo-triangulation were coined by Pocchiola and Vegter in 1993[3]. A pseudo-triangle is a simple polygon with exactly three convex vertices, called corners and three concave chains of edges joining the corners [4].

Let S be a set of random n points p_0, \dots, p_{n-1} in the plane. The goal is generation of a random simple polygon with a uniform distribution. A uniformly random polygon on S is a polygon generated with probability of $1/k$ if there exist k simple polygons on S in total [2, 5]. Since the generation of a polygon from a set of random points is frequently used to consider the performance of proposed algorithms in the context of polygons such as the Art gallery problem; therefore, the generation of similar polygons cannot show well the quality of the performance of the algorithms. Thus,

we are looking for algorithms having the production ability of kinds of polygons with different structures, and up to now no solutions with the polynomial time to generate uniform random polygons have been known.

The following subjects of this paper have been organized in this way: In section 2 the initial definitions are presented. In section 3 the generation manner of convex hull layers are expressed. In section 4 the suggested algorithm to generate pseudo-triangulated spiral simple polygon and the analysis of its time complexity is proposed. Finally in section 5, conclusion will be presented.

2 Preliminaries

A sequence of line segments, such that the end of each one is the beginning of the following one, is referred as polygon and a polygon whose edges don't intersect one another is called *simple polygon*.

A simple polygon is called a *convex polygon* when all the internal angles are less than π . According to this definition, the set of points S on a plane is called convex if and only if in exchange for both the points $p, q \in S$, the line segment pq completely lies inside S ($pq \subseteq S$).

The most applicable structure in robotic geometry is *convex hull*. Convex hull of the given points p_0, \dots, p_{n-1} is the smallest convex set on the plane which contains the points.

Let three points $p_1(x_1, y_1)$, $p_2(x_2, y_2)$ and $p_3(x_3, y_3)$ are given in the plane. Hence matrix A is defined as follows:

$$A = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \quad (1)$$

Let $\det(A)$ refers to determinant of matrix A . Three cases can be occurred.

- Case a: $\det(A) > 0$, Sequence p_1, p_2, p_3 are counter-clockwise (left turn).

- Case b: $\det(A) < 0$, Sequence p_1, p_2, p_3 are clockwise (right turn).
- Case c: $A = 0$ implies that the three points p_1, p_2, p_3 are collinear.

Two points p and q on the Euclidean plane are *visible* towards each other if the line segment pq doesn't intersect any other line segments.

Let p_0, \dots, p_{n-1} be the vertices of a simple polygon P which lie in counter-clockwise direction (Fig. 1). We call $\delta(p_i, p_j)$, the shortest path between the two vertices p_i, p_j from the vertices of P . $\delta(p_i, p_j)$ path is called convex chain If we move from vertex p_i towards vertex p_j on the path, the relevant path will be counter-clockwise, otherwise the $\delta(p_i, p_j)$ path is called concave chain.

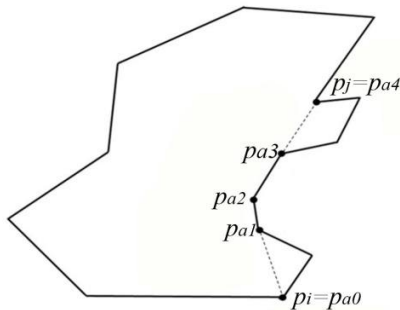


Fig. 1 Convex and concave chain

- Step 3- Line segment p_0p_1 definitely lies on the convex hull. Thus these two vertices are pushed into a stack so that p_1 lies on the top of the stack. Two top of stack vertices together with the following vertex (p_2) are considered and the clockwise or the counter-clockwise directions of these consecutive three vertices are determined. If the angle is counter-clockwise, the vertex will be pushed into the stack and the next vertex is considered, otherwise the top of stack is popped and similarly the algorithm is continued. Eventually, all the vertices which lie on the stack are the same vertices sorted on the most external convex hull layer. The time complexity of the presented algorithm is $O(n \log n)$ (Fig. 2-b).

Pseudo code of the Graham algorithm version B:

```

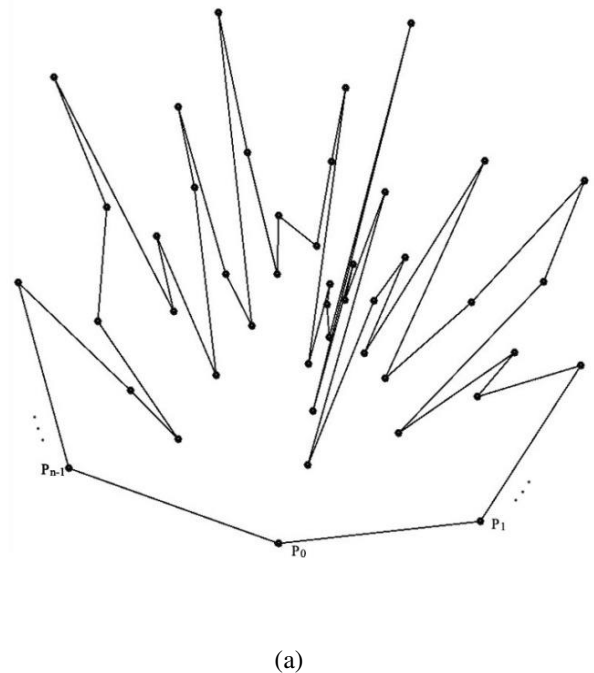
Procedure Graham
 $p_0 \leftarrow$  find the point whose y coordinate is minimum
Sort the other points around  $p_0$  and call them  $p_1, \dots, p_{n-1}$ 
Push ( $p_0$ )
Push ( $p_1$ )
for  $i \leftarrow 2$  to  $n-1$  do
    while Right (stack [top-1], stack [top],  $p_i$ ) do
        Pop
    Repeat
        Push ( $p_i$ )
    Repeat
    
```

3 The generation of convex hull layers

In this section we will consider algorithm of the generation of convex hull. In order to generate convex hull, the coordinates of vertices and the order of their connections are required. There are different algorithms in order to generate convex hull. In this paper the Graham algorithm version B has been used. In order to compute convex hull, first one should find boundary points. In this algorithm, the lowest point is the first starting extreme point.

The set S with n points on a plane is given. According to Graham scan algorithm version B, the following steps are taken:

- Step 1- Find the lowest point and call it point p_0 .
- Step 2- The remaining points are put in order based on the angle around point p_0 . If two points have the same angle with p_0 , (i.e. they are collinear) then the point which has a larger distance from p_0 is taken into consideration. We call these points p_1, \dots, p_{n-1} and connecting these points to one another generating a star shaped polygon (Fig. 2-a).



(a)

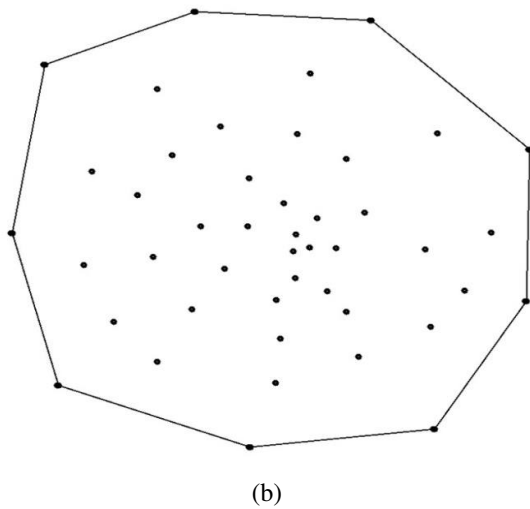


Fig. 2 (a) Star shape polygon (b) the most external convex hull layer

By extracting the convex hull points, the algorithm is repeated on the remaining points, a new convex hull is generated and this action goes on until it comes to less than three points. This means that just one or two points remains. Hence, the convex hull layers are generated. In the following section these generated convex layers will be typically used. It means that these layers are not depicted for the set of points S on the plane and they are computed as preprocessing (Fig. 3).

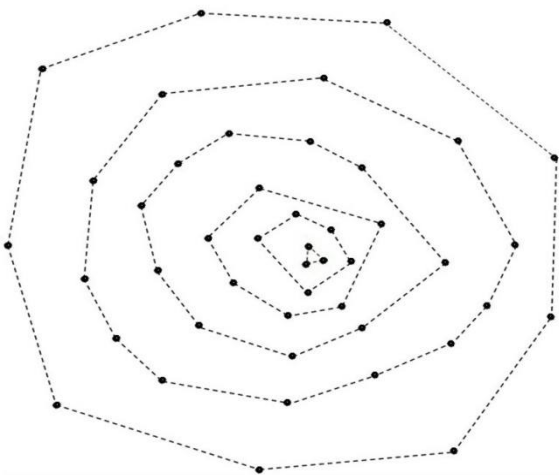


Fig. 3 The convex hull layers

4 The generation of pseudo-triangulated spiral simple polygon

In this section a new algorithm is presented for the generation of spiral simple polygon which is also pseudo-triangulated simultaneously.

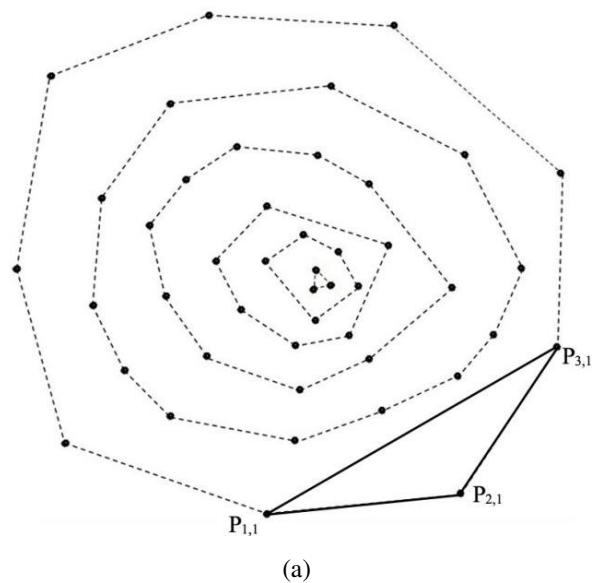
4.1 The suggested algorithm

The suggested algorithm consists of two stages. Applying these two stages to the generated convex layers in the preceding stage, and by generating consecutive pseudo-triangles, pseudo-triangulated spiral polygon is eventually obtained.

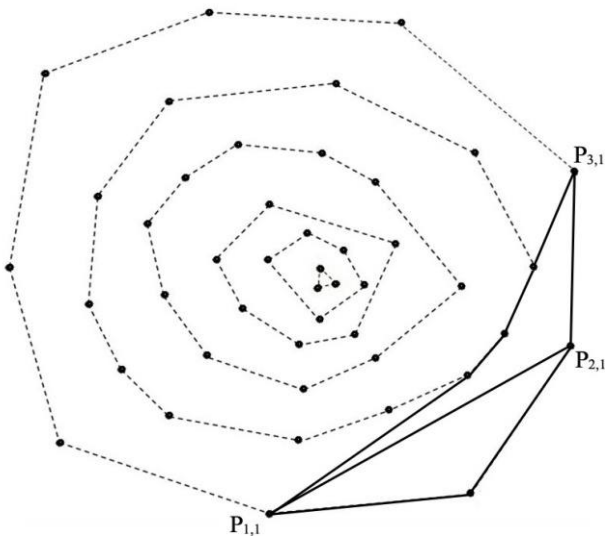
Let M be the number of convex layers. L_j is the j th layer that $j = 1, \dots, M$ and $P_{i,j}$ is the i th vertex in the j th layer.

4.1.1 The first stage of the algorithm:

- *Step 1-* Take $j \leftarrow 1$ into account and choose a point on L_1 (The most external convex layers) as the starting point and call it $p_{1,j}$.
- *Step 2-* Choose two other points in counter-clockwise direction respectively and call them $p_{2,j}$ and $p_{3,j}$. Take these three points as the pseudo-triangle vertices into account and generate the line segments $p_{1,j}p_{2,j}$ and $p_{2,j}p_{3,j}$.
- *Step 3-* In this step, in order to generate the connecting line segment between two vertices of $p_{1,j}$ and $p_{3,j}$, in case of non existing intersection with layer L_{j+1} , the foregoing line segment is depicted. Otherwise, we should choose and depict points of layer L_{j+1} from vertex $p_{1,j}$ to vertex $p_{3,j}$ which form a concave chain with these two vertices. In this stage of algorithm, a pseudo-triangle has been generated (Fig. 4).



(a)



(b)

Fig. 4 (a) Non existing intersection with layer L_{j+1} . (b) Intersection with layer L_{j+1} .

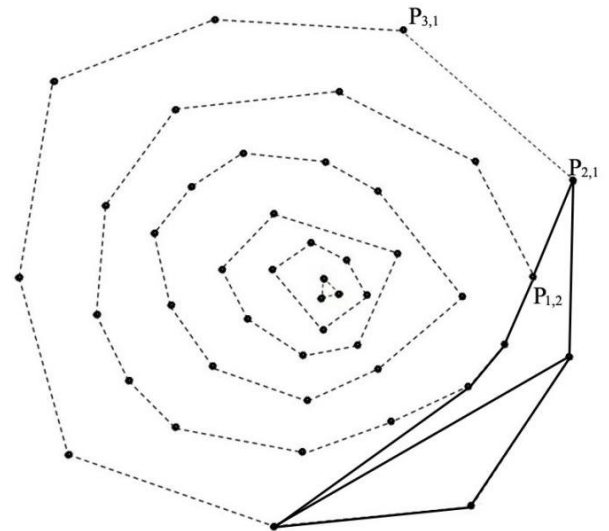


Fig. 5 Change the local position of triangle vertices (Intersection with layer L_{j+1}).

- Step 4-* In order to generate the following pseudo-triangle the local position of vertices $p_{1,j}$, $p_{2,j}$ and $p_{3,j}$ should be changed by considering the following conditions:

 - The local position of point $p_{1,j}$ changes in case of intersection of line segment $p_{1,j}p_{3,j}$ with layer L_{j+1} and is exchanged to the neighboring point $p_{3,j}$ on the concave chain (Fig. 5).
 - The vertex $p_{2,j}$ is transferred to the local position of the present vertex $p_{3,j}$ and call its neighboring point in counter-clockwise direction on the layer L_j vertex $p_{3,j}$ and generate line segment $p_{2,j}p_{3,j}$ (Fig. 5 and Fig. 6).

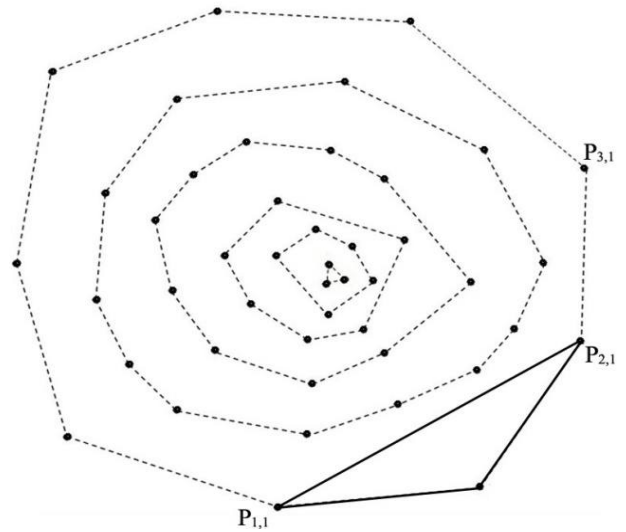


Fig. 6 Change the local position of triangle vertices (Non-existing intersection with layer L_{j+1}).

Repeat steps 3 and 4 as far as the last remaining point on the layer L_j .

4.1.2 The second stage of the algorithm:

Meeting the last point on layer L_j , in this stage among the remaining points on layer L_{j+1} which haven't been used to generate concave chain in the first stage of algorithm, find the farthest visible point from the last point on the layer L_j that is vertex $p_{3,j}$ and call it v , then depict the connecting line segment between the two points. Also, move from vertex $p_{1,j}$

to the visible point on layer L_{j+1} and depict the line segments among the existing points one by one in this path.

After finishing this stage, taking $j \leftarrow j+1$ into account, enter the following layer and consider the point neighboring the farthest visible point (in the second stage of algorithm) in clockwise direct as the starting point of this layer, and from the second step of the first stage we continue the algorithm with the remaining points (except for the farthest visible point) in this layer and repeat the stages till $j < M$ (Fig. 7).

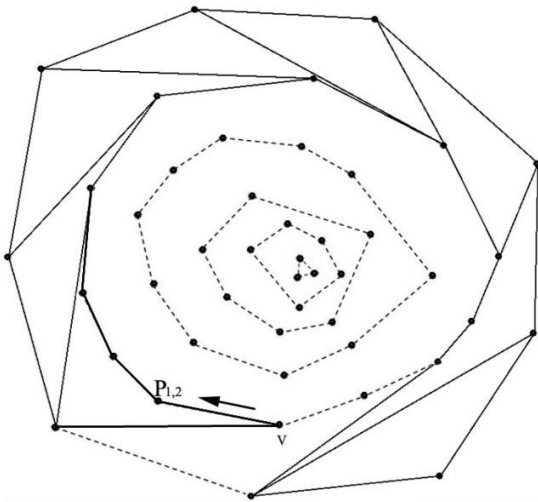


Fig. 7 Entering the next layer (L_{j+1}).

Hence by repeating the stages of the suggested algorithm until $j < M$, a pseudo-triangulated spiral simple polygon is generated (Fig. 8) that in subsection of the following section deal with analyzing the suggested algorithm.

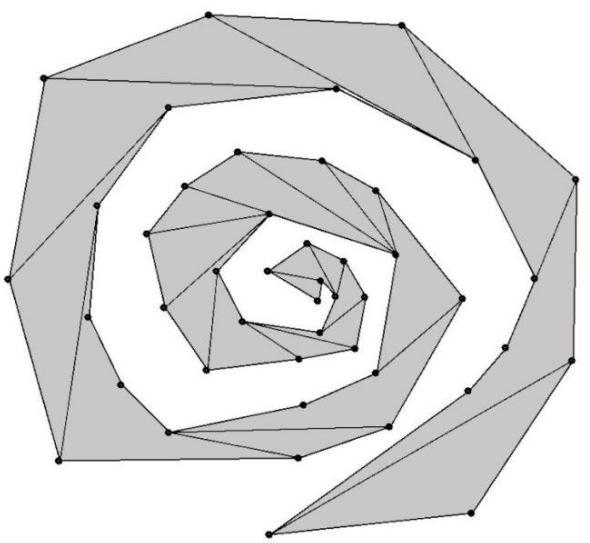


Fig. 8 Pseudo-triangulated spiral simple polygon.

Theorem 1. The presented algorithm produces random simple polygons twice as much as the number of the existing points on the most external convex layer.

Proof: Since every point of the convex polygon of the most external layer can be the starting point of algorithm and it can be selected clockwise or counter-clockwise in order to be run, so simple random polygons will be produced twice as much as the number of the existing points on the most external convex layer. ■

Pseudo code of the counter-clockwise algorithm:

Algorithm Random Polygon Generation

```

j ← 1
i ← 1
P1,j ← pi,j
while j < M do
  while i < nj do
    i ← i + 1
    P2,j ← pi,j
    Draw Line (P1,j, P2,j)
    i ← i + 1
    P3,j ← pi,j
    Draw Line (P2,j, P3,j)
    if (P1,j, P3,j) ∩ lj+1 ≠ ∅ then
      find the reflex chain on lj+1 between P1,j, P3,j and draw it
      P1,j ← last element on the reflex chain
    else
      Draw Line (P1,j, P3,j)
      i ← i + 1
      P2,j ← pi,j
      P3,j ← pi,j
    end if
  end while
  v ← find last point which is visible from P3,j among all remain
  points in lj+1
  Draw Line (P3,j, v)
  Draw a chain from P1,j to v
  i ← index of the nearest point to v in clockwise direction
  j ← j + 1
end while
end of algorithm

```

4.2 Analysing the suggested algorithm

The suggested algorithm requires a preprocessing stage called visibility graph, then the generation of convex hulls and eventually the implementation of the suggested algorithm in section 4. The generation of visibility graph can be done in $O(n \log n)$ [1]. The generation of convex hull layers can be done in $O(n \log n)$ [6]. In section 4, implementing the suggested algorithm to generate the line segment between the two points, the issue of the visibility of the two points should be considered that by performing the preprocessing stage, its time complexity is $O(n \log n)$. With regard to the planarity of the pseudo-triangulation graph to generating all of the pseudo-triangles $O(n)$ is required. Thus, the algorithm can be done in $O(n \log n)$.

5 Conclusions

In this paper a new algorithm was presented to generate pseudo-triangulated spiral simple polygons from the set of random points S on the plane. The work trend was such that first the convex hull layers were generated for the set of random points S . By using this suggested algorithm from the most external convex layer to the most internal layer respectively, by creating consecutive pseudo-triangles, pseudo-triangulated spiral polygon whose time complexity is $O(n \log n)$. The generation of simple polygons out of the set of random points, have such applications as the consideration of heuristic algorithms in issues like Art gallery, and thus algorithms to produce polygon are very efficient in this affair.

6 References

- [1] Berg M. D., *Computational Geometry: Algorithms and Applications*, 3rd edition, published by Springer-Verlag, 2008.
- [2] Aure T., and Held M., "Heuristic for generation of random polygons" 8th Canadian Conference On Computational Geometry (CCCG), Ottawa, Canada, pp.38-44, 1996.
- [3] Rote G., Santos F., and Streinu I., "Pseudo-Triangulation – a Survey" *Discrete Comput. Geom.* 2007.
- [4] Aichholzer O., Aurenhammer F., Krasser H., and Speckmann B., "Convexity minimizes pseudo-triangulations" *Computational Geometry* 28.2004.3-10.
- [5] Dailey D., and Whitfield D., "Constructing Random Polygons" SIGITE⁰⁸, USA, pp.119-124, 2008.
- [6] Chazelle B., "On the Convex Layers of a Planar Set" *IEEE Tran. Information Theory*, Vol. IT-31, No. 4, pp. 509-517, 1985.

Dynamic LZW for Compressing Large Files

Chung-E Wang

Department of Computer Science
California State University, Sacramento
Sacramento, CA 95819-6021

Abstract. *The amount of data stored digitally continues to grow dramatically across many fields, along with the need for algorithms to efficiently compress this data for storage and transmission. In this paper, we describe an improvement of LZW data compression. We employ a dynamic dictionary, in which least recently used and aging algorithms are used to replace infrequently used entries. We demonstrate that these pruning techniques result in significant gains in compression ratios for large data files.*

Keywords. LZW data compression, dynamic dictionary, table pruning, least recently used, aging replacement.

1. Introduction

Data compression algorithms are widely used for data storage and data transmission. A popular lossless method known as Lempel-Ziv (LZ) compression [1] replaces a string of characters with an index into a dictionary that is built during the compression process. There are many modifications of the original LZ compression algorithm, many of which are feature different implementations of the dictionary [1]-[6].

Lempel-Ziv-Welch (LZW) compression [4] is Terry Welch's modification of LZ compression. This algorithm uses a string table to implement the dictionary. Initially, the string table contains all strings of length 1. During the process of compression, the algorithm adds every new string it sees to the string table. To compress, the algorithm scans the input data for the longest matching string in the string table and outputs the index of that string as the result of the compression. Compression occurs when a long string of characters is replaced by a shorter index.

One difficulty in using LZW compression on large data files is in managing the dictionary, as the size of the string table often surpasses that of available memory. Here we propose a new method called table pruning for managing the dictionary. We have demonstrated our method with least recently used and aging replacement algorithms and improved the compression ratio obtained from using LZW alone. Finally, we discuss some factors we observed to be crucial to compression ratios.

2. Handling the Ever Growing String Table

One drawback to be considered in implementing the LZW algorithm is the ever-growing string table; as more data is analyzed the dictionary becomes increasingly large. The table must be managed, as computer memory is limited. Two existing methods for handling the ever-growing string table [1], [9] are discussed below.

2.1 Table Freezing

This is the method used by the original LZW algorithm. This method picks a size of the string table and does not allow the table to grow beyond that size. Instead, it continues the compression according to the frozen table. It is simple and easy but it doesn't work well with large files.

2.2 Table Flushing

This is the method used in [9]. This method computes the current compression ratio periodically. When the table is full and the current compression ratio drops below some predetermined threshold value, it flushes the string table. That is, the algorithm abandons the current string table and builds a new one when compressing the remaining input data.

Flushing can get rid of infrequently used entries. However, this drastic operation also flushes out frequently used entries. Thus, it doesn't improve compression ratios for a lot of input files.

2.3 Table Pruning

We propose to prune the string table. Once the string table becomes full and an additional entry is needed, we replace an infrequently used entry with the new entry and the compression continues. However, the problem of selecting an infrequently used entry for pruning is non-trivial.

3. Selecting an Infrequently Used Entry for Replacement

Many strategies exist for selecting infrequently used entries, a problem similar to selecting replacement pages for virtual memory management systems. Here we utilize principals from two of these so-called “page replacement algorithms”: Least Recently Used and Aging Replacement.

3.1 Least Recently Used (LRU)

In LRU, the entry which has not been accessed for the longest is selected as the replacement entry. In our implementation, we use a self-organizing list to select the least recently used entry. This list contains an index to every entry of the string table. During the compression, every time an entry is accessed, the corresponding index is moved to the front of the list. When a replacement entry is needed, it's selected from the end of the list.

3.2 Aging Replacement

In addition to LRU, we use the aging replacement algorithm to manage the string table. In this algorithm, we keep a value called time to live (TTL) for every table entry. When an entry is created the corresponding TTL is initialized to some predetermined value. Periodically, the TTL is decreased. When the TTL becomes zero, the entry is deleted from the string table. In order to let table accesses closer to the present time have more impact than table accesses long ago, when an entry is accessed, its TTL is reset to $(\text{current value}/2 + \text{initially value})$. When a replacement entry is needed, an unused entry or the one with the smallest TTL will be selected.

4. Implementation Complicatedness

The implementation of our idea is somewhat complicated mainly due to the representation and management of the string table.

In order to speed up the process of searching the string table, the double hashing technique is used to implement the string table. In order to achieve a good performance of the hash table, the size of the hash table is 25% bigger than the needed size of the string table.

Because of hashing, deleting or replacing entry of the string table cannot be done directly. To replace an entry, we need to mark an entry as deleted and use an unused entry for the new entry. Because of this, we need to clean up marked entries before the hash table gets full. To do so, we need to recreate the hash table periodically.

Moreover, if LRU algorithm is used to select

infrequently used entries, a linked list is added to implement the self-organizing list. If the aging replacement algorithm is used, a heap is added to accelerate the process of finding the entry with the smallest TTL.

5. Factors That Affect the Compression Ratio

We found the following factors to be crucial to the resulting compression ratio, the ratio of the compressed file size to the original file size.

5.1 The maximum size of the string table

The maximum size of the string table determines the number of bits needed to represent a code word, i.e. an index to the string table. The larger the size the greater number of bits will be required to represent an index. To compress a small file, a smaller table results in a smaller compressed file. To compress a large file, a smaller table holds less strings and thus less chance of using an index to encode a long string of characters and thus reduce the compression efficacy. Algorithms in [7]-[9] reduce the size of the compressed file by using variable length tables. According to [9], the maximum number of bits can be saved is 3840. For large files with millions of bytes, this is insignificant.

To fully utilize all possible combinations of bits of compressed codeword, the size of the string table is a power of 2. After experimenting with different table sizes ranging from 2^{12} through 2^{22} , we found that a table of size 2^{16} , i.e. 65536 works well with large text files.

5.2 The period of recreating the hash table

The hash table must be recreated before the hash table becomes full. However, if the table is recreated too often, the program speed is greatly decreased. Moreover, according to our observations, different lengths of period result in different compression ratios.

According to our study, for a table of size 65536, the optimal period to recreate the string table is after compressing 4096 strings.

5.3 The interval of decreasing TTLs

Recreating the hash table is a time consuming process in which every entries of the table must be accessed. In order to reduce the speed impact of managing the hash table, we paired the task of recreating the hash table with the task of decreasing TTLs. That is, recreating the hash table and decreasing the TTLs are done at the same time.

5.4 The initial value of TTLs

If the initial value of TTLs is too small, many entries of the string table will be deleted too soon and thus the table pruning method has the same draw back as the table flushing method.

After some experiments, we found the optimal initial TTL value to be the size of the table divided by 1024. That is, for a table of 65536, the best initial TTL value is 64.

6. Empirical Results

To evaluate the effectiveness of our methods, we test our methods with test files from the web site Canterbury Corpus. (<http://corpus.canterbury.ac.nz>). The Canterbury Corpus is a benchmark to enable researchers to evaluate lossless compression methods.

We present our results in the following tables. The three test files *E.coli*, *bible.txt* and *world192.txt* are in the large corpus collection of the Canterbury Corpus. In these experiments, we have used string tables of size 65536, hash table recreating period of 4096, and TTL initial value of 64.

Table 1: Compressed file sizes

	E.coli	bible.txt	world192.txt
Original file size (bytes)	4,638,690	4,047,392	2,473,400
LZW	1,213,588	1,417,762	925,826
LZW/Aging	1,199,245	1,242,153	804,493
LZW /LRU	1,234,866	1,291,120	850,560

Table 2: Compression ratios

	E.coli	bible.txt	world192.txt
LZW	3.82	2.85	2.67
LZW/aging	3.87 (+1%)	3.26 (+12%)	3.07 (+13%)
LZW /LRU	3.76 (-1%)	3.13 (+9%)	2.91 (+8%)

Besides the test files from The Canterbury Corpus, we have also tested our methods with other text files. Compression tests on these files yielded the following findings:

- LZW/aging does better than LZW/LRU 90% of the time.
- LZW/aging can improve the compression ratio over LZW by 10-15% for 90% of the files tested.

Preliminary tests of our methods with video and image files also gave promising results. The original LZW consistently inflate video and image files by about 25%. Our LZW/aging can deflate video and image files by 1% consistently. In other words, LZW/aging can improve the compression gain by 26% for large video or image files over the original LZW.

7. Decompression

Decompression is a simple task relative to compression. Since there is no need to search the string table, the hashing technique is not required and thus there is no need to recreate the hash table periodically. However, a heap or a self-organizing list is still needed for LZW/aging and LZW/LRU respectively. The purpose of including a heap or a self-organizing list is to synchronize the decompression string table with the compression string table so the two tables use the same sequence of replacement entries.

8. Conclusions

We have described an improvement of LZW data compression which use table pruning techniques. With more efficient management of the dynamic dictionary, a better compression ratio may be achieved. Specifically, we show that LZW/aging can significantly improve the compression ratio for most large files.

According to our experiments, we identified four factors that are crucial to the compression ratios of LZW/aging and LZW/LRU. These factors are the size of the string table, the period of recreating the hash table, the interval of decreasing TTLs and the initial value of TTLs. Further work needs be done to characterize the combinatorial effects of these factors and determine their optimal combinations.

While the aging algorithm provided considerable improvement over LZW compression alone, additional replacement algorithms should be explored. Finally, we will explore more on how the compression methods perform on different types of data files such as video and image files.

9. References

- [1] Ziv, J. and Lempel A. 1977. "A universal algorithm for sequential data compression". IEEE Trans. Inf. Theory 23, 3

- (May), 337-343.
- [2] Ziv, J., & Lempel, A. 1978. "Compression of individual sequences via variable-rate coding", *IEEE Trans. Inform. Theory*, 24(5), 530-536.
 - [3] Storer, J.A., & Szymanski, T.G. (1982) "Data Compression via Textual Substitution," *Journal of ACM*, 29(4), 928-951.
 - [4] Welch, T. A. 1984. "A technique for high-performance data compression". *Computer* 17, 6 (June), 8-19.
 - [5] Willard, L., Lempel, A., Ziv, J. & Cohn, M. (1984) "Apparatus and method for compressing data signals and restoring the compressed data signals", US patent - US4464650.
 - [6] Horspool, R.N. (1991) "Improving LZW," *Proc. Data Compression Conference (DCC 91)*, Snowbird, Utah, IEEE Computer Society Press, Los Alamitos, CA, pp. 332-341.
 - [7] Ouaisa, K., Abdat, M. and Plume, P. 1995. "Adaptive limitation of the dictionary size in LZW data compression". *Proceedings 1995 IEEE International Symposium on Information Theory*.
 - [8] Chai, Z. and Chen W. 2004. "An adaptive LZWCompression algorithm using changeable maximum-code-length". *Fourth International Conference on Computer and Information Technology (CIT'04)* pp. 1175-1180.
 - [9] Raghuwanshi, B.S., Jain, S. Chawda, D. and Varma, B. 2009. "New dynamic approach for LZW data compression". *IJCNS Vol. 1, No. 1 (October)*, 22-26.

SESSION

THEORY + PROOF + VERIFICATION METHODS + INTERESTING RESULTS

Chair(s)

TBA

Formal Verification of DES Using the Mizar Proof Checker

Hiroyuki Okazaki¹, Kenichi Arai², and Yasunari Shidama¹

¹Shinshu University, 4-17-1 Wakasato Nagano-city, Nagano 380-8553, Japan

²Department of Information Technology, Nagano Technical High School,
3-9-1 sasideminami Nagano-city, Nagano 380-0948, Japan

Abstract—*In this paper, we introduce our formalization of Data Encryption Standard (DES) algorithm. DES, which was formerly the most widely used symmetric cryptosystem in the world, is a block cipher that was selected by the National Bureau of Standards as an official Federal Information Processing Standard for the United States in 1976. We prove the correctness of our formalization by using the Mizar proof checking system as a formal verification tool. Mizar is a project that formalizes mathematics with a computer-aided proving technique. The main objective of this work is to prove the security of cryptographic systems by using the Mizar proof checker.*

Keywords: Formal Verification, Mizar, Cryptology, Data Encryption Standard (DES)

1. Introduction

Mizar[1], [2] is a project that formalizes mathematics with a computer-aided proving technique. The objective of this study is to prove the security of cryptographic systems by using the Mizar proof checker. To achieve this, we are intend to formalize some topics concerning cryptology.

In this paper, we introduce our formalization of the Data Encryption Standard (DES). DES, which was formerly the most widely used symmetric cryptosystem in the world, is a block cipher that was selected by the National Bureau of Standards as an official Federal Information Processing Standard for the United States in 1976[3]. DES is now considered to be insecure and has already been superseded by the Advanced Encryption Standard (AES)[4]. Please see [5] and [6] about recent information on DES. However, DES is a typical block cipher, and it has a strong influence on the design of its successors. Thus, we will verify another block cipher system that we will develop in the future by using a method similar to our formalization of DES with the Mizar system. We formalized the DES algorithm as shown in FIPS46-3[3] in the Mizar language. We then verified the correctness of the formalized algorithm that the ciphertext encoded by the algorithm can be decoded uniquely by the same algorithm by using the Mizar proof checker.

The remainder of this study is organized as follows. In Section 2, we briefly introduce the Mizar project. In Section 3, we briefly introduce the Data Encryption Standard (DES). In Section 4, we discuss our strategy for formalizing DES in Mizar. In Sections 5 and 6, we propose a formalization

of DES. We conclude our discussion in Section 7. The definitions and theorems in this study have been verified for correctness by using the Mizar proof checker.

2. Mizar

Mizar[1], [2] is an advanced project of the Mizar Society led by Andrzej Trybulec that formalizes mathematics with a computer-aided proving technique. The Mizar project describes mathematical proofs in the Mizar language, which is created to formally describe mathematics. The Mizar proof checker operates in both Windows and UNIX environments, and registers the proven definitions and theorems in the Mizar Mathematical Library (MML).

Furthermore, the objective of the Mizar project is to create a check system for mathematical theses. What formalizes the proof of mathematics by Mizar and describes it is called “article”. When an article is newly described, it is possible to advance it by referring to articles registered in the MML that have already been inspected as proof. Likewise, other articles can refer to an article after it has been registered in the MML. Although the Mizar language is based on the description method for general mathematical proofs, the reader should consult the references for its grammatical details, because Mizar uses a specific, unique notation[1], [2], [7], [8], [9].

3. Data Encryption Standard

In this section, we review the outline of the DES algorithm. The DES algorithm takes a 64bits length plaintext block and a 64 bits length secret key, and transforms into a 64 bits length ciphertext block. Decryption must be performed using the same key as used for encryption, however it should be performed with the key scheduling process altered so that the decryption is the reverse of the encryption. Figure 1 shows a sketch of the structure of DES.

DES is a type of iterated block cipher with the Feistel structure. The Feistel structure ensures that the encryption and decryption are similar processes, except that the round keys are used in the reverse order when decrypting. The algorithm is composed of the Feistel structure and a key scheduling function. In the Feistel structure of DES, there are 16 rounds of processing iterations. Before the main iterations, a given block of plaintext is permuted by IP and is then divided into two 32 bits length blocks, L_0 and

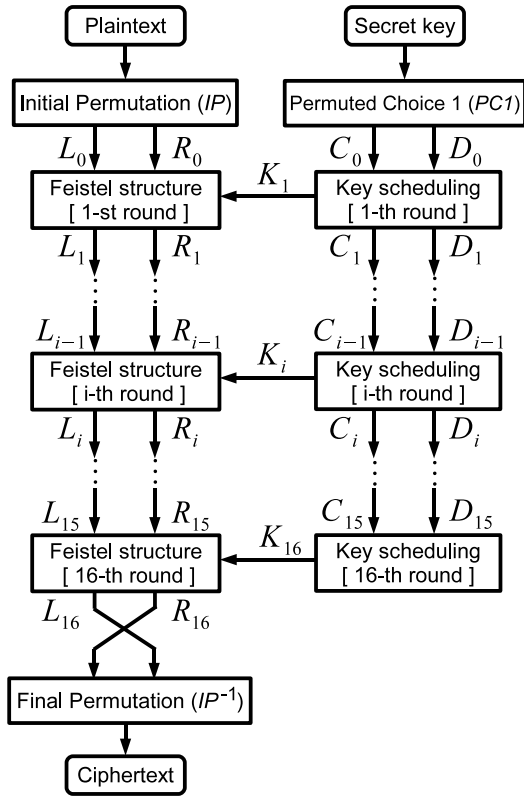


Figure 1: Structure of DES

R_0 . The i -th round is performed as follows:

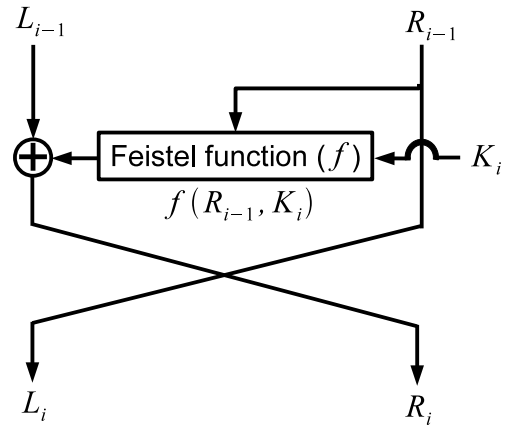
$$\begin{aligned} L_i &= R_{i-1}, \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i), \end{aligned}$$

where $1 \leq i \leq 16$, f is the Feistel function of DES, and K_i is the i -th round key that is yielded by the key schedule function KS from the given secret key. Figure 2 shows a sketch of the i -th round of Feistel structure. Finally, the final permutation IP^{-1} transforms the concatenation of L_{16} and R_{16} into the ciphertext.

4. Strategy of Formalizing DES in Mizar

In Mizar, there are two ways to define computational routines in an algorithmic sense. One way is by defining a routine as a functor. A functor is a relation between the input and output of a routine in Mizar. It is easy to write and understand the formalization of a routine as a functor, because the format of a functor in Mizar is similar to that of a function in certain programming languages.

The other way is by defining a routine as a Function. A Function is a map from the space of the input onto that of the output. We can handle a Function as an element of the set of Functions. Note that both functor and Function can take a Function as their substitutable subroutines.

Figure 2: i -th round of Feistel structure

In Section 5, we will formalize the algorithm of generalized DES as a functor that takes substitutional subroutines. This generalized definition of DES is easily reusable for the formalization of other ciphers. In Section 6, we first formalize the subroutines, that is, the primitives of DES, according to FIPS46-3[3]. We will then formalize the DES algorithm by using the formalization of the generalized definition in Section 5 and the primitives in Section 6.1.

5. Formalization of Generalized DES

First, we formalize the generalized algorithm of DES as a functor in the Mizar language as follows:

Definition 5.1: (Codec of generalized DES)

```

let n,m,k be non empty Element of NAT,
    RK be Element of (k-tuples_on
        (m-tuples_on BOOLEAN)),
    F be Function of [:n-tuples_on BOOLEAN,
        m-tuples_on BOOLEAN:],
    n-tuples_on BOOLEAN,
    IP be Permutation of (2*n)-tuples_on
        BOOLEAN,
    M be Element of (2*n)-tuples_on BOOLEAN;
func DES-like-CoDec(M,F,IP,RK) ->
    Element of (2*n)-tuples_on BOOLEAN
means
ex
L,R be sequence of (n-tuples_on BOOLEAN)
st
L.0=SP-Left(IP.M) & R.0=SP-Right(IP.M) &
(for i be Element of NAT st 0<=i &
    i<=k-1 holds L.(i+1)=R.i &
    R.(i+1)=Op-XOR(L.i,F.(R.i,RK/.(i+1))))
& it=IP".((R.k)^(L.k));

```

□

Note that we can express the algorithm of general Feistel ciphers¹ by using the functor DES-like-CoDec if we give the identical permutation of $(2*n)$ tuples_on BOOLEAN IP. Moreover, SP-Left and SP-Right are functions that divide a finite sequence into two 32 bits length blocks (Figure 3).

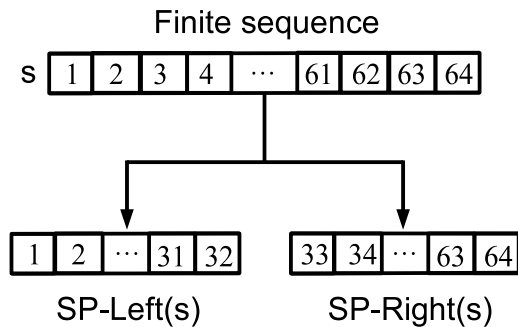


Figure 3: SP-Left and SP-Right

We then prove the following theorem:

Theorem 5.1: (Correctness of generalized DES)

for n, m, k be non empty Element of NAT,
 RK be Element of k -tuples_on
 (m -tuples_on BOOLEAN),
 F be Function of $[:n$ -tuples_on BOOLEAN,
 m -tuples_on BOOLEAN:],
 n -tuples_on BOOLEAN,
 IP be Permutation of $(2*n)$ -tuples_on
 BOOLEAN,
 M be Element of $(2*n)$ -tuples_on BOOLEAN
 holds
 DES-like-CoDec (DES-like-CoDec (M, F, IP,
 RK), F, IP, Rev (RK)) = M

□

Thus, we proved in the Mizar system that the ciphertext encoded by any Feistel cipher algorithm can be decoded uniquely with the same algorithm and secret key that were used in encryption.

6. Formalization of DES

In this section, we formalize the DES algorithm according to FIPS46-3[3] in the Mizar language. First, we will formalize the DES primitives according to FIPS46-3[3]. Next, we will formalize and prove the correctness of the DES algorithm.

¹General Feistel ciphers are composed only of iterated rounds. In other words, General Feistel ciphers do not have initial and final permutations.

6.1 DES Primitives

6.1.1 S-Boxes

We formalize the S-BOX S_1 as the following functor in the Mizar language:

Definition 6.1: (S-Box S_1)

```
func DES-SBOX1 -> Function of 64,16
means
it.0=14 & it.1=4 & it.2=13 &
      :
      (omitted)
      :
it.61=0 & it.62=6 & it.63=13;
```

□

We similarly defined the other S-Boxes, DES-SBOX2,.....,and DES-SBOX8.

6.1.2 Initial Permutation

We formalize the initial permutation IP as the following functor in the Mizar language:

Definition 6.2: (IP as functor)

```
let r be Element of 64-tuples_on
BOOLEAN;
func DES-IP(r) ->
Element of 64-tuples_on BOOLEAN
means
it.1=r.58 & it.2=r.50 & it.3=r.42 &
      :
      (omitted)
      :
it.62=r.23 & it.63=r.15 & it.64=r.7;
```

□

We then formalize the initial permutation as the following function:

Definition 6.3: (IP as function)

```
func DES-PIP ->
Function of 64-tuples_on BOOLEAN,
64-tuples_on BOOLEAN
means
for i be Element of 64-tuples_on
BOOLEAN
holds
it.i=DES-IP(i);
```

□

We similarly defined the functor of the final permutation DES-IPINV and the function of the DES-PIPINV. Note that the final permutation is the inverse of IP.

6.1.3 Feistel Function

Figure 4 shows a sketch of the Feistel function.

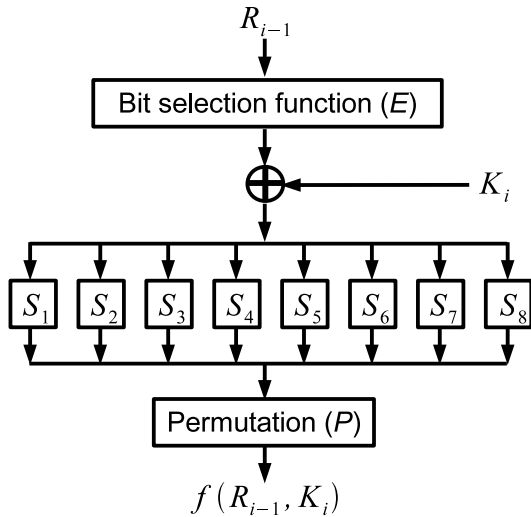


Figure 4: Feistel function

We formalize the bit selection function E as the following functor in the Mizar language:

Definition 6.4: (E as functor)

```
let r be Element of 32-tuples_on
  BOOLEAN;
func DES-E(r) ->
  Element of 48-tuples_on BOOLEAN
means
it.1=r.32 & it.2=r.1 & it.3=r.2 &
  :
  (omitted)
  :
it.46=r.31 & it.47=r.32 & it.48=r.1;
```

□

We then formalize the permutation P as follows:

Definition 6.5: (P as functor)

```
let r be Element of 32-tuples_on
  BOOLEAN;
func DES-P(r) ->
  Element of 32-tuples_on BOOLEAN
means
it.1=r.16 & it.2=r.7 & it.3=r.20 &
  :
  (omitted)
  :
it.30=r.11 & it.31=r.4 & it.32=r.25;
```

□

Next, we formalize the Feistel function F as the following functor in the Mizar language:

Definition 6.6: (Feistel function F as functor)

```
let R be Element of 32-tuples_on
```

```
BOOLEAN,
RKey be Element of 48-tuples_on
  BOOLEAN;
func DES-F(R, RKey) ->
  Element of 32-tuples_on BOOLEAN
means
ex
D1, D2, D3, D4, D5, D6, D7, D8 be Element of
  6-tuples_on BOOLEAN,
x1, x2, x3, x4, x5, x6, x7, x8 be Element of
  4-tuples_on BOOLEAN,
C32 be Element of 32-tuples_on BOOLEAN
st
D1= (DES-DIV8 (Op-XOR (DES-E (R) , RKey) ) ) .1 &
D2= (DES-DIV8 (Op-XOR (DES-E (R) , RKey) ) ) .2 &
D3= (DES-DIV8 (Op-XOR (DES-E (R) , RKey) ) ) .3 &
D4= (DES-DIV8 (Op-XOR (DES-E (R) , RKey) ) ) .4 &
D5= (DES-DIV8 (Op-XOR (DES-E (R) , RKey) ) ) .5 &
D6= (DES-DIV8 (Op-XOR (DES-E (R) , RKey) ) ) .6 &
D7= (DES-DIV8 (Op-XOR (DES-E (R) , RKey) ) ) .7 &
D8= (DES-DIV8 (Op-XOR (DES-E (R) , RKey) ) ) .8 &
Op-XOR (DES-E (R) , RKey) =
  D1^D2^D3^D4^D5^D6^D7^D8 &
x1=N16toB4. (DES-SBOX1. (B6toN64 (D1) ) ) &
x2=N16toB4. (DES-SBOX2. (B6toN64 (D2) ) ) &
x3=N16toB4. (DES-SBOX3. (B6toN64 (D3) ) ) &
x4=N16toB4. (DES-SBOX4. (B6toN64 (D4) ) ) &
x5=N16toB4. (DES-SBOX5. (B6toN64 (D5) ) ) &
x6=N16toB4. (DES-SBOX6. (B6toN64 (D6) ) ) &
x7=N16toB4. (DES-SBOX7. (B6toN64 (D7) ) ) &
x8=N16toB4. (DES-SBOX8. (B6toN64 (D8) ) ) &
C32=x1^x2^x3^x4^x5^x6^x7^x8 &
it=DES-P (C32);
```

□

Here, the function DES-DIV8 divides the 48-bits length input into eight 6-bits length blocks. The function N16toB4 yields a 4-bits length block from a natural number less than 16. The function B6toN64 yields a natural number less than 64 from a 6-bits length input.

Finally, we formalize the Feistel function F as the following function:

Definition 6.7: (Feistel function F as function)

```
func DES-FFUNC ->
  Function of [:32-tuples_on BOOLEAN,
    48-tuples_on BOOLEAN:],
    32-tuples_on BOOLEAN
means
for z be Element of [:32-tuples_on
  BOOLEAN, 48-tuples_on BOOLEAN:]
holds
it.z=DES-F(z`1 , z`2);
```

□

6.1.4 Key Scheduling Function

Figure 5 shows a sketch of the key scheduling function.

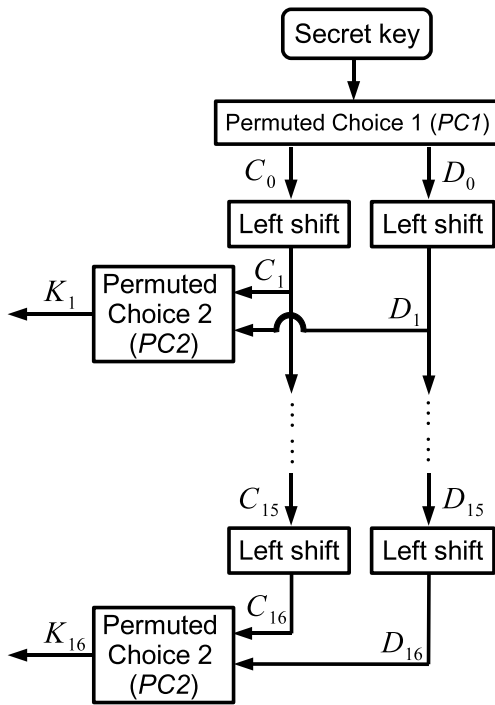


Figure 5: Key Scheduling Function

We formalize the permutation PC1 as the following functor in the Mizar language:

```

Definition 6.8: (PC1 as functor)
let r be Element of 64-tuples_on
  BOOLEAN;
func DES-PC1(r) ->
  Element of 56-tuples_on BOOLEAN
means
it.1=r.57 & it.2=r.49 & it.3=r.41 &
  :
  (omitted)
  :
it.54=r.20 & it.55=r.12 & it.56=r.4;
    
```

We similarly defined the functor of PC2 as DES-PC2.

Next, we formalize the table of the numbers of Left-Shift as the following functor:

```

Definition 6.9: (Table of Left-Shift)
func bitshift_DES -> FinSequence of NAT
means
it is 16-long & it.1=1 & it.2=1 &
it.3=2 & it.4=2 & it.5=2 & it.6=2 &
it.7=2 & it.8=2 & it.9=1 & it.10=2 &
    
```

```

it.11=2 & it.12=2 & it.13=2 & it.14=2 &
it.15=2 & it.16=1;
    
```

□

Finally, we formalize the key scheduling function as the following functor:

Definition 6.10: (Key Scheduling function)

```

let Key be Element of 64-tuples_on
  BOOLEAN;
func DES-KS(Key) ->
  Element of (16-tuples_on
  (48-tuples_on BOOLEAN))
means
ex
C,D be sequence of (28-tuples_on
  BOOLEAN)
st
C.0=Op-Left(DES-PC1(Key),28) &
D.0=Op-Right(DES-PC1(Key),28) &
(for i be Element of NAT st 0<=i &
  i<=15 holds it.(i+1)=
  DES-PC2((C.(i+1))^(D.(i+1))) &
  C.(i+1)=Op-Shift(C.i,bitshift_DES.i) &
  D.(i+1)=Op-Shift(D.i,bitshift_DES.i));
    
```

□

6.2 DES Algorithm

In this section, we formalize the DES algorithm according to FIPS46-3[3] in the Mizar language by using our formalization of the generalized DES algorithm in Section 5 and the DES primitives in Section 6.1.

Definition 6.11: (DES Algorithm)

```

let RK be Element of (16-tuples_on
  (48-tuples_on BOOLEAN)),
F be Function of [:32-tuples_on BOOLEAN,
  48-tuples_on BOOLEAN:],
  32-tuples_on BOOLEAN,
IP be Permutation of 64-tuples_on
  BOOLEAN,
M be Element of 64-tuples_on BOOLEAN;
func DES-CoDec(M,F,IP,RK) ->
  Element of 64-tuples_on BOOLEAN
means
ex
IPX be Permutation of (2*32)-tuples_on
  BOOLEAN,
MX be Element of (2*32)-tuples_on
  BOOLEAN
st
IPX=IP & MX=M &
it=DES-like-CoDec(MX,F,IPX,RK);
    
```

□

Definition 6.12: (Encode Algorithm of DES)

```

let plaintext, secretkey be Element of
  64-tuples_on BOOLEAN;
func DES-ENC(plaintext, secretkey) ->
  Element of 64-tuples_on BOOLEAN
equals
DES-CoDec(plaintext, DES-FFUNC, DES-PIP,
  DES-KS(secretkey));

```

□

Definition 6.13: (Decode Algorithm of DES)

```

let ciphertext, secretkey be Element of
  64-tuples_on BOOLEAN;
func DES-DEC(ciphertext, secretkey) ->
  Element of 64-tuples_on BOOLEAN
equals
DES-CoDec(ciphertext, DES-FFUNC, DES-PIP,
  Rev(DES-KS(secretkey)));

```

□

Finally, we then prove the following theorem:

Theorem 6.1: (Correctness of DES)

```

for message, secretkey be Element of
  64-tuples_on BOOLEAN
holds
DES-DEC(DES-ENC(message, secretkey),
  secretkey)=message

```

□

Thus, we proved using the Mizar system that the ciphertext encoded by the DES algorithm can be decoded uniquely with the same algorithm and secret key that were used in encryption.

7. Conclusion

In this study, we introduced our formalization of the DES algorithm in Mizar. We also proved the correctness of the DES algorithm by using the Mizar proof checking system as a formal verification tool. Currently, we are attempting to analyze the security of DES.

Acknowledgments

This work was supported by JSPS KAKENHI 2124000101.

References

- [1] *Mizar Proof Checker*. Available at <http://mizar.org/>.
- [2] E. Bonarska, *An Introduction to PC Mizar*, Mizar Users Group, Fondation Philippe le Hodey, Brussels, 1990.
- [3] U.S. Department of Commerce/National Institute of Standards and Technology, *FIPS PUB 46-3, DATA ENCRYPTION STANDARD (DES)*, Federal Information Processing Standards Publication, 1999. Available at <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>.
- [4] U.S. Department of Commerce/National Institute of Standards and Technology, *FIPS PUB 197, Advanced Encryption Standard (AES)*, Federal Information Processing Standards Publication, 2001. Available at <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.
- [5] NIST Special Publication 800-67 Version 1.1, *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*, National Institute of Standards and Technology, 2008. Available at <http://csrc.nist.gov/publications/nistpubs/800-67/SP800-67.pdf>.
- [6] ISO/IEC 18033-3:2010, *Information technology – Security techniques – Encryption algorithms – Part 3: Block ciphers*, 2010. Available at http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54531.
- [7] M. Muzalewski, *An Outline of PC Mizar*, Fondation Philippe le Hodey, Brussels, 1993.
- [8] Y. Nakamura, T. Watanabe, Y. Tanaka, and P. Kawamoto, *Mizar Lecture Notes (4th Edition)*, Shinshu University, Nagano, 2001. Available at <http://markun.cs.shinshu-u.ac.jp/kiso/projects/proofchecker/mizar/index-e.html>.
- [9] A. Grabowski, A. Kornilowicz, and A. Naumowicz, *Mizar in a Nutshell*, Journal of Formalized Reasoning 3(2), pp.153-245, 2010.

Reasoning about Hybrid States

Angel Rivera

Computer Science Department, Utica College, Utica, NY, U.S.A.

Abstract—In this paper, we present a new semantics for the well-known normal system of modal logic \mathcal{K} based on the notion of convergence spaces. The purpose is to use convergence spaces to model and reason about systems with both discrete and continuous states, so-called hybrid state spaces. \mathcal{K} is sound and complete with respect to the class of convergence space models, and we show that Kripke frames and McKinsey and Tarski's topological frames are special cases of our convergence space models.

Keywords: Modal logic, convergence spaces, hybrid states, convergence space models

1. Introduction

In Computer Science, a variety of logic-based formal techniques are available for rigorously reasoning about the evolution of systems in discrete time using directed graphs for their semantics. Some of these logics, most notably **S4**, have been interpreted to act on dynamical systems evolving in continuous time using topological spaces as the base for their semantic models. In order to deal with systems with both *digital* (discrete) and *analog* (continuous) components, we need a suitable framework. One such candidate can be found in the notion of *convergence spaces*. Convergence spaces are flexible, and generalize both directed graphs and topological spaces.

In this paper, we present convergence spaces as semantic for the (smallest) normal system of modal logic \mathcal{K} which we would like to use for reasoning about *hybrid* states. Our convergence space models generalize two well-known semantics for \mathcal{K} and **S4**, respectively: Kripke frames and McKinsey and Tarski's topological models. Our hope is that convergence space models can serve as a bridge between Kripke frames and topological models so they can both benefit from the wealth of work in both areas.

This paper is organized as follows. First, we introduce convergence spaces, and the notions of *interior* and *closure* which we will use in defining the \Box and \Diamond modal operators. Next, we present the modal logics \mathcal{K} in one of its classical axiomatic forms. We then define a new semantic for \mathcal{K} based convergence spaces. We present an example of a system with a hybrid space and show some properties using our formalism. Last, we show that Kripke models and topological models are special cases of convergence models, and comment on the relation of our models and the well-known Montague-Scott's *neighborhood semantics*.

This paper overviews the framework of convergence space models as presented in [1].

2. Preliminaries

In this section, we present the basic notation and the different ideas that form the foundation for the rest of this paper. We introduce the framework of Convergence spaces starting with the notion of filters, which form their building blocks, and present some basic properties. We end this section with a presentation of the Modal Logic of "necessity" and "possibility" in one of its classical axiomatic forms.

2.1 Convergence Spaces

Given a set X , a *filter* is a collection of subsets of X such that it is closed under finite intersections, closed under arbitrary unions, and it contains X . The following definition formalizes this notion.

Definition 2.1 (Filters [2]): Let X be a non-empty set. A collection \mathcal{F} of subsets of X is called a *filter* if and only if, for $A, B \subseteq X$,

- (F1) If $A, B \in \mathcal{F}$ then $(A \cap B) \in \mathcal{F}$
- (F2) If $A \in \mathcal{F}$ and $A \subseteq B$ then $B \in \mathcal{F}$
- (F3) $X \in \mathcal{F}$
- (F4) $\emptyset \notin \mathcal{F}$

We use $\Phi(X)$ to denote the set of filters of X . A non-empty collection \mathcal{B} is called a *filter basis* if it only satisfies (F1) and (F4). We can create filters from subsets and filter basis on X . The *principal filter* of $A \subseteq X$, $A \neq \emptyset$, denoted by $[A]$, is the filter containing all the super-sets of A . If $A = \{x\}$, a singleton, we use $[x]$ instead and call this the *point filter* of x . If \mathcal{B} is a filter basis, then $[\mathcal{B}]$ is the filter consisting of all the super-sets of elements in \mathcal{B} . For filters $\mathcal{F}, \mathcal{G} \in \Phi(X)$, If $\mathcal{F} \subseteq \mathcal{G}$ then we say that \mathcal{F} is *coarser* than \mathcal{G} , and \mathcal{G} is *finer* than \mathcal{F} .

2.2 Convergence Spaces

A convergence space consists of a set X and a binary relation \downarrow that associates filters with points in X . The following definition formalize this notion.

Definition 2.2 (Convergence Space [2]): Let X be a non-empty set. A binary relation \downarrow on $\Phi(X) \times X$ is a *convergence structure* if it is "closed under super-filters;" that is,

- (C1) if $\mathcal{F} \downarrow x$ and $\mathcal{F} \subseteq \mathcal{G}$ then $\mathcal{G} \downarrow x$.

Whenever $\mathcal{F} \downarrow x$ we say that filter " \mathcal{F} converges to x ." We call the pair (X, \downarrow) a *convergence space*, and we shall use **Conv** to denote the set of all convergence spaces.

The definition of *convergence spaces* varies in the literature. For instance, the definition found in [3], [4] and [5] has an additional axiom; and in [6], [7] and [8] convergence spaces are called *filter spaces*. We follow the definition of [2], and

others, as it is the most general and compatible with existing models for modal logics.

Notice that, aside from (C1), there is great flexibility in the creation of a convergence relation. This allows us to see many well-known structures "under the lens" of convergence spaces; graphs, pretopologies and topologies, for example, form proper subcategories of the category of convergence spaces [5].

2.3 Digraphs, topologies and convergence spaces

For computer science, an important result on convergence spaces is that all directed graphs, or *digraphs*, can induce, or be seen as, convergence spaces. In [5], Blair et al. point out that there are (possibly infinitely) many ways of doing this embedding. We present the way commonly found in the literature (cf. [5], [9], [10] and [11]).

The approach relies on using the *graph-neighborhood* of a vertex as the basis for the smallest filter converging to that vertex in the convergence structure. For digraph $G = (V, E)$, with vertex-set V and edge-set E , we can induce the convergence space (V, \downarrow_E) by demanding that, for all $v \in V$ and filter $\mathcal{F} \in \Phi(V)$, $\mathcal{F} \downarrow v$ iff $\{v' \in V \mid vEv'\} \in \mathcal{F}$.

The relation between digraphs and induced convergence spaces works both ways. We can recover the edge-set E of a digraph (V, E) from its induced convergence space representation (V, \downarrow) by demanding that vEv' if and only if $\{v'\} \downarrow v$, for all $v, v' \in V$ (cf. [4], [11]–[13]).

We can also induce a convergence space from a topology. The idea is to use the collection of open sets containing a point x as the filter basis for the smallest filter attached to x ; we then close the convergence relation under super filters. We shall call the generated space a *topological* convergence space [12], [14]–[16].

2.4 Discrete, continuous and hybrid spaces

The notions of discrete, continuous and hybrid spaces rely on the idea of the *neighborhood of a point*. Following [12], the neighborhood filter of a point x , $\mathcal{N}(x)$, is the filter obtained as the intersection of all the filters converging to that point; an element $N \in \mathcal{N}(x)$ is called a *neighborhood* of x .

We then stipulate that, if $\mathcal{N}(x)$ has a bottom element, for every $x \in X$, then the space is discrete; for example, all convergence spaces induced by digraphs are discrete. On the other hand, if for every $x \in X$, $\mathcal{N}(x)$ does not have a bottom element, then X is continuous; for example, let $X = \mathbb{R}$, and use its well-known *standard* topology as the basis of the following convergence structure: for each $x \in X$, let $\mathcal{F} \downarrow x$ if \mathcal{F} contains the collection of all open intervals containing x , i.e. $\{(x', x'') \mid x' < x < x''\}$, then the neighborhood filter does not have a bottom element and, hence, X is continuous.

However, if X does not satisfy either condition, that is some points are discrete and other continuous, we then call X a *hybrid* space. We provide an example of a hybrid space in its own section below.

2.5 Interior, Closure, Open sets and Closed sets

We now define the two operators we shall later use in the semantics of modal logic presented below.

Definition 2.3 (interior and closure, [2]): Let (X, \downarrow) be a convergence space.

- 1) $\text{int}_{\downarrow}(A) = \{x \in X \mid \forall \mathcal{F} \in \Phi(X) : \text{if } \mathcal{F} \downarrow x \text{ then } A \in \mathcal{F}\}$
- 2) $\text{cl}_{\downarrow}(A) = \{x \in X \mid \exists \mathcal{F} \in \Phi(X) : A \in \mathcal{F} \text{ and } \mathcal{F} \downarrow x\}$

We say a set $A \subseteq X$ is *open* in (X, \downarrow) if and only if $A = \text{int}_{\downarrow}(A)$. We say A is *closed* if and only if $A = \text{cl}_{\downarrow}(A)$.

The definition above is compatible with the notions of interior and closure in other spaces. In particular, Stadler and Stadler in [2] show how these operators agree with their topological counterparts whenever the convergence space is induced by a topological space.

The following propositions present some useful properties, and give us an insight on the soundness of the axioms of \mathcal{K} . We point out that the theorem below generalizes Theorem 1 in [2].

Theorem 2.4 (cf. theorem 1, [2]): Let (X, \downarrow) be a convergence space and let $A \subseteq X$. Then the following are true:

- 1) $X - \text{int}_{\downarrow}(A) = \text{cl}_{\downarrow}(X - A)$
- 2) $X - \text{cl}_{\downarrow}(A) = \text{int}_{\downarrow}(X - A)$
- 3) $\text{int}_{\downarrow}(X) = X$, $\text{cl}_{\downarrow}(\emptyset) = \emptyset$
- 4) If $A \subseteq B$ then $\text{cl}_{\downarrow}(A) \subseteq \text{cl}_{\downarrow}(B)$ and $\text{int}_{\downarrow}(A) \subseteq \text{int}_{\downarrow}(B)$.
- 5) Let $B \subseteq X$, then $\text{cl}_{\downarrow}(A) \cup \text{cl}_{\downarrow}(B) = \text{cl}_{\downarrow}(A \cup B)$
- 6) Let $B \subseteq X$, then $\text{int}_{\downarrow}(A) \cap \text{int}_{\downarrow}(B) = \text{int}_{\downarrow}(A \cap B)$

Corollary 2.5: $\text{cl}_{\downarrow}(A) = X - \text{int}_{\downarrow}(X - A)$

2.6 Modal Logic

In this section we introduce the fundamentals of (classical) *modal logic* in its syntactical form. The material in this section can be found in graduate level books on modal logic. Here, we follow the presentation [17].

2.6.1 Syntax

Definition 2.6 (The language): Let \mathbb{P} be an indefinitely continuable list of symbols $\mathbb{P}_0, \mathbb{P}_1, \dots, \mathbb{P}_n, \dots$, indexed by the natural numbers \mathbb{N} . The language \mathcal{L} is the set of sentences generated recursively by the following *BNF-style* grammar:

$$\varphi ::= \mathbb{P}_i \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Box\varphi$$

where $\mathbb{P}_i \in \mathbb{P}$.

We shall use the terms *sentences* or *formulas* for the elements in \mathcal{L} . Elements in \mathbb{P} are called *atomic* sentences. Sentences of the form $\Box A$ or $\Diamond A$ are called *modal* sentences and we shall say that a sentence is *propositionally atomic* if is atomic or modal.

As usual, we define the other common Boolean connectives in the following way, for $\varphi, \psi \in \mathcal{L}$: $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$, $\varphi \rightarrow \psi := \neg(\varphi \wedge \neg\psi)$, and $\varphi \leftrightarrow \psi := (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$. The logical constants *true* and *false* are defined as $\top := \varphi \vee \neg\varphi$ and $\perp := \varphi \wedge \neg\varphi$, respectively, for any $\varphi \in \mathcal{L}$. A formula of the form $\Box\varphi$ is called *necessitation* and φ is called its *necessitate*. The classical negation dual of a necessitation is

defined by $\diamond\varphi := \neg\Box\neg\varphi$ and it is called a *possibilitation* where φ is the *possibilitate*.

We shall provide meaning to these formulas by assigning subsets of a convergence space to the atomic sentences, and relating set operations for finding the meaning of more complex sentences. We shall then say, for instance, that the meaning of a negation, conjunction and disjunction of sentences corresponds to, respectively, the set-complement, intersection and union of the meanings of their constituent parts. In case of the modal operators, a necessitation is interpreted as the *interior* of the meaning of its necessitate and a possibilitation as the *closure* of the meaning of its possibilitate.

2.7 Normal Systems of Modal Logic

As in [17], we shall call a system of modal logic any set of sentences that contains all propositional tautologies on the language \mathcal{L} and it is closed under the inference rule *modus ponens* (MP). This makes classical propositional logic (PL) the smallest system of modal logic.

We now introduce one axiomatic form of the well-known system of modal logic known as \mathcal{K} .

Definition 2.7 (System \mathcal{K} [17]): Let \mathcal{K} be the smallest system of modal logic containing all instances of the following schema:

- Df \diamond .** $\diamond A \leftrightarrow \neg\Box\neg A$
M. $\Box(A \wedge B) \rightarrow (\Box A \wedge \Box B)$
C. $(\Box A \wedge \Box B) \rightarrow \Box(A \wedge B)$
N. $\Box\top$
K. $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$

And closed under the following inference rules:

- RN.** if A then $\Box A$
RE. if $A \leftrightarrow B$ then $\Box A \leftrightarrow \Box B$

There are different ways to present system \mathcal{K} (cf. [17], [18], [19]). For instance, by just having all instances of **Df \diamond** to \mathcal{K} and closing it under the inference rule **RK** (if $(A_1 \wedge \dots \wedge A_n) \rightarrow A$ then $(\Box A_1 \wedge \dots \wedge \Box A_n) \rightarrow \Box A$), we obtain all the instances of **M**, **C**, **N** and **K**; alternatively, these four axioms and the inference rule **RE** can be obtained from using **K** and **RN**. We do, however, choose the presentation above as it makes more clear the role of the structure of convergence spaces as models for \mathcal{K} .

3. Convergence Space Models

A convergence space model \mathcal{M} consists of a convergence space (W, \downarrow) , and a valuation function P assigning to each atomic proposition a subset of W . The meaning of more complex sentences can be determined using set-theoretic operations associated to each of the Boolean and modal operators.

3.1 Models

Definition 3.1: A *convergence space model*, or *convergence model*, is a structure $\mathcal{M} = \langle W, \downarrow, P \rangle$ where:

- 1) W is a set called the *set of possible worlds*

- 2) (W, \downarrow) is a convergence space; and
- 3) $P : \mathbb{P} \rightarrow 2^W$ is a valuation function mapping atomic propositions to sets of possible worlds. Since \mathbb{P} is indexed by \mathbb{N} , we shall write P_n for $P(\mathbb{P}_n)$, with $n \in \mathbb{N}$.

For simplicity, we shall say that α is a **world** in a convergence model \mathcal{M} whenever α is an element of W ; we shall write $\alpha \in \mathcal{M}$ to indicate this fact. We shall also use **Conv** for the class of all convergence models.

3.2 Truth and Validity

We now define what it means for a sentence A to be *valid* at a world α in a model \mathcal{M} , denoted by $\mathcal{M}, \alpha \models A$. We present the validity of sentence A based on its structural form.

Definition 3.2: Let α be a world in a convergence model $\mathcal{M} = \langle W, \downarrow, P \rangle$.

- 1) $\mathcal{M}, \alpha \models \mathbb{P}_n$ iff $\alpha \in P_n$ for $n \in \mathbb{N}$
- 2) $\mathcal{M}, \alpha \models \top$ always
- 3) $\mathcal{M}, \alpha \models \neg A$ iff $\mathcal{M}, \alpha \not\models A$
- 4) $\mathcal{M}, \alpha \models A \wedge B$ iff both $\mathcal{M}, \alpha \models A$ and $\mathcal{M}, \alpha \models B$
- 5) $\mathcal{M}, \alpha \models \Box A$ iff $(\exists O \subseteq W)(\forall \mathcal{F} \in \Phi(W)) : \text{if } \mathcal{F} \downarrow \alpha, \text{ then } O \in \mathcal{F} \text{ and } \forall \beta \in O : \mathcal{M}, \beta \models A$

We shall also say that $\mathcal{M} \models A$ iff $\mathcal{M}, \alpha \models A$, for every world $\alpha \in \mathcal{M}$; and that **Conv** $\models A$ iff $\mathcal{M} \models A$, for every model $\mathcal{M} \in \mathbf{Conv}$.

In this definition, items 1–4 form the standard definition of validity for non-modal formulas (i.e. without \Box or \diamond) in classical propositional calculus (cf. [17]). Item 5 defines the meaning of the modal formulas in our language.

In order to complement the definition above, we introduce the interpretation of validity using set-theoretic operations with the help of the well-known notion of the *truth set* of a sentence.

Definition 3.3 (cf. [17] definition 2.9): Let \mathcal{M} be model and A a sentence. The *truth set* of A , $\|A\|^{\mathcal{M}}$, is defined as:

$$\|A\|^{\mathcal{M}} = \{ \alpha \in \mathcal{M} \mid \mathcal{M}, \alpha \models A \}$$

Combining definitions 3.2 and 3.3, we obtain the characterization below which resembles the well-known topological semantics of **S4** of Tarski and McKinsey [20].

Theorem 3.4 (cf. theorem 2.10 [17]): Let $\mathcal{M} = \langle W, \downarrow, P \rangle$ be a convergence model. Then:

- 1) $\|\mathbb{P}_n\|^{\mathcal{M}} = P_n$, for $n \in \mathbb{N}$
- 2) $\|\top\|^{\mathcal{M}} = W$
- 3) $\|\neg A\|^{\mathcal{M}} = W - \|A\|^{\mathcal{M}}$
- 4) $\|A \wedge B\|^{\mathcal{M}} = \|A\|^{\mathcal{M}} \cap \|B\|^{\mathcal{M}}$
- 5) $\|\Box A\|^{\mathcal{M}} = \text{int}_{\downarrow}(\|A\|^{\mathcal{M}})$

As a corollary, it follows that, for example, $\|A \rightarrow B\|^{\mathcal{M}} = (W - \|A\|^{\mathcal{M}}) \cup \|B\|^{\mathcal{M}}$ and $\|\diamond A\|^{\mathcal{M}} = \text{cl}_{\downarrow}(\|A\|^{\mathcal{M}})$

We can now establish the well-known relation between truth sets and the validity of a sentence: a sentence A is valid at a world α if α belongs to the truth set of A , $\|A\|^{\mathcal{M}}$. Formally:

Theorem 3.5: Let \mathcal{M} be a convergence model and A a sentence. Then

$$\mathcal{M}, \alpha \models A \quad \text{iff} \quad \alpha \in \|A\|^{\mathcal{M}}$$

At this point, we have everything we need to reason about convergence spaces using normal systems of modal logic. We would now like present an example using this apparatus.

4. Example

We would like to be able to reason about the functioning of, say, an electronic device whose behavior evolves in so-called "real" time. Figure 1 illustrates the states considered in the functioning of the device. The device can be in one of the discrete states *Off*, *On*, or *Fail* if something caused the device to stop functioning; notice *Fail* is a *sink* state in order to model the fact that the device cannot auto-recover from failures.

Once the device is in operation, we use the non-negative reals to represent the evolution of the running time of the device: when the device starts, it moves to (time) state 0, and advances to higher positive real values following, say, a set of differential equations modeling the continuous behavior of the device. In this example, and in the interest of simplicity, we are not concerned on how this continuous evolution takes place; we are only concerned with the running time of the device. (Instead of the running time, we could consider, for instance, the space \mathbb{R}^n for modeling n continuous variables representing the "real" state of the device.)

At any time during the *running* of the device, it can be turned off (arrows from $x \in \mathbb{R}_0^+$ to the *Off* state), or may stop working because of a failure (arrows from $x \in \mathbb{R}_0^+$ to the *Fail* state). Notice that even at time state 0 the device may not work (a dead battery, perhaps).

We start by constructing a convergence space for Figure 1. We will augment the standard topology on \mathbb{R} , restricted to the non-negative reals, with the discrete states in the system, and use this to create the smallest filters for each $x \in \mathbb{R}_0^+$. For the discrete states, we will follow [5] in attaching point filters to vertices based on their graph neighborhoods.

Let (W, \downarrow) be a convergence space where $W = \mathbb{R}_0^+ \cup \{\text{On}, \text{Off}, \text{Fail}\}$ and the convergence structure \downarrow is as follows:

- For all $x \in \mathbb{R}^+$,
 $[\{(x', x'') \cup \{\text{Off}, \text{Fail}\} \mid x' < x < x''\}] \downarrow x$.
 For 0, let $[\{[0, x) \cup \{\text{Off}, \text{Fail}\} \mid x > 0\}] \downarrow 0$
- $[\text{Fail}] \downarrow \text{Fail}$
- $[\text{On}] \downarrow \text{Off}$
- $0 \in \mathbb{R}_0^+$, $[0] \downarrow \text{On}$.
- Apply (C1) to close \downarrow under super-filter inclusion.

Notice this space is *hybrid*: the states *On*, *Off* and *Fail* are discrete, but the states $x \in \mathbb{R}_0^+$ are all continuous.

We leave to the reader to verify that the point filters $[\text{Off}]$, $[\text{Fail}]$, and $[x]$ also converge to each $x \in \mathbb{R}_0^+$.

We now define the model $\mathcal{M} = \langle W, \downarrow, P \rangle$, where the valuation function P is as follows:

- $P_0 = \mathbb{R}_0^+$.
- $P_1 = \{\text{On}\}$, $P_2 = \{\text{Off}\}$, $P_3 = \{\text{Fail}\}$; and
- $P_n = \emptyset$, for $n > 3$.

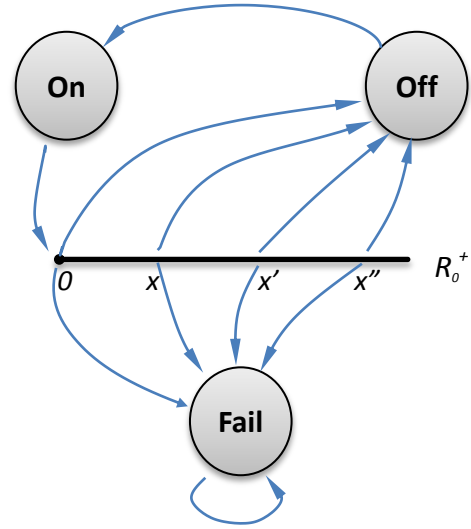


Fig. 1: Hybrid state space of a device. The device can be in one of the discrete states ("On", "Off", or "Fail") or *running* in "real" time represented by the non-negative reals.

That is, the atomic proposition \mathbb{P}_0 is identified with \mathbb{R}_0^+ , and \mathbb{P}_1 , \mathbb{P}_2 , and \mathbb{P}_3 with the discrete states $\{\text{On}\}$, $\{\text{Off}\}$ and $\{\text{Fail}\}$, respectively.

We can then show, for example, that the following sentences are valid in this model at all worlds $x \in \mathbb{R}_0^+$:

- $\mathbb{P}_0 \rightarrow \diamond \mathbb{P}_2$: "while running, we can turn the device off."
 We have that: $\|\diamond \mathbb{P}_2\|^{\mathcal{M}} = \text{cl}_{\downarrow}(\|\mathbb{P}_2\|^{\mathcal{M}}) = \mathbb{R}_0^+$, and therefore $\|\mathbb{P}_0 \rightarrow \diamond \mathbb{P}_2\|^{\mathcal{M}} = (W - \|\mathbb{P}_0\|^{\mathcal{M}}) \cup \|\diamond \mathbb{P}_2\|^{\mathcal{M}} = W$.
- $\mathbb{P}_0 \rightarrow \diamond \mathbb{P}_3$: "while running, the device may fail"
 We have that: $\|\diamond \mathbb{P}_3\|^{\mathcal{M}} = \text{cl}_{\downarrow}(\|\mathbb{P}_3\|^{\mathcal{M}}) = \mathbb{R}_0^+ \cup \{\text{Fail}\}$, and therefore $\|\mathbb{P}_0 \rightarrow \diamond \mathbb{P}_3\|^{\mathcal{M}} = W - \|\mathbb{P}_0\|^{\mathcal{M}} \cup \|\diamond \mathbb{P}_3\|^{\mathcal{M}} = (W - \mathbb{R}_0^+) \cup (\mathbb{R}_0^+ \cup \{\text{Fail}\}) = W$
- $\mathbb{P}_0 \rightarrow \diamond \square \mathbb{P}_1$: "while running, it is possible to turn the device off, and then back on again."
 We have that: $\|\diamond \square \mathbb{P}_1\|^{\mathcal{M}} = \text{cl}_{\downarrow}(\text{int}_{\downarrow}(\|\mathbb{P}_1\|^{\mathcal{M}})) = \text{cl}_{\downarrow}(\text{int}_{\downarrow}(\{\text{On}\})) = \text{cl}_{\downarrow}(\{\text{Off}\}) = \mathbb{R}_0^+ \cup \{\text{On}\}$. Therefore, $\|\mathbb{P}_0 \rightarrow \diamond \square \mathbb{P}_1\|^{\mathcal{M}} = W - \|\mathbb{P}_0\|^{\mathcal{M}} \cup \|\diamond \square \mathbb{P}_1\|^{\mathcal{M}} = (W - \mathbb{R}_0^+) \cup (\mathbb{R}_0^+ \cup \{\text{On}\}) = W$.
- $\mathbb{P}_3 \rightarrow \square \mathbb{P}_3$: "once the device fails, it will not auto-recover."
 We have that: $\|\square \mathbb{P}_3\|^{\mathcal{M}} = \text{int}_{\downarrow}(\|\mathbb{P}_3\|^{\mathcal{M}}) = \{\text{Fail}\}$. Therefore, $\|\mathbb{P}_3 \rightarrow \square \mathbb{P}_3\|^{\mathcal{M}} = W - \|\mathbb{P}_3\|^{\mathcal{M}} \cup \|\square \mathbb{P}_3\|^{\mathcal{M}} = W - \{\text{Fail}\} \cup \{\text{Fail}\} = W$.

5. Properties

System \mathcal{K} is sound and complete with respect to the class of convergence space models [1]. Soundness, as it is often the case, is easier to prove than completeness. Looking at the definition and properties of filters and the interior and closure operators we have that, for instance, $\mathbf{Df}\diamond$ is supported by corollary 2.5, axioms \mathbf{M} and \mathbf{C} follow from 2.4(5), \mathbf{N} follows

from 2.4(3) and **K** follows from the definition of the interior operator and filter axiom (F2).

Since convergence space models include relational and topological models as subcategories (see below), it inherits many desired properties found in those models, like the Finite Model property.

6. Other models for Normal Systems

In this section we briefly discuss the relationship between Convergence models and well-known models for normal systems. These are Kripke models, Topological models, and Neighborhood models. We will show that the first two are special cases of Convergence model semantics, and how our semantics relates to that of Neighborhood models.

6.1 Kripke models

A simple inspection of the literature on modal logic in Computer Science will reveal that Kripke semantics is by far the most popular semantics for normal systems of modal logic in the field. A Kripke model is a triple, consisting of a set W , a binary relation R , and a valuation P that assigns subsets of W to the atomic propositions in the logic. Whenever $\alpha R \beta$, for worlds $\alpha, \beta \in W$, we say that β is a world accessible from α [21]. Kripke models can also be found in the literature with the name *Standard Models* [17].

Definition 6.1 (Kripke models [17]): $\mathcal{M}_{\mathbf{K}} = \langle W, R, P \rangle$ is a Kripke model if and only if

- 1) W is a set called the *set of possible worlds*.
- 2) $R \subseteq W \times W$ is a binary relation known as the *accessibility relation*.
- 3) $P : \mathbb{P} \rightarrow 2^W$ is a valuation function mapping atomic propositions to sets of possible worlds.

In section 2.1, we discussed the relationship between relational and convergence spaces. As we mentioned there, it is well-known that a relation on a set, or a digraph, can induce a convergence space in (possibly infinite) many ways while preserving the structural properties of the graph. Kripke models are then a special case of convergence models.

Proposition 6.2: Every Kripke model $\mathcal{M}_{\mathbf{K}} = \langle W, R, P \rangle$ induces an convergence model $\mathcal{M} = \langle W, \downarrow_R, P \rangle$ where \downarrow_R is induced by R , and vice versa.

6.2 Topological models

Although not nearly as popular as Kripke frames, topological spaces have been shown to be useful in computer science when a *continuous* element is needed. For example, in reasoning about systems with continuous dynamics [22], and in artificial intelligence for the representation and reasoning of spatial information [23], [24], for example. The use of topological spaces for modal logics can be traced back to the seminal work of Tarski and McKinsey as semantics for intuitionistic logic and the logic **S4** [20], [25].

Definition 6.3 (Topological model): $\mathcal{M}_{\mathbf{T}} = \langle W, O, P \rangle$ is a topological model if and only if

- 1) W is a set called the *set of possible worlds*.

- 2) $O \subseteq 2^W$ a collections of subsets of possible worlds such that (i) $\{\emptyset, W\} \subseteq O$, (ii) O is closed under arbitrary unions, and (iii) O is closed under finite intersections. (W, O) is what is called a *topological space*.

- 3) $P : \mathbb{P} \rightarrow 2^W$ is a valuation function mapping atomic propositions to sets of possible worlds.

We mentioned in section 2.3 that topological spaces induce (topological) convergence spaces, and since the definition of the valuation function in topological models and convergence models are the same, inducing a convergence model from a topological model is rather simple.

Proposition 6.4: Every topological model $\mathcal{M}_{\mathbf{T}} = \langle W, O, P \rangle$ induces a (topological) convergence space $\mathcal{M} = \langle W, \downarrow_O, P \rangle$.

We then have that topological semantics is a special case of convergence space semantics.

6.3 Neighborhood models

As a generalization of Kripke models and Topological models, it is well-known that Richard Montague [26] and Dana Scott [27] independently defined the *neighborhood semantics* for modal logic in which an arbitrary collection of subsets of possible worlds, or *neighborhood system*, is attached to each world in a model.

The notion of validity that we obtain from neighborhood models is more general, but far weaker than the one we have with our models and Kripke models. In fact, axiom schema **M**, **C**, **N** and **K** are not valid in the class of neighborhood models (see, for example, [17]). The reason derives from the fact that these axioms require a structure on the elements of the neighborhood systems which does not necessarily exist. These collections could be internally organized like our filters, or just as a disparate bag of subsets. Neighborhood models can also be found in the literature with the name *minimal models* [17].

Definition 6.5 (Neighborhood model): A triple $\mathcal{M}_{\mathbf{N}} = \langle W, N, P \rangle$ is a neighborhood model if and only if

- 1) W is a set called the *set of possible worlds*.
- 2) $N : W \rightarrow 2^{2^W}$ is a mapping that relates each element in W with a collection of subsets of W .
- 3) $P : \mathbb{P} \rightarrow 2^W$ is a valuation function mapping atomic propositions to sets of possible worlds.

We can see from the preceding definition that, if the neighborhood system of a world α in a model $\mathcal{M}_{\mathbf{N}}$ is a filter (2.1), then it could be used as the smallest filter converging to α , thus inducing a convergence space on W . The following proposition formalizes this idea.

Proposition 6.6: Every neighborhood model $\mathcal{M}_{\mathbf{N}} = \langle W, N, P \rangle$ such that, for every $\alpha \in W$, if $N_{\alpha} \neq \emptyset$ then N_{α} is a filter, induces a *pretopological* convergence model $\mathcal{M} = \langle W, \downarrow_N, P \rangle$.

A *pretopological* convergence spaces is one in that, if $\mathcal{F} \downarrow x$, for some $x \in W$, then \mathcal{N}_x , the neighborhood filter of x , also converges to x . Pretopological spaces are a proper subset of convergence spaces [2], [5]. So, the relationship between

convergence models and neighborhood models is close, but not one-to-one. In a neighborhood model, N_α does not have to be a filter which makes these models more generic, but at the expense of giving us a far weaker logic. On the other hand, the flexibility in the construction of convergence structures (for example, there are many different structures representing the same digraph) allows us to fine-tune these models and work with a much richer logic.

7. Summary and Further Work

In this paper we showed the use of convergence spaces as the basis for a semantics for the smallest normal system of modal logic, system \mathcal{K} . Convergence models are generalizations of Kripke models, and Topological models, and, as such, it inherit their properties, like the Finite Model Property.

The use of convergence spaces opens up the possibility of generalizing results found with Kripke and Topological models, and of as a bridge between these two areas.

Specifically, convergence spaces could be used to model hybrid systems of the kind designed with digital and analog components, currently modeled with languages like VHDL-AMS. This could provide one more formal tool in the validation of these systems which could ultimately help in reducing their time-to-market.

We would like to investigate the relationship between our work and that of Blair et al.'s in [5]. There, a framework is given for the differentiation of continuous functions between convergence spaces. (For example, a continuous function with domain \mathbb{R} and an automaton as codomain would model a *continuous* automaton. This an important notion as right now such mapping is possible at the level of topological spaces if the automaton is reflexive and transitively closed, which they often are not). We would like to know how the validity of sentences is affected as a continuous function between two models is being differentiated.

References

- [1] A. Rivera, "Reasoning about co-evolving discrete, continuous and hybrid states," Ph.D. dissertation, Syracuse University, Syracuse, New York, 2008.
- [2] P. F. Stadler and B. M. R. Stadler, "Basic properties of filter convergence spaces," preprint, 2001. [Online]. Available: <http://bioinf.uni-leipzig.de/studia/Publications/PREPRINTS/01-pfs-007-sub11.pdf>
- [3] D. C. Kent, "Convergence functions and their related topologies," *Fundamenta Mathematicae*, vol. 54, pp. 125–133, 1964.
- [4] R. Heckmann, "A non-topological view of dcpo's as convergence spaces," *Theoretical Computer Science*, vol. 305, pp. 159–186, 2003.
- [5] H. A. Blair, D. W. Jakel, R. J. Irwin, and A. J. Rivera, "Elementary differential calculus on discrete and hybrid structures," in *Logical Foundations of Computer Science: International Symposium, LICS 2007, New York, NY, USA, June 4–7, 2007, Proceedings*, ser. Lecture Notes in Computer Science, S. N. Artemov and A. Nerode, Eds., no. 4514. Springer-Verlag, 2007, pp. 41–53.
- [6] J. M. E. Hyland, "Filter spaces and continuous functionals," *Annals of Mathematical Logic*, vol. 16, pp. 101–143, 1979.
- [7] D. C. Kent and N. Rath, "Filter spaces," *Applied Categorical Structures*, vol. 1, no. 3, pp. 297–309, 1993.
- [8] R. Heckmann, "On the relationship between filter spaces and equilogical spaces," 1998, draft report. [Online]. Available: <http://rw4.cs.uni-sb.de/heckmann/papers/fil.ps.gz>
- [9] B. M. R. Stadler, P. F. Stadler, G. P. Wagner, and W. Fontana, "The topology of the possible: formal spaces underlying patterns of evolutionary change," *Journal of Theoretical Biology*, vol. 213, no. 2, pp. 241–274, 2001.
- [10] C. M. Reidys and P. F. Stadler, "Combinatorial landscapes," Santa Fe Institute, Tech. Rep., 2003. [Online]. Available: <http://www.santafe.edu/sfi/publications/Working-Papers/01-03-014.pdf>
- [11] P. S. Alexandrov, "Discrete Räume," *Math. Sb. (New Series)*, vol. 2 (44), no. 3, pp. 501–519, 1937.
- [12] H. A. Blair, D. W. Jakel, R. J. Irwin, and A. J. Rivera, "Calculus on discrete and hybrid structures. part i: Foundations and calculi," in preparation.
- [13] S. Vickers, *Topology via Logic*, ser. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1989, no. 6.
- [14] C. Choquet, "Convergences," *Ann. Univ. Grenoble*, vol. 23, pp. 55–112, 1947.
- [15] N. Bourbaki, *Topologie Générale*. Actualités Sci. Ind., **858** (1940), **916** (1942), **1029** (1947), **1045** (1948), **1084** (1949).
- [16] J. L. Kelley, *General Topology*. Van Nostrand Reinhold, 1955.
- [17] B. F. Chellas, *Modal logic. An introduction*. Cambridge University Press, 1980.
- [18] E. J. Lemmon, *An Introduction to Modal Logic (The "Lemmon Notes")*, ser. American Philosophical Quarterly, K. Segerberg, Ed. Basil Blackwell, 1977, no. 11.
- [19] G. E. Hughes and M. J. Cresswell, *A New Introduction to Modal Logic*. Routledge, 1968, reprinted in 1968, 2001, 2003.
- [20] J. C. C. McKinsey and A. Tarski, "Some theorems about the sentential calculi of lewis and heyting," *The Journal of Symbolic Logic*, no. 13, pp. 1–15, 1948.
- [21] S. A. Kripke, "A semantical analysis of modal logic I: Normal modal propositional calculi," *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, vol. 9, pp. 67–96, 1963.
- [22] V. Couillard and J. Davoren, "Spatio-temporal logics for continuous dynamical systems," January 2004, conference paper submission. [Online]. Available: <http://www.ee.unimelb.edu.au/staff/davoren/00-lics04-couillard.pdf>
- [23] M. Aiello, J. van Benthem, and G. Bezhanishvili, "Reasoning about space: the modal way," *Journal of Logic and Computation*, vol. 13, no. 6, pp. 889–920, 2003, first published as a manuscript: Manuscript UvA, pp 01 – 18, 2001. [Online]. Available: <http://www.cs.rug.nl/aiellom/publications/topax.pdf>
- [24] M. Aiello, "The topo-approach to spatial representation and reasoning," in *AI*IA Notizie XVI*, 2003, vol. 4, pp. 5–14. [Online]. Available: <http://www.cs.rug.nl/aiellom/publications/aiiaPhDaward.pdf>
- [25] J. C. C. McKinsey and A. Tarski, "The algebra of topology," *Annals of Mathematics*, vol. 45, pp. 141–191, 1944.
- [26] R. Montague, "Universal grammar," *Theoria*, no. 36, pp. 373–398, 1970.
- [27] D. Scott, "Advice in modal logic," in *Philosophical Problems in Logic*, K. Lambert, Ed. Reidel, Dordrecht, 1970, pp. 143–173.

An Automated Derivation of Two Alternate Axiomatic Bases for Łukasiewicz's Sentential Calculus

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA

Abstract

The optimization of computing systems incorporating Boolean-circuit-based computing equipment must be expressed at some level in Boolean behaviors and operations. Boolean behaviors and operations are part of a larger family of logics -- the logic of sentences, also known as the "sentential calculus". Two logics are implicationally equivalent if the axioms and inference rules of each imply the axioms of the other. Characterizing the inferential equivalences of various formulations of the sentential calculi is thus foundational to the optimization of Boolean-oriented computing systems. Using an automated deduction system, I show that one of the most austere formulations of the sentential calculus, Łukasiewicz's CN, has at least two alternate axiomatic bases; the bases appears to be novel. The proofs further demonstrate a natural proving order that both informs and constrains optimization strategies.

Keywords: propositional logic, automated deduction, sentential calculus

1.0 Introduction

The optimization of computing systems incorporating Boolean-circuit-based computing equipment must be expressed at some level in Boolean behaviors and operations. Boolean behaviors and operations are part of a larger family of logics -- the logic of sentences, also known as the "sentential calculus". Two logics are implicationally equivalent if the axioms and

inference rules of each imply the axioms of the other. Characterizing the inferential equivalences of various formulations of the sentential calculi is thus foundational to the optimization of Boolean-oriented computing systems ([1],[3]-[7],[9]-[10],[12]-[15]).

"CN", the formulation of the sentential calculus in [1], is among the most austere: its vocabulary contains only two logical connectives (C, and N) and sentence variables (p, q, r, \dots). It has two inference rules (condensed detachment and substitution), and three axioms.

In CN, any expression of the form Cxy or Nz , where $x, y,$ and z are sentences, is a sentence. Cpq is interpreted as "sentence p implies sentence q "; Np is interpreted as "not- p ". C and N are right-associative; N has higher associative precedence than C. For example,

$$CCqrCpNr$$

translates to the more common "arrow-and-parenthesis" notation as

$$(q \rightarrow r) \rightarrow (p \rightarrow \sim r)$$

where " \rightarrow " designates "implies" and " \sim " designates "not".

The axioms of CN in [1] are:

CN1. $CCpqCCqrCpr$

CN2. $CCNppp$

CN3. $CpCNpq$

The main result of this paper is that either

CN13. $CCNpqCtCCqpp$

or

CN14. $CCCtCCqpprCCNpqr$

can be substituted for CN2 to obtain an alternate basis for CN.

2.0 Method

To prove that substituting CN13 for CN2 yields an axiomatic basis for CN, it suffices to show that CN13 can be derived from {CN1, CN2, CN3}, and that CN2 can be derived from {CN1, CN3, CN13}.

To show that CN13 is derivable from CN1-CN3, [1] was implemented in the *prover9* ([2]) script shown in Figure 1 and executed under on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 @ 2.33 GHz and 8.00 GB RAM, running under the *Windows Vista Home Premium (SP2)/Cygwin* operating environment.

```

set(pos_hyper_resolution).

formulas(usable).
P(i(i(x,y),i(i(y,z),i(x,z)))) # label("CN1").
P(i(i(-x,x),x)) # label("CN2").
P(i(x,i(-x,y))) # label("CN3").
end_of_list.

formulas(sos).
-P(i(x,y) | -P(x) | P(y)) # label("InfConDet").
end_of_list.

formulas(goals).
P(i(i(-x,y),i(v,i(i(y,x),x)))) # label("CN13").
end_of_list.

```

Figure 1. The *prover9* script used to show the conjunction of CN1-CN3 implies CN13. The implementation of condensed detachment is the formula in the "sos" list; substitution is derived from *prover9*'s hyperresolution rule (introduced in the "set" command at the top of the script). Details of *prover9*'s syntax and semantics can be found in [2].

To show that CN2 is derivable from the conjunction of CN1, CN3, and CN13, [1], with CN13 substituted for CN2, was implemented in the *prover9* ([2]) script shown in Figure 2 and executed on the platform described above.

```

set(pos_hyper_resolution).

formulas(usable).
P(i(i(x,y),i(i(y,z),i(x,z)))) # label("CN1").
P(i(i(-x,y),i(v,i(i(y,x),x)))) # label("CN13").

```

```

P(i(x,i(-x,y))) # label("CN3").
end_of_list.

formulas(sos).
-P(i(x,y)) | -P(x) | P(y) # label("InfConDet").
end_of_list.

formulas(goals).
P(i(i(-x,x),x)) # label("CN2").
end_of_list.

```

Figure 2. The *prover9* script used to show the conjunction of CN1, CN3, and CN13 imply CN2. Details of the syntax and semantics of the notation can be found in [2]. The implementation of condensed detachment is the formula in the "sos" list; substitution is derived from *prover9*'s hyperresolution rule (introduced in the "set" command at the top of the script).

In this paper, showing that CN14 can be substituted for CN2 to form an alternate basis for CN requires some intermediate results from the proof that CN13 can be substituted for CN2 to form such a basis. The details of that argument are accordingly deferred to Section 3.0.

3.0 Results

Figure 3 shows that CN13 can be derived from the conjunction of CN1-CN3.

```

===== PROOF =====

% Proof 1 at 0.09 (+ 0.05) seconds: "CN13".

1 P(i(i(-x,y),i(v,i(i(y,x),x)))) # label("CN13") # label(non_clause) #
label(goal). [goal].
2 P(i(i(x,y),i(i(y,z),i(x,z)))) # label("CN1"). [assumption].
3 P(i(i(-x,x),x)) # label("CN2"). [assumption].
4 P(i(x,i(-x,y))) # label("CN3"). [assumption].
5 -P(i(x,y)) | -P(x) | P(y) # label("InfConDet"). [assumption].
6 -P(i(i(-c1,c2),i(c3,i(i(c2,c1),c1)))) # label("CN13") #
answer("CN13"). [deny(1)].
10 P(i(i(i(-x,y),z),i(x,z))). [hyper(5,a,2,a,b,4,a)].
11 P(i(i(x,y),i(i(-x,x),y))). [hyper(5,a,2,a,b,3,a)].
12 P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))). [hyper(5,a,2,a,b,2,a)].
20 P(i(x,x)). [hyper(5,a,10,a,b,3,a)].
22 P(i(-i(x,x),y)). [hyper(5,a,4,a,b,20,a)].
24 P(i(i(x,y),i(-i(z,z),y))). [hyper(5,a,2,a,b,22,a)].
87 P(i(x,i(-i(y,y),z))). [hyper(5,a,10,a,b,24,a)].
93 P(i(i(i(-i(x,x),y),z),i(u,z))). [hyper(5,a,2,a,b,87,a)].
330 P(i(x,i(y,y))). [hyper(5,a,93,a,b,3,a)].
337 P(i(i(i(x,x),y),i(z,y))). [hyper(5,a,2,a,b,330,a)].
357 P(i(x,i(i(-y,y),y))). [hyper(5,a,337,a,b,11,a)].
391 P(i(i(i(i(-x,x),x),y),i(z,y))). [hyper(5,a,2,a,b,357,a)].

```

```

856 P(i(i(x,i(-y,y)),i(z,i(x,y)))) . [hyper(5,a,12,a,b,391,a)].
1206 P(i(i(-x,y),i(z,i(i(y,x),x)))) . [hyper(5,a,12,a,b,856,a)].
1207 $F # answer("CN13") . [resolve(1206,a,6,a)].

```

=====
===== end of proof =====

Figure 3. Summary of a *prover9* ([2]) proof showing that Proposition CN13 is derivable from CN1-CN3. The proof assumes condensed detachment (Line 5) and substitution (implied by the "set" command) as inference rules. The general form of a line in this proof is "*line_number conclusion [derivation]*", where *line_number* is a unique identifier of a line in the proof, and *conclusion* is the result of applying the *prover9* inference rules noted (denoting the *derivation*), to the lines cited in those brackets. All lines annotated as "assumption" are axioms or definitions. All *prover9* proofs are proofs by contradiction; note in particular that Line 6 is the denial of Line 1. Further detail of the syntax and semantics of *prover9* notation used in this study can be found in [2] and [4]. Note the derivation of the Law of Identity at Line 20.

Figure 4 shows that CN2 can be derived from the conjunction of CN1, CN3, and CN13 of [1]. This is sufficient to show that these three propositions form an alternative axiomatic basis for CN.

=====
===== PROOF =====

```

1 P(i(i(-x,x),x)) # label("CN2") # label(non_clause) # label(goal) .
[goal].
2 P(i(i(x,y),i(i(y,z),i(x,z)))) # label("CN1") . [assumption].
3 P(i(i(-x,y),i(z,i(i(y,x),x)))) # label("CN13") . [assumption].
4 P(i(x,i(-x,y))) # label("CN3") . [assumption].
5 -P(i(x,y)) | -P(x) | P(y) # label("InfConDet") . [assumption].
6 -P(i(i(-c1,c1),c1)) # label("CN2") # answer("CN2") . [deny(1)].
10 P(i(x,i(i(i(-y,z),y),y))) . [hyper(5,a,3,a,b,4,a)].
11 P(i(i(i(-x,y),z),i(x,z))) . [hyper(5,a,2,a,b,4,a)].
12 P(i(i(i(x,i(i(y,z),z)),u),i(i(-z,y),u))) . [hyper(5,a,2,a,b,3,a)].
61 P(i(i(i(-x,y),x),x)) . [hyper(5,a,10,a,b,10,a)].
115 P(i(i(-x,y),i(i(y,x),x))) . [hyper(5,a,12,a,b,61,a)].
128 P(i(x,i(i(y,x),x))) . [hyper(5,a,11,a,b,115,a)].
151 P(i(i(i(i(x,y),y),z),i(y,z))) . [hyper(5,a,2,a,b,128,a)].
256 P(i(x,x)) . [hyper(5,a,151,a,b,61,a)].
274 P(i(i(-x,x),x)) . [hyper(5,a,115,a,b,256,a)].
275 $F # answer("CN2") . [resolve(274,a,6,a)].

```

=====
===== end of proof =====

Figure 4. Summary of a *prover9* ([2]) proof showing that Proposition CN2 (Axiom 2 in the default formulation of CN) is derivable from CN1, CN3, and CN13. Note the derivation of the Law of Identity at Line 256.

Figures 3 and 4 demonstrate that CN1, CN3, and CN13 collectively form an alternate basis for CN.

Figure 5 shows that CN14 is derivable from CN1 and CN13. Because CN13 is derivable from {CN1, CN2, CN3}, CN14 is therefore derivable from {CN1, CN2, CN3}.

```

===== PROOF =====
1 P(i(i(i(u,i(i(y,x),x)),z),i(i(-x,y),z))) # label("CN14") #
label(non_clause) # label(goal). [goal].
2 P(i(i(-x,y),i(z,i(i(y,x),x)))) # label("CN13"). [assumption].
3 P(i(i(x,y),i(i(y,z),i(x,z)))) # label("AxCN1"). [assumption].
6 -P(i(x,y)) | -P(x) | P(y) # label("InfConDet"). [assumption].
7 -P(i(i(i(c1,i(i(c2,c3),c3)),c4),i(i(-c3,c2),c4))) # label("CN14") #
answer("CN14"). [deny(1)].
15 P(i(i(i(x,i(i(y,z),z)),u),i(i(-z,y),u))). [hyper(6,a,3,a,b,2,a)].
16 $F # answer("CN14"). [resolve(15,a,7,a)].

===== end of proof =====

```

Figure 5. Summary of prover9 proof showing that CN14 is derivable from CN1 and CN13.

```

===== PROOF =====
1 P(i(i(-x,y),i(v,i(i(y,x),x)))) # label("CN13") # label(non_clause) #
label(goal). [goal].
2 P(i(i(i(x,i(i(y,z),z)),u),i(i(-z,y),u))) # label("CN14").
[assumption].
3 P(i(i(x,y),i(i(y,z),i(x,z)))) # label("AxCN1"). [assumption].
4 P(i(x,i(-x,y))) # label("AxCN3"). [assumption].
5 -P(i(x,y)) | -P(x) | P(y) # label("InfConDet"). [assumption].
6 -P(i(i(-c1,c2),i(c3,i(i(c2,c1),c1)))) # label("CN13") #
answer("CN13"). [deny(1)].
10 P(i(i(i(-x,y),z),i(x,z))). [hyper(5,a,3,a,b,4,a)].
20 P(i(i(-x,y),i(z,i(i(y,x),x)))) # label("CN13"). [hyper(5,a,2,a,b,10,a)].
21 $F # answer("CN13"). [resolve(20,a,6,a)].

===== end of proof =====

```

Figure 6. Summary of prover9 proof that CN14, CN1, and CN13 jointly imply CN13.

Figure 6 shows that CN13 is derivable from CN14, CN1, and CN13 collectively. Because Figure 4 shows that CN13, CN1, and CN3 jointly imply CN2, {CN14, CN1,

CN3} implies CN2. This is sufficient to show that {CN1, CN3, CN14} is an alternate basis for CN.

4.0 Conclusions and discussion

Section 3 demonstrates that CN1, CN3, and CN13 (or CN14) collectively form an alternate axiomatic basis for CN; these bases appear to be novel.

In [1], CN13 (or CN14) is treated as a lemma to help derive CN15 (which, [1] notes, can be substituted for CN2 to form a basis). In [1], the derivation of CN13 (or CN14) assumes only on CN1 and CN3; the *prover9* script for deriving CN13, however, "spontaneously" deployed CN2, in addition to CN1 and CN3, to derive CN13. This result hints (but of course does not prove) that CN13 might be substituted for CN2 to form an alternate basis. Similarly, [1] uses CN13 to derive CN14, hinting (but not proving) that CN14 could be substituted for CN2 to form an alternate basis for CN.

These derivation relationships both inform and constrain optimization strategies on Boolean-based computing systems.

5.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, and Alberto Coffa. For any problems that remain, I am wholly responsible.

6.0 References

- [1] Łukasiewicz J. *Elements of Mathematical Logic*. Second Edition (1958). Trans. by Wojtasiewicz O. Pergamon Press. 1963.
- [2] McCune WW. *prover9 and mace4*. <http://www.cs.unm.edu/~mccune/prover9/>. 2009.
- [3] Aristotle. *Prior Analytics*. Trans. by A. J. Jenkinson. In Aristotle. *The Basic Works of Aristotle*. Ed. by R. McKeon. Random House. 1941. pp. 62-107.
- [4] Aristotle. *Posterior Analytics*. Trans. by G. R. G. Mure. In Aristotle. *The Basic Works of Aristotle*. Ed. by R. McKeon. Random House. 1941. pp. 108-186.
- [5] Tarski A. *Introduction to Logic*. Trans. by O. Helmer. Dover. 1941.
- [6] Hempel C. Studies in the logic of explanation. In Hempel C. *Aspects of Scientific Explanation and Other Essays in the Philosophy of Science*. Free Press. 1965. pp. 245-290.
- [7] Quine WVO. *Philosophy of Logic*. Second Edition. Harvard. 1986.
- [8] Chang CC and Keisler HJ. *Model Theory*. North-Holland. 1990.
- [9] Russell B and Whitehead AN. *Principia Mathematica*. Volume I (1910). Merchant Books. 2009.
- [10] Frege G. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle. 1879. Translated in van Heijenoort J. *Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought. From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*. Harvard. 1967. pp. 3-82.
- [11] Horn A. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic* 16 (1951), 14–21.
- [12] Church A. *Introduction to Mathematical Logic*. Volume I. Princeton. 1956.
- [13] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-843.
- [14] Kant I. *Kant's Introduction to Logic* (1800). Trans. by Abbott TK. Greenwood Press. 1963.
- [15] Cohen MR and Nagel E. *An Introduction to Logic and Scientific Method*. Harcourt, Brace, and Company. 1934.

An Automated Derivation of Łukasiewicz's *CN* from the Sentential Calculus of *Principia* *Mathematica*

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA

Abstract

The optimization of computing systems that incorporate Boolean-circuit-based computing equipment must be expressed at some level in Boolean behaviors and operations. Boolean behaviors and operations are part of a larger family of logics -- the logic of sentences, also known as the "sentential calculus". Two logics are implicationally equivalent if the axioms and inference rules of each imply the axioms of the other. Characterizing the inferential equivalences of various formulations of the sentential calculi is thus foundational to the optimization of Boolean-based computing systems. Two logics are implicationally equivalent if the axioms and inference rules of each imply the axioms of the other. Characterizing the inferential equivalences of various formulations of the sentential calculi is foundational to the study of logic. Using an automated deduction system, I show that Łukasiewicz's CN can be derived from the sentential calculus of the Principia Mathematica; the proof appears to be novel.

Keywords: propositional logic, automated deduction, sentential calculus

1.0 Introduction

The optimization of computing systems that incorporate Boolean-circuit-based computing equipment must be expressed at some level in Boolean behaviors and operations. Boolean behaviors and operations are part of a larger family of

logics -- the logic of sentences, also known as the "sentential calculus". Two logics are implicationally equivalent if the axioms and inference rules of each imply the axioms of the other. Characterizing the inferential equivalences of various formulations of the sentential calculi is thus foundational to the optimization of Boolean computing systems. Two logics are implicationally equivalent if the axioms and inference rules of each imply the axioms of the other. Characterizing the inferential equivalences of various formulations of the sentential calculi is foundational to the study of logic. Characterizing equivalences of various formulations of the sentential calculi is foundational to the optimization of Boolean-oriented computing systems ([1],[3]-[7],[9]-[10],[12]-[15]).

"CN", the formulation of the sentential calculus in [1], is among the most austere: its vocabulary contains only two logical connectives (C, and N) and sentence variables (p, q, r, \dots). It has two inference rules (condensed detachment and substitution), and three axioms.

In CN, any expression of the form Cxy or Nz , where $x, y,$ and z are sentences, is a sentence. Cpq is interpreted as "sentence p implies sentence q "; Np is interpreted as "not- p ". C and N are right-associative; N has higher associative precedence than C. For example,

$$CCqrCpNr$$

translates to the more common "arrow-and-parenthesis" notation as

$$(q \rightarrow r) \rightarrow (p \rightarrow \sim r)$$

where " \rightarrow " designates "implies" and " \sim " designates "not".

The axioms of CN in [1] are:

- CN1. $CCpqCCqrCpr$
- CN2. $CCNppp$
- CN3. $CpCNpq$

Cast in CN notation, the axioms of the sentential calculus of *Principia Mathematica* (PM, [9]) are

- CN73. $CqApq$
- CN74. $CAppp$
- CN75. $CApqAqp$

CN76. $CCqrCApqApr$

CN78. $CApAqrAqApr$

where $Apq \equiv CNpq$. The main result of this paper is that the axioms of [9] implies the axioms of [1].

2.0 Method

To show that the axioms of [9] imply the axioms of [1], the *prover9* ([2]) script shown in Figures 1 was executed under on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 @ 2.33 GHz and 8.00 GB RAM, running under the *Windows Vista Home Premium (SP2)/Cygwin* operating environment.

```

set(hyper_resolution).

formulas(usable).
% put axioms and previously proven theorems here.
P ( i(i(x, i(y,z)), i(y, i(x,z))) )      # label("PM 2.04").
P ( i(i(y,z), i(i(x,y), i(x,z))) )      # label("PM 2.05").
P ( i(-x, i(x,y)) )                      # label("PM 2.21").
P ( i(y, i(-x,y)) )                      # label("CN73").
P ( i(i(-x,x), x) )                      # label("CN74").
P ( i(i(-x,y), i(-y,x)) )                # label("CN75").
P ( i(i(y,z), i(i(-x,y), i(-x,z))) )    # label("CN76").
P ( i(i(-x, i(-y,z)), i(-y, i(-x,z))) ) # label("CN78").
end_of_list.

formulas(sos).
% put inference rules here.
-P(i(x,y) | -P(x) | P(y))                # label("InfConDet").
end_of_list.

formulas(goals).
% put item(s) to be proven here.
P( i(i(x,y), i(i(y,z), i(x,z))) )        # label("AxCN1").
P( i(i(-x,x), x) )                        # label("AxCN2").
P( i(x, i(-x,y)) )                        # label("AxCN3").
end_of_list.

```

Figure 1. The *prover9* script (in Horn clause ([11]) form), used to show that the axioms of PM imply the axioms of CN. The implementation of condensed detachment is the formula in the "sos" list;

substitution is derived from *prover9's* hyperresolution rule (introduced in the "set" command at the top of the script). PM Theorems 2.04, 2.05 and 2.21 were added to the axioms of PM to facilitate the derivation. Details of *prover9's* syntax and semantics can be found in [2].

3.0 Results

Figure 2 shows that PM implies CN.

```

===== PROOF =====

% Proof 1 at 0.01 (+ 0.05) seconds: "AxCN2".
% Length of proof is 4.
% Level of proof is 2.
% Maximum clause weight is 7.
% Given clauses 0.

2 P(i(i(-x,x),x)) # label("AxCN2") # label(non_clause) # label(goal).
[goal].
8 P(i(i(-x,x),x)) # label("CN74"). [assumption].
14 -P(i(i(-c4,c4),c4)) # label("AxCN2") # answer("AxCN2"). [deny(2)].
15 $F # answer("AxCN2"). [resolve(14,a,8,a)].

===== end of proof =====

===== PROOF =====

% Proof 2 at 0.01 (+ 0.05) seconds: "AxCN3".
% Length of proof is 7.
% Level of proof is 2.
% Maximum clause weight is 8.
% Given clauses 1.

3 P(i(x,i(-x,y))) # label("AxCN3") # label(non_clause) # label(goal).
[goal].
4 P(i(i(x,i(y,z)),i(y,i(x,z)))) # label("PM 2.04"). [assumption].
6 P(i(-x,i(x,y))) # label("PM 2.21"). [assumption].
12 -P(i(x,y)) | -P(x) | P(y) # label("InfConDet"). [assumption].
16 -P(i(c5,i(-c5,c6))) # label("AxCN3") # answer("AxCN3"). [deny(3)].
50 P(i(x,i(-x,y))). [hyper(12,a,4,a,b,6,a)].
51 $F # answer("AxCN3"). [resolve(50,a,16,a)].

===== end of proof =====

===== PROOF =====

% Proof 3 at 0.01 (+ 0.05) seconds: "AxCN1".
% Length of proof is 7.
% Level of proof is 2.
% Maximum clause weight is 12.
% Given clauses 1.

```

```

1 P(i(i(x,y),i(i(y,z),i(x,z)))) # label("AxCN1") # label(non_clause) #
label(goal). [goal].
4 P(i(i(x,i(y,z)),i(y,i(x,z)))) # label("PM 2.04"). [assumption].
5 P(i(i(x,y),i(i(z,x),i(z,y)))) # label("PM 2.05"). [assumption].
12 -P(i(x,y)) | -P(x) | P(y) # label("InfConDet"). [assumption].
13 -P(i(i(c1,c2),i(i(c2,c3),i(c1,c3)))) # label("AxCN1") #
answer("AxCN1"). [deny(1)].
52 P(i(i(x,y),i(i(y,z),i(x,z)))) . [hyper(12,a,4,a,b,5,a)].
53 $F # answer("AxCN1"). [resolve(52,a,13,a)].

===== end of proof =====

```

Figure 2. Summary of a prover9 ([2]) proof showing that PM ([9]) implies CN ([1]).

The total time to complete the proofs shown in Figure 2 was ~0.2 seconds on the platform described in Section 2.0.

4.0 Conclusions and discussion

Section 3 demonstrates that PM implies CN. The proof in Figure 2 appears to be novel.

A companion paper ([16]) proves CN implies PM. These relationships both inform and constrain optimization strategies on Boolean-circuit-based computing systems.

5.0 References

[1] Łukasiewicz J. *Elements of Mathematical Logic*. Second Edition (1958). Trans. by Wojtasiewicz O. Pergamon Press. 1963.

[2] McCune WW. *prover9 and mace4*. <http://www.cs.unm.edu/~mccune/prover9/>. 2009.

[3] Aristotle. *Prior Analytics*. Trans. by A. J. Jenkinson. In Aristotle. *The Basic Works of Aristotle*. Ed. by R. McKeon. Random House. 1941. pp. 62-107.

[4] Aristotle. *Posterior Analytics*. Trans. by G. R. G. Mure. In Aristotle. *The Basic*

Works of Aristotle. Ed. by R. McKeon. Random House. 1941. pp. 108-186.

[5] Tarski A. *Introduction to Logic*. Trans. by O. Helmer. Dover. 1941.

[6] Hempel C. Studies in the logic of explanation. In Hempel C. *Aspects of Scientific Explanation and Other Essays in the Philosophy of Science*. Free Press. 1965. pp. 245-290.

[7] Quine WVO. *Philosophy of Logic*. Second Edition. Harvard. 1986.

[8] Chang CC and Keisler HJ. *Model Theory*. North-Holland. 1990.

[9] Russell B and Whitehead AN. *Principia Mathematica*. Volume I (1910). Merchant Books. 2009.

[10] Frege G. Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. Halle. 1879. Translated in van Heijenoort J. Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*. Harvard. 1967. pp. 3-82.

- [11] Horn A. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic* 16 (1951), 14–21.
- [12] Church A. *Introduction to Mathematical Logic*. Volume I. Princeton. 1956.
- [13] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-843.
- [14] Kant I. *Kant's Introduction to Logic* (1800). Trans. by Abbott TK. Greenwood Press. 1963.
- [15] Cohen MR and Nagel E. *An Introduction to Logic and Scientific Method*. Harcourt, Brace, and Company. 1934.
- [16] Horner JK. An automated derivation of the sentential calculus of *Principia Mathematica* from Łukasiewicz's *CN*. *Proceedings of the 2011 International Conference on Artificial Intelligence*. CSREA Press. Forthcoming.

An Automated Derivation of the Sentential Calculus of *Principia Mathematica* from Łukasiewicz's *CN*

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA

Abstract

The optimization of computing systems hosted on Boolean-circuit-based computing equipment must be expressed at some level in Boolean behaviors and operations. Boolean behaviors and operations are part of a larger family of logics -- the logic of sentences, also known as the "sentential calculus". Two logics are implicationally equivalent if the axioms and inference rules of each imply the axioms of the other. Characterizing the inferential equivalences of various formulations of the sentential calculi is thus foundational to the optimization of Boolean computing systems. Using an automated deduction system, I show that the sentential calculus of the Principia Mathematica (PM) can be derived from Łukasiewicz's CN; the proof appears to be novel. The proofs variously demonstrate, furthermore, a natural proving order.

Keywords: propositional logic, automated deduction, sentential calculus

1.0 Introduction

The optimization of computing systems hosted on Boolean-circuit-based computing equipment must be expressed at some level in Boolean behaviors and operations. Boolean behaviors and operations are part of a larger family of logics -- the logic of sentences, also known as the "sentential calculus". Two logics are implicationally equivalent if the axioms and inference rules

of each imply the axioms of the other. Characterizing the inferential equivalences of various formulations of the sentential calculi is thus foundational to the optimization of Boolean-oriented computing systems ([1],[3]-[7],[9]-[10],[12]-[15]).

"CN", the formulation of the sentential calculus in [1], is among the most austere: its vocabulary contains only two logical connectives (C, and N) and sentence variables (p, q, r, \dots). It has two inference rules (condensed detachment and substitution), and three axioms.

In CN, any expression of the form Cxy or Nz , where x, y , and z are sentences, is a sentence. Cpq is interpreted as "sentence p implies sentence q "; Np is interpreted as "not- p ". C and N are right-associative; N has higher associative precedence than C. For example,

$$CCqrCpNr$$

translates to the more common "arrow-and-parenthesis" notation as

$$(q \rightarrow r) \rightarrow (p \rightarrow \sim r)$$

where " \rightarrow " designates "implies" and " \sim " designates "not".

The axioms of CN in [1] are:

CN1. $CCpqCCqrCpr$

CN2. $CCNppp$

CN3. $CpCNpq$

Cast in CN notation, the axioms of the sentential calculus of *Principia Mathematica* (PM, [9]) are

CN73. *CqApq*
CN74. *CAppp*
CN75. *CApqAqp*
CN76. *CCqrCApqApr*
CN78. *CApAqrAqApr*

The main result of this paper is that [1] implies [9].

2.0 Method

To show that [1] implies [10], the *prover9* ([2]) script shown in Figures 1 was executed under on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 @ 2.33 GHz and 8.00 GB RAM, running under the *Windows Vista Home Premium (SP2)/Cygwin* operating environment.

```
set(hyper_resolution).

formulas(usable).
% put axioms and previously proven theorems here.
P ( i(i(u, i(y, i(x,z))), i(i(x,u), i(y, i(x,z)))) ) # label("CN34").
P ( i(i(x,y), i(i(y,z), i(x,z))) ) # label("AxCN1").
P ( i(i(-x,x), x) ) # label("AxCN2").
P ( i(x, i(-x,y)) ) # label("AxCN3").
end_of_list.

formulas(sos).
% put inference rules here.
-P(i(x,y) | -P(x) | P(y)) # label("InfConDet").
end_of_list.

formulas(goals).
% put item(s) to be proven here.
P ( i(y, i(-x,y)) ) # label("CN73").
P ( i(i(-x,x), x) ) # label("CN74").
P ( i(i(-x,y), i(-y,x)) ) # label("CN75").
P ( i(i(y,z), i(i(-x,y), i(-x,z))) ) # label("CN76").
P ( i(i(-x, i(-y,z)), i(-y, i(-x,z))) ) # label("CN78").
end_of_list.
```

Figure 1. The *prover9* script used to show that CN implies PM. The implementation of condensed detachment is the formula in the "sos" list; substitution is derived from *prover9*'s hyperresolution rule (introduced in the "set" command at the top of the script). CN34, a theorem of [1], has been added to the axioms of [1] to facilitate the derivation. Details of *prover9*'s syntax and semantics can be found in [2].

3.0 Results

Figure 2 shows that CN implies PM.

```
===== PROOF =====

% Proof 1 at 0.03 (+ 0.01) seconds: "CN74".

2 P(i(i(-x,x),x)) # label("CN74") # label(non_clause) # label(goal). [goal].
8 P(i(i(-x,x),x)) # label("AxCN2"). [assumption].
12 -P(i(i(-c3,c3),c3)) # label("CN74") # answer("CN74"). [deny(2)].
13 $F # answer("CN74"). [resolve(12,a,8,a)].

===== end of proof =====
```

```

===== PROOF =====
% Proof 2 at 0.80 (+ 0.01) seconds: "CN75".

3 P(i(i(-x,y),i(-y,x))) # label("CN75") # label(non_clause) # label(goal). [goal].
6 P(i(i(x,i(y,i(z,u))),i(i(z,x),i(y,i(z,u)))) # label("CN34"). [assumption].
7 P(i(i(x,y),i(i(y,z),i(x,z)))) # label("AxCN1"). [assumption].
8 P(i(i(-x,x),x)) # label("AxCN2"). [assumption].
9 P(i(x,i(-x,y))) # label("AxCN3"). [assumption].
10 -P(i(x,y)) | -P(x) | P(y) # label("InfConDet"). [assumption].
14 -P(i(i(-c4,c5),i(-c5,c4))) # label("CN75") # answer("CN75"). [deny(3)].
21 P(i(i(i(-x,y),z),i(x,z))). [hyper(10,a,7,a,b,9,a)].
22 P(i(i(x,y),i(i(-x,x),y))). [hyper(10,a,7,a,b,8,a)].
23 P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))). [hyper(10,a,7,a,b,7,a)].
25 P(i(i(x,y),i(-y,i(x,z)))). [hyper(10,a,6,a,b,9,a)].
39 P(i(x,x)). [hyper(10,a,21,a,b,8,a)].
42 P(i(-i(x,x),y)). [hyper(10,a,9,a,b,39,a)].
45 P(i(i(x,y),i(-i(z,z),y))). [hyper(10,a,7,a,b,42,a)].
95 P(i(i(i(-x,i(y,z)),u),i(i(y,x),u))). [hyper(10,a,7,a,b,25,a)].
234 P(i(x,i(-i(y,y),z))). [hyper(10,a,21,a,b,45,a)].
241 P(i(i(i(-i(x,x),y),z),i(u,z))). [hyper(10,a,7,a,b,234,a)].
3643 P(i(x,i(y,y))). [hyper(10,a,241,a,b,8,a)].
3662 P(i(i(i(x,x),y),i(z,y))). [hyper(10,a,7,a,b,3643,a)].
3720 P(i(i(x,y),i(z,i(x,y)))). [hyper(10,a,23,a,b,3662,a)].
3738 P(i(x,i(i(-y,y),y))). [hyper(10,a,3662,a,b,22,a)].
3882 P(i(i(x,i(x,y)),i(z,i(x,y)))). [hyper(10,a,6,a,b,3720,a)].
5584 P(i(x,i(i(y,i(y,z)),i(y,z)))). [hyper(10,a,3882,a,b,3882,a)].
6335 P(i(i(x,i(x,y)),i(x,y))). [hyper(10,a,5584,a,b,5584,a)].
6383 P(i(i(i(x,y),x),i(i(x,y),y))). [hyper(10,a,23,a,b,6335,a)].
6752 P(i(i(i(-x,x),x),y,y)). [hyper(10,a,6383,a,b,3738,a)].
6796 P(i(i(x,i(-y,y)),i(x,y))). [hyper(10,a,23,a,b,6752,a)].
7345 P(i(i(-x,y),i(-y,x))). [hyper(10,a,95,a,b,6796,a)].
7346 $F # answer("CN75"). [resolve(7345,a,14,a)].

```

```

===== end of proof =====

```

```

===== PROOF =====
% Proof 3 at 1.04 (+ 0.01) seconds: "CN73".

1 P(i(y,i(-x,y))) # label("CN73") # label(non_clause) # label(goal). [goal].
6 P(i(i(x,i(y,i(z,u))),i(i(z,x),i(y,i(z,u)))) # label("CN34"). [assumption].
7 P(i(i(x,y),i(i(y,z),i(x,z)))) # label("AxCN1"). [assumption].
8 P(i(i(-x,x),x)) # label("AxCN2"). [assumption].
9 P(i(x,i(-x,y))) # label("AxCN3"). [assumption].
10 -P(i(x,y)) | -P(x) | P(y) # label("InfConDet"). [assumption].
11 -P(i(c1,i(-c2,c1))) # label("CN73") # answer("CN73"). [deny(1)].
21 P(i(i(i(-x,y),z),i(x,z))). [hyper(10,a,7,a,b,9,a)].
22 P(i(i(x,y),i(i(-x,x),y))). [hyper(10,a,7,a,b,8,a)].
23 P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))). [hyper(10,a,7,a,b,7,a)].
25 P(i(i(x,y),i(-y,i(x,z)))). [hyper(10,a,6,a,b,9,a)].
39 P(i(x,x)). [hyper(10,a,21,a,b,8,a)].
42 P(i(-i(x,x),y)). [hyper(10,a,9,a,b,39,a)].
45 P(i(i(x,y),i(-i(z,z),y))). [hyper(10,a,7,a,b,42,a)].
95 P(i(i(i(-x,i(y,z)),u),i(i(y,x),u))). [hyper(10,a,7,a,b,25,a)].
234 P(i(x,i(-i(y,y),z))). [hyper(10,a,21,a,b,45,a)].
241 P(i(i(i(-i(x,x),y),z),i(u,z))). [hyper(10,a,7,a,b,234,a)].
3643 P(i(x,i(y,y))). [hyper(10,a,241,a,b,8,a)].
3662 P(i(i(i(x,x),y),i(z,y))). [hyper(10,a,7,a,b,3643,a)].
3720 P(i(i(x,y),i(z,i(x,y)))). [hyper(10,a,23,a,b,3662,a)].
3738 P(i(x,i(i(-y,y),y))). [hyper(10,a,3662,a,b,22,a)].
3882 P(i(i(x,i(x,y)),i(z,i(x,y)))). [hyper(10,a,6,a,b,3720,a)].
5584 P(i(x,i(i(y,i(y,z)),i(y,z)))). [hyper(10,a,3882,a,b,3882,a)].
6335 P(i(i(x,i(x,y)),i(x,y))). [hyper(10,a,5584,a,b,5584,a)].
6383 P(i(i(i(x,y),x),i(i(x,y),y))). [hyper(10,a,23,a,b,6335,a)].

```



```

6752 P(i(i(i(-x,x),x),y),y)). [hyper(10,a,6383,a,b,3738,a)].
6796 P(i(i(x,i(-y,y)),i(x,y))). [hyper(10,a,23,a,b,6752,a)].
7345 P(i(i(-x,y),i(-y,x))). [hyper(10,a,95,a,b,6796,a)].
9633 P(i(x,i(-y,x))). [hyper(10,a,21,a,b,7345,a)].
9634 $F # answer("CN73"). [resolve(9633,a,11,a)].

===== end of proof =====

===== PROOF =====

% Proof 4 at 2.65 (+ 0.03) seconds: "CN78".

5 P(i(i(-x,i(-y,z)),i(-y,i(-x,z)))) # label("CN78") # label(non_clause) # label(goal).
[goal].
6 P(i(i(x,i(y,i(z,u))),i(i(z,x),i(y,i(z,u)))) # label("CN34"). [assumption].
7 P(i(i(x,y),i(i(y,z),i(x,z)))) # label("AxCN1"). [assumption].
8 P(i(i(-x,x),x)) # label("AxCN2"). [assumption].
9 P(i(x,i(-x,y))) # label("AxCN3"). [assumption].
10 -P(i(x,y) | -P(x) | P(y) # label("InfConDet"). [assumption].
16 -P(i(i(-c9,i(-c10,c11)),i(-c10,i(-c9,c11)))) # label("CN78") # answer("CN78").
[deny(5)].
21 P(i(i(i(-x,y),z),i(x,z))). [hyper(10,a,7,a,b,9,a)].
22 P(i(i(x,y),i(i(-x,x),y))). [hyper(10,a,7,a,b,8,a)].
23 P(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))). [hyper(10,a,7,a,b,7,a)].
25 P(i(i(x,y),i(-y,i(x,z)))). [hyper(10,a,6,a,b,9,a)].
39 P(i(x,x)). [hyper(10,a,21,a,b,8,a)].
42 P(i(-i(x,x),y)). [hyper(10,a,9,a,b,39,a)].
45 P(i(i(x,y),i(-i(z,z),y))). [hyper(10,a,7,a,b,42,a)].
95 P(i(i(i(-x,i(y,z)),u),i(i(y,x),u))). [hyper(10,a,7,a,b,25,a)].
100 P(i(-x,i(x,y))). [hyper(10,a,25,a,b,39,a)].
116 P(i(i(x,y),z),i(-x,z))). [hyper(10,a,7,a,b,100,a)].
145 P(i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z)))). [hyper(10,a,23,a,b,23,a)].
234 P(i(x,i(-i(y,y),z))). [hyper(10,a,21,a,b,45,a)].
241 P(i(i(i(-i(x,x),y),z),i(u,z))). [hyper(10,a,7,a,b,234,a)].
342 P(i(-x,x)). [hyper(10,a,116,a,b,8,a)].
349 P(i(i(x,y),i(--x,y))). [hyper(10,a,7,a,b,342,a)].
406 P(i(x,i(--x,y))). [hyper(10,a,21,a,b,349,a)].
3643 P(i(x,i(y,y))). [hyper(10,a,241,a,b,8,a)].
3662 P(i(i(x,x),y),i(z,y))). [hyper(10,a,7,a,b,3643,a)].
3720 P(i(i(x,y),i(z,i(x,y)))). [hyper(10,a,23,a,b,3662,a)].
3738 P(i(x,i(i(-y,y),y))). [hyper(10,a,3662,a,b,22,a)].
3882 P(i(i(x,i(x,y)),i(z,i(x,y)))). [hyper(10,a,6,a,b,3720,a)].
5584 P(i(x,i(i(y,z),i(y,z)))). [hyper(10,a,3882,a,b,3882,a)].
6335 P(i(i(x,i(x,y)),i(x,y))). [hyper(10,a,5584,a,b,5584,a)].
6383 P(i(i(i(x,y),x),i(i(x,y),y))). [hyper(10,a,23,a,b,6335,a)].
6752 P(i(i(i(-x,x),x),y),y)). [hyper(10,a,6383,a,b,3738,a)].
6796 P(i(i(x,i(-y,y)),i(x,y))). [hyper(10,a,23,a,b,6752,a)].
6911 P(i(i(-x,y),i(i(y,x),x))). [hyper(10,a,23,a,b,6796,a)].
6944 P(i(x,--x)). [hyper(10,a,6796,a,b,406,a)].
6959 P(i(x,i(y,--y))). [hyper(10,a,3720,a,b,6944,a)].
7345 P(i(i(-x,y),i(-y,x))). [hyper(10,a,95,a,b,6796,a)].
8122 P(i(i(i(x,--x),y),y)). [hyper(10,a,6383,a,b,6959,a)].
9633 P(i(x,i(-y,x))). [hyper(10,a,21,a,b,7345,a)].
9707 P(i(i(x,-y),i(z,i(x,z)))). [hyper(10,a,145,a,b,9633,a)].
18826 P(i(x,i(y,x))). [hyper(10,a,8122,a,b,9707,a)].
18906 P(i(i(i(x,y),z),i(y,z))). [hyper(10,a,7,a,b,18826,a)].
20404 P(i(x,i(i(x,y),y))). [hyper(10,a,18906,a,b,6911,a)].
20871 P(i(i(x,i(y,z)),i(y,i(x,z)))). [hyper(10,a,145,a,b,20404,a)].
20872 $F # answer("CN78"). [resolve(20871,a,16,a)].

===== end of proof =====

===== PROOF =====

% Proof 5 at 7.67 (+ 0.31) seconds: "CN76".

4 P(i(i(y,z),i(i(-x,y),i(-x,z)))) # label("CN76") # label(non_clause) # label(goal).
[goal].
6 P(i(i(x,i(y,i(z,u))),i(i(z,x),i(y,i(z,u)))) # label("CN34"). [assumption].

```

```

7 P(i(i(x,y),i(i(y,z),i(x,z)))) # label("AxCN1"). [assumption].
8 P(i(i(-x,x),x)) # label("AxCN2"). [assumption].
9 P(i(x,i(-x,y))) # label("AxCN3"). [assumption].
10 -P(i(x,y) | -P(x) | P(y)) # label("InfConDet"). [assumption].
15 -P(i(i(c6,c7),i(i(-c8,c6),i(-c8,c7)))) # label("CN76") # answer("CN76"). [deny(4)].
21 P(i(i(i(-x,y),z),i(x,z))). [hyper(10,a,7,a,b,9,a)].
22 P(i(i(x,y),i(i(-x,x),y))). [hyper(10,a,7,a,b,8,a)].
23 P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))). [hyper(10,a,7,a,b,7,a)].
25 P(i(i(x,y),i(-y,i(x,z)))). [hyper(10,a,6,a,b,9,a)].
39 P(i(x,x)). [hyper(10,a,21,a,b,8,a)].
42 P(i(-i(x,x),y)). [hyper(10,a,9,a,b,39,a)].
45 P(i(i(x,y),i(-i(z,z),y))). [hyper(10,a,7,a,b,42,a)].
95 P(i(i(i(-x,i(y,z)),u),i(i(y,x),u))). [hyper(10,a,7,a,b,25,a)].
100 P(i(-x,i(x,y))). [hyper(10,a,25,a,b,39,a)].
116 P(i(i(i(x,y),z),i(-x,z))). [hyper(10,a,7,a,b,100,a)].
145 P(i(i(x,i(y,z),i(i(u,y),i(x,i(u,z))))). [hyper(10,a,23,a,b,23,a)].
234 P(i(x,i(-i(y,y),z))). [hyper(10,a,21,a,b,45,a)].
241 P(i(i(i(-i(x,x),y),z),i(u,z))). [hyper(10,a,7,a,b,234,a)].
342 P(i(--x,x)). [hyper(10,a,116,a,b,8,a)].
349 P(i(i(x,y),i(--x,y))). [hyper(10,a,7,a,b,342,a)].
406 P(i(x,i(--x,y))). [hyper(10,a,21,a,b,349,a)].
3643 P(i(x,i(y,y))). [hyper(10,a,241,a,b,8,a)].
3662 P(i(i(i(x,x),y),i(z,y))). [hyper(10,a,7,a,b,3643,a)].
3720 P(i(i(x,i(x,y),i(z,i(x,y)))). [hyper(10,a,23,a,b,3662,a)].
3738 P(i(x,i(i(-y,y),y))). [hyper(10,a,3662,a,b,22,a)].
3882 P(i(i(x,i(x,y)),i(z,i(x,y)))). [hyper(10,a,6,a,b,3720,a)].
5584 P(i(x,i(i(y,i(y,z)),i(y,z))). [hyper(10,a,3882,a,b,3882,a)].
6335 P(i(i(x,i(x,y),i(x,y))). [hyper(10,a,5584,a,b,5584,a)].
6383 P(i(i(i(x,y),x),i(i(x,y),y))). [hyper(10,a,23,a,b,6335,a)].
6752 P(i(i(i(i(-x,x),x),y),y)). [hyper(10,a,6383,a,b,3738,a)].
6796 P(i(i(x,i(-y,y),i(x,y))). [hyper(10,a,23,a,b,6752,a)].
6911 P(i(i(-x,y),i(i(y,x),x))). [hyper(10,a,23,a,b,6796,a)].
6944 P(i(x,--x)). [hyper(10,a,6796,a,b,406,a)].
6959 P(i(x,i(y,--y))). [hyper(10,a,3720,a,b,6944,a)].
7345 P(i(i(-x,y),i(-y,x))). [hyper(10,a,95,a,b,6796,a)].
8122 P(i(i(i(x,--x),y),y)). [hyper(10,a,6383,a,b,6959,a)].
9633 P(i(x,i(-y,x))). [hyper(10,a,21,a,b,7345,a)].
9707 P(i(i(x,-y),i(z,i(x,z)))). [hyper(10,a,145,a,b,9633,a)].
18826 P(i(x,i(y,x))). [hyper(10,a,8122,a,b,9707,a)].
18906 P(i(i(i(x,y),z),i(y,z))). [hyper(10,a,7,a,b,18826,a)].
20404 P(i(x,i(i(x,y),y))). [hyper(10,a,18906,a,b,6911,a)].
20871 P(i(i(x,i(y,z)),i(y,i(x,z)))). [hyper(10,a,145,a,b,20404,a)].
44253 P(i(i(x,y),i(i(z,x),i(z,y)))). [hyper(10,a,20871,a,b,7,a)].
44254 $F # answer("CN76"). [resolve(44253,a,15,a)].

===== end of proof =====

```

Figure 2. Summary of a *prover9* ([2]) proof showing that CN ([1]) implies PM ([9]).

The total time to complete the proofs shown in Figure 2 was ~12 seconds on the platform described in Section 2.0.

4.0 Conclusions and discussion

Section 3 demonstrates that CN implies PM. A companion paper ([16]) proves PM implies CN. The proof in Figure 2 appears to be novel.

The proof of CN74 is trivial because CN74 is identical to CN2.

There are some interesting relationships among four of the five axioms of PM. In particular, Lines 21-7345 of the proof of CN75 are a subproof of the proof of CN73; this subproof can therefore be regarded as a proof of a set of lemmas for the proof of CN73. Similarly, Lines 21-20871 of the proof of CN78 are a subproof of the proof of CN73; this subproof can therefore be regarded as the proof of a set of lemmas for

the proof of CN78. Lines 21-95 of CN75 are a subproof of CN75, CN73, CN76, and CN78; this subproof can therefore be regarded as the proof of a set of lemmas for the proof of all PM axioms other than CN74. These relationships both inform and constrain optimization strategies on Boolean-circuit-bases computing systems.

5.0 References

- [1] Łukasiewicz J. *Elements of Mathematical Logic*. Second Edition (1958). Trans. by Wojtasiewicz O. Pergamon Press. 1963.
- [2] McCune WW. *prover9 and mace4*. <http://www.cs.unm.edu/~mccune/prover9/>. 2009.
- [3] Aristotle. *Prior Analytics*. Trans. by A. J. Jenkinson. In Aristotle. *The Basic Works of Aristotle*. Ed. by R. McKeon. Random House. 1941. pp. 62-107.
- [4] Aristotle. *Posterior Analytics*. Trans. by G. R. G. Mure. In Aristotle. *The Basic Works of Aristotle*. Ed. by R. McKeon. Random House. 1941. pp. 108-186.
- [5] Tarski A. *Introduction to Logic*. Trans. by O. Helmer. Dover. 1941.
- [6] Hempel C. Studies in the logic of explanation. In Hempel C. *Aspects of Scientific Explanation and Other Essays in the Philosophy of Science*. Free Press. 1965. pp. 245-290.
- [7] Quine WVO. *Philosophy of Logic*. Second Edition. Harvard. 1986.
- [8] Chang CC and Keisler HJ. *Model Theory*. North-Holland. 1990.
- [9] Russell B and Whitehead AN. *Principia Mathematica*. Volume I (1910). Merchant Books. 2009.
- [10] Frege G. Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. Halle. 1879. Translated in van Heijenoort J. Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*. Harvard. 1967. pp. 3-82.
- [11] Horn A. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic* 16 (1951), 14–21.
- [12] Church A. *Introduction to Mathematical Logic*. Volume I. Princeton. 1956.
- [13] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-843.
- [14] Kant I. *Kant's Introduction to Logic* (1800). Trans. by Abbott TK. Greenwood Press. 1963.
- [15] Cohen MR and Nagel E. *An Introduction to Logic and Scientific Method*. Harcourt, Brace, and Company. 1934.
- [16] Horner JK. An automated derivation of the sentential calculus of *Principia Mathematica* from Łukasiewicz's CN. *Proceedings of the 2011 International Conference on Artificial Intelligence*. CSREA Press. Forthcoming.

SESSION
GRAPH BASED METHODS AND RELATED
ISSUES

Chair(s)

TBA

Graph representation of hierarchical Alvis model structure

L. Kotulski¹, and M. Szyrka¹

¹Department of Automatics, AGH University of Science and Technology, Kraków, Poland

Abstract—*Alvis Toolkit supports the development of embedded systems. A result of the toolkit use is not only an Alvis model, but also a formal model representation in the form of a Labelled Transition System is generated in parallel. This allows a designer to make a formal verification of the developed embedded system behaviour. The modularisation concept is expressed by the hierarchical agents structure. In the paper, we propose a graph representation of this hierarchical structure, that supports its transformation into a flat one (equivalent to the generated LTS model) with linear time of computational complexity.*

Keywords: Alvis, embedded system, graph representation

1. Introduction

A software for an embedded system should be verified due to a high cost of the service of equipment, where such a software is installed. Defining a formal model that after the verification will be used for an automatic code generation, is counter-intuitive for engineers without a solid mathematical experience. Alvis [1], [2], a new language for supporting embedded software development, allows us to design a system but in parallel with a model an abstract representation, using the Labelled Transition System notation, is created. The basic abstraction of this language is an agent, that communicates with its environment via ports. *Communication diagrams* are the visual part of the Alvis modelling language. They are used to represent the structure of the system under consideration. They are a way to point out agents that communicate one with the other. Moreover, the diagrams allow programmers to combine sets of agents into modules, described by an abstraction called page, that are also represented as agents (called *hierarchical agents*). Finally, the system is represented by the *hierarchical communication diagram* [3] being a set of pages and the substitution function (that shows, which page structure will substitute each *hierarchical agent*). Each agent in an Alvis model is either a hierarchical agent with a substitution page assigned or a simple agent, whose behaviour is described with the Alvis Code Language. Hierarchical structure supports the modularisation concept. However, it is difficult to understand the behaviour of the whole system and impossible to generate the formal model directly from it. For this purpose, the concept of a *flat representation* (in which each element of a system is described by exactly one agent) has been introduced in [3]. The formal definition uses the *hierarchical dependency* and *substitution* relations, what

raises a fear on the effectiveness of the proposed solutions. In the paper the consistent *graph representation* of the above problems is introduced and it is proved the polynomial computational complexity of:

- the generation of the *flat representation*,
- the execution of the synthesis (analysis) operation that allows us to move between *flat representations* to achieve a more abstract (detailed) description of the considered embedded system.

2. Hierarchical agents structure

The key concept of Alvis is an *agent* that denotes any distinguished part of the system under consideration with defined identity persisting in time. An agent can communicate with other agents through *ports*. Each agent port must have a unique identifier (name) assigned, but ports of different agents may have the same identifier assigned. Thus, each port in a model is identified using its name and its agent name. For simplicity, we will use the so-called *dot notation* – $X.p$ denotes the port p of the agent X .

Let us define the following symbols:

- $\mathcal{P}(X)$ denotes the set of ports of agent X .
- $\mathcal{P}(D)$ denotes the set of ports of page D .
- $\mathcal{N}(W)$ denotes the set of names of ports belonging to set W and $card(W)$ returns the number of W elements.

For example, if a diagram contains only agents X_1 with port p and X_2 also with port p , then $\mathcal{P}(D) = \{X_1.p, X_2.p\}$, and $\mathcal{N}(\mathcal{P}(D)) = \{p\}$.

Alvis provides hierarchical communication diagrams used to describe an embedded system from the control and data flow point of view. A hierarchical diagram enables designers to distribute parts of a diagram across multiple subdiagrams called *pages*.

Definition 1: A *Page* is a triple $D = (\mathcal{A}^i, \mathcal{C}^i, \sigma^i)$, where:

- 1) $\mathcal{A}^i = \{X_1^i, \dots, X_n^i\}$ is a set of *agents* with subsets of *active agents* \mathcal{A}_A^i , *passive agents* \mathcal{A}_P^i , and *hierarchical agents* \mathcal{A}_H^i , such that $\mathcal{A}^i = \mathcal{A}_A^i \cup \mathcal{A}_P^i \cup \mathcal{A}_H^i$, and \mathcal{A}_A^i , \mathcal{A}_P^i , \mathcal{A}_H^i are pairwise disjoint.
- 2) $\mathcal{C}^i \subseteq \mathcal{P}^i \times \mathcal{P}^i$ is the *communication relation*, such that $\forall j = 1, \dots, n (\mathcal{P}^i(X_j) \times \mathcal{P}^i(X_j)) \cap \mathcal{C}^i = \emptyset$, where $\mathcal{P}^i = \bigcup_{j=1, \dots, n} \mathcal{P}^i(X_j)$. Each element of the relation \mathcal{C}^i is called a *connection* or a *communication channel*.
- 3) $\sigma^i: \mathcal{A}_A^i \rightarrow \{False, True\}$ is the *start function* that points out initially activated agents.

We forbid designers to connect ports of one agent (see point 2). The start function σ , from point 3, makes delaying

activation of some agents possible – we can make them active later with the *start* statement.

Pages are combined using the so-called *substitution mechanism*. A *hierarchical agent* at one level can be replaced by a page on the lower level, which usually gives a more precise and detailed description of the subsystem represented by the agent. Let a hierarchical agent Y be given and let $join_Y(D^i)$ denote the set of all *join ports* of the page D^i with respect to Y , i.e. $join_Y(D^i) = \{X_j^i.p \in \mathcal{P}(D^i) : \mathcal{N}(X_j^i.p) \in \mathcal{N}(\mathcal{P}(Y))\}$. In other words, $join_Y(D^i)$ is the set of all ports from the page D^i whose names are the same as those of the hierarchical agent Y .

Definition 2: Let a hierarchical agent Y and a page $D^i = (\mathcal{A}^i, \mathcal{C}^i, \sigma^i)$ be given. The agent Y and the page D^i satisfy the *substitution requirements* iff $card(\mathcal{P}(Y)) \leq card(join_Y(D^i))$, and $(join_Y(D^i) \times join_Y(D^i)) \cap \mathcal{C}^i = \emptyset$

We will consider a *binding function* π that maps ports of a hierarchical agent to the join ports of the corresponding page.

An example of a communication diagram is shown in Fig. 1. Active agents are drawn as rounded boxes while passive ones as rectangles. Ports are drawn as circles placed at the edges of the corresponding rounded box or rectangle. Communication channels are drawn as lines (or broken lines). An arrowhead points out the input port for the particular connection. Communication channels without arrowheads represent pairs of connections with opposite directions. Black triangles indicate hierarchical agents.

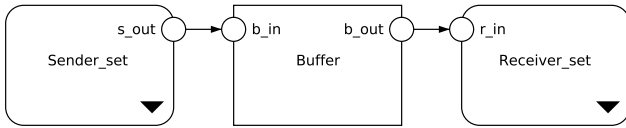


Fig. 1

SENDER-RECEIVER SYSTEM WITH BUFFER – COMMUNICATION DIAGRAM.

Definition 3: A *hierarchical communication diagram* is a pair $H = (\mathcal{D}, \gamma)$, where:

- $\mathcal{D} = \{D^1, \dots, D^k\}$ is a set of *pages* of the hierarchical communication diagram, such that sets of agents \mathcal{A}^i ($i = 1, \dots, k$) are pairwise disjoint.
- $\gamma: \mathcal{A}_H \rightarrow \mathcal{D}$, where $\mathcal{A}_H = \bigcup_{i=1, \dots, k} \mathcal{A}_H^i$, is the *substitution function*, such that:

- 1) γ is an injection.
- 2) For any $X_j^i \in \mathcal{A}_H$, X_j^i and $\gamma(X_j^i)$ satisfy the requirements of the substitution.
- 3) Labelled directed graph $\mathcal{G} = (\mathcal{D}, E, \mathcal{A}_H)$, (where \mathcal{D} is a set of nodes, $E = \{(D^i, X_k^i, D^j) : \gamma(X_k^i) = D^j\}$ is a set of edges and \mathcal{A}_H is a set of labels), called *page hierarchy graph* is a tree or a forest.

We assume that a system definition starts from a page or a set of pages, thus the number of pages must be greater than the number of hierarchical agents. Formally pages from the set $\mathcal{D} - \gamma(\mathcal{A}_H)$ are called *primary pages*, they are roots of trees that constitute a page hierarchy graph.

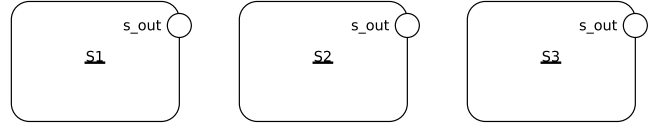


Fig. 2

SENDER-RECEIVER SYSTEM – PAGE *Sender_page*.

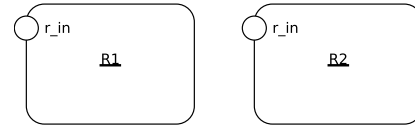


Fig. 3

SENDER-RECEIVER SYSTEM – PAGE *Receiver_page*.

An example of the substitution pages is shown in Fig. 2 and 3. The page hierarchy graph for the readers-writers model is shown in Fig. 4.

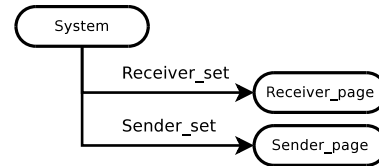


Fig. 4

PAGE HIERARCHY GRAPH

Let us focus on the *Sender_set* agent. Thus, we have:

- $\mathcal{P}(Sender_set) = \{Sender_set.r_out\}$
- $join_{Sender_set}(Sender_page) = \{S1.r_out, S2.s_out, S3.s_out\}$
- $\mathcal{N}(\mathcal{P}(Readers)) = \{r_out\} = join_{Sender_set}(Sender_page)$

In this case, the binding function π is defined as follows: $\pi(Sender_set.s_out) = \{S1.s_out, \dots, S3.s_out\}$

Following symbols are valid for hierarchical communication diagrams:

- $\mathcal{A}_A = \bigcup_{i=1, \dots, k} \mathcal{A}_A^i$,
- $\mathcal{A}_P = \bigcup_{i=1, \dots, k} \mathcal{A}_P^i$,
- $\mathcal{A} = \mathcal{A}_A \cup \mathcal{A}_P$,
- $\sigma: \mathcal{A}_A \rightarrow \{False, True\}$ and $\forall i = 1, \dots, k \forall X_j^i \in \mathcal{A}_A^i: \sigma(X_j^i) = \sigma^i(X_j^i)$.

3. Graph representation

In this section we introduce the *graph representation* of a system defined by a hierarchical communication diagram, that will support effective reasoning about the system properties.

Definition 4: A labelled directed graph is a tuple $\mathcal{G} = (V, E, \Sigma, \Gamma)$, where:

- V is the a of nodes.
- Σ is a set of node labels.
- Γ is a set of edge labels.
- $E \subseteq V \times \Gamma \times V$ is a set of direct edges.

Edges in a graph are directed, and for a given edge $\alpha = (X, a, Y)$ the following functions return its elements: $pred(\alpha) = X$, $succ(\alpha) = Y$ and $lab(\alpha) = a$. An undirected edge is considered as a pair of directed edges. For any node X , $lab(X)$ return its label and $name(X)$ returns the name associated with this node.

We use labelled directed graphs to describe the structure of an Alvis model. We assume that each agent is represented as a node labelled by A (agent) and a set of nodes labelled by P (port); the node representing the agent is connected with port(s) by edge(s) labelled by b (belongs to). Two ports (but only belonging to different agents) can be connected by an edge labelled by c (communication channel).

For a page $D^i = \gamma(Y)$ all nodes, representing agents belonging to \mathcal{A}^i , are connected with the node representing Y by an edge labelled by γ ; analogously, appropriate nodes representing ports are connected by edges labelled by π .

Definition 5: A hierarchical graph $G(H)$ representing a given hierarchical communication diagram $H = (\mathcal{D}, \gamma)$ is a labelled directed graph $\mathcal{G} = (V, E, \{A, P\}, \{b, c, \gamma, \pi\})$ such that:

- There exists an isomorphic mapping $\delta: V \rightarrow \mathcal{A} \cup \mathcal{P}$.
- $\forall X \in V$ such that $lab(X) = P$, $\exists! Y \in V: lab(Y) = A, (X, b, Y) \in E$ and $\delta(X) \in \mathcal{P}(\delta(Y))$.
- $X, Y \in V$ and $(\delta(X), \delta(Y) \in C) \Rightarrow (X, c, Y) \in E$.
- $X, Y \in V$, $\gamma(\delta(Y)) = D^i$ and $\delta(X) \in \mathcal{A}^i \Rightarrow (X, \gamma, Y) \in E$ (in such a case X is said to be directly dependent on Y and denoted by $X \succ Y$).
- $X, Y \in V$ and $\pi(\delta(Y)) = \delta(X) \Rightarrow (X, \pi, Y) \in E$.

Let us note that in the graph representation, the direction in edges is opposite to the introduced in γ and π functions, due to a possibility of representing injective functions by edges.

Figure 5 represents the graph of the Sender-Receiver example, assuming that the *Sender_set* is substituted by *Sender_page* (consisting of three S agents) and *Receiver_set* is substituted by *Receiver_page* (consisting of two R agents).

Such a graph can be modified using *transformation rules* represented graphically as *productions*, that consist of two graphs L – left side graph and R – right side graph. To apply a production $P: L \Rightarrow R$ in the context of a graph \mathcal{G} , we should perform the following steps:

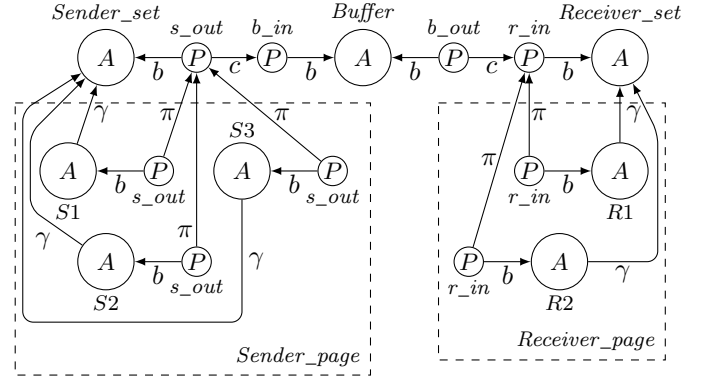


Fig. 5

EDG GRAPH REPRESENTING A SYSTEM

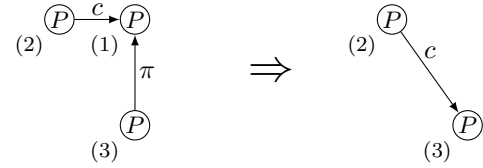


Fig. 6

PRODUCTION P_0

- find an occurrence of L in the graph \mathcal{G} ;
- remove from \mathcal{G} all nodes $a \in V_L - V_R$, all edges α such that $succ(\alpha) = a$ or $pred(\alpha) = a$ and all edges $\alpha \in E_L - E_R$;
- add to \mathcal{G} all nodes $a \in V_R - V_L$ and all edges $\alpha \in E_R - E_L$.

For example, in production shown in Fig. 6, the node indexed by (1) is removed together with edges coincident with it and a new edge is added.

4. Hierarchy elimination

The possibility of substitution of an abstract description of an agent by a more detailed one represented by a submodel (subpage) is very common in a system design. It is, however, difficult when we would like to understand (or verify) the behaviour of the whole system, associations among their components and so on. Thus, in this section we introduce the *flat* (non-hierarchical) abstraction of a system represented by its *hierarchical communication diagram*. In this representation we will use only agents and connections among them inherited from the *hierarchical communication diagram*.

To define the global set of connections, we have to take into account not only connections from sets \mathcal{C}^i , but also connections resulting from replacing hierarchical agents with subpages. For any page D^i we define a set of hierarchical

connections \mathcal{C}_H^i as follows:

$$\begin{aligned} \mathcal{C}_H^i = & \{(X_l^j.p, X_m^i.q) : \exists X_n^j \in \mathcal{A}_H^j \wedge \\ & (X_l^j.p, X_n^j.q) \in \mathcal{C}^j \wedge \gamma(X_n^j) = D^i\} \cup \\ & \{(X_m^i.q, X_l^j.p) : \exists X_n^j \in \mathcal{A}_H^j \wedge \\ & (X_n^j.q, X_l^j.p) \in \mathcal{C}^j \wedge \gamma(X_n^j) = D^i\} \end{aligned} \quad (1)$$

Finally, the global set of hierarchical connections \mathcal{C}_H is the sum $\mathcal{C}_H = \bigcup_{i=1, \dots, k} \mathcal{C}_H^i \cup \mathcal{C}_H^i$.

Definition 6: For any two agents $X \in \mathcal{A}_H$ and $Y \in \mathcal{A}$, X is said to be *hierarchically dependent on* Y , denoted as $X \succeq Y$, iff $X = Y$ or $\exists k : X = Y_1 \succ \dots \succ Y_k = Y$ for some $Y_1, \dots, Y_k \in \mathcal{A}$.

Definition 7: A *flat representation* of a communication diagram $H = (\mathcal{D}, \gamma)$ is the triple $(\mathcal{F}, \mathcal{C}', \sigma')$ such that:

- 1) $\forall X, Y \in \mathcal{F} \subseteq \mathcal{A} : \neg(X \succeq Y)$,
- 2) $\forall X \in \mathcal{A} - \mathcal{A}_H \exists Y \in \mathcal{F} : Y \succeq X$,
- 3) $\mathcal{C}' = \{(X.p, Y.q) \in \mathcal{C}_H : X, Y \in \mathcal{F}\}$,
- 4) $\sigma' = \sigma|_{\mathcal{A}_A \cap \mathcal{F}}$.

Theorem 1: Let \mathcal{F} be a set of active agents. The verification if there exists \mathcal{C}' and σ' such that $(\mathcal{F}, \mathcal{C}', \sigma')$ is a *flat representation* of a communication diagram H can be done with linear computational complexity with respect to the number of agents in the system (i.e. $\mathcal{O}(\text{card}(\mathcal{A}))$) and it can be generated with linear computational complexity with respect to the number of ports in the system (i.e. $\mathcal{O}(\text{card}(\mathcal{P}))$).

Proof:

- 1) For all $X \in \mathcal{F}$:
 - a) We visit nodes (marking them by *mu*) moving in the direction pointed by edges labelled by γ , until no node is pointed or the destination node is already marked.
 - b) We visit nodes (marking them by *md*) moving in the opposite direction to the one pointed by edges labelled by γ , until no node can be pointed. Let us note that while traversing in the opposite direction, we are traversing the tree structure. When we find a node that has been earlier marked, we generate negative answer and skip the algorithm.
- 2) We check if there exists a node (representing an agent) that has not been marked. If such a node is found, then we return a negative answer. Otherwise, we verify the set of agents \mathcal{F} as the base for generation of the flat graph positively.

To generate \mathcal{C}' we copy the graph representation of H and denote it as CGR.

- 1) Next we remove the following nodes and edges:
 - a) nodes representing agents marked by *mu*,
 - b) nodes representing agents marked by *md* and all nodes representing ports associated with them,
 - c) edges coincident with removed nodes.

- 2) For all nodes representing ports in CGR' graph (generated in the 1 phase), we apply the production P_0 represented in Fig. 6. For the node indexed as (1), we apply the production P , if we find nodes (2) and (3) such that they are connected in the way shown in the left side of the production. Then, we replace this subgraph by the one presented on the right side of the production.

Let us note that the verification phase needs visiting all agents, thus its computational complexity is $\mathcal{O}(\text{card}(\mathcal{A}))$. The generation of the flat representation needs visiting all ports and all agents ($\text{card}(\mathcal{P}) > \text{card}(\mathcal{A})$) thus the computational complexity of this algorithm is $\mathcal{O}(\text{card}(\mathcal{P}))$. ■

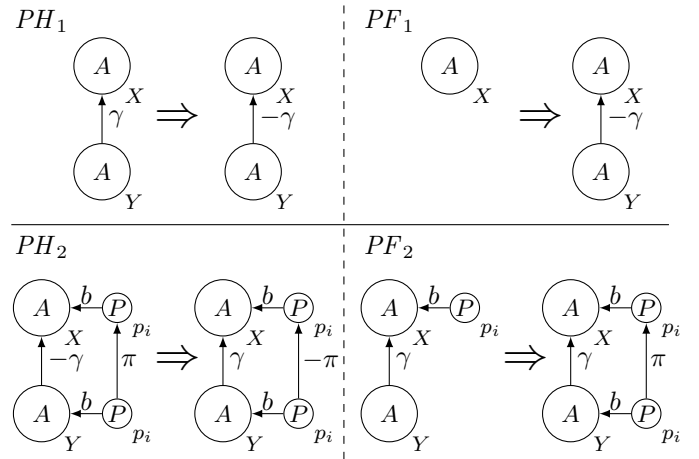


Fig. 7

PRODUCTIONS SUPPORTING ANALYSIS GRAPH TRANSFORMATION – COHESION MODE

It is easy to check that the set of *primary pages* is a *flat representation* of a system represented by a hierarchical communication diagram.

We can move from one flat system representation to another, a more detailed one, using the analysis operation.

Definition 8: Let H be a hierarchical communication diagram, $F = (\mathcal{F}, \mathcal{C}', \sigma')$ be a flat representation of H , $X \in \mathcal{A}_H \cap \mathcal{F}$ and $\gamma(X) = D^i = (\mathcal{A}^i, \mathcal{C}^i, \sigma^i)$. *Analysis of the flat representation* $(\mathcal{F}, \mathcal{C}', \sigma')$ of the hierarchical diagram H in context of X is the flat representation $(\mathcal{F}^*, \mathcal{C}^*, \sigma^*)$ (denoted $\text{AN}(H, \mathcal{F}, X)$), such that:

- 1) $\mathcal{F}^* = \mathcal{F} - \{X\} \cup \mathcal{A}^i$,
- 2) $\mathcal{C}^* = \{(Z.p, Z'.q) \in \mathcal{C}_H : Z, Z' \in \mathcal{F}^*\}$,
- 3) $\sigma^* = \sigma|_{\mathcal{A}_A \cap \mathcal{F}^*}$.

Theorem 2: The computational complexity of the analysis operation is linear with respect to the number of join ports of the analysed agent X (i.e. $\mathcal{O}(\text{card}(\text{join}_X(\gamma(X))))$).

Proof: The algorithm of the designation flat representation preserves the same indexation of nodes in the hierarchical graph $G(H)$ and the generated flat one. Thus, for the notation simplicity, we will not differ node X in the hierarchical graph $G(H)$ with its copy in the F graph. While generation of the $AN(H, \mathcal{F}, X)$:

- 1) For every $Y \in \gamma(X)$ we apply the productions PH_1 and PF_1 presented in Fig. 7. These productions are performed in a cohesion mode (see [4], [5] for details), which means that either both of them are applied or none of them is executed. Note that because the production PH_1 changes label γ onto $-\gamma$, we can apply these pairs of productions only $card(\gamma(X))$ times.
- 2) We try to apply in the same cohesion mode productions PH_2 and PF_2 – in this case left sides of both productions should be matched with $G(H)$ and \mathcal{F} . Note that because the production PH_2 changes label π onto $-\pi$, we can apply these pairs of productions only $card(join_X(\gamma(X)))$ times.
- 3) In the \mathcal{F} graph we apply production P_0 (from Fig. 6) until the edges labelled by π are eliminated.
- 4) We remove node X from the graph F .
- 5) We exchange labels $-\pi$ onto π and $-\gamma$ onto γ in the graph $G(H)$.

Each of the transformation rules applied to generation of the $AN(H, \mathcal{F}, X)$ is applied at most $card(join_X(\gamma(X)))$ times so the final computational complexity of this algorithm is $\mathcal{O}(card(join_X(\gamma(X))))$. ■

Definition 9: Let H be a hierarchical communication diagram, $(\mathcal{F}, \mathcal{C}', \sigma')$ be a flat representation of H , $Y \in \mathcal{F}$ and $\exists X \in A_H$ such that $X \succ Y$ and $\gamma(X) = D^i = (A^i, C^i, \sigma^i)$. *Synthesis of the flat representation* $(\mathcal{F}, \mathcal{C}', \sigma')$ of the hierarchical diagram H in context of Y is the flat representation $(\mathcal{F}^*, \mathcal{C}^*, \sigma^*)$ (denoted as $SN(H, \mathcal{F}, Y)$) such that:

- 1) $\mathcal{F}^* = \mathcal{F} - A^i \cup \{X\}$,
- 2) $\mathcal{C}^* = \{(Z.p, Z'.q) \in \mathcal{C}_H : Z, Z' \in \mathcal{F}^*\}$,
- 3) $\sigma^* = \sigma|_{A \cap \mathcal{F}^*}$.

Theorem 3: The computational complexity of the synthesis operation is linear with respect to the number of join ports of the considered agent X (i.e. $\mathcal{O}(card(\mathcal{P}(\gamma(X))))$).

Proof: Analogously as in the analysis case, for the notation simplicity, we will not differ node X in the hierarchical graph $G(H)$ with its copy in F graph. To generate $SN(H, \mathcal{F}, Y)$ we have to perform the following steps:

- 1) For a node Y , we designate the representation of a hierarchical agent X as exactly the node that belongs to edge $(Y, \gamma, X) \in E_{G(H)}$. Note that $X \succeq Y$.
- 2) For every $Y \in \gamma(X)$ we apply the productions PH_1 and PF_1 presented in Fig. 8. These productions are performed in a cohesion mode, which means that

either both of them are applied or none of them is executed. Note that because of the fact that the production PH_1 change a label γ onto $-\gamma$, we can apply the pair of productions only $card(\gamma(X))$ times.

- 3) We try to apply in the same cohesion mode the productions PH_2 and PF_2 – in this case left sides of both productions should be matched with $G(H)$ and F . Note that because of the fact that the production PH_2 changes a label π onto $-\pi$, we can apply the pair of productions only $card(join_X(\gamma(X)))$ times.
- 4) For every edge $(p, \pi, q) \in \mathcal{F}'$ and for every edge $(r, c, q) \in \mathcal{F}$ we apply the production PF_3 . The edges, such that $lab(\alpha) = \pi$ has been added in the previous step and their number is limited by $card(join_X(\gamma(X)))$. These edges are removed in the PF_3 production, thus PF_3 can be applied no more than $card(join_X(\gamma(X)))$ times.
- 5) For every edge $(p, \pi, q) \in \mathcal{F}'$ and for every edge $(q, c, r) \in \mathcal{F}$ we apply the production PF_4 . Analogously as in the previous step, PF_4 can be applied no more than $card(join_X(\gamma(X)))$ times.
- 6) The production PF_5 removes ports belonging to nodes Y , such that $X \succeq Y$. The production can be applied no more than $card(\mathcal{P}(\gamma(X)))$ times.
- 7) The production PF_6 removes nodes Y , such that $X \succeq Y$. The production can be applied no more than $card(\gamma(X))$ times. ■

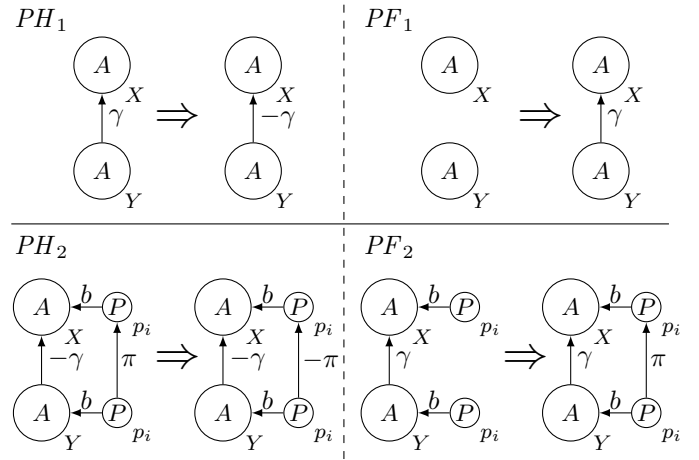


Fig. 8

PRODUCTIONS SUPPORTING SYNTHESIS GRAPH TRANSFORMATION – COHESION MODE

In the next section we consider a flat representation without hierarchical agents, such a representation is the maximal one from the analysis point of view.

Definition 10: A flat representation $(\mathcal{F}, \mathcal{C}', \sigma')$ is called

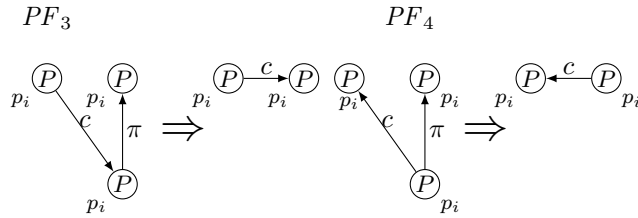


Fig. 9

ASSOCIATION PORTS OF THE REST SYSTEM WITH THE PORTS OF AGENT X

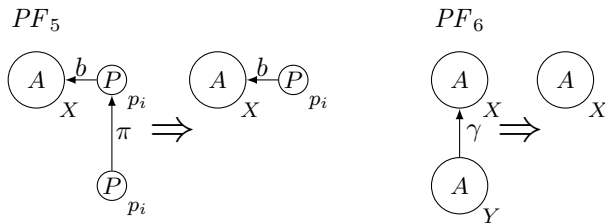


Fig. 10

REMOVING AGENTS AND PORTS PAGE $\gamma(X)$

the *maximal flat representation* iff $\forall X \in \mathcal{A} \exists Y \in \mathcal{F}: X \succeq Y$.

5. Usefulness of the flat representation

Even though the main topic of the paper is connected with graphs representation, it should be underlined that a complete Alvis model contains three layers [1]. Communication diagrams are used to define interconnections among agents. Each non-hierarchical agent must be also defined in the code layer. Such a code layer for the considered example is presented in Fig. 11.

The behaviour of each active agent in the model under consideration is defined using an infinite loop. Senders repeat the following steps: 1) entering the loop, 2) preparing a new value for the k parameter (the remainder of division $k + 1$ by 2) and 3) sending the current value of k via the s_out port. Receivers repeat two steps: 1) entering the loop, 2) collecting a value for the k parameter via the r_in port. The only passive agent *Buffer* provides two procedures for putting or getting a value from it. For more details about the Alvis Code Language see [1] or [2].

A state of a model is represented as a sequence of agents states. To describe the current state of an agent, we need a tuple with four pieces of information: *agent mode* (am), *program counter* (pc), *context information list* (ci) and *parameters values tuple* (pv) [3]. The mode is used to indicate whether an agent is, for example, *running* or *waiting* for an event. The program counter points out

```

agent S1, S2, S3 {
  k :: Int = 0;
  loop {
    k = rem (k + 1) 2;
    out s_out k; }
}

agent R1, R2 {
  k :: Int = 0;
  loop { in r_in k; }
}

agent Buffer {
  i :: Int = 0;
  proc b_in { in b_in i; }
  proc b_out { out b_out i; }
}

```

Fig. 11
CODE LAYER

the current or the next step to be executed. The context information list contains additional information about the current agent's state, e.g. the name of the port used in the current communication. The parameters values tuple contains values of the agent's parameters. The initial state for the model under consideration is as follows:

```

S1: (running, 1, [], (0))
S2: (running, 1, [], (0))
S3: (running, 1, [], (0))
R1: (running, 1, [], (0))
R2: (running, 1, [], (0))
Buffer: (waiting, 0, [in(b_in), out(b_out)], (0))

```

Agents $S1, S2, \dots, R2$ are about executing their first step. The agent *Buffer* is waiting for calling one of its procedures.

We consider behaviour of Alvis models at the level of detail of single steps. Each step is realised in the context of one active agent. Also procedures of passive agents are realised in the context of active agents that called them. The third layer of an Alvis model is called *system layer*. It is the predefined layer and provides, among other things, information about the number of processors accessible in a considered embedded system. If more than one processor is accessible, then some agents can execute their steps concurrently. Suppose that the α^0 system layer is considered i.e. there is unlimited number of processors (each active agent has access to its own processor). Moreover, assume that executing *loop* and *exec* steps takes 1 millisecond and executing *in* and *out* steps takes 2 milliseconds. Thus, after executing 5 *loop* steps concurrently, after 1 millisecond the considered model reaches the following state:

```

S1: (running, 2, [], (0))
S2: (running, 2, [], (0))
S3: (running, 2, [], (0))
R1: (running, 2, [], (0))
R2: (running, 2, [], (0))

```

```
Buffer: (waiting, 0, [in(b_in), out(b_out)], (0))
```

Similarly, after the next 1 ms we have:

```
S1: (running, 3, [], (1))
S2: (running, 3, [], (1))
S3: (running, 3, [], (1))
R1: (running, 2, [sft(1)], (0))
R2: (running, 2, [sft(1)], (0))
Buffer: (waiting, 0, [in(b_in), out(b_out)], (0))
```

The *stf* abbreviation stands for *step finish time* and means that the agents *R1* and *R2* need 1 ms more to finish their current step. Such a state is called *snapshot*. Of course, we can take a snapshot every 1 millisecond, but we are interested only in these snapshots where at least one step has finished its execution. For the sake of simplicity, states where all agents have finished their steps are also called snapshots. The set of all reachable snapshots and transitions among them is represented in the form of directed graph called *snapshot reachability graph* or SR-graph for short. Nodes of such a graph represent reachable snapshots, while edges sets of concurrently executed steps that lead from one snapshot to another. An SR-graph is a kind of LTS graph i.e. *Labelled Transition System*.

The possibility of a formal model verification makes Alvis a formal modelling language. As it has already been said, Alvis modelling environment creates in parallel a model of the considered system and a labelled transition system (SR-graph) that is its formal representation. The SR-graph can be formally verified with the CADP toolbox [6].

There are two approaches considered for the SR-graph generation. The first one uses Haskell representation of an Alvis model. *Alvis Translator* that is part of the Alvis modelling environment called *Alvis Toolkit* translates an Alvis model into a Haskell program that generates the SR-graph. The second approach uses *Alvis VM* (Virtual Machine). In this approach, Alvis Translator is used to translate an Alvis model into object representation suitable for the machine and Alvis VM is used to generate the SR-graph.

Both approaches are still under development, but what is more important, they need the maximal flat representation of the corresponding communication diagram. The proposed graph representation not only seems to be the most suitable one for this purpose, but also is convenient from implementation point of view (such a graph can be represented, for example, in the form of a single matrix with integer elements). The analysis operation translates one matrix representation of a graph into another equivalent one. Alvis Translator takes the graph representation of a model communication diagram as its input and applies the analysis operation until the *maximal flat representation* is received. Then, the representation is used for next steps of the transformation algorithms.

6. Conclusion

The formal LTS model, representing the behaviour of the designed system, is equivalent to the some *flat representation* of the hierarchical Alvis system structure described both in Alvis Code Language and Alvis Communication Diagrams. The introduced graph representation seems to be suitable for implementation, but what is more important, it allows one to formally prove the polynomial computational complexity of the basic operations on this structure. It was proved that:

- the verification of a *flat representation* of a communication diagram H can be done with the linear computational complexity with respect to the number of agents in the system (i.e. $\mathcal{O}(\text{card}(\mathcal{A}))$) and it can be generated with the linear computational complexity with respect to the number of ports in the system (i.e. $\mathcal{O}(\text{card}(\mathcal{P}))$).
- the computational complexity of the analysis operation is linear with respect to the number of join ports of the considered agent X (i.e. $\mathcal{O}(\text{card}(\text{join}_X(\gamma(X))))$).
- the computational complexity of the synthesis operation is linear with respect to the number of join ports of the considered agent X (i.e. $\mathcal{O}(\text{card}(\mathcal{P}(\gamma(X))))$).

Acknowledgement

The paper is supported by the Alvis Project funded from 2009-2010 resources for science as a research project.

References

- [1] M. Szpyrka, P. Matyasik, and R. Mrówka, "Alvis – modelling language for concurrent systems," in *Intelligent Decision Systems in Large-Scale Distributed Environments*, ser. Studies in Computational Intelligence, P. Bouvry, H. Gonzalez-Velez, and J. Kołodziej, Eds. Springer-Verlag, 2011, (in press).
- [2] M. Szpyrka, *Alvis On-line Manual*, AGH University of Science and Technology, 2011. [Online]. Available: <http://fm.ia.agh.edu.pl/alvis:manual>
- [3] M. Szpyrka, P. Matyasik, R. Mrówka, L. Kotulski, and K. Balicki, "Formal introduction to Alvis modelling language," *International Journal of Applied Mathematics and Computer Science*, 2011, (to appear).
- [4] L. Kotulski and L. Fryz, "Conjugated graph grammars as a mean to assure consistency of systems of conjugated graphs," in *Proceedings of the 2008 Third International Conference on Dependability of Computer Systems DepCoS-RELCOMEX*. Szklarska Poreba, Poland: IEEE Computer Society, June 26-28 2008, pp. 119–126.
- [5] L. Kotulski, "GRADIS – multiagent environment supporting distributed graph transformations," in *Computational Science – ICCS 2008*, ser. LMCS, M. Bubak, G. van Albada, J. Dongarra, and P. Sloot, Eds. Springer-Verlag, 2008, vol. 5103, pp. 644–653.
- [6] H. Garavel, F. Lang, R. Mateescu, and W. Serwe, "CADP 2006: A toolbox for the construction and analysis of distributed processes," in *Computer Aided Verification (CAV'2007)*, ser. LNCS, vol. 4590. Berlin, Germany: Springer, 2007, pp. 158–163.

⁰For more details see Alvis web site: <http://fm.ia.agh.edu.pl>.

Inapproximability of Maximum r -Regular Induced Connected Subgraph Problems

Yuichi Asahiro¹, Hiroshi Eto², and Eiji Miyano²

¹Department of Information Science, Kyushu Sangyo University,
Fukuoka 813-8503, Japan. asahiro@is.kyusan-u.ac.jp

²Department of System Design and Informatics, Kyushu Institute of Technology,
Fukuoka 820-8502, Japan. {eto@theory., miyano@}ces.kyutech.ac.jp

Abstract—Given a graph $G = (V, E)$ on n vertices, the MAXIMUM r -REGULAR INDUCED CONNECTED SUBGRAPH (r -MaxRICS) problems ask for a maximum sized subset of vertices $S \subseteq V$ such that the induced subgraph $G[S]$ on S is connected and r -regular. For $r = 2$, it is known that 2-MaxRICS is \mathcal{NP} -hard and cannot be approximated within a factor of $n^{1-\varepsilon}$ in polynomial time for any $\varepsilon > 0$ if $\mathcal{P} \neq \mathcal{NP}$. In this paper, we show that r -MaxRICS is \mathcal{NP} -hard for any fixed integer $r \geq 3$, and furthermore r -MaxRICS cannot be approximated within a factor of $n^{1/6-\varepsilon}$ in polynomial time for any $\varepsilon > 0$ if $\mathcal{P} \neq \mathcal{NP}$.

Keywords: induced connected subgraph, regularity, \mathcal{NP} -hardness, inapproximability

1. Introduction

In this paper we only consider simple undirected graphs with no loops and no multiple edges. Let $G = (V(G), E(G))$ be a graph, where $V(G)$ and $E(G)$ denote the set of vertices and the set of edges in G , respectively. A graph G_S is a subgraph of a graph G if $V(G_S) \subseteq V(G)$ and $E(G_S) \subseteq E(G)$. For a subset of vertices $S \subseteq V(G)$, by $G[S]$, we mean the subgraph of G induced on S , which is called the *induced subgraph*.

The problem MAXIMUM INDUCED SUBGRAPH (MaxIS) of finding the maximum number of vertices that induces a subgraph satisfying some properties is one of the most fundamental problems in the fields of graph theory and combinatorial optimization, and thus extensively studied in these decades. Unfortunately, however, it is well known that the MaxIS problem is \mathcal{NP} -hard for a large class of interesting properties. For example, in [7], Lund and Yannakakis prove that the MAXIMUM INDUCED SUBGRAPH problem for the natural properties such as *acyclicity*, *planarity*, and *bipartiteness* cannot be approximated within a factor of $n^{1-\varepsilon}$ in polynomial time for any positive constant ε if $\mathcal{P} \neq \mathcal{NP}$, where n is the number of the vertices in the input graph.

1.1 Our Problems and Results

A graph is r -regular if the degree of every vertex is exactly r . The *regularity* of graphs must be one of the most basic properties. In this paper we consider the following variant of the MaxIS problem, i.e., the desired properties the induced subgraph must satisfy are *regularity* and *connectivity*:

MAXIMUM r -REGULAR INDUCED CONNECTED SUBGRAPH (r -MaxRICS)

Input: A graph $G = (V, E)$ and an integer r .

Goal: Find a maximum subset of vertices $S \subseteq V$ such that the induced subgraph $G[S]$ on S is connected and r -regular.

Since a *clique* is connected and regular, the MAXIMUM CLIQUE problem may be regarded as a special one of r -MaxRICS. The MAXIMUM CLIQUE is very difficult even to approximate [5]. Clearly, however, the problem of finding a clique of a constant degree is solvable in polynomial time. On the other hand, r -MaxRICS is hard even if r is a small constant as follows: The problem 2-MaxRICS is known as LONGEST INDUCED CYCLE problem since a 2-regular subgraph means a cycle in the input graph. In [6] Kann shows the following inapproximability for 2-MaxRICS:

Theorem 1 ([6]): 2-MaxRICS cannot be approximated in polynomial time within a factor of $n^{1-\varepsilon}$ for any constant $\varepsilon > 0$ if $\mathcal{P} \neq \mathcal{NP}$, where n is the number of vertices in the input graph.

In [3] Bonifaci, Di Iorio, and Laura consider the following problem and show its \mathcal{NP} -hardness:

MAXIMUM REGULAR INDUCED SUBGRAPH (MaxRIS)

Input: A graph $G = (V, E)$.

Goal: Find a maximum subset of vertices $S \subseteq V$ such that the induced subgraph $G[S]$ on S is regular.

Strictly speaking, MaxRIS is slightly different from r -MaxRICS, but the same reduction introduced in [3] shows the following intractability when $r = 3$:

Theorem 2 ([3]): 3-MaxRICS is \mathcal{NP} -hard.

However, it would be hard to show the hardness of approximating r -MaxRICS for $r \geq 3$ by using a similar reduction with small modification to the reduction in [3]. In this paper, by using a different *gap-preserving* reduction, we first show the following inapproximability of 3-MaxRICS.

Theorem 3: 3-MaxRICS cannot be approximated in polynomial time within a factor of $n^{1/6 - \varepsilon}$ for any constant $\varepsilon > 0$ if $\mathcal{P} \neq \mathcal{NP}$, where n is the number of vertices in the input graph.

Furthermore, by using additional ideas to the reduction, we show the same inapproximability of r -MaxRICS for any fixed integer $r \geq 4$.

Corollary 1: For any fixed integer $r \geq 4$, r -MaxRICS cannot be approximated in polynomial time within a factor of $n^{1/6 - \varepsilon}$ for any constant $\varepsilon > 0$ if $\mathcal{P} \neq \mathcal{NP}$, where n is the number of vertices in the input graph.

The proofs of Theorem 3 and Corollary 1 will be given in Section 3.

1.2 Related Work

Recently, the problem of finding a maximum induced subgraph having regularity is very popular. Many researchers study the following variant, that is, the connectivity property is not imposed on the induced subgraph.

MAXIMUM r -REGULAR INDUCED SUBGRAPH (r -MaxRIS)

Input: A graph $G = (V, E)$ and an integer r .
Goal: Find a maximum subset of vertices $S \subseteq V$ such that the induced subgraph $G[S]$ on S is r -regular.

If we does not require the connectivity constraint, then the problems when $r = 0$ and $r = 1$ correspond to the well studied MAXIMUM INDEPENDENT SET and MAXIMUM INDUCED MATCHING problems, respectively. The former problem is hard even to approximate [5]. The \mathcal{NP} -hardness of the latter problem is also shown in [1], [10]. In [9] Orlovich, Finke, Gordon, and Zverovich prove the MAXIMUM INDUCED MATCHING cannot be approximated within a factor of $|V|^{1/2 - \varepsilon}$ in polynomial time for any $\varepsilon > 0$. The parameterized complexity and exact exponential algorithms of r -MaxRIS are studied in [8] and [4], respectively. Very recently, in [2] Cardoso, Kamiński, and Lozin prove that r -MaxRIS is \mathcal{NP} -hard for any value of $r \geq 3$. Motivated by this result, we investigate the complexity of the connected version problem r -MaxRICS for $r \geq 3$ in this paper.

2. Notation

By (u, v) we denote an edge with endpoints u and v . For a vertex u , the set of vertices adjacent to u in G is denoted by $N_G(u)$ or simply by $N(u)$, and $(u, N_G(u))$ denotes the set $\{(u, v) \mid v \in N_G(u)\}$ of edges. Let the degree of a vertex u be denoted by $\deg(u)$, i.e., $|N(u)| = \deg(u)$. A (simple) path P of length ℓ from a vertex v_0 to a vertex v_ℓ is represented as a sequence of vertices such that $P = \langle v_0, v_1, \dots, v_\ell \rangle$, and $|P|$ denotes the length of P . A cycle C of length ℓ is similarly denoted by $C = \langle v_0, v_1, \dots, v_{\ell-1}, v_0 \rangle$, and $|C|$ denotes the length of C . A *chord* of a path (cycle) $\langle v_0, \dots, v_\ell \rangle$ ($\langle v_0, \dots, v_{\ell-1}, v_0 \rangle$) is an edge between two vertices of the path (cycle) that is not an edge of the path (cycle). A path (cycle) is *chordless* if it contains no chords, i.e., an induced cycle must be chordless. Let G_1, G_2, \dots, G_ℓ be ℓ graphs and also let v_i be some vertex in G_i for $1 \leq i \leq \ell$. Then, $\langle G_1, G_2, \dots, G_\ell \rangle$ denotes the subgraph $G = (V(G_1) \cup V(G_2) \cup \dots \cup V(G_\ell), E(G_1) \cup E(G_2) \cup \dots \cup E(G_\ell) \cup \{(v_1, v_2), (v_2, v_3), \dots, (v_{\ell-1}, v_\ell)\})$. That is, two adjacent graphs G_{i-1} and G_i are connected by only one edge and G roughly forms a path. Similarly, $\langle G_1, G_2, \dots, G_\ell, G_1 \rangle$ roughly forms a cycle.

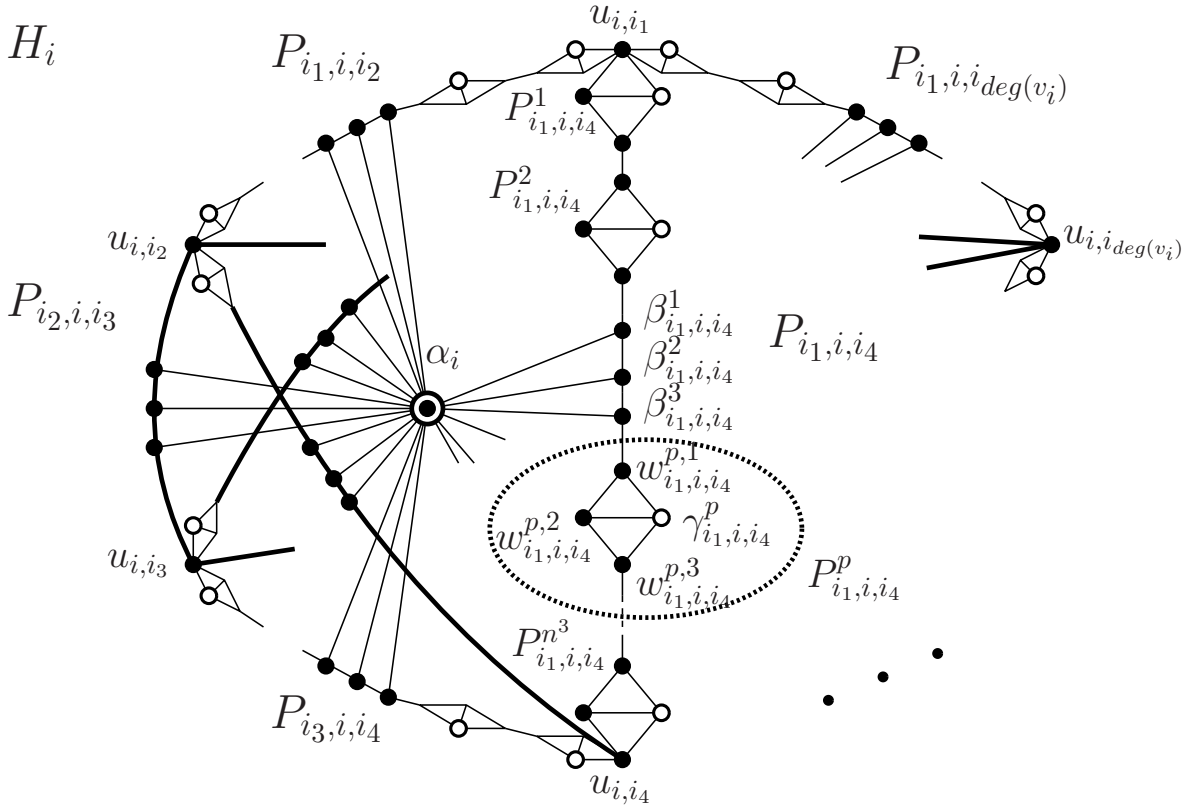
3. Hardness of Approximating r -MaxRICS

In this section we give the proofs of Theorem 3 and Corollary 1. The hardness of approximating r -MaxRICS for $r \geq 3$ is shown via a gap-preserving reduction from LONGEST INDUCED CYCLE problem, i.e., 2-MaxRICS. Consider an input graph $G = (V(G), E(G))$ of 2-MaxRICS with n vertices and m edges. Then, we construct a graph $H = (V(H), E(H))$ of r -MaxRICS. First we show the $O(n^{1/6 - \varepsilon})$ -inapproximability of 3-MaxRICS and then the same $O(n^{1/6 - \varepsilon})$ -inapproximability of the general r -MaxRICS problem for $r \geq 4$.

Let $OPT_1(G)$ (and $OPT_2(H)$, respectively) denote the number of vertices of an optimal solution for G of 2-MaxRICS (and H of r -MaxRICS, respectively). Let $V(G) = \{v_1, v_2, \dots, v_n\}$ of n vertices and $E(G) = \{e_1, e_2, \dots, e_m\}$ of m edges. Let $g(n)$ be a parameter function of the instance G . Then we provide the gap preserving reduction such that (C1) if $OPT_1(G) \geq g(n)$, then $OPT_2(H) \geq 4(n^3 + 1) \times g(n)$, and (C2) if $OPT_1(G) < \frac{g(n)}{n^{1-\varepsilon}}$ for a positive constant ε , then $OPT_2(H) < 4(n^3 + 1) \times \frac{g(n)}{n^{1-\varepsilon}}$. As we will explain it, the number of vertices in the reduced graph H is $O(n^6)$. Hence the approximation gap is $n^{1-\varepsilon} = O(|V(H)|^{1/6 - \varepsilon})$ for any constant $\varepsilon > 0$.

3.1 Reduction for $r = 3$

Without loss of generality, we can assume that there is no vertex whose degree is one in the input graph G

Figure 1: Subgraph H_i

of 2-MaxRICS. The reason is that such a vertex does not contribute to any feasible solution, i.e., a cycle, of 2-MaxRICS and can be removed from G .

The constructed graph H consists of (i) n subgraphs, H_1 through H_n , which are associated with n vertices, v_1 through v_n , respectively, and (ii) m edge sets, E_1 through E_m , which are associated with m edges, e_1 through e_m , respectively.

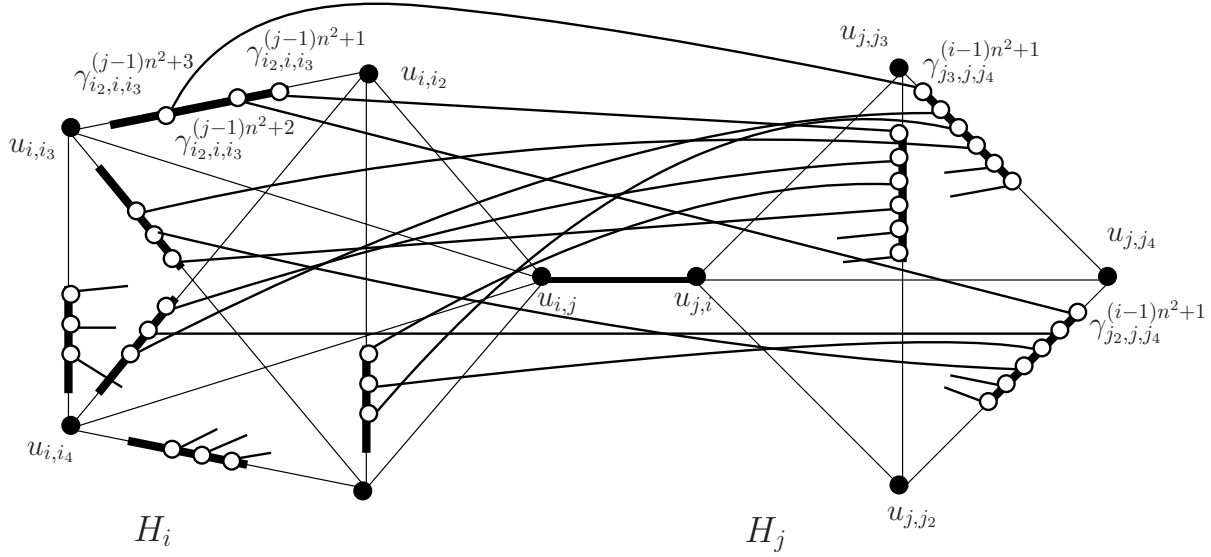
(i) Here we describe the construction of the i th subgraph H_i in detail for some i ($1 \leq i \leq n$). See Figure 1, which illustrates H_i . Suppose that the set of vertices adjacent to v_i is $N(v_i) = \{v_{i_1}, v_{i_2}, \dots, v_{i_{deg(v_i)}}\}$, where $i_j \in \{1, 2, \dots, n\} \setminus \{i\}$ for $1 \leq j \leq deg(v_i)$. The subgraph $H_i = (V(H_i), E(H_i))$ includes $deg(v_i)$ vertices, u_{i, i_1} through $u_{i, i_{deg(v_i)}}$ that correspond to the vertices adjacent to v_i , and $deg(v_i)(deg(v_i) - 1)/2$ path gadgets, $P_{i_1, i, i_2}, P_{i_1, i, i_3}, \dots, P_{i_1, i, i_{deg(v_i)}}, P_{i_2, i, i_3}, \dots, P_{i_{deg(v_i)-1}, i, i_{deg(v_i)}}$, where two vertices u_{i, i_j} and u_{i, i_k} are connected via the path gadget P_{i_j, i, i_k} for $v_{i_j}, v_{i_k} \in N(v_i)$. As an example, in Figure 1, the top vertex u_{i, i_1} and the bottom u_{i, i_4} are connected via P_{i_1, i, i_4} . Each path gadget P_{i_j, i, i_k} includes n^3 subgraphs,

P_{i_j, i, i_k}^1 through $P_{i_j, i, i_k}^{n^3}$, where, for each $1 \leq p \leq n^3$,

$$V(P_{i_j, i, i_k}^p) = \{w_{i_j, i, i_k}^{p,1}, w_{i_j, i, i_k}^{p,2}, w_{i_j, i, i_k}^{p,3}, \gamma_{i_j, i, i_k}^p\},$$

$$E(P_{i_j, i, i_k}^p) = (\gamma_{i_j, i, i_k}^p, \{w_{i_j, i, i_k}^{p,1}, w_{i_j, i, i_k}^{p,2}, w_{i_j, i, i_k}^{p,3}\}) \cup \{(w_{i_j, i, i_k}^{p,1}, w_{i_j, i, i_k}^{p,2}), (w_{i_j, i, i_k}^{p,2}, w_{i_j, i, i_k}^{p,3})\}.$$

In the path gadget P_{i_j, i, i_k} , two vertices $w_{i_j, i, i_k}^{p,1}$ and $w_{i_j, i, i_k}^{p,3}$ are respectively identical to the vertices u_{i, i_j} and u_{i, i_k} prepared in the above. For $2 \leq p \leq n^3$, contiguous two subgraphs P_{i_j, i, i_k}^{p-1} and P_{i_j, i, i_k}^p are connected by one edge $(w_{i_j, i, i_k}^{p-1,3}, w_{i_j, i, i_k}^{p,1})$ except for a pair P_{i_j, i, i_k}^{q-1} and P_{i_j, i, i_k}^q for some q : the two subgraphs P_{i_j, i, i_k}^{q-1} and P_{i_j, i, i_k}^q are connected by a path of length four $\langle w_{i_j, i, i_k}^{q-1,3}, \beta_{i_j, i, i_k}^1, \beta_{i_j, i, i_k}^2, \beta_{i_j, i, i_k}^3, w_{i_j, i, i_k}^{q,1} \rangle$. This q can be arbitrary since we just want to insert the path of length four into the path gadget, and as an example, $q = 3$ in the path gadget P_{i_1, i, i_4} in Fig. 1. Finally, we prepare a special vertex α_i , and α_i is connected to all $\{\beta_{i_1, i, i_4}^1, \beta_{i_1, i, i_4}^2, \beta_{i_1, i, i_4}^3\}$'s. In the following, $\alpha_1, \alpha_2, \dots, \alpha_n$ are called α -vertices. Similarly, β -vertices and γ -vertices mean the vertices labeled


 Figure 2: E_k connecting H_i and H_j

by β and γ , respectively. Since each path gadget has $4n^3 + 3$ vertices (two of which are shared with other path gadgets), the total number of vertices in H_i is

$$|V(H_i)| = \frac{\deg(v_i)(\deg(v_i) - 1)(4n^3 + 1)}{2} + n + 1,$$

i.e., there are $O(n^5)$ vertices in H_i .

(ii) Next we explain construction of the edge sets E_1 through E_m . Now suppose that e_k connects v_i with v_j for $i \neq j$. Also suppose that the sets of vertices adjacent to v_i and v_j are $N(v_i) = \{j, i_2, \dots, i_{\deg(v_i)}\}$ and $N(v_j) = \{i, j_2, \dots, j_{\deg(v_j)}\}$, respectively. Then, $(u_{i,j}, u_{j,i}) \in E_k$ where $u_{i,j} \in V(H_i)$ in the i th subgraph H_i and $u_{j,i} \in V(H_j)$ in the j th subgraph H_j . Furthermore, by the following rules, γ -vertices in the path gadgets are connected: See Figure 2. Every vertex in the path gadget $P_{x,i,y}$ for $x = j$ or $y = j$ in H_i is not connected to any vertex in H_j , except for $u_{i,j}$. Similarly, every vertex in $P_{s,j,t}$ for $s = i$ or $t = i$ in H_j is not connected to H_i , except for $u_{j,i}$. For a path gadget $P_{x,i,y}$ in H_i , where $j \notin \{x, y\}$ we prepare a set of edges as follows. Let $D = \min_{k \in \{i, j\}} \{\deg(v_k)(\deg(v_k) - 1)/2 - (\deg(v_k) - 1)\}$.

- In $P_{x,i,y}$, there are n^3 γ -vertices, $\gamma_{x,i,y}^1$ through $\gamma_{x,i,y}^{n^3}$. Consider D γ -vertices among those n^3 γ -vertices, the $((j-1)n^2 + 1)$ th vertex $\gamma_{x,i,y}^{(j-1)n^2+1}$ through the $((j-1)n^2 + D)$ th vertex $\gamma_{x,i,y}^{(j-1)n^2+D}$.
- Next take a look at the j th subgraph H_j and the path gadgets $P_{s,j,t}$'s for $i \notin \{s, t\}$. Note that the number of such gadgets is $\deg(v_j)(\deg(v_j) - 1)/2 - (\deg(v_j) - 1)$ and hence at least D . Then, consider the $((i-1)n^2 + 1)$ th vertex $\gamma_{s,j,t}^{(i-1)n^2+1}$ in each $P_{s,j,t}$. Here, the term

“+1” in the superscript of γ comes from the assumption that $j_1 = i$; if $j_k = i$, we consider the $((i-1)n^2 + k)$ th γ -vertex.

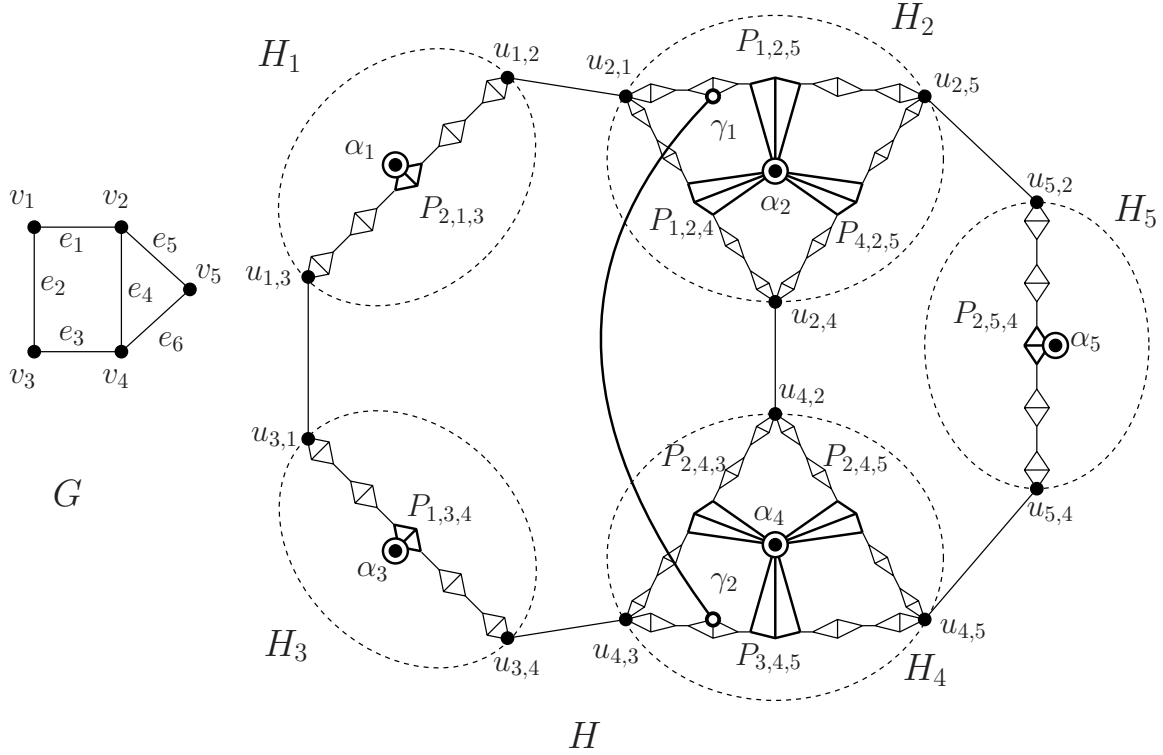
- Then, we can choose any function f which assigns each element in $\{1, \dots, D\}$ to a string s, j, t such that $i \notin \{s, t\}$ and it holds $f(b) \neq f(c)$ if $b \neq c$. Finally, we connect $\gamma_{x,i,y}^{(j-1)n^2+k}$ with $\gamma_{f(k)}^{(i-1)n^2+1}$ for $1 \leq k \leq D$. It is important that the path gadget $P_{x,i,y}$ is connected to $P_{s,j,t}$ via only one edge.

Just to make the above construction clear, see Figure 3. For example, if an input instance G is the left graph, then the reduced graph H is illustrated in the right graph, where some details on the path gadgets are omitted due to the space. For example, since two vertices v_1 and v_2 are connected via the edge e_1 in G , $u_{1,2}$ in H_1 is connected to $u_{2,1}$ in H_2 . Similarly to e_2 through e_6 , there are five edges, $(u_{1,3}, u_{3,1})$, $(u_{3,4}, u_{4,3})$, $(u_{2,4}, u_{4,2})$, $(u_{2,5}, u_{5,2})$, and $(u_{4,5}, u_{5,4})$ in H . Furthermore, two path gadgets $P_{1,2,5}$ and $P_{3,4,5}$ are connected by one edge (γ_1, γ_2) .

Each subgraph H_i has $O(n^5)$ vertices and thus the total number of vertices $|V(H)| = O(n^6)$. Clearly, this reduction can be done in polynomial time. In the next two subsections, we show that both conditions (C1) and (C2) are satisfied by the above reduction.

3.2 Proof of Condition (C1)

Without loss of generality, suppose that a longest induced cycle in G is $C^* = \langle v_1, v_2, \dots, v_\ell, v_1 \rangle$ of length ℓ , and thus $OPT_1(G) = |C^*| = \ell \geq g(n)$. Then we select the following subset S of $4(n^3 + 1) \times \ell$ vertices and the induced

Figure 3: Input graph G (left) and reduced graph H (right)

subgraph $G[S]$:

$$S = V(P_{\ell,1,2}) \cup \{\alpha_1\} \cup V(P_{1,2,3}) \cup \{\alpha_2\} \\ \cup \dots \cup V(P_{\ell-1,\ell,1}) \cup \{\alpha_\ell\}.$$

For example, take a look at the graph G illustrated in Figure 3 again. One can see that the longest induced cycle in G is $\langle v_1, v_3, v_4, v_2, v_1 \rangle$. Then, we select the connected subgraph induced on the following set of vertices:

$$V(P_{2,1,3}) \cup \{\alpha_1\} \cup V(P_{1,3,4}) \cup \{\alpha_3\} \\ \cup V(P_{2,4,3}) \cup \{\alpha_4\} \cup V(P_{1,2,4}) \cup \{\alpha_2\}$$

It is easy to see that the induced subgraph is 3-regular and connected. Hence, the reduction satisfies the condition (C1).

3.3 Proof of Condition (C2)

We show that the reduction satisfies the condition (C2) by showing its contraposition. Suppose that $OPT_2(H) \geq 4(n^3 + 1) \cdot \frac{g(n)}{n^{1-\varepsilon}}$ holds for a positive constant ε , and S^* is an optimal set of vertices such that the subgraph $H[S^*]$ induced on S^* is connected and 3-regular. In the following, one of the crucial observations is that we can select at most one path gadget from each subgraph H_i into the optimal set S^* of vertices, and if a portion of the path gadget is only selected, then the induced subgraph cannot be 3-regular.

(I) See Figure 1 again. Suppose for example that two path gadgets P_{i_1,i,i_4} and P_{i_2,i,i_3} are selected, and put their vertices into S^* . In order to make the degree of β -vertices three, we need to also select α_i . However, the degree of α_1 becomes six. This implies that we can select at most three β -vertices from each subgraph H_i .

(II) From the above observation (I), we consider the case that at most two of $\beta_{j,i,k}^1$, $\beta_{j,i,k}^2$, and $\beta_{j,i,k}^3$ are selected for some i, j, k . Let us assume that we select $\beta_{j,i,k}^1$ and $\beta_{j,i,k}^2$ ($\beta_{j,i,k}^1$ and $\beta_{j,i,k}^3$, resp.) are put into S^* , but $\beta_{j,i,k}^3$ ($\beta_{j,i,k}^2$, resp.) is not selected. Then, the degree of $\beta_{j,i,k}^2$ ($\beta_{j,i,k}^1$ and $\beta_{j,i,k}^3$, resp.) is at most 2 even if we select α_i , i.e., the induced subgraph cannot be 3-regular. By a similar reason, we can not select only one of the β -vertices. Hence, if we select β -vertices, all of the three β -vertices in one path gadget must be selected.

As for w -vertices, a similar discussion can be done: For example, if we select $w_{j,i,k}^{p,1}$ and $w_{j,i,k}^{p,3}$ for some i, j, k, p , but $w_{j,i,k}^{p,2}$ ($\gamma_{j,i,k}^p$, resp.) is not selected, then the degree of $\gamma_{j,i,k}^p$ ($w_{j,i,k}^{p,2}$, resp.) is only 2. Thus, we need to select all the vertices of the part $P_{k,i,j}^p$ if we select some vertices from it.

Combining two observations above, one can see that the edges connecting $P_{k,i,j}^{p,1}$ and $P_{k,i,j}^p$, or w -vertices and β -vertices are necessary to make the degrees of the vertices three. As a result, we can conclude that if only a part of one path gadget is chosen, then the induced subgraph obtained

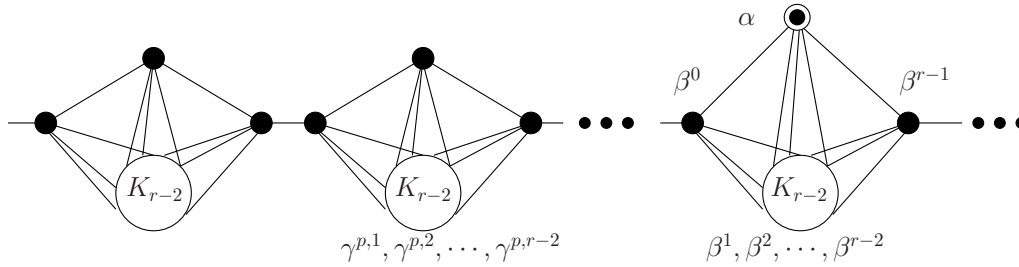


Figure 4: Modified path gadget in the proof of Corollary 1

cannot be 3-regular.

(III) From (I) and (II), we can assume that if some vertices of a path gadget are selected into S^* , it means that the whole vertices of the path gadget is selected. For example, suppose that P_{i_1, i_4} is selected. Since the degree of the endpoint u_{i_1} (u_{i_4}) of P_{i_1, i_4} is only 2, we have to put at least one vertex into S^* from another subgraph adjacent to H_{i_1} , say, a vertex u_{j, i_1} in H_j . This implies that the induced subgraph $H[S^*]$ forms a cycle-like structure $\langle H_{i_1}, H_{i_2}, \dots, H_{i_j}, H_{i_1} \rangle$ connecting $H_{i_1}, H_{i_2}, \dots, H_{i_j}, H_{i_1}$ in order, where $\{i_1, i_2, \dots, i_j\} \subseteq \{1, 2, \dots, n\}$.

We mention that such an induced subgraph $H[S^*]$ is 3-regular if and only if the corresponding subgraph in the original graph G is an induced cycle. The if-part is clear by the discussion of the previous section. Let us look at the induced subgraph $H[V(P_{2,1,3}) \cup V(P_{1,3,4}) \cup V(P_{3,4,5}) \cup V(P_{2,5,4}) \cup V(P_{1,2,5})]$ in the right graph H shown in Figure 3. Then, the induced subgraph includes the chord edge (γ_1, γ_2) and thus the degree of γ_1 and γ_4 is 4. The reason why the induced subgraph cannot be 3-regular comes from the fact that the cycle $\langle v_1, v_3, v_4, v_5, v_2, v_1 \rangle$ includes the chord edge (v_1, v_4) in the original graph G . The edges between γ -vertices are placed because there is an edge between their corresponding vertices in G . As a result, the assumption that $H[S^*]$ is an optimal solution, i.e., 3-regular, implies that the corresponding induced subgraph in the original graph G forms a cycle $\langle v_{i_1}, v_{i_2}, \dots, v_{i_j}, v_{i_1} \rangle$.

Since the number of vertices in each path gadget is $4(n^3 + 1)$, $OPT_1(G) \geq \frac{g(n)}{n^{1-\epsilon}}$ holds by the assumption $OPT_2(H) \geq 4(n^3 + 1) \cdot \frac{g(n)}{n^{1-\epsilon}}$. Therefore, the condition (C2) is also satisfied.

3.4 Reduction for $r \geq 4$

In this section, we give a brief sketch of the ideas to prove Corollary 1, i.e., the $O(n^{1/6 - \epsilon})$ inapproximability for r -MaxRICS for any fixed integer $r \geq 4$.

The proof is very similar to that of Theorem 3. The main difference between those proofs is the structure of each path gadget. See Figure 4, which shows the modified

path gadget. (i) We replace each of γ -vertices in Figure 1 with the complete graph K_{r-2} of $r-2$ vertices, and then connect one γ -vertex in H_i and one γ -vertex in H_j for $i \neq j$ by a similar manner to the reduction for the case $r = 3$. (ii) As for β -vertices, we prepare K_{r-2} of $r-2$ vertices, say, $\beta^1, \dots, \beta^{r-2}$, and two vertices, say, β^0 and β^{r-1} , such that each of the two vertices β^0 and β^{r-2} is adjacent to all the vertices in K_{r-2} . Then, all of the β -vertices are connected to the α -vertex similar to the reduction for $r = 3$. Since the reduction requires n^3 γ -vertices to connect all the pairs of H_i 's, which is independent of the value of r , the path gadget consists of $\lceil \frac{n^3}{r-2} \rceil$ subgraphs, say, $P_{j,i,k}^1$ through $P_{j,i,k}^{\lceil \frac{n^3}{r-2} \rceil}$. Further details are omitted here.

Acknowledgment

This work is partially supported by Grant-in-Aid for Scientific Research (KAKENHI), 22700019 and 23500020.

References

- [1] K. Cameron. Induced Matchings. *Discrete Applied Math*, 24, pp.97–102 (1989)
- [2] D.M. Cardoso, M. Kamiński, and V. Lozin. Maximum k -regular induced subgraphs. *J. Combinatorial Optimization*, 14(5), pp.455–463 (2007)
- [3] V. Bonifaci, U. Di Iorio, and L. Laura. The complexity of uniform Nash equilibria and related regular subgraph problems. *Theoretical Computer Science*, 401, pp.144–152 (2008)
- [4] S. Gupta, V. Raman, and S. Saurabh. Fast exponential algorithms for maximum r -regular induced subgraph problems. In *Proc. FSTTCS 2006*, pp.139–151 (2006)
- [5] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182 (1), pp.105–142, 1999.
- [6] V. Kann. Strong lower bounds on the approximability of some NPO PB-complete maximization problems. In *Proc. MFCS 1995*, pp.227–236 (1995)
- [7] C. Lund and M. Yannakakis. The approximation of maximum subgraph problems. In *Proc. ICALP 1993*, pp.40–51 (1993)
- [8] H. Moser and S. Sikdar. The parameterized complexity of the induced matching problem. *Discrete Applied Mathematics*, 157, pp.715–727 (2009)
- [9] Y. Orlovich, G. Finke, V. Gordon, and I. Zverovich. Approximability results for the maximum and minimum maximal induced matching problems. *Discrete Optimization*, 5, pp.584–593 (2008)
- [10] L.J. Stockmeyer and V.V. Vazirani. \mathcal{NP} -completeness of some generalizations of the maximum matching problem. *Information Processing Letters*, 15(1), pp.14–19 (1982)

Formal specification of semantics of UML 2.0 activity diagrams by using Graph Transformation Systems

Somayeh Azizi¹, Vahid Panahi²

Computer science department, Sama Technical and vocational, Training School, Islamic university, Arak Branch

Arak, Iran, s.azizi2011@gmail.com

² Arak, Iran ,v.p1386@gmail.com

Abstract - Graphical structures of various kinds (like graphs, diagrams, visual sentences) are very useful to describe complex structures and systems. The field of Graph transformation and Abstract State Machine has been widely used for modeling. Graphs are well suited to describe the underlying structures of models. They provide a good method to carry out the analysis and verification activities and use from the AGG (Attributed Graph Grammar) tools for design them. So the Abstract State Machine (ASM) is a modern computation model. ASM based tools are used in academia and industry, albeit on a modest scale. They allow you to give high-level operational semantics to computer artifacts and to write executable specifications of software and hardware at the desired abstraction level

Keywords: Graph Transformation, Abstract state machine, Activity Diagram, Semantics, Verification and Validation

I. INTRODUCTION

Recently modeling is significant department of activities that it available introduction a proper software for security user requirement. Select proper model is base of modeling. For complete understanding of systems and specific relation between different stages of them, they should model.

They are some approach for modeling such as:

- Unified modeling language(UML)
- Petri Nets

The token flow semantics of UML2.0 activity diagrams is formally defined using Abstract State Machines and Graph Transformation System. The state of the art in semantics for UML 2.0 activity diagrams covers three distinct approaches: mapping to Petri-nets, using graph transformation rules, or providing pseudo-code. ASM using pseudo- code and graph transformation system using graph transformation rules for determine semantics. A major goal of this paper is ability to determine the correctness behavior and formal semantics of UML2.0 activity diagram by Graph Transformation System and Abstract state machine Graph Transformation system (GTS)

- Process Algebra
- State Diagrams

The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software system. UML Activity diagram is a visual representation of any system's

activities and flows of data or control between activities. They describe the workflow behavior of a system. Activity diagrams are similar to state diagrams because activities are the state of doing something. The diagrams describe the state of activities by showing the sequence of activities performed. State diagram and activity diagrams both describe state transitions and share many of the same elements. The main reason to use activity diagrams is to model the workflows. Activity Diagrams are also useful for analyzing a use case by describing what actions need to take place and when they should occur. Currently, the UML semantics is informally defined in plain text and it is often unclear, ambiguous or it contains contradictory assertions. It is difficult to present the precise semantics which are taken as important in workflow system with the guide provided by OMG to the UML activity diagram. In this paper, the alternative approach of using Abstract State Machines and Graph Transformation System to formalize UML activity diagrams is presented. We propose the workflow modeling methodology by applying ASM and GTS semantics to the activity diagram. Through the exact definition to formal semantics based on ASM and GTS, it is possible to effectively model the workflow.

II. RELATED WORK

There is research done about formal specification semantics of uml2.0 activity diagram using different formal language. **Harel** defines state diagrams to model activities behavior in STATEMENT structured analysis notation.

In resumption **Eshuis** used this method to describe the behavior of UML1.5 activity diagram. He defines concept of strong fairness that based on model should not be indefinite loops. He also indicated modeled in two levels:

- Requirement level semantics: This level is easy for analysis.
- Implementation-level semantics: This level is difficult for analysis but it provides a real vision of system

Bogor used abstract state machine to describe UML2.0 activity diagrams. This method is based on event and per state is algebra. In ASM transition from one status to another status is done by rule *if- then*. **Hausmann** defines concept Dynamic Meta Modeling (DMM) using graph transformation systems. He developed the old graph rules by defining a new concept named rule invocation. In DMM there are two types of rules: big-step and small-step rules.

Big-step rules act as traditional rules but small-step rules should be invoked by big-step rules. Hausmann then defines

semantics for Activity diagrams using concept of DMM.

Engels use DMM and semantics defined by Haussmann for modeling and verification of workflows. For verification, he use GROOVE but as GROOVE does not support attributed typed graphs and rule invocation, they change the rules to be verifiable by GROOVE they check deadlock freeness and action reach ability properties on the modeled workflows. In contrast to this work, our approach has more flexibility to support user defined properties.

Furthermore, event and exception modeling can be supported by our approach. Additionally, the extension defined by Haussmann (small/big step rules and rule invocation) cannot be modeled directly in existing graph transformation tools; hence it is not so easy for designers to use this approach.

III. ACTIVITY DIAGRAM

UML2.0 Activity Diagram modeling behavior aspect of software systems particular Data Flow and control Flow.

Data Flow specific data transform from source path to destination and Control Flow specific existing paths for data transform an activity is operation sequence from start to end the system done and per activity can be transaction on data or process.

A. Kind Of Activity Nodes

1. **Action Node:** An action node is a atomic stage in activity such as math function that they can manipulate data
2. **Control Nodes:** A control node is Responsible for routing of tokens. They routing based on decision node, fork node and join node. The tokens are production and consumption.

B. Kind Of Control Nodes

1. **Initial Node:** An initial node Defines start of activity. This node has no input edge and it has only output edge.
2. **Decision Node:** A decision node has an input edge and more than one output edge. Input tokens are moving based on constraints on one of the output edges. The node will select different outputs based on a given Boolean expression.
3. **Merge Node:** A merge node have more than one input edges and only one output edge. This node each token to pass to side its output edge and they lead several workflow activities to a flow activity.
4. **Fork Node:** A fork node has only one input edge and more than output edge. Each input token is copied and passes through all the output edges. They are divided a flow to multiple simultaneous flow in an activity.
5. **Join Node:** A join node has more than input edge and only one output edge. If all incoming edges carry tokens In this case these nodes are used as a synchronization point.
6. **Final Node**
 - **Activity Final:** An activity final node has one or more Than one input edge. If the first token to reach the node then sequence of all tokens immediately across the activity can be used and enforcement activity will stop.
 - **Flow Final:** A flow final node has one or more than one input edge. This node uses each token entered

and Lead to a path is ending.

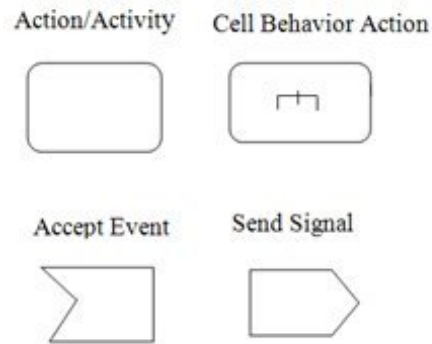


Figure 1 Kind Of Action Nodes

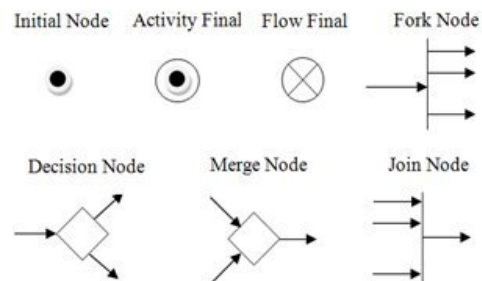


Figure 2 Kind Of Control Nodes

IV. GRAPH TRANSFORMATION

Graph transformation is applied for simulating the behavior of models. It consists of three set: (i) type graph,(ii) host graph,(iii) rules. Hence per Graph transformation represents formally with triple AGT (TG, HG, R).

- The type graph (Meta model) defines the abstract syntax of a modeling language. Formally, it can be represented by a type graph (TG). Nodes in it called classes. Per class have attributes and functions.
- A host graph (Instance Models) describes the system define in modeling language. In fact it is a well-formed instance of the Meta model.
- A graph transformation rule describes dynamic behavior of graph transformation.

Formally, A graph Transformation rule $p = (L,R,N)$ consists of: a graph L being the left hand side (LHS) of the rule a graph R being the right hand side (RHS) of the rule; a set of graphs N being the negative application conditions (NACs). The application of a graph transformation rule

transforms a graph G, the source graph, into a graph H, the target graph, by looking for an occurrence of L in G and then replacing that occurrence of L with R, resulting in H. The role of the NACs is that they can still prevent application of the rule when an occurrence of the LHS has been found, namely if there is an occurrence of some $N \in N$ in G that extends th the candidate occurrence of L. [1,3]

V. ABSTRACT STATE MACHINE

The Abstract State Machine (ASM) Project (formerly known as the Evolving Algebras Project) was started by Yuri Gurevich as an attempt to bridge the gap between

formal models of computation and practical specification methods.

A sequential ASM is defined as a set of transition rules of form: $\{ \text{If Condition then Updates} \}$, which transform first-order structures (the states of the machine), where the guard Condition, which has to be satisfied for a rule to be applicable, is a variable free first-order formula, and Updates is a finite set of function updates (containing only variable free terms) of form: $f(t_1, \dots, t_n) := t$. The execution of these rules is understood as updating, in the given state and in the indicated way, the value of the function f at the indicated parameters, leaving everything else unchanged. (This proviso avoids the frame problem of declarative approaches.) In every state, all the rules which are applicable are simultaneously applied (if the updates are consistent) to produce the next state. If desired or useful, declarative features can be built into an ASM by integrity constraints and by assumptions on the state, on the environment, and on the applicability of rules. The ASM is formal and can therefore serve as a foundation for the implementation of tools. Finally, it helps to ensure that the specified behavior meets the intuition of the modeler. Abstract State Machine has a main rule that computation transition and uses from it for determine token flow semantics. The semantics of activity diagram determine base token flow. When token

available in initial, object and action nodes then calling transition of rule. if guards evaluation true then token move toward destination nodes. [2]

VI. WORKFLOW MODELING

The modeling of workflow should treat of each element of the activity diagram was determined uses each node in activity diagram is equivalent to a class in this method. Each class consists of three parts, the class name, attributes and functions. Class name is equivalent to the node name and attributes of each class determine according to class name and action it. In the proposed method of functions are obtained from pseudo-code of state machine abstraction and attributes are Combination of characteristics in the graph transformation system and abstract state machine. In UML2.0 semantic specification is based tokens. These diagrams explain each activity and its interactions in detail, including how it is triggered, what resources are needed and what deliverables will be created. This knowledge will enable you to discover and address any unstated requirements prior to finalizing the project plan. These workflow diagrams are key to effective analysis and communications. to model workflows we consider these parts of Activity diagrams: *Init* node, *Final* node, *Action* node, *Fork* node, *Join* node, *Merge*

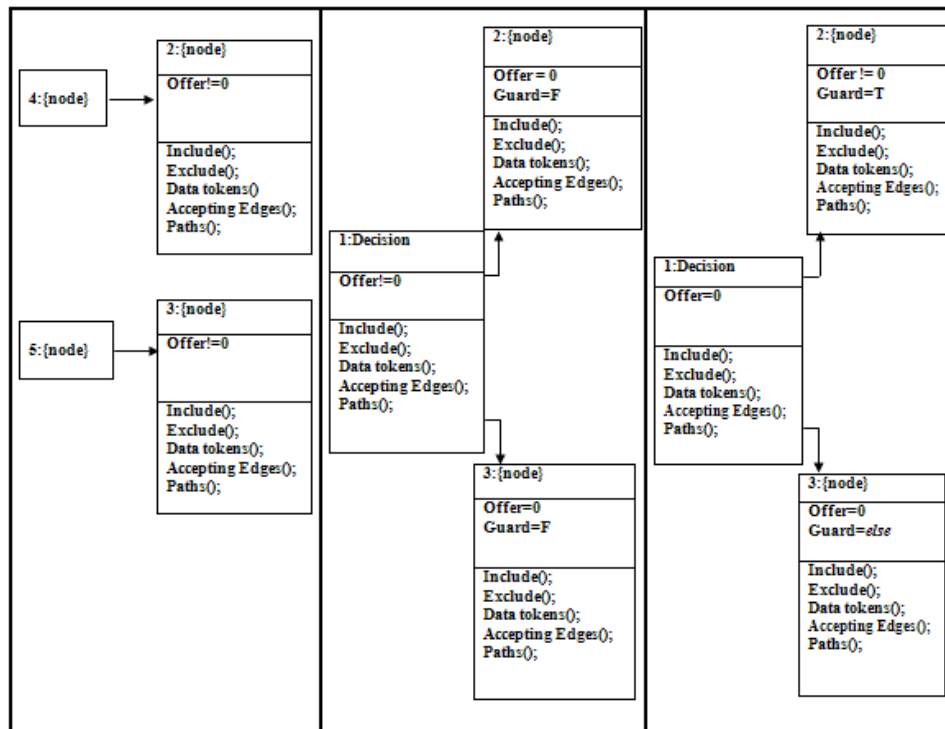


Figure 3 rules showing the semantics of Decision node

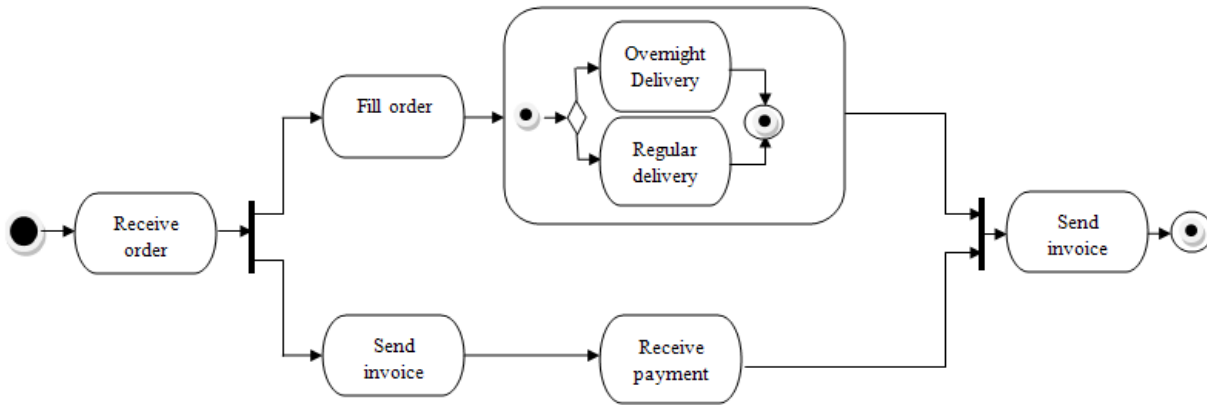


Figure 4 A sample activity diagram

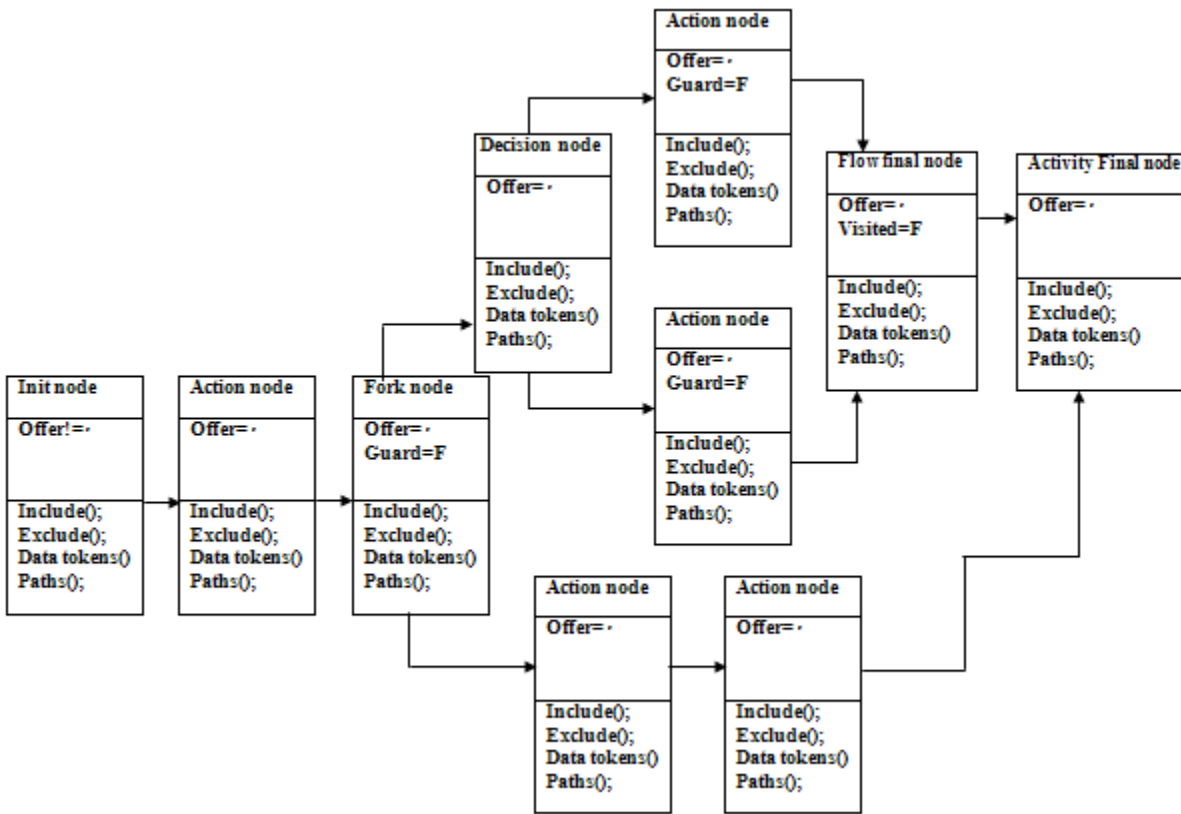


Figure 5 The sample activity diagram in fig3 as host graph with ASM and GTS

node, *Decision node* We will use token-flow semantics in our graphs. In this paper we show determine semantics of decision node with ASM and GTS.

Decision node: pseudo-code of decision node

forall I with $1 < I < |accepting\ Edges|$ *do* $t(I) := new(Token\ Offer) seq$

forall I with $1 < I < |accepting\ Edges|$ *do* $t(i).offered$

Token := t.offered Token

$t(i).paths := \{p \dashv\vdash element\ At(accepting\ Edges, i) \mid p \in t.$

$path \}$

$t(i).exclude := t.exclude \cup \{t(j) \mid 1 \leq j \leq |accepting\ Edges| \wedge I \neq j\}$
 $t(i).include := t.include \cup \{t\}$ *add* $t(i)$ *to* *offers* (*element* $At(accepting\ Edges, I)$)

In Figure3:

- The NAC of this rule states that both of the following nodes should not be *Join* or *Merge* nodes, because these nodes have several output edge, but

decision node chooses the output of a edge between its output edges. This restriction is implemented by the function Exclude. This function cause the tokens do, not clash together. We have different rules for cases that the following nodes are *Join* or *Merge* nodes.

- The LHS shows the precondition of this rule. If *Decision* node has the token and both of the following nodes have not any token, then this rule can be applied on the model and token will be routed to only one of the following nodes. Thus offer token entered into decision node and the value of the attribute inequality is false. Yet offers token not enter within nodes that they are outputs decision nodes. Guard adjective in one of the output nodes is **True** and order output node Labeled with **else**.

As it is shown, the RHS says that offer token must be routed to only one of the following Nodes. Thus Is offer token out of the decision node and based on current conditions enter one of the output nodes.

We will use Activity diagram of figure 3 as a running example for the rest of this paper. It describes the processing of orders in a company.

Figure4 represent a workflow modeled with activity diagram .this activity can be image as host graph in graph transformation system. For each node in activity diagram, there is a node (class) in host graph .Class diagrams have three parts: (i) class name, (ii) attributes, (iii) functions. Class name is name of node and one of attribute for node is token offer attribute. Token offers computation in initial node and this attribute values in initial node is not null and other nodes is null. By moving token in path and inter it at a node, calling functions and computation token offer it. When token arrives final node, the token offer attribute for preview nodes is null and for final node is not null. Therefore when token flow will be terminated that token arrives final node or there is not any path for token to be routed.

VII. VERIFICATION AND VALIDATION

The use of formal verification methods is essential in the design process of dependable computer controlled systems. The efficiency of applying these formal methods will be highly increased if the underlying mathematical background is hidden from the designer in such an integrated system effective techniques are needed to transform the system model to different sort of mathematical models supporting the assessment of system characteristics.

To verification of activities must be design an approach. The theory and application of visual languages is also based on the strong paradigm of graph transformation. Therefore For analyze designed activities, graph transformation system (type graph, host graph, rules) and properties gets as input of verification approach. Graph transformation system must be design in AGG; also properties define by special rules. If designers are expert in graph transformation, they can directly to model workflows by using graph transformation system. In this approach designers do not need to learn of formal method. In other case, they can model workflows by UML2.0 activity diagram, then by transformer, designed activities in UML transformed to graph transformation.

In this approach verification done automatically and designers do not need to define of rules for verification. VIATRA (Visual Automated Transformations) is a model transformation framework developed mainly for the formal dependability analysis of UML models. In VIATRA, Meta modeling is conceived specially: the instantiation is based on mathematical formalisms and called Visual Precise Meta modeling. The attribute transformation is performed by abstract state machine statements, and there is built-in support for attributes of basic Java types. The model constraints can be expressed by graph patterns with arbitrary levels of negation. The rule constraints are also specified by graph patterns. VIATRA uses abstract state machines (ASM) to define the control flow of the system.[1,3]

VIII. CONCLUSION

This paper proposes a formal approach base compose Graph Transformation Systems (GTS) and Abstract State Machine (ASM). This approach determines behavior of UML2.0 activity diagrams base token flow. Rules of ASM that define with *pseudo-code*, display by fundamental element of GTS (LHS, RHS, NAC).

REFERENCES

- [1] Vahid Rafe , Adel T. Rahmani, "Formal Analysis of Workflows Using UML 2.0 Activities and Graph Transformation Systems" ICTAC 2008, LNCS 5160, pages. 305–318, 2008.
- [2] Stefan Sarstedt, Walter Guttmann, "An ASM Semantics of Token Flow in UML2.0 Activity Diagrams", 2008.
- [3] Laszlo Lengyel, Tihamer Levendovszky, Gergely Mezei and Hassan Charaf," Model Transformation with a Visual Control Flow Language", International Journal of Computer Science Volume1 Number 1, 2005.

SESSION

PROGRAMMING ISSUES AND TOOLS + OS + CONCURRENCY + MODEL CHECKING

Chair(s)

TBA

Towards a Multi-Formalism Model Checker Based on SDES Description

Behrang Mehrparvar¹, Mohammad Abdollahi Azgomi¹

¹ School of Computer Engineering, Iran University of Science and Technology, Tehran, Tehran, Iran

Abstract - *The role of critical concurrent systems is increasing in today ICT systems. Formal modeling techniques and tools provide adequate and comprehensive solutions for verification of these systems. High-level modeling formalisms support system designers to express systems in an abstract manner, using multiple formalisms to model various aspects of complex systems, eases the process of system modeling. A multi-formalism model checker is intended to provide various methods of model checking regardless of the formalism defining the model. The aim has been to use SDES description as the base of a multi-formalism model checking framework. By translation from various high-level models, the interface model is used for generating low-level state spaces. In this paper, a new multi-formalism model checking approach is introduced based on SDES description as an interface formalism. Furthermore, an architecture for a multi-formalism model checking component integrated in PDETool, an existing modeling tool, is provided.*

Keywords: SDES description; Model checking; Multi-formalism; Discrete event systems; State space generation.

1 Introduction

Model checking as an instance of system verification was independently introduced by Clarke and Emerson [1] and Queille and Sifakis [2]. It provides an automated efficient search in the state graph representing all reachable states of the model of a system. The algorithmic search is intended to indicate whether a specific temporal property, P , is satisfied in a model structure, M , or not. Moreover, it can provide a counterexample for further debugging of the system [3]. Comparing to other verification techniques, model checking is a systematic, adequate and comprehensive method to verify critical concurrent systems.

As mentioned before, the main inputs of model checking approach are system model and property specification. Most model checking tools support low-level formalisms like transition systems, discrete-time and continuous-time Markov chains as models of systems. Others, such as ALPINA [4], UPPAAL [5] and SMART [6], support high-level modeling formalisms. These formalisms are more adequate for modeling large scale systems in an abstract manner. In such cases, the model checker initially applies a state generation algorithm in

order to transform the high-level formalism into the low-level transition system or Markov chain.

Another primary consideration that should be taken into account in choosing an appropriate model checking tool is the type of logics provided to specify the properties being checked within the modeled system. From this point of view, various model checking tools provide nondeterministic model checking methods based on LTL [7] and CTL [1] temporal logics. The tools, such as PRISM [8] [9], provide probabilistic and stochastic model checking based on PCTL [10] and CSL [11]. MRMC [12] also supports model checking for reward models [13].

Additionally, relying on a single high-level formalism is not either illustrative enough to express various aspects of large and complex systems. A multi-formalism modeling tool can support the designers by providing multiple high-level formalisms to model the system. In such cases, the developer can usually extend the modeling capability by defining new modeling languages based on an interface formalism. For instance, SMART provides a specific language for modeling systems as the core language. The user can specify extended models by defining formalism-specific types and functions [14]. Also, in Möbius [15], an abstract functional interface (AFI) is provided in the form of C++ abstract classes. New extended models can implement basic variables and actions defined in the AFI by use of class inheritance [16].

Although multi-formalism multi-solution tools provide modeling, simulation and evaluation solutions for multiple formalisms, they rarely support various methods of model checking such as nondeterministic, probabilistic and stochastic methods in an integrated framework. Having considered the basic steps in model checking process as mentioned in [17], a multi-formalism model checking tool is assumed to provide four basic features. These features include the support of multiple formalisms to provide high-level models of systems, integrated simulation engine regardless of the formalism representing the model, multiple property specification logics and multiple methods of model checking.

Accordingly, by applying a unified formalism as the core interface modeling language of an integrated tool, a multi-formalism model checker can be developed providing different methods of model checking for a wider range of applications. Introduced by Zimmermann [18], SDES description provides

a unified abstract formalism for stochastic discrete-event systems. Popular model classes, such as different extensions of Petri nets, queuing networks and timed automata, can be translated into SDES description.

Furthermore, having applied SDES description as the core formalism, PDETool [19] provides an integrated tool to model, simulate and evaluate discrete-event systems. Also, some important extensions of Petri nets, such as SPNs, GSPNs, SANs and CSANs are implemented based on SDES description [20]. Therefore, with respect to SDES as the core interface formalism, a multi-formalism model checking component can be integrated into PDETool to support nondeterministic, probabilistic and stochastic model checking. In this paper we introduce extension of PDETool with multi-formalism model checking capability. For this purpose, a general state space structure is defined and generated from SDES description which can be used as the low-level input model for various model checking methods.

The remainder of the paper is organized as follows. SDES description is described in section 2. The proposed approach is introduced in section 3, by introducing property specification approach, state space structure, generation approach and finally the model checking methods. Section 4 provides an example to illustrate the proposed approach. In section 5, an architecture for multi-formalism model checker integrated in PDETool is proposed. Finally section 6 concludes the paper.

2 SDES Description

Different classes of stochastic discrete-event systems share common characteristics. SDES description [18] is a unified abstract modeling formalism that represents these common characteristics.

DEFINITION 1. (SDES Description). SDES description is defined as a tuple $SDES = (SV^*, A^*, S^*, RV^*)$, where:

- SV^* is a finite set of state variables,
- A^* is the finite set of actions,
- S^* is the sort function defining the range of state variables or action variables. The sort of a variable specifies the values that might be assigned to it,
- RV^* is defined as the set of reward variables corresponding the quantitative evaluation of the model. Every element in RV^* specifies one reward variable and maps the stochastic process to a real value.

By associating values to each state variable allowed by the sort function, Σ is defined as all theoretically possible states of a certain SDES model:

$$\Sigma = \prod_{sv \in SV^*} S^*(sv) \quad (1)$$

A state variable sv_i usually corresponds to a passive element of the SDES, like a place of a Petri net or a queue of a

queuing model. Each state variable has the following attributes:

$$sv = (Cond^*, Val_0^*) \quad (2)$$

where Val_0^* is a function representing the initial value of each state variable and $Cond^*$ indicates whether a state variable is allowed in a specific model state or not.

$$Cond^*: SV^* \times \Sigma \rightarrow \mathbb{B} \quad (3)$$

An action $a \in A^*$ of SDES describes possible state changes of the modeled system. It is composed of the following attributes:

$$a = (Pri^*, Deg^*, Vars^*, Ena^*, Delay^*, Weight^*, Exec^*) \quad (4)$$

Each item is defined as follows:

- Pri^* associates a global priority to every action,
- The enabling degree Deg^* of an action specifies the number of activities that are permitted to run concurrently in any state,
- The action variables $Vars^*$ define a model-dependent set of variables $Vars^*(a)$ of an action a with individual sorts,
- The value of the Boolean enabling function Ena^* of an action for a state returns if it is enabled or not,
- $Delay^*$ describes the time that must elapse while an action is enabled in an activity until it finishes,
- The $Weight^*$ of an action is a real number that defines the probability to select it for execution in relation to other weights,
- $Exec^*$ defines the state changes that occur as a result of an action execution. So actions change the state and $Exec^*$ is a function that associates a destination state to a source state for each action.

A reward variable $rvar^* \in RV^*$ is defined as a tuple composed of the following attributes:

$$rvar^* = (rrate^*, rimp^*, rint^*, ravg^*) \quad (5)$$

where the items respectively denote state rewards, impulse rewards, the observation interval and a variable that determines whether the resulting measure should be computed as an average over time or as accumulated.

3 The Proposed Approach

The basic model checking process consists of four main steps of system modeling, property formalization, model checking, and counterexample simulation. However, a multi-formalism model checker is intended to support the following features:

Feature 1: Multiple formalisms to provide high-level models of systems, e.g. SPNs, SANs and PEPA.

Feature 2: Integrated simulation engine regardless of the formalism representing the model.

Feature 3: Multiple property specification logics, e.g. CTL, PCTL and CSL.

Feature 4: Multiple methods of model checking, e.g. nondeterministic, probabilistic and stochastic.

The proposed approach applies three adaptations in the basic model checking process in order to gain the above features. Figure 1 illustrates the process of multi-formalism model checking.

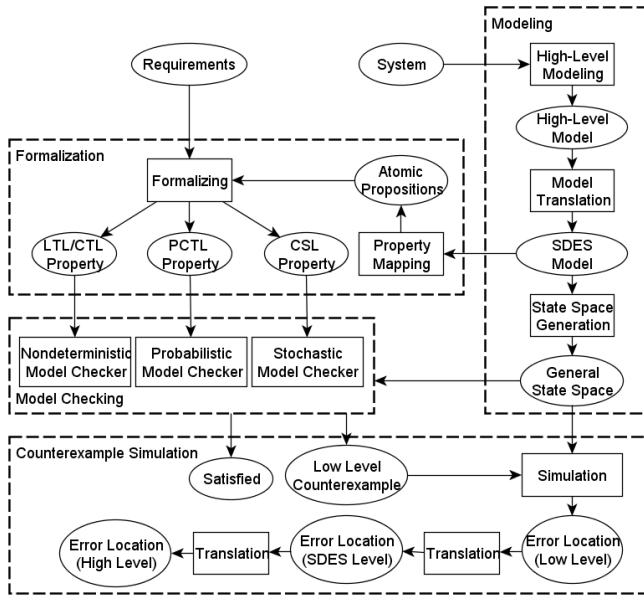


Figure 1. The proposed multi-formalism model checking approach

In order to support multiple property specifications, property mapping functions are defined to enhance the model checker in order to derive atomic propositions. Also, the modeling step itself is broken down into three separate phases to support multiple formalisms as input. After modeling the system in high-level formalisms, these models are later translated into SDES description. Afterwards, a state space generation algorithm generates a low-level general state space (GS), which is suited for various model checking algorithms.

Using a general state space as model structure, various methods of model checking can be applied on multiple high-level formalisms. In case of dissatisfaction of the model checking algorithm, a counterexample is provided in terms of low-level formalism which can be later retranslated into SDES description. Using SimGine [19] interface integrated in PDETool, the counterexample can be simulated and also animated in the respective high-level formalism.

3.1 Property Specification

Properties in temporal logic are specified based on atomic propositions. Atomic propositions are simple known facts that formalize basic temporal characteristics of the system. Regardless of the type of logic representing the specification, the model checking method should be able to

check properties in a labeled model. A labeled model is a model in which all states are labeled by items of a set containing atomic propositions. While the existence of an atomic proposition in a set is generally represented by Boolean values, the states generated from SDES static model are based on the values of not necessarily Boolean state variables. As a result, each state variable cannot be directly considered as an atomic proposition. Therefore, a set of mapping functions, Map^* , is defined as follows to derive each atomic proposition, ap_i , regarding the values of state variables:

$$Map_{ap_i}: SV^* \rightarrow \mathbb{B} \quad (6)$$

Each atomic proposition and the corresponding mapping function are directly derived from the property being checked. As an example, consider the following temporal property in CTL:

$$AG((x + y < 2) \text{ or } (z > 3)) \quad (7)$$

where x , y and z are state variables in SDES static model. Consequently, two atomic propositions, ap_1 and ap_2 , are either automatically or manually derived from the property with the following mapping functions:

$$Map_{ap_1}(x, y, z) = (x + y < 2) \quad (8)$$

$$Map_{ap_2}(x, y, z) = (z > 3) \quad (9)$$

Finally the model checking algorithm checks the provided specification, regarding the set of derived mapping functions. The resulted property is mapped as follows:

$$AG(ap_1 \text{ or } ap_2) \quad (10)$$

By defining the mapping functions considering the input property specifications, the labeling of each state is simultaneously applied while the state space is generated.

3.2 State Space Generation

As mentioned before, in order to check different types of properties on models, the state space should be generated in a suitable form of structure. For nondeterministic, probabilistic and stochastic model checking, labeled transition systems, discrete-time and continuous-time Markov chains are respectively used by model checking tools. However, in order to have a multi-formalism model checker, a general structure should be defined to cover all the items provided in the basic formalisms mentioned above.

DEFINITION 2. (General State Space). A general state space is a tuple $GS = (S, Act, trans, I, AP, L, R, P, \rho, \ell)$, where:

- S is a set of states,
- Act is a set of actions,
- $trans \subseteq S \times Act \times S$ is a set of transitions, with $(s_1, a, s_2) \in trans$, $a \in Act$ and $s_1, s_2 \in S$,
- $I \subseteq S$ is a set of initial states,
- AP is a set of atomic propositions,

- $L: S \rightarrow 2^{AP}$ is a labeling function which assigns to each state $s \in S$ the set $L(s)$ of atomic propositions that are valid in the state,
- $R: S \times S \rightarrow \mathbb{R}^{\geq 0}$ is the transition rate matrix,
- $P: S \times S \rightarrow [0, 1]$ is the transition probability matrix where $\sum_{s' \in S} P(s, s') = 1$ for all $s \in S$,
- $\underline{\rho}: S \rightarrow \mathbb{R}_{\geq 0}$ is a state reward function which defines the reward acquired in a state,
- $l: S \times S \rightarrow \mathbb{R}_{\geq 0}$ is a transition reward function which defines the reward acquired each time a transition is fired.

Considering the items defined in Definition 2, the GS structure is inclusively considered as a super class for transition system, DTMC, CTMC and MRM. Consequently, it is used as the structure for representing the state space capable for nondeterministic, probabilistic and stochastic model checking.

In order to use GS as the core state space structure for multi-formalism model checking, the items mentioned in Definition 2 should be directly generated from SDES description as follows:

- All states in the SDES dynamic model are members of the state set in GS :

$$\forall \sigma \in \Sigma: \sigma \in S \quad (11)$$

- All actions in the SDES model are members of the actions set in GS :

$$\forall av \in AV: av \in Act \quad (12)$$

- Any transition in GS is resulted by firing an SDES execution function:

$$\begin{aligned} \forall \sigma_1, \sigma_2 \in \Sigma, av \in AV: (\sigma_1, av, \sigma_2) \in Trans \\ | Exec(av, \sigma_1) = \sigma_2 \end{aligned} \quad (13)$$

- The initial states in GS are generated by setting the values of each state variable in SDES static model to its initial value:

$$\forall \sigma \in \Sigma: \sigma \in I: \sigma \in \prod_{sv_i \in SV} Val_0(sv_i) \quad (14)$$

- Regarding each mapping function derived from the given property, an atomic proposition is defined as follows:

$$\forall map_i \in Map: i \in AP \quad (15)$$

- The result of the labeling function for each state on a specific atomic proposition is derived from its respective mapping function:

$$\forall \sigma \in \Sigma, ap \in AP: ap \in L(\sigma) | map_{ap}(\sigma) = true \quad (16)$$

- The probability matrix is derived from the weight of executed actions:

$$\begin{aligned} \forall \sigma, \sigma' \in \Sigma: P(\sigma, \sigma') = p | \exists av \in AV, \\ Exec(av, \sigma) = \sigma', Weight(av) = p \end{aligned} \quad (17)$$

- The transition rate matrix is derived from the delay of executed actions:

$$\begin{aligned} \forall \sigma, \sigma' \in \Sigma: R(\sigma, \sigma') = p | \exists av \in AV, \\ Exec(av, \sigma) = \sigma', Delay(av) = p \end{aligned} \quad (18)$$

- State rewards can be directly generated from $rrate^*$:

$$\forall \sigma \in \Sigma: \underline{\rho}_i(\sigma) = r | rrate_i(\sigma) = r \quad (19)$$

- Transition rewards can be directly generated from $rimp^*$ one executed action variants AV^* :

$$\begin{aligned} \forall av \in AV^*, \forall \sigma_1, \sigma_2 \in \Sigma: l_i(\sigma_1, \sigma_2) = r | \\ Exec(av, \sigma_1) = \sigma_2, rimp_i(av) = r \end{aligned} \quad (20)$$

According to the mappings mentioned above, the state generation algorithm for SDES description is defined as in Figure 2.

```

Algorithm#1: State Space Generation
Input: SDES description
Output: General State Space GS

For all svi in SV* do Ii = Val0(svi)
  S ← I; N ← I
  For all api in AP
    If Mapapi(I) Then
      L(I) ← L(I) ∪ api
While N != {}
Begin
  NS ← N; N ← {}
  For all Oi in NS
    For all avi in AV*
      If Ena(avi, Oi) AND Count(avi.a) < Deg(avi.a)
        Then
          Begin
            σ = Exec(avi, Oi);
            If (S ← S ∪ σ) Then
              Begin
                Act ← Act ∪ avi
                Trans ← trans ∪ (Oi, avi, σ)
                N ← N ∪ σ
                P(Oi, σ) = Weight(avi)
                R(Oi, σ) = Delay(avi)
                ρ(σ) = rrate(σ)
                l(oi, σ) = rimp(avi)
                For all api in AP
                  If Mapapi(σ) Then
                    L(σ) ← L(σ) ∪ api
              End If
            End For
          End While

```

Figure 2. State space generation algorithm

3.3 The Model Checking Method

A multi-formalism model checker should provide three basic methods of model checking including nondeterministic, probabilistic and stochastic methods. Each method is applied on specific logics and state space structures.

Nondeterministic model checking algorithm checks the properties specified in LTL [7] or CTL [1] logics in labeled transition systems. In LTL model checking, a Büchi automata A is constructed from the LTL property and by constructing a product transition system of $TS \otimes A$ the algorithm tries to disprove the satisfaction of the property [21]. However, in CTL model checking, the algorithm recursively calculates the satisfaction set of states for the property [1]. In probabilistic

and stochastic model checking, the system is modeled in labeled discrete-time or continuous-time Markov chain and the algorithm is able to check PCTL [10] or CSL [11] properties on the defined model in a recursive manner [23]. However, model checking of reward models can also be applied on discrete-time or continuous-time Markov reward models. In this case, PRCTL [22] or CSRL [11] logic is used to specify properties [22]. MRMC [12] model checker provides probabilistic and stochastic model checking on DMRMs and CMRRMs [13].

As general state space is a super class of labeled transition system, DTMC, CTMC, and MRM, it can be used as the low-level model structure in a multi-formalism model checker that provides all nondeterministic, probabilistic and stochastic model checking methods mentioned above. However, as some of the elements in general state space are not used in all methods of model checking, Table I illustrates how each type of action is treated in various methods of model checking.

Table I . The view of types of actions in different methods of model checking

Method of Model Checking	Type of Action		
	Nondeterministic	Probabilistic	Stochastic
Nondeterministic	ND	ND	ND
Probabilistic	ND	PB	ND
Stochastic	ND	ND	SC

4 An Illustrative Example

In this section an example for multi-formalism model checking on SDES description is provided. In this example, the SDES model is translated from an SPN high-level model. This model showed in Figure 3 represents a system containing two processors, two memory blocks and one common bus.

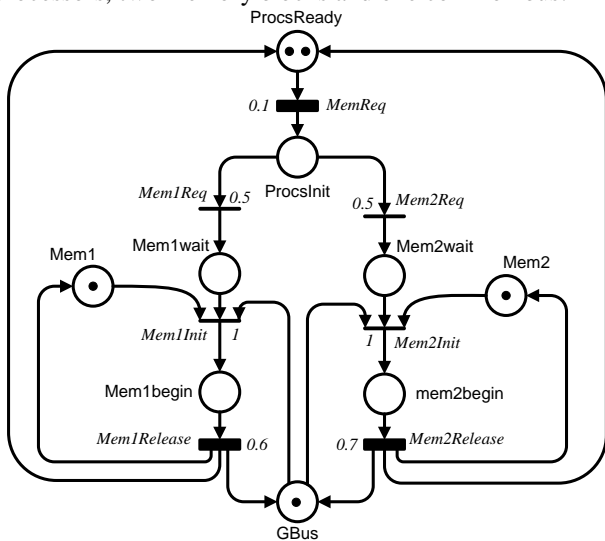


Figure 3 . SPN representation of a system with two processors and two memory blocks

As mentioned before, the main inputs of a model checker are the state space and the property specifications. Section 4.1 defines the SDES description of the above example and also the regarding generated state space. Section 4.2 provides the property specifications and the final results.

4.1 Modeling the system

The SPN model is firstly transformed into SDES description using the approach introduced in [18]. The corresponding SDES model of the above example is shown in Figure 4.

State variables:

$$SV = \{ProcsReady, ProcsInit, Mem1wait, Mem2Wait, Mem1, Mem2, Mem1begin, Mem2begin, GBus\}$$

Actions:

$$A = \{MemReq, Mem1Req, Mem2Req, Mem1Init, Mem2Init, Mem1Release, Mem2Release\}$$

Sort functions: $S(.) = \{0, 1, 2\}$

Condition functions: $Cond(.,.) = True$

Initial values:

$$Val_0(ProcsReady) = 2 \quad Val_0(ProcsInit) = 0$$

$$Val_0(Mem2wait) = 0 \quad Val_0(Mem1) = 1$$

$$Val_0(Mem1begin) = 0 \quad Val_0(Mem2begin) = 0$$

$$Val_0(Mem1wait) = 0 \quad Val_0(Mem2) = 1$$

$$Val_0(GBus) = 1$$

Degrees: $Deg(.) = 1$

Action variables: $Vars(.) = \{ \}$

Delays:

$$Delay(MemReq) = 0.1 \quad Delay(Mem1Req) = 0$$

$$Delay(Mem1Init) = 0 \quad Delay(Mem2Init) = 0$$

$$Delay(Mem2Release) = 0.7 \quad Delay(Mem2Req) = 0$$

$$Delay(Mem1Release) = 0.6$$

Weights:

$$Weight(MemReq) = 1 \quad Weight(Mem1Req) = 0.5$$

$$Weight(Mem1Init) = 1 \quad Weight(Mem2Init) = 1$$

$$Weight(Mem2Release) = 1 \quad Weight(Mem2Req) = 0.5$$

$$Weight(Mem1Release) = 1$$

All possible states: $\Sigma = \{0,1,2\}^3$

Figure 4. SDES description of a system with two processors and two memory blocks

Considering the algorithm of Figure 2, the general state space, GS, regarding the above SDES description is generated as shown in Figure 5.

4.2 Property Specification and the Results

For the model of the example in Figure 3, the following property specifications are checked on the model:

1. The system will reach to a state where both memory blocks are accessed.
2. The system will reach to a state with 50% probability where a processor would wait for memory block 1 in 5

next steps while it is waiting for the common bus at the current time.

3. The system will reach to a state where a processor would be available in 4 seconds.

Initial states:	$I = \{1\}$
States:	$S = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18\}$
Actions:	$Act = \{MemReq, Mem1Req, Mem2Req, Mem1Init, Mem2Init, Mem1Release, Mem2Release\}$
Probabilities:	$\forall (\sigma_1, trans, \sigma_2) \in Trans:$
	$P(\sigma_1, \sigma_2) = \begin{cases} 0.5, & trans \in \{Mem1Req, Mem2Req\} \\ 1, & else \end{cases}$
Rates:	$\forall (\sigma_1, trans, \sigma_2) \in Trans: R(\sigma_1, \sigma_2):$
	$f(x) = \begin{cases} 0.1, & trans = MemReq \\ 0.6, & trans = Mem1Acc \\ 0.7, & trans = Mem2Acc \\ 1, & else \end{cases}$
State rewards:	$\rho(.) = 0$
Transition rewards:	$\ell(.,.) = 0$

Figure 5. GS representation of a system with two processors and two memory blocks

According to section 3.1, the set of atomic propositions and the regarding mapping functions for the properties are defined as follows:

$$AP = \{mm, m, b, c\} \quad (21)$$

$$Map_{mm}(\cdot) = (Mem1begin > 0 \text{ and } Mem2begin > 0) \quad (22)$$

$$Map_m(\cdot) = (Mem1 = 0 \text{ and } Mem1wait > 0) \quad (23)$$

$$Map_b(\cdot) = (CBus = 0) \quad (24)$$

$$Map_c(\cdot) = (ProcsReady > 0) \quad (25)$$

Consequently, properties are respectively specified in CTL, PCTL and CSL logics as follows:

1. $p_1: EF(mm)$ (26)

2. $p_2: P_{>0.5}[m U^{\leq 5} b]$ (27)

3. $p_3: P_{>0.0}[true U^{\leq 4} c]$ (28)

Finally, the system model and the property specification are provided to the multi-formalism model checker. Using the approach introduced in 3.3, the model checker applies specific model checking methods on the unique GS regarding the logic representing the properties. Therefore, nondeterministic, probabilistic and stochastic model checking algorithms are respectively applied on p_1 , p_2 and p_3 in order to find the satisfaction sets of states. Table II shows the result using MRMC model checker.

Table II . Model checking results for the example

Model checking method	Property	Satisfaction set
Nondeterministic	p_1	$\{\}$
Probabilistic	p_2	$\{6, 8, 9, 10, 14, 16, 17, 18\}$
Stochastic	p_3	$\{1, 2, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18\}$

5 The Proposed Architecture

In order to provide a multi-formalism model checker, the tool must be integrated into a framework providing multi-formalism modeling and simulation engine. On the other hand, a high-level interface modeling formalism as the core structure eases the collaboration of these components. The architecture of PDETool [19] contains Model Editor and Simulation Engine.

Having considered the multi-formalism model checking approach proposed in Figure 1, the architecture of the multi-formalism model checker integrated in PDETool is illustrated in Figure 6.

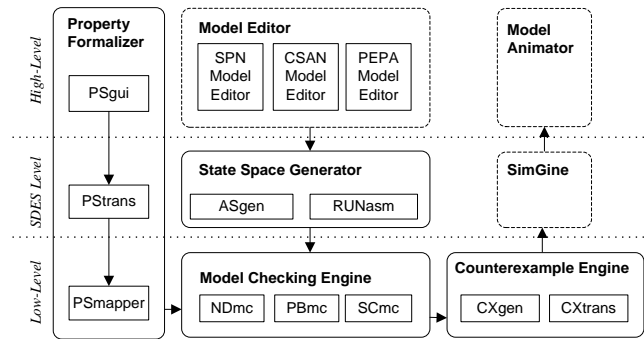


Figure 6 . The proposed architecture for a multi-formalism model checker integrated into PDETool

The model checking component integrated in PDETool framework is composed of four basic components arranged in three layers. Each component is composed of the basic components with specific functionalities.

The *property formalizer* component which is responsible for defining system requirements and is composed of the following components:

- *PSgui*: Property specification graphical user interface provides an environment to specify temporal logic properties based on the regarding high-level formalisms.
- *PStrans*: Property specification translator is responsible for translating properties to SDES formalism.
- *PSmapper*: Atomic propositions are derived through *PSmapper* and the mapping functions are defined according to the approach introduced in 3.1.

- The state space generation component which is responsible for providing low-level state space structures for the models includes the following components:
- *ASgen*: Assembly generator generates executable runtime assemblies (*RUNasm*) based on the SDES static models.
- *RUNasm*: Runtime assemblies are executable assemblies that form SDES dynamic models. Running the *RUNasm* assembly using the approach introduced in 3.2, the low-level general state space of the model is generated.

The *model checking component* implements the multi-formalism model checking approach introduced in 3.3 using the following components:

- *NDmc*: This component applies nondeterministic model checking methods on LTL and CTL temporal logics.
- *PBmc*: Probabilistic model checking is applied using *PBmc* for PCTL and PRCTL temporal logics.
- *SCmc*: Properties formalized in CSL and CSRL temporal logics are checked in stochastic model checking component.

While the model checking engine provides the satisfaction sets of states, the counter example engine is responsible for providing counterexamples using the following components:

- *CXgen*: *Counterexample* generator is responsible for providing the states not satisfying the properties.
- *CXtrans*: Counterexample translator gets the counterexamples provided by *CXgen* and translates them to sequences of SDES actions that lead to the non-satisfying states.

The SDES level counterexamples provided by *CXtrans* are used in SimGine in order to simulate the execution path that did not satisfy the specified property. Later, the animator component shows the counterexample in the high-level model by re-translating it from SDES level.

6 Conclusions

In this paper, we proposed a multi-formalism model checking approach that provides four basic features in PDETool framework. These features consist of supporting multiple high-level formalisms for modeling, multiple property specification logics, multiple methods of model checking and providing a simulator engine independent of the model class.

Firstly, by integrating the model checking tool in PDETool, multiple model classes such as SPNs, CSANs and PEPAs can be modeled and translated into SDES description. Secondly, by defining a set of mapping functions, atomic propositions can be derived from SDES variables. Various property specifications can be defined in LTL, CTL, PCTL and CSL, based on defined atomic propositions. Thirdly, by defining a general state space structure as a super-class of labeled transition systems, DTMCs, CTMCs, DMRMs or CMRMs,

nondeterministic, probabilistic and stochastic model checking techniques can be applied, respectively. And finally, by using SDES as the base interface formalism in PDETool, the simulation engine, SimGine, can be used in order to determine the counterexamples regardless of the high-level formalism describing the model.

7 References

- [1] E. M. Clarke and E. A. Emerson. "Design and synthesis of synchronization skeletons using branching time temporal logic" in *Logic of Programs*, vol. 131 of LNCS, pp. 52–71, Springer, 1981.
- [2] J. P. Queille and J. Sifakis. "Specification and verification of concurrent systems in CESAR". In *Proc. 5th International Symposium on Programming*, 1982, pp. 337–351.
- [3] E. Emerson. "Meanings of Model Checking". In *Concurrency, Compositionality, and Correctness, Design and synthesis of synchronization skeletons using branching time temporal logic*, vol. 5930 of LNCS, pp. 237–249. Springer, 2010.
- [4] "AlPiNA: an Algebraic Petri Net Analyzer". Internet: www.alpina.unige.ch, [Sep. 18, 2010].
- [5] "UPPAAL". Internet: www.uppaal.com, [Aug. 24, 2010].
- [6] "SMART project". Internet: www.cs.ucr.edu/~ciardo/SMART, [Oct. 10, 2010].
- [7] A. Pnueli. "The temporal logic of programs". In *Proc. 18th IEEE Symposium on Foundations of Computer Science (FOCS)*, 1977, pp. 46–67.
- [8] "PRISM - Probabilistic Symbolic Model Checker". Internet: www.prismmodelchecker.org, [Aug. 26 2010].
- [9] M. Kwiatkowska, G. Norman, D. Parker. "PRISM: probabilistic model checking for performance and reliability analysis". *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, n.4, Mar. 2009.
- [10] H. Hansson and B. Jonsson. "A logic for reasoning about time and reliability". *Formal Aspects of Computing*, vol. 6, pp. 512–535, 1994.
- [11] A. Aziz, K. Sanwal, V. Singhal, R. K. Brayton, "Verifying Continuous Time Markov Chains" in *Proc. 8th International Conference on Computer Aided Verification*, 1996, pp. 269–276.
- [12] "Markov Reward Model Checker". Internet: www.mrmc-tool.org/trac, [Sep. 10, 2010].
- [13] J. Katoen, I. Zapreev, E. Hahn, H. Hermanns and D. Jansen. "The ins and outs of the probabilistic model checker MRMC". *QEST IEEE CS Press*, pp. 167–176, 2009.
- [14] G. Ciardo, R. Jones, A. Miner, R. Siminiceanu. "Logic and stochastic modeling with SMART". In *Computer Performance*, Vol. 2794 of LNCS, 2003, pp. 78–97.
- [15] "The Möbius Tool - Overview of Features". Internet: www.mobius.illinois.edu, [Aug. 15, 2010].
- [16] J. M. Doyle, "Abstract Model Specification Using the Möbius Modeling Tool", M.S. thesis, University of Illinois, USA, 2000.
- [17] C. Baier, J.P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.
- [18] A. Zimmermann. *Stochastic Discrete-Event Systems: Modeling, Evaluation and Applications*. Heidelberg: Springer, 2008.
- [19] A. Khalili, A. Jalaly Bidgoly, M. Abdollahi Azgomi, "PDETool: A Multi-formalism Modeling Tool for Discrete-Event Systems Based on SDES Description", in *Proc. 30th International Conference on Applications and Theory of Petri Nets*, 2009, pp. 22–26.
- [20] A. Jalaly Bidgoly, A. Khalili, M. Abdollahi Azgomi, "Implementation of Coloured Stochastic Activity Networks within the PDETool Framework", in *Proc. of Third Asia International Conference on Modeling & Simulation*, 2009, pp.710–715.
- [21] M. Y. Vardi and P. Wolper. "An automata-theoretic approach to automatic program verification", In *Proc. of 1st Annual Symposium on Logic in Computer Science (LICS)*, 1986, pages 332–344.
- [22] A. Bianco and L. de Alfaro. "Model checking of probabilistic and nondeterministic systems". In *Proc. Foundations of Software Technology and Theoretical Computer Science*, 1995, pp. 499–513.
- [23] M. Kwiatkowska, G. Norman, and D. Parker. "Stochastic model checking", in *Formal Methods for Performance Evaluation*, vol. 4486 of LNCS, pp. 220–270, Springer, 2007.

A Higher-Order Computational Model for Cooperative Constraint Programming

Rafael del Vado Vírveda and Fernando Pérez Morente

Departamento de Sistemas Informáticos y Computación

Universidad Complutense de Madrid

Facultad de Informática, Madrid, Spain

rdelvado@sip.ucm.es fperezmo@fdi.ucm.es

Abstract—This paper presents a theoretical framework for the integration of the cooperative constraint solving of several algebraic domains into higher-order functional and logic programming on λ -abstractions, using the instance *CFLP*(\mathcal{C}) of the generic Constraint Functional Logic Programming (*CFLP*) scheme [7] over a so-called higher-order coordination domain \mathcal{C} . We provide this framework as a powerful computational model for the higher-order cooperation of algebraic constraint domains over real numbers \mathcal{R} and integers \mathcal{FD} , which has been useful in practical applications involving the hybrid combination of its components, so that more declarative and efficient solutions can be promoted. Our proposal of computational model has been proved sound and complete with respect to the declarative semantics provided by the *CFLP* scheme, and enriched with new mechanisms for modeling the intended cooperation among the algebraic domains and a higher-order constraint domain λ equipped with a sound and complete constraint solver for solving higher-order equations.

Keywords: higher-order cooperation, constraint domains, functional and logic programming, logic in computer science, models of computation, hybrid computation

1. Introduction

The effort to identify suitable theoretical frameworks for *higher-order functional logic programming* has grown in recent years [2], [8], [9], [15]. The high number of approaches in this area and their different scopes and objectives indicate the high potential of such a paradigm in modeling complex real-world problems [12].

Functional logic programming [3] is the result of integrating two of the most successful declarative programming styles: functional and logic programming, in a way that captures the main advantages of both. Whereas higher-order programming is standard in functional programming, logic programming is in large part still tied to the first-order world. Only a few higher-order logic programming languages, most notably λ -*Prolog* [10], use higher-order logic for logic programming and have shown its practical utility, although the definition of evaluable functions is not supported. Moreover, higher-order constructs such as

function variables and λ -abstractions of the form $\lambda x. e$ are widely used in functional programming and higher-order logic programming languages, where λ -terms are used as data structures to obtain more of the expressivity of higher-order functional programming.

In this research area, [14], [15] proposes a complete theoretical framework for higher-order functional logic programming as an extension to the setting of the simply typed lambda calculus of a first-order rewriting logic, where programs are presented by *Conditional Pattern Rewrite Systems* (*CPRS* for short) on lambda abstractions. For a first impression of this higher-order programming framework, the following *CPRS* illustrates the syntax of patterns on lambda abstractions to define a classical higher-order function *map* for the application of a given function to a list of elements.

$$\begin{aligned} \text{map } (\lambda u. F(u), []) &= [] \\ \text{map } (\lambda u. F(u), [X | Xs]) &= [F(X) | \text{map } (\lambda u. F(u), Xs)] \end{aligned}$$

The first contribution of this paper is to present a theoretical framework for the integration of higher-order functional logic programming with *constraint solving*, extending the programming language with the capacity of solving constraints over a given algebraic constraint domain. The term *constraint* is intuitively defined as a relationship required to hold among certain entities as variables and values (e.g., $X + Y \leq 0$). We can take for instance the set of integers or the set of real numbers with addition, multiplication, equality, and perhaps other functions and predicates. Among the formalisms for the integration of constraints in functional logic programming we use in this work the *Constraint Functional Logic Programming* scheme *CFLP*(\mathcal{D}) [7] which supports a powerful combination of functional and constraint logic programming and can be instantiated by any constraint domain \mathcal{D} given as parameter which provides specific data values, constraints based on specific primitive operations, and a dedicated constraint solver. There are different *instances* of the scheme for various choices of \mathcal{D} , providing a declarative framework for any chosen domain. Useful constraint domains include the *Herbrand domain* \mathcal{H} which supplies *equality constraints* over symbolic terms, the algebraic domain \mathcal{R} which supplies

arithmetic constraints over *real numbers*, and the algebraic domain \mathcal{FD} which supplies arithmetic and finite domain constraints over *integers*. As a concrete example of a *CPRS* integrating higher-order functional logic programming with algebraic constraints in \mathcal{R} , we can consider the following variant of a classical higher-order function *diff* to compute the differential of a function f at some numeric value X under some arithmetic constraints over real numbers in the *conditional part* (\Leftarrow) of program rules.

$$\begin{aligned} \text{diff} &:: (\text{real} \rightarrow \text{real}) \rightarrow \text{real} \rightarrow \text{real} \\ \text{diff} (\lambda u. u, X) &= 1 \\ \text{diff} (\lambda u. \sin(F(u)), X) &= \cos(F(X)) * \text{diff} (\lambda u. F(u), X) \\ &\quad \Leftarrow \pi/4 \leq F(x) \leq \pi/2 \\ \text{diff} (\lambda u. \ln(F(u)), X) &= \text{diff} (\lambda u. F(u), X) / F(X) \\ &\quad \Leftarrow F(X) \neq 0 \end{aligned}$$

In contrast to first-order programming, we can easily formalize functions to be differentiated, or to compute the inverse operation of the differentiation (integration) by means of *narrowing* [14] as a suitable operational semantic, a transformation rule which combines the basic execution mechanism of functional and logic languages, namely *rewriting* with *unification*. For instance, we can compute by narrowing the substitution $\{F \mapsto \lambda u. \sin(u)\}$ as a solution of the goal $\lambda x. \text{diff} (\lambda u. \ln(F(u)), x) == \lambda x. \cos(x) / \sin(x)$ because the constraint $\lambda x. (\pi/4 \leq x \leq \pi/2 \rightarrow \sin(x) \neq 0)$ is evaluated to *true* by an \mathcal{R} -constraint solver.

Practical applications in higher-order functional logic programming, however, often involve more than one “pure” domain (i.e., \mathcal{H} , \mathcal{R} , \mathcal{FD} , etc.), and sometimes problem solutions have to be artificially adapted to fit a particular choice of domain and solver. The cooperative combination of constraint domains and solvers has evolved during the last decade as a relevant research issue that is raising an increasing interest in the constraint programming community. An important idea emerging from the research in this area is that of *hybrid constraint domain* (e.g., $\mathcal{H} \oplus \mathcal{R} \oplus \mathcal{FD}$ [1]), built as a combination of simpler pure domains and designed to support the cooperation of its components, so that more declarative and efficient solutions for practical problems can be promoted.

2. Higher-Order Algebraic Constraint Cooperation

The second contribution of this work is to present a formal framework for the cooperation of the algebraic constraints domains \mathcal{FD} and \mathcal{R} in an improved version of the *CFLP(D)* scheme [7], now useful for higher-order functional and logic programming on lambda abstractions. As a result, we provide a powerful theoretical framework for higher-order constraint functional logic programming with lambda abstractions and decidable higher-order unification in a new *higher-order constraint domain* λ , which leads to greater expressivity. As a motivation for the rest of the paper, we present in this section an example of *CPRS*-program

involving the cooperation of the algebraic constraint domains \mathcal{FD} and \mathcal{R} to illustrate the different cooperation mechanisms that are supported by our theoretical framework, as well as the benefits resulting from the cooperation in the higher-order functional logic programming setting.

In engineering, a common problem is the approximation of a complicated continuous function by a simple discrete function (e.g., the approximation of *GPS* satellite coordinates). Suppose we know a real function (given by a lambda abstraction $\lambda u. F(u)$) but it is too complex to evaluate efficiently. Then we could pick a few approximated (integer) data points from the complicated function, and try to interpolate those data points to construct a simpler function, for example, a polynomial $\lambda u. P(u)$. Of course, when using this polynomial function to calculate new (real) data points we usually do not receive the same result as when using the original function, but depending on the problem domain and the interpolation method used the gain in simplicity might offset the error.

$$\begin{aligned} \text{disc} &:: (\text{real} \rightarrow \text{real}) \rightarrow (\text{int} \rightarrow \text{int}) \\ \text{disc} (\lambda u. F(u)) &= \lambda u. P(u) \Leftarrow \\ &\quad \text{domain } [X] \ 0 \ N, \text{ labeling } [\text{ff}] [X], \\ &\quad X \Leftarrow RX, Y \Leftarrow RY, \\ &\quad |F(RX) - RY| < 1, \\ &\quad \text{collection } [X, Y] \ C, \text{ interpolation } [\text{lg}] \ C \ P \end{aligned}$$

The aim of this example is to approximate a continuous function represented by a lambda abstraction $\lambda u. F(u)$ over *real numbers* by a discrete polynomial function $\lambda u. P(u)$ over *integer numbers*. In this case, we use the \mathcal{FD} -constraints *domain* $[X] \ 0 \ N$, *labeling* $[\text{ff}] [X]$ to generate each value of the *discrete* interval $[0..N]$, according to a *first-fail* (or *ff*) labeling option [7]. In order to model the intended cooperation between the constraint domains \mathcal{FD} and \mathcal{R} we use a special kind of *hybrid constraints* \Leftarrow called *bridges*, as a key tool for communicating constraints between different algebraic constraint domains. The first bridge constraint $X \Leftarrow RX$ maps each *integer value* of X into an equivalent *real value* in RX . By applying the higher-order functional variable F to RX we obtain the \mathcal{R} -constraint $|F(RX) - RY| < 1$. From this constraint, the \mathcal{R} -solver computes (*infinite*) real values for RY . However, because of the second bridge constraint $Y \Leftarrow RY$, each *real value* assigned to RY by the constraint solving process causes the variable Y to be bound only to an equivalent *integer value*. By means of the primitive constraint *collection* $[X, Y] \ C$ we can collect all the pairs (X, Y) generated by the labeling-solving process into a set C . Finally, *interpolation* $[\text{lg}] \ C \ P$ finds a polynomial which goes exactly through the points collected in C by means of the *Lagrange Interpolation* (*Ig*) method. For instance, we can consider the following goal $\text{disc} (\lambda u. 4 * u - u^2) == \lambda u. P(u)$ involving the continuous function F as $\lambda u. 4 * u - u^2$ with $N = 4$. We obtain the set of integer pairs (x_i, y_i) in $C = \{(0, 0), (1, 3), (2, 4), (3, 3), (4, 0)\}$. For this particular

case, it is easy to check that this computed answer is simply $\{P \mapsto \lambda u. 4 * u - u^2\}$.

3. A Higher-Order Constraint Domain

Taking the generic scheme $CFLP(\mathcal{D})$ as a formal basis for foundational and practical issues concerning the cooperation of algebraic constraint domains, in this section we focus on the formalization of a *higher-order constraint domain* λ which supplies λ -abstractions and equality constraints over λ -terms in the instance $CFLP(\lambda)$. First, we introduce the preliminary notions of our higher-order theoretical framework to formalize the constraint domain λ along with a suitable λ -constraint solver based on an approach similar to the Huët's procedure of higher-order pre-unification [8], [9], [12].

3.1 Preliminary notions

We assume the reader is familiar with the notions and notations pertaining to λ -calculus (see, e.g., [12] for more examples and motivations). The set of types for simply typed λ -terms is generated by a set \mathcal{B} of *base types* (as e.g., *bool*, *real*, *int*) and the function type constructor " \rightarrow ". Simply typed λ -terms are generated in the usual way from a signature \mathcal{F} of *function symbols* and a countably infinite set \mathcal{V} of *variables* by successive operations of *abstraction* and *application*. We also consider the enhanced signature $\mathcal{F}_\perp = \mathcal{F} \cup \text{Bot}$, where $\text{Bot} = \{\perp_b \mid b \in \mathcal{B}\}$ is a set of distinguished \mathcal{B} -typed constants. The constant \perp_b is intended to denote an *undefined value* of type b . We employ \perp as a generic notation for a constant from Bot . A sequence of syntactic objects o_1, \dots, o_n , where $n \geq 0$, is abbreviated by $\overline{o_n}$. For instance, the simply typed λ -term $\lambda x_1, \dots, \lambda x_k. (\dots (a \ t_1) \dots t_n)$ is abbreviated by $\lambda \overline{x_k}. a(\overline{t_n})$. Substitutions $\gamma \in \text{Subst}(\mathcal{F}_\perp, \mathcal{V})$ are finite type-preserving mappings from variables to λ -terms, denoted by $\{\overline{X_n} \mapsto \overline{t_n}\}$, and extended homomorphically from λ -terms to λ -terms. By convention, we write $\{\}$ for the *identity substitution*, $t\gamma$ instead of $\gamma(t)$, and $\gamma\gamma'$ for the composition $\gamma' \circ \gamma$.

The long $\beta\eta$ -normal form of a λ -term t , denoted by $t_{\downarrow\beta}^\eta$, is the η -expanded form of the β -normal form of t . It is well-known that $s =_{\alpha\beta\eta} t$ if $s_{\downarrow\beta}^\eta =_\alpha t_{\downarrow\beta}^\eta$. Since $\beta\eta$ -normal forms are always defined, we will in general assume that λ -terms are in long $\beta\eta$ -normal form and are identified modulo α -conversion. For brevity, we may write variables and constants from \mathcal{F} in η -normal form, e.g., X instead of $\lambda \overline{x_k}. X(\overline{x_k})$. We assume that the transformation into long $\beta\eta$ -normal form is an implicit operation, e.g., when applying a substitution to a λ -term. With these conventions, every λ -term t has an unique long $\beta\eta$ -normal form $\lambda \overline{x_k}. a(\overline{t_n})$, where $a \in \mathcal{F}_\perp \cup \mathcal{V}$ and $a()$ coincides with a . The symbol a is called the *root* of t and is denoted by $hd(t)$. We distinguish between the set $\mathcal{T}(\mathcal{F}_\perp, \mathcal{V})$ of *partial* λ -terms and the set $\mathcal{T}(\mathcal{F}, \mathcal{V})$ of *total* λ -terms. The set $\mathcal{T}(\mathcal{F}_\perp, \mathcal{V})$ is a poset with respect to

the *approximation ordering* \sqsubseteq , defined as the least partial ordering such that:

$$\lambda \overline{x_k}. \perp \sqsubseteq \lambda \overline{x_k}. t \quad t \sqsubseteq t \quad \frac{s_1 \sqsubseteq t_1 \cdots s_n \sqsubseteq t_n}{\lambda \overline{x_k}. a(\overline{s_n}) \sqsubseteq \lambda \overline{x_k}. a(\overline{t_n})}$$

A *pattern* [9] is a λ -term t for which all subterms $t|_p = X(\overline{t_n})$, with $X \in \mathcal{FV}(t)$ a *free variable* of t and $p \in \text{MPos}(t)$ a *maximal position* in t , satisfy the condition that $t_1 \downarrow_\eta, \dots, t_n \downarrow_\eta$ is a sequence of distinct elements of the set $\mathcal{BV}(t, p)$ of *bound variables* abstracted on the path to position p in t . Moreover, if all such subterms of t satisfy the additional condition $\mathcal{BV}(t, p) \setminus \{t_1 \downarrow_\eta, \dots, t_n \downarrow_\eta\} = \emptyset$, then the pattern t is *fully extended*. It is well known that unification of patterns is decidable and unitary [9]. Therefore, for every $t \in \mathcal{T}(\mathcal{F}_\perp, \mathcal{V})$ and pattern π , there exists at most one matcher between t and π , which we denote by *matcher*(t, π).

3.2 The higher-order constraint domain λ

Intuitively, a *constraint domain* \mathcal{D} provides data values and constraints oriented to some particular application domain. In our higher-order setting, we need to formalize a special *higher-order constraint domain* λ to support computations with symbolic equality over λ -terms of any type. Formally, it is defined as follows:

Definition 1 (λ -domain): The higher-order constraint domain λ is a structure $\langle D_\lambda, ==^\lambda \rangle$ such that the carrier set D_λ coincides with the set of ground patterns (i.e., patterns without free variables) over any type, and the function symbol $==$ is interpreted as *strict equality* over D_λ , so that for all $t_1, t_2, t \in D_\lambda$, one has $==^\lambda \subseteq D_\lambda^2 \times D_\lambda$, where $t_1 ==^\lambda t_2 \rightarrow t$ (i.e., $(t_1, t_2, t) \in ==^\lambda$) iff some of the following three cases hold:

- (1) t_1 and t_2 are one and the same total λ -term in D_λ , and *true* $\sqsupseteq t$.
- (2) t_1 and t_2 have no common upper bound in D_λ w.r.t. the approximation ordering \sqsubseteq , and *false* $\sqsupseteq t$.
- (3) $t = \perp$.

An *equality constraint* (or simply, λ -constraint) is a multiset $\{\{s, t\}\}$, written $s == t$, where $s, t \in \mathcal{T}(\mathcal{F}_\perp, \mathcal{V})$ are λ -terms of the same type. The set of *solutions* of an equality constraint $s == t$ is defined as follows: $\text{Soln}(s == t) = \{\gamma \in \text{Subst}(\mathcal{F}_\perp, \mathcal{V}) \mid t\gamma ==^\lambda s\gamma \rightarrow \text{true}\}$. Any set E of strict equations is interpreted as conjunction, and therefore $\text{Soln}(E) = \bigcap_{(s==t) \in E} \text{Soln}(s == t)$.

3.3 The λ -constraint solver

Solving equality and disequality constraints in first-order term algebras (which is also known as *unification*) is the most famous symbolic constraint solving problem. In the higher-order case, *higher-order unification* is a powerful method for solving equality λ -constraints between λ -terms

and is currently used in *theorem provers* [15]. However, one of the major obstacles for reasoning in the higher-order case is that unification is undecidable. However, in this subsection we examine a decidable higher-order unification case of patterns by means of the development of a λ -constraint solver for the higher-order constraint domain λ , now supporting an improved treatment of the strict equality $==$ as a built-in primitive function symbol, rather than a defined function.

Definition 2 (States): The constraint solver $Solver^\lambda$ for the higher-order domain λ acts on *states* of the form $P \equiv \langle E | \mathcal{K} \rangle$, where E is a set of strict equality constraints $s == t$ between λ -terms s, t , and \mathcal{K} is a set of patterns intended to represent and store computed *values* in the sense of [14], [15] during the constraint solving process. The meaning of a state $P \equiv \langle E | \mathcal{K} \rangle$ is as follows: $\llbracket \langle E | \mathcal{K} \rangle \rrbracket = \{ \gamma \in Soln(E) \mid \mathcal{K}\gamma \text{ is a set of values} \}$. We note that $\llbracket \langle E | \mathcal{K} \rangle \rrbracket = \emptyset$ whenever \mathcal{K} is not a set of values. In the sequel, we denote this state by `fail` and call it *failure state*.

Solving a set of strict equality λ -constraints amounts to computing λ -derivations, i.e., sequences of transformation steps.

Definition 3 (Derivations): A λ -derivation of a set E of strict equality λ -constraints is a maximal finite sequence of transformation steps: $P_0 \equiv \langle E | \emptyset \rangle \equiv \langle E_0 | \mathcal{K}_0 \rangle \Rightarrow_{\sigma_1} P_1 \equiv \langle E_1 | \mathcal{K}_1 \rangle \Rightarrow_{\sigma_2} \dots \Rightarrow_{\sigma_m} P_m \equiv \langle E_m | \mathcal{K}_m \rangle$, between states P_0, P_1, \dots, P_m , such that $P_m \neq \text{fail}$ is a *final state*, i.e., a non failure state which can not be transformed anymore.

Definition 4 (λ -constraint solver):

- (1) Each transformation step in a λ -derivation Π corresponds to an instance of some transformation rule of the λ -constraint solver $Solver^\lambda$ described below. We abbreviate Π by $P_0 \Rightarrow_\sigma^* P_m$, where $\sigma = \sigma_1 \dots \sigma_m$.
- (2) Given such a set E of strict equality λ -constraints, the set of *computed answers* produced by the λ -constraint solver $Solver^\lambda$ is $\mathcal{A}(E) = \{ \sigma\gamma \upharpoonright_{\mathcal{FV}(E)} \mid \langle E | \emptyset \rangle \Rightarrow_\sigma^* P \text{ is a } \lambda\text{-derivation and } \gamma \in \llbracket P \rrbracket \}$, where $\mathcal{FV}(E)$ is the set of *free variables* of E .

In the sequel, we will describe the transformation rules of the λ -constraint solver and analyze its main properties. The general idea is to ensure the computation of solutions from λ -equations which are correct with respect to the semantics given in λ . Since the design considerations are quite involved and the analysis techniques quite complicated, we consider useful to precede our presentation with a brief outline of our design considerations and techniques. Typical requirements in the design of such a solver (see,

e.g., [1], [2], [7]) are *soundness*: every computed answer is a solution, i.e., $\mathcal{A}(E) \subseteq Soln(E)$, and *completeness*: for any $\gamma \in Soln(E)$ there exists $\gamma' \in \mathcal{A}(E)$ such that $\gamma' \preceq \gamma$ [$\mathcal{FV}(E)$]. Note that the completeness requirement demands the capability to compute a minimal complete set of solutions. It is easy to see that if the higher-order λ -constraint solver is complete then it suffices to enumerate minimal complete set of solutions of the final states. Therefore, an important design issue is to guarantee that minimal complete sets of solutions are easy to read off for the final states. In the design of first-order solvers for the Herbrand domain \mathcal{H} [1], this is achieved by ensuring that final states have empty components; thus the minimal complete set of solutions of a final state consists of the identity substitution $\{\}$. Unfortunately, things are much more complicated in the higher-order case. This problem is inevitably related to the problem of unifying *flex* λ -terms (i.e., λ -terms t such that $hd(t) \in \mathcal{FV}(t)$), which is in general intractable. We adopt an approach similar to Huët's procedure of higher-order pre-unification [9], [12]: we refrain from solving equations between flex λ -terms as much as possible. As a consequence, our final states will be a class of states whose λ -equations are only between flex λ -terms. This guarantee that the final states are meaningful and that it is relatively easy to read off some of their solutions.

(an) **annotation**

$$\langle \{ \{ s == t, E \} \mid \mathcal{K} \} \Rightarrow_{\{ \} } \{ \{ s ==_H t, E \} \mid \mathcal{K} \cup \{ H \} \} \rangle$$

where H is a fresh variable of a suitable type.

(sg) **strict guess**

$$\langle \{ \{ \lambda \bar{x}_k . a(\bar{s}_n) ==_H t, E \} \mid \mathcal{K} \} \Rightarrow_\sigma \langle \{ \{ \lambda \bar{x}_k . a(\bar{s}_n) ==_{H\sigma} t, E \} \mid \mathcal{K}\sigma \} \rangle \rangle$$

where $a \in \mathcal{F} \cup \{ \bar{x}_k \}$, and $\sigma = \{ H \mapsto \lambda \bar{x}_k . a(\overline{H_n(\bar{x}_k)}) \}$.

(d) **decomposition**

$$\langle \{ \{ \lambda \bar{x}_k . a(\bar{s}_n) ==_u \lambda \bar{x}_k . a(\bar{t}_n), E \} \mid \mathcal{K} \} \Rightarrow_\sigma \langle \{ \{ \lambda \bar{x}_k . s_n ==_{H_n} \lambda \bar{x}_k . t_n, E \} \mid \mathcal{K}\sigma \} \rangle \rangle$$

where $a \in \mathcal{F} \cup \{ \bar{x}_k \}$, and either

- $u \equiv H$ and $\sigma = \{ H \mapsto \lambda \bar{x}_k . a(\overline{H_n(\bar{x}_k)}) \}$, or
- $u \equiv \lambda \bar{x}_k . a(\overline{H_n(\bar{x}_k)})$ and $\sigma = \{ \}$.

(i) **imitation**

$$\langle \{ \{ \lambda \bar{x}_k . X(\bar{s}_p) ==_u \lambda \bar{x}_k . f(\bar{t}_n), E \} \mid \mathcal{K} \} \Rightarrow_\sigma \langle \{ \{ \lambda \bar{x}_k . X_n(\bar{s}_p) ==_{H_n} \lambda \bar{x}_k . t_n, E \} \mid \mathcal{K} \cup \{ X \} \} \rangle \rangle$$

where $X \in \mathcal{V}$, and either

- $u \equiv H$ and $\sigma = \{ X \mapsto \lambda \bar{y}_p . f(\overline{X_n(\bar{y}_p)}) \}$,
 $H \mapsto \lambda \bar{x}_k . f(\overline{H_n(\bar{x}_k)}) \}$, or
- $u \equiv \lambda \bar{x}_k . f(\overline{H_n(\bar{x}_k)})$ and $\sigma = \{ X \mapsto \lambda \bar{y}_p . f(\overline{X_n(\bar{y}_p)}) \}$.

(p) **projection**

$$\langle \{ \{ \lambda \bar{x}_k . X(\bar{s}_p) ==_u t, E \} \mid \mathcal{K} \} \Rightarrow_\sigma \langle \{ \{ \lambda \bar{x}_k . X(\bar{s}_p) ==_u t, E \} \mid \mathcal{K} \cup \{ X \} \} \rangle \rangle$$

where $X \in \mathcal{V}$, t is not flex, and $\sigma = \{ X \mapsto \lambda \bar{y}_p . y_i(\overline{X_n(\bar{y}_p)}) \}$.

(fs) **flex same**

$$\langle \{ \{ \lambda \bar{x}_k . X(\bar{y}_p) ==_H \lambda \bar{x}_k . X(\bar{y}'_p), E \} \mid \mathcal{K} \} \Rightarrow_\sigma \langle \{ \{ E \} \mid \mathcal{K} \cup \{ X \} \} \rangle \rangle$$

where $X \in \mathcal{V}$, $\lambda \bar{x}_k . X(\bar{y}_p)$, and $\lambda \bar{x}_k . X(\bar{y}'_p)$ are patterns, $\sigma = \{ X \mapsto \lambda \bar{y}_p . Z(\bar{z}_q), H \mapsto \lambda \bar{x}_k . Z(\bar{z}_q) \}$ with $\{ \bar{z}_q \} = \{ y_i \mid y_i = y'_i, 1 \leq i \leq n \}$.

(fd) **flex different**

$$\langle \{ \{ \lambda \bar{x}_k . X(\bar{y}_p) ==_H \lambda \bar{x}_k . Y(\bar{y}'_q), E \} \mid \mathcal{K} \} \Rightarrow_\sigma \langle \{ \{ E \} \mid \mathcal{K} \cup \{ X, Y \} \} \rangle \rangle$$

where $X, Y \in \mathcal{V}$, $\lambda\overline{x_k}.X(\overline{y_p})$, and $\lambda\overline{x_k}.Y(\overline{y'_q})$ are patterns, $X \neq Y$, $\sigma = \{X \mapsto \lambda\overline{y_p}.Z(\overline{z_r}), Y \mapsto \lambda\overline{y'_q}.Z(\overline{z_r}), H \mapsto \lambda\overline{x_k}.Z(\overline{z_r})\}$ with $\{\overline{z_r}\} = \{\overline{y_p}\} \cap \{\overline{y'_q}\}$.

- (cf) **clash failure**
 $\langle \{\{\lambda\overline{x_k}.a(\overline{s_n}) =_{=u} \lambda\overline{x_k}.a'(\overline{t_m}), E\} \mid \mathcal{K}\} \Rightarrow \{\} \text{fail}$
 if $a, a' \in \mathcal{F}_c \cup \{\overline{x_k}\}$ (where the notation \mathcal{F}_c will be explained in Section 4), and either (i) $a \neq a'$ or (ii) $hd(u) \notin \mathcal{V} \cup \{a, a'\}$.
- (oc) **occur check**
 $\langle \{\{\lambda\overline{x_k}.s =_{=u} \lambda\overline{x_k}.X(\overline{y_n}), E\} \mid \mathcal{K}\} \Rightarrow \{\} \text{fail}$
 if $X \in \mathcal{V}$, $\lambda\overline{x_k}.X(\overline{y_n})$ is a flex pattern, $hd(\lambda\overline{x_k}.s) \neq X$ and $(\lambda\overline{x_k}.s)|_p = X(\overline{z_n})$, where $\overline{z_n}$ is a sequence of distinct bound variables and p is a maximal safe position of $\lambda\overline{x_k}.s$ (i.e., $hd((\lambda\overline{x_k}.s)|_q) \in \mathcal{BV}(\lambda\overline{x_k}.s, q) \cup \mathcal{F}_c$ for all $q \leq p$).

In order to illustrate the overall behavior of our constraint solver $Solver^\lambda$, we consider the following λ -derivation involving the function symbols given in the signature of the *diff*-example presented in Section 1: $\langle \{\{\lambda x. \sin(F(x)) =_{=} \lambda x. \sin(\cos(x))\} \mid \emptyset\} \Rightarrow_{\{F \mapsto \lambda x. \cos(x)\}}^{(an),(d),(i)} \langle \emptyset \mid \{\lambda x. \sin(\cos(x)), \lambda x. \cos(x)\} \rangle$. Therefore, we have computed the substitution $\{F \mapsto \lambda x. \cos(x)\}$ as the only answer in $\mathcal{A}(\lambda x. \sin(F(x)) =_{=} \lambda x. \sin(\cos(x)))$.

The main properties of the λ -constraint solver, *soundness* and *completeness*, relate the solutions of a set of strict equality λ -constraints to the answers computed by our system of transformation rules for higher-order unification.

Theorem 1 (Properties of the λ -solver):

- (1) **Soundness:** Let $\langle E \mid \emptyset \rangle \Rightarrow_{\sigma}^* P$ be a λ -derivation. Then, $\sigma\gamma \in \text{Soln}(E)$ whenever $\gamma \in \llbracket P \rrbracket$.
- (2) **Completeness:** Let E be a finite set of λ -constraints. Then, $\mathcal{A}(E) = \{\gamma \upharpoonright_{\mathcal{FV}(E)} \mid \gamma \in \text{Soln}(E)\}$.

The proof can be found at <http://www.fdi.ucm.es/profesor/rdelvado/FCS2011/Proofs.pdf>. As we have commented, the generic scheme *CFLP*(\mathcal{D}) presented in [7] serves in this work as a logical and semantic framework for lazy Constraint Functional Logic Programming over a parametrically given constraint domain \mathcal{D} . In order to model the coordination of algebraic constraint domains in the higher-order functional logic programming framework [14], [15], we propose the construction of a *higher-order coordination domain* \mathcal{C} , as a special kind of hybrid domain tailored to the cooperation of the algebraic domains \mathcal{R} and \mathcal{FD} with the higher-order constraint domain λ which supplies lambda abstractions as data values and equalities over lambda terms as constraints. Following the methodology of [1], we obtain a suitable theoretical framework for the cooperation of algebraic constraint domains with their respective solvers in higher-order functional and logic programming using instances *CFLP*(\mathcal{C}).

A *coordination domain* \mathcal{C} is a kind of hybrid constraint domain built from various component domains (as, e.g., $\mathcal{H}, \lambda, \mathcal{R}, \mathcal{FD}, \dots$) intended to cooperate. The construction

of coordination domains involves a so-called *mediatorial domain* \mathcal{M} , whose purpose is to supply mechanisms for communication among the component domains via bridges, projections, functional variable applications, interpolations, and some more ad hoc operations. In this work, the component domains will be chosen as the pure domains λ , \mathcal{R} , and \mathcal{FD} , equipped with constraint solvers, in such a way that the communication provided by the mediatorial domain will also benefit the solvers. In the remain of this section we briefly explain the construction of this *higher-order coordination domain* \mathcal{C} , mathematically represented as the sum $\mathcal{C} = \mathcal{M} \oplus \lambda \oplus \mathcal{FD} \oplus \mathcal{R}$.

The construction of the coordination domain \mathcal{C} relies on a combined algebraic constraint domain $\mathcal{FD} \oplus \mathcal{R}$, which represents the *amalgamated sum* of the two joinable algebraic domains \mathcal{FD} and \mathcal{R} . In this case, the *joinability condition* asserts that the only primitive function symbol allowed to belong to \mathcal{FD} and \mathcal{R} is the strict equality $=_{=}$, where the interpretation of this operator will behave as defined for the higher-order constraint domain λ . As a consequence, the amalgamated sum $\lambda \oplus \mathcal{FD} \oplus \mathcal{R}$ is always possible, and give rise to compound a higher-order algebraic domain that can profit from the higher-order λ -constraint solver. However, in order to construct a more interesting sum for higher-order algebraic cooperation tailored to the communication among pure domains λ , \mathcal{R} , and \mathcal{FD} , *mediatorial domains* are needed.

The *higher-order mediatorial domain* \mathcal{M} serves as a basis for useful cooperation facilities among λ , \mathcal{FD} , and \mathcal{R} , including the projection of \mathcal{R} -constraints to the \mathcal{FD} -solver (and vice versa) using bridges (see [1] for more details), the specialization of λ -constraints to become \mathcal{R} - or \mathcal{FD} -constraints, the definition of algebraic constraints in \mathcal{R} and \mathcal{FD} from the application of higher-order functional variables in the domain λ , the gathering of numeric data values to construct a λ -abstraction in λ which closely fits the data points by means of interpolation techniques, and some other special mechanisms designed for processing the mediatorial constraints occurring in functional and logic computations.

4. Higher-Order Cooperative Programming in *CFLP*(\mathcal{C})

We are now ready to present our computation framework for higher-order functional and logic programming with cooperation of algebraic constraint domains within the *CFLP*(\mathcal{C}) instance of the *CFLP* scheme. Building upon the notion of higher-order coordination domain \mathcal{C} described in the previous section, we have designed a formal operational semantics by means of a higher-order cooperative constraint lazy narrowing calculus provided by *CFLP*(\mathcal{C}) which is *sound* and *complete*, extending the formal properties presented in Section 3 for the λ -constraint solver. The calculus embodies computation

rules for creating bridges, invoking constraint solvers, and performing constraint projections as well as other more ad hoc operations for communications among the higher-order domain λ and the algebraic constraint domains \mathcal{FD} and \mathcal{R} . Moreover, the calculus uses *higher-order demand-driven narrowing with definitional trees* for processing calls to program defined functions, ensuring that function calls are evaluated only as far as demanded by the resolution of the \mathcal{C} -constraints involved in the current computation. After introducing *CFLP(C)-programs* and *goals*, we present the goal-solving rules of the calculus and results concerning the formal properties of our higher-order cooperative computation model.

Definition 5 (CFLP(C)-Programs): A Constrained Pattern Rewrite System over the higher-order coordination domain $\mathcal{C} = \mathcal{M} \oplus \lambda \oplus \mathcal{FD} \oplus \mathcal{R}$ (*CPRS(C)* for short) is a finite set of \mathcal{C} -constrained rewrite rules of the form $f(\bar{l}_n) = r \Leftarrow C$, where

- $f(\bar{l}_n)$ and r are total λ -terms of the same base type.
- $f(\bar{l}_n)$ is a fully extended linear pattern.
- C is a (possibly empty) finite sequence of \mathcal{C} -constraints. More precisely, each \mathcal{C} -constraints is exactly of one of the following cases:
 - λ -constraint (C_λ): equations $s == t$, with $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.
 - \mathcal{M} -constraint ($C_{\mathcal{M}}$): bridge constraints $X \rightleftharpoons Y$, with $X :: \text{int}$ and $Y :: \text{real}$.
 - \mathcal{R} -constraint ($C_{\mathcal{R}}$): arithmetic constraints over real numbers.
 - \mathcal{FD} -constraint ($C_{\mathcal{FD}}$): arithmetic and finite domain constraints over integers.

A goal C for a given *CPRS(C)* is a set of \mathcal{C} -constraints. Each *CPRS(C)* \mathcal{R} induces a partition of the underlying signature \mathcal{F} into \mathcal{F}_d (*defined function symbols*) and \mathcal{F}_c (*data constructors*): $\mathcal{F}_d = \{f \in \mathcal{F} \mid \exists(f(\bar{l}_n) = r \Leftarrow C) \in \mathcal{R}\}$ and $\mathcal{F}_c = \mathcal{F} \setminus \mathcal{F}_d$. We say that \mathcal{R} is a *constructor-based CPRS(C)* if each conditional pattern rewrite rule $f(\bar{l}_n) = r \Leftarrow C$ satisfies the condition that $l_1, \dots, l_n \in \mathcal{T}(\mathcal{F}_c, \mathcal{V})$.

Our higher-order cooperative computation model works by transforming initial goals C_0 into final goals C , which serve as computed answers from a set of values \mathcal{K} . We represent the computation as a *CFLP(C)-derivation* $\langle C_0 \mid \emptyset \rangle \Rightarrow_\sigma^* \langle C \mid \mathcal{K} \rangle$, extending the notation previously introduced by the λ -constraint solver in Section 3. The core of the computational model in *CFLP(C)* consists of the *Higher-Order Lazy Narrowing calculus with Definitional Trees* presented in [14] for higher-order (unconstrained) functional logic programming. We can use this calculus in a modular way, ignoring at this moment algebraic

domain cooperation and solver invocation, in order to deal with calls to defined functions and to apply a program rule. More precisely, if the goal includes a constraint C_λ (analogously, $C_{\mathcal{R}}$ or $C_{\mathcal{FD}}$) of the form $\lambda \bar{x}_k. f(\bar{s}_n) == t$ with $f \in \mathcal{F}_d$, we can apply the rules of the calculus to perform a demand-driven evaluation of the function call, represented by $\lambda \bar{x}_k. \langle f(\bar{s}_n), \mathcal{T}_f \rangle \mapsto R$ (see [14] for more details). Then, higher-order narrowing is applied in an optimized way by using an associated *higher-order definitional tree* \mathcal{T}_f to ensure an optimal choice of needed narrowing steps by means of the selection of a suitable (possibly non-deterministic) conditional pattern rewrite rule $\{\pi = r_i \Leftarrow C_i\}_{1 \leq i \leq m}$ of the *CFLP(C)*-program \mathcal{R} . Therefore, we transform C_λ into a flattened form $R == t$. The following three rules formalize these transformations.

- rigid narrowing**
 $\langle \{\{\lambda \bar{x}_k. f(\bar{s}_n) == t, C\} \mid \mathcal{K}\} \Rightarrow \{\}\rangle$
 $\langle \{\{\lambda \bar{x}_k. \langle f(\bar{s}_n), \mathcal{T}_f \rangle \mapsto R, R == t, C\} \mid \mathcal{K}\} \Rightarrow \{\}\rangle$
 where $f \in \mathcal{F}_d$.
- flex narrowing**
 $\langle \{\{\lambda \bar{x}_k. X(\bar{s}_m) == t, C\} \mid \mathcal{K}\} \Rightarrow \sigma$
 $\langle \{\{\lambda \bar{x}_k. \langle X(\bar{s}_m), \mathcal{T}_f \rangle \mapsto R, R == t, C\} \mid \mathcal{K} \cup \{X\}\} \sigma \rangle$
 where $\sigma = \{X \mapsto \lambda \bar{y}_m. f(X_n(\bar{y}_m))\}$ with $f \in \mathcal{F}_d$.
- evaluation**
 $\langle \{\{\lambda \bar{x}_k. \langle \pi', \text{rule}(\pi, \{r_i \Leftarrow C_i\}_{1 \leq i \leq m}) \rangle \mapsto R, C\} \mid \mathcal{K}\} \Rightarrow \{\}\rangle$
 $\langle \{\{\lambda \bar{y}_{k_q}. s_q == R_q, C_i, \lambda \bar{x}_k. r_i == R, C\} \mid \mathcal{K} \cup \{\bar{R}_q\}\} \rangle$
 if $1 \leq i \leq m$, $\text{matcher}(\lambda \bar{x}_k. \pi', \lambda \bar{x}_k. \pi) = \{R_q \mapsto \lambda \bar{y}_{k_q}. s_q\}$,
 and $\{\bar{R}_q\} = \mathcal{FV}(\lambda \bar{x}_k. \pi)$.

The following two rules describe the possible transformation in a goal of a finite subset $C_{\mathcal{D}}$ of \mathcal{D} -constraints (where \mathcal{D} is the pure domain λ , \mathcal{M} , \mathcal{FD} or \mathcal{R}) by a \mathcal{D} -solver's invocation, including the detection of *failure* by the corresponding solver.

- constraint solving**
 $\langle \{\{C_{\mathcal{D}}, C\} \mid \mathcal{K}\} \Rightarrow_\sigma \langle \{C'_{\mathcal{D}}, C\} \mid \mathcal{K}' \rangle \rangle$
 if the \mathcal{D} -constraint solver $\text{Solver}^{\mathcal{D}}$ performs a *successful* \mathcal{D} -derivation
 $\langle C_{\mathcal{D}} \mid \mathcal{K} \rangle \Rightarrow_\sigma^* \langle C'_{\mathcal{D}} \mid \mathcal{K}' \rangle$.
- solving failure**
 $\langle \{C_{\mathcal{D}}, C\} \mid \mathcal{K} \rangle \Rightarrow_\sigma \text{fail}$
 if the \mathcal{D} -constraint solver $\text{Solver}^{\mathcal{D}}$ performs a *failure* \mathcal{D} -derivation
 $\langle C_{\mathcal{D}} \mid \mathcal{K} \rangle \Rightarrow_\sigma^* \text{fail}$.

The availability of the \mathcal{M} -solver means that solving mediatorial constraints contributes to the cooperative goal-solving process, in addition to the role of bridges for guiding *projections*. Projected constraints improve the performance of the corresponding solver, enabling certain solvers to profit from (the projected forms) of constraints originally intended for other solvers. The last two rules take care of this idea of domain cooperation, and are used to generate new bridges and to compute projected constraints to be added to the goal.

- set bridges**
 $\langle \{C_{\mathcal{D}}, C_{\mathcal{M}}, C\} \mid \mathcal{K} \rangle \Rightarrow \{\} \langle \{C_{\mathcal{D}}, C_{\mathcal{M}}, C'_{\mathcal{M}}, C\} \mid \mathcal{K}' \rangle$
 where \mathcal{D} is the algebraic constraint domain \mathcal{R} (resp. \mathcal{FD}) and \mathcal{D}' the domain \mathcal{FD} (resp. \mathcal{R}), and $\text{bridges}^{\mathcal{D} \rightarrow \mathcal{D}'}(C_{\mathcal{D}}, C_{\mathcal{M}}) = C'_{\mathcal{M}}$.

(pp) **propagate projections**
 $\langle \{C_{\mathcal{D}}, C_{\mathcal{M}}, C\} \mid \mathcal{K} \rangle \Rightarrow_{\{\}} \langle \{C_{\mathcal{D}'}, C'_{\mathcal{D}'}, C_{\mathcal{M}}, C\} \mid \mathcal{K} \rangle$
 where \mathcal{D} is the algebraic constraint domain \mathcal{R} (resp. \mathcal{FD}) and \mathcal{D}' the domain \mathcal{FR} (resp. \mathcal{R}), and $proj^{\mathcal{D} \rightarrow \mathcal{D}'}(C_{\mathcal{D}}, C_{\mathcal{M}}) = C'_{\mathcal{D}'}$.

The concrete example given in Section 2 illustrate the behavior of this goal-solving calculus in $CFLP(\mathcal{C})$. We conclude this section with theoretical results now ensuring *soundness* and *completeness* for $CFLP(\mathcal{C})$ -derivations. Both properties are presented w.r.t. the declarative semantics of the instance $CFLP(\mathcal{C})$, provided by the generic $CFLP(\mathcal{D})$ scheme [7] and the semantic framework for higher-order functional logic programs on λ -abstractions [14], [15]. For instance, the set of solutions $Soln(\mathcal{C})$ of a goal \mathcal{C} and the meaning $\llbracket P \rrbracket$ of a state P now refer to the existence of *logical proofs* in the corresponding \mathcal{D} -instance of a generic *Constrained ReWriting Logic*, whose inference rules can be found in [7], [15] for each \mathcal{D} -constraint $C_{\mathcal{D}}$ in \mathcal{C} , where \mathcal{D} is \mathcal{R} , \mathcal{FD} , \mathcal{M} , or λ .

Theorem 2 (Properties of the calculus):

- (1) **Soundness:** Let $\langle C \mid \emptyset \rangle \Rightarrow_{\sigma}^* P$ be a $CFLP(\mathcal{C})$ -derivation. Then, $\sigma\gamma \in Soln(\mathcal{C})$ whenever $\gamma \in \llbracket P \rrbracket$.
- (2) **Completeness:** Let \mathcal{C} be an initial goal given by a finite set of \mathcal{C} -constraints, and $\mathcal{A}(\mathcal{C}) = \{\sigma\gamma \upharpoonright_{\mathcal{FV}(\mathcal{C})} \mid \langle C \mid \emptyset \rangle \Rightarrow_{\sigma}^* P \text{ is a finite } CFLP(\mathcal{C})\text{-derivation with } \gamma \in \llbracket P \rrbracket\}$. Then, $\mathcal{A}(\mathcal{C}) = \{\gamma \upharpoonright_{\mathcal{FV}(\mathcal{C})} \mid \gamma \in Soln(\mathcal{C})\}$.

Thanks to the *soundness* and *completeness* results modularly presented in Section 3 for the new constraint domain λ and the higher-order narrowing calculus in [14] for declarative programming in $CFLP(\lambda)$, THEOREM 2 can be proved in the cooperative setting of $CFLP(\mathcal{C})$.

5. Conclusions

In this work we have presented a suitable use of cooperative algebraic constraint domains and solvers in a higher-order functional and logic programming framework on λ -abstractions. We have investigated foundational issues concerning a sound and complete computational framework for the cooperation of algebraic constraint domains. For this purpose, we have designed an improved higher-order instance $CFLP(\mathcal{C})$ of an already existing generic scheme [7] for constraint functional logic programming, now over a higher-order coordination domain \mathcal{C} .

In addition to already mentioned works, an important related work in this area is the $CFLP$ scheme developed by Mircea Marin in his PhD Thesis [8]. This work introduces $CFLP(\mathcal{D}, \mathcal{S}, \mathcal{L})$, a family of languages parameterized by a constraint domain \mathcal{D} , a strategy \mathcal{S} which defines the cooperation of several constraint solvers over \mathcal{D} , and a constraint lazy narrowing calculus \mathcal{L} for solving constraints involving functions defined by user given constrained rewrite

rules. The main difference with respect to our approach is the lack of declarative (model-theoretic and fixpoint) semantics provided by the rewriting logic underlying our $CFLP(\mathcal{C})$ instance (see [15] for more details). Another difference is the intended application domain. The instance of $CFLP$ developed by Marin combines four solvers over a constraint domain for algebraic symbolic computation.

In the future, we would like to improve some of the limitations of our current approach to higher-order algebraic domain cooperation, concerning both the formal foundations and the implemented system. For instance, the computational model should be generalized to allow for an arbitrary higher-order coordination domain \mathcal{C} in place of the concrete choice $\mathcal{M} \oplus \lambda \oplus \mathcal{R} \oplus \mathcal{FD}$, and the implemented prototype should be properly developed, maintained and improved in various ways. In particular, the experimentation with *benchmarks* and application cases should be further developed.

Acknowledgements

This work has been partially supported by the Spanish projects STAMP (TIN2008-06622-C03-01), Prometidos-CM (S2009TIC-1465) and GPD (UCM-BSCH-GR35/10-A-910502).

References

- [1] S. Estévez et al. *On the cooperation of the constraint domains \mathcal{H} , \mathcal{R} , and \mathcal{FD} in $CFLP$* . Journal of TPLP, vol. 9, pp. 415–527, 2009.
- [2] J.C. González, M.T. Hortalá, and M. Rodríguez. *A higher-order rewriting logic for functional logic programming*. In Proc. ICLP'97, pp. 153–167, 1997.
- [3] M. Hanus. *The Integration of Functions into Logic Programming: From Theory to Practice*. Journal of Logic Programming 19&20, pp. 583–628, 1994.
- [4] M. Hanus and C. Prehofer. *Higher-order narrowing with definitional trees*. Journal of Functional Programming, vol. 9, pp. 33–75, 1999.
- [5] J.R. Hindley and J.P. Seldin. *Introduction to Combinatorics and λ -Calculus*. Cambridge University Press, 1986.
- [6] F.J. López and J. Sánchez. *TCY: A Multiparadigm Declarative System*. In Proc. RTA'99, Springer LNCS 1631, pp. 244–247, 1999.
- [7] F.J. López, M. Rodríguez, and R. del Vado. *A New Generic Scheme for Functional Logic Programming with Constraints*. Journal of HOSC 20, 1/2, pp. 73–122, 2007.
- [8] M. Marin. *Functional Logic Programming with Distributed Constraint Solving*. PhD. Thesis, 2000.
- [9] D. Miller. *A logic programming language with λ -abstraction, function variables, and simple unification*. Journal of Logic and Computation, 1(4):497–536, 1991.
- [10] G. Nadathur and D. Miller. *An overview of λ -Prolog*. In Proc. Int. Conf. on Logic Programming (ICLP'88), The MIT Press, pp. 810–827, 1988.
- [11] L.C. Paulson. *Isabelle: A Generic Theorem Prover*. Springer LNCS, vol. 828, 1994.
- [12] C. Prehofer. *Solving Higher-Order Equations. From Logic to Programming*. Found. of Computing, 1998.
- [13] T. Suzuki, K. Nakagawa, and T. Ida. *Higher-Order Lazy Narrowing Calculus: A Computation Model for a Higher-Order Functional Logic Language*. In Proc. of ALP'97, vol. 1298 of LNCS, pp. 99–113, 1997.
- [14] R. del Vado. *A Higher-Order Demand-Driven Narrowing Calculus with Definitional Trees*. In Proc. ICTAC'07, Springer LNCS 4711, pp. 169–184, 2007.
- [15] R. del Vado. *A Higher-Order Logical Framework for the Algorithmic Debugging and Verification of Declarative Programs*. In PPDP'09, ACM, pp. 49–60, 2009.

On Compilation of Higher-Order Concurrent Programs into First Order Programs Preserving Scope Equivalence

Masaki Murakami

Department of Computer Science,
Graduate School of Natural Science and Technology, Okayama University
3-1-1 Tsushima-Naka, Okayama, 700-0082, Japan
murakami@momo.cs.okayama-u.ac.jp

Abstract—This paper discusses the expressive power of a graph rewriting model of concurrent processes with higher-order communication. As we reported before, it is difficult to represent the scopes of names using models based on process algebra. Then we presented a model of concurrent systems based on graph rewriting. The model makes it possible to represent the scopes of names precisely. We defined an equivalence relation called scope equivalence. Two systems are scope equivalent not only in their behavior but in extrusion of scopes of names also. This paper presents a result that there is no compilation mapping from the higher-order model into the first-order model that is homomorphic wrt input context and full abstract wrt the scope equivalence. As reported, it is possible to compile LHO_{π} processes into first-order π -calculus processes preserving a behavioral equivalence. In that sense, the first-order calculus is as expressive as the higher-order calculus when we focus on the behavioral equivalence. On the other hand, this paper shows that the higher-order model is strictly more expressive than the first-order model if we focus on scope equivalence.

Keywords: concurrency, graph rewriting, higher-order communication

1. Introduction

LHO_{π} (Local Higher-Order π -calculus) [11] is a formal model of concurrent systems with higher-order communication. It is a subcalculus of higher-order π -calculus[10] with asynchronous communication capability. The calculus has the expressive power to represent practically and/or theoretically interesting examples that include program code transfer.

On the other hand, as we reported in [4], [5], [6], it is difficult to represent the scopes of names of communication channels precisely using models based on process algebra such as LHO_{π} . We presented a model that is based on graph rewriting instead of process algebra as a solution for the problem on representation of scopes of names [4].

Our model of concurrent systems is based on graph rewriting system such as [1], [2], [3], [12]. We represent a concurrent program consists of a number of processes (and messages on the way) using a bipartite directed acyclic graph. A bipartite graph is a graph whose nodes are

decomposed into two disjoint sets: source nodes and sink nodes such that no two graph nodes within the same set are adjacent. Every edge is directed from a source node to a sink node. The system three processes b_1, b_2 and b_3 and two names $a_i (i = 1, 2)$ shared by b_i and b_{i+1} is represented with a graph as Fig.1.

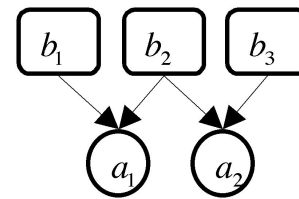


Fig. 1. A Bipartite Directed Acyclic Graph

Processes and messages on the way are represented with source nodes. We call source nodes as behaviors. In Fig. 1., b_1, b_2 and b_3 are behaviors.

We define the operational semantics of the model as a set of rules to rewrite graphs. The intuitive description of the rewriting rules is presented in [5], [7], [8].

We defined an equivalence relation of processes called “scope equivalence” on the model. Intuitively, two systems are scope equivalent not only in their behavior but in extrusions of scopes of names. We showed the congruence results of the scope equivalence for the first-order case[6]. We extended the the model for systems with higher-order communication[5].

This paper discusses the expressive power of the higher-order model. As [11] showed, it is possible to compile LHO_{π} processes into first-order processes preserving a behavioral equivalence. Namely there exists a compilation mapping from LHO_{π} processes into first-order processes that is full abstract wrt a behavioral equivalence and homomorphic wrt first-order context. Namely there exists a compilation mapping $\llbracket _ \rrbracket$ such that for any higher-order processes P and Q , $P \approx Q$ iff $\llbracket P \rrbracket \approx \llbracket Q \rrbracket$ and for any process P and any first order context $R[_]$, $\llbracket R[P] \rrbracket = \llbracket R \rrbracket \llbracket P \rrbracket$. In that sense, the first-order calculus is as expressive as the higher-order

calculus when we focus on a behavioral equivalence.

This paper presents a result that there is no compilation mapping that is full abstract wrt the scope equivalence and homomorphic wrt input context. Namely, there exist two programs P and Q which are scope equivalent and for any mapping $\llbracket _ \rrbracket$ that is homomorphic wrt input-context, their images $\llbracket P \rrbracket$ and $\llbracket Q \rrbracket$ are not scope equivalent or for some context $R[_]$, $R[P]$ and $R[Q]$ are not scope equivalent but $\llbracket R[P] \rrbracket$ and $\llbracket R[Q] \rrbracket$ are scope equivalent. Thus the higher-order model is strictly more expressive than the first-order model if we focus on scope equivalence.

2. Formal Definitions

2.1 Programs

First, a countably-infinite set of *names* is presupposed.

Definition 1 (program, behavior) Programs and behaviors are defined recursively as follows.

(i) Let a_1, \dots, a_k are distinct names. A *program* is a bipartite directed acyclic graph with source nodes b_1, \dots, b_m and sink nodes a_1, \dots, a_k such that

Each source node $b_i (1 \leq i \leq m)$ is a behavior. Duplicated occurrences of the same behavior are possible.

Each sink node is a name $a_j (1 \leq j \leq k)$. All a_j 's are distinct.

Each edge is directed from a source node to a sink node. Namely, an edge is an ordered pair (b_i, a_j) of a source node and a name. For any source node b_i and a name a_j there is at most one edge from b_i to a_j .

For a program P , we denote the multiset of all source nodes of P as $\text{src}(P)$, the set of all sink nodes as $\text{snk}(P)$ and the set of all edges as $\text{edge}(P)$. Note that the empty graph $\mathbf{0}$ such that $\text{src}(\mathbf{0}) = \text{snk}(\mathbf{0}) = \text{edge}(\mathbf{0}) = \emptyset$ is a program.

(ii) A *behavior* is an application, a message or a node consists of the *epidermis* and the *content* defined as follows. In the following of this definition, we assume that any element of $\text{snk}(P)$ nor x does not occur in anywhere else in the program.

- 1) A node labeled with a tuple of a name n (called the *subject of the message*) and an object o is a *message* and denoted as $n\langle o \rangle$.
- 2) A tuple of a variable x and a program P is an *abstraction* and denoted as $(x)P$. An *object* is a name or an abstraction.
- 3) A node labeled with a tuple of an abstraction and an object is an *application*. We denote an application as $A\langle o \rangle$ where A is an abstraction and o is an object.
- 4) A node whose epidermis is labeled with “!” and the content is a program P is a *replication*, and denoted as $!P$.

5) An *input prefix* is a node (denoted as $a(x).P$) that the epidermis is labeled with a tuple of a name a and a variable x and the content is a program P .

6) A τ -*prefix* is a node (denoted as $\tau.P$) that the epidermis is labeled with a silent action τ and the content is a program P .

A program P is *first-order* if any abstraction never occurs anywhere in P .

Definition 2 (local program) A program P is *local* if for any input prefix $c(x).Q$ and any abstraction $(x)Q$ occurring in P , x does not occur in the epidermis of any input prefix in Q . An abstraction $(x)P$ is local if P is local. A local object is a local abstraction or a name.

Though the locality condition affects the expressive power of the model, we do not consider that this restriction significantly damages the expressive power. For the detail, see [6], [7].

Definition 3 (free/bound name)

- 1) For a behavior or an object p , the *set of free names of p* : $\text{fn}(p)$ is defined as : $\text{fn}(\mathbf{0}) = \emptyset$, $\text{fn}(a) = \{a\}$ for a name a , $\text{fn}(a\langle o \rangle) = \text{fn}(o) \cup \{a\}$, $\text{fn}((x)P) = \text{fn}(P) \setminus \{x\}$, $\text{fn}(!P) = \text{fn}(P)$, $\text{fn}(\tau.P) = \text{fn}(P)$, $\text{fn}(a(x).P) = (\text{fn}(P) \setminus \{x\}) \cup \{a\}$ and
- 2) $\text{fn}(o_1\langle o_2 \rangle) = \text{fn}(o_1) \cup \text{fn}(o_2)$.
- 3) For a program P where $\text{src}(P) = \{b_1, \dots, b_m\}$, $\text{fn}(P) = \bigcup_i \text{fn}(b_i) \setminus \text{snk}(P)$.

The set of *bound names* of P (denoted as $\text{bn}(P)$) is the set of all names that occur in P but not in $\text{fn}(P)$ (including elements of $\text{snk}(P)$ even if they do not occur in any element of $\text{src}(P)$).

The notion of free name in our model is a little bit different from that of process algebras such as π -calculus. For example, a free name x occurs in Q is used as a variable in $(x)Q$ or $a(x).Q$. A channel name that is used for communication with the environments is an element of snk , so it is not a free name.

Definition 4 (normal program) A program P is *normal* if for any $b \in \text{src}(P)$ and for any $n \in \text{fn}(b) \cap \text{snk}(P)$, $(b, n) \in \text{edge}(P)$ and any program occurs in b is normal.

In the rest of this paper we consider normal programs only.

Definition 5 (composition) Let P and Q be programs such that $\text{src}(P) \cap \text{src}(Q) = \emptyset$ and $\text{fn}(P) \cap \text{snk}(Q) = \text{fn}(Q) \cap \text{snk}(P) = \emptyset$. The *composition* $P\|Q$ of P and Q is the program such that $\text{src}(P\|Q) = \text{src}(P) \cup \text{src}(Q)$, $\text{snk}(P\|Q) = \text{snk}(P) \cup \text{snk}(Q)$ and $\text{edge}(P\|Q) = \text{edge}(P) \cup \text{edge}(Q)$.

Intuitively, $P\|Q$ is the parallel composition of P and Q . Note that we do not assume $\text{snk}(P) \cap \text{snk}(Q) = \emptyset$. Obviously $P\|Q = Q\|P$ and $((P\|Q)\|R) = (P\|(Q\|R))$ for any P, Q and R from the definition. The empty graph $\mathbf{0}$ is the unit of “ $\|$ ”. Note that $\text{src}(P) \cup \text{src}(Q)$ and $\text{edge}(P) \cup \text{edge}(Q)$ denote the multiset unions while $\text{snk}(P) \cup \text{snk}(Q)$ denotes the set union. It is easy to show that for normal and local programs P and Q , $P\|Q$ is normal and local.

Definition 6 (N -closure) For a normal program P and a set of names N such that $N \cap \text{bn}(P) = \emptyset$, the N -closure $\nu N(P)$ is the program such that $\text{src}(\nu N(P)) = \text{src}(P)$, $\text{snk}(\nu N(P)) = \text{snk}(P) \cup N$ and $\text{edge}(\nu N(P)) = \text{edge}(P) \cup \{(b, n) \mid b \in \text{src}(P), n \in N\}$.

We denote $\nu N_1(\nu N_2(P))$ as $\nu N_1 \nu N_2(P)$ for a program P and sets of names N_1 and N_2 .

Definition 7 (deleting a behavior) For a normal program P and $b \in \text{src}(P)$, $P \setminus b$ is a program that is obtained by deleting a node b and edges that are connected with b from P . Namely, $\text{src}(P \setminus b) = \text{src}(P) \setminus \{b\}$, $\text{snk}(P \setminus b) = \text{snk}(P)$ and $\text{edge}(P \setminus b) = \text{edge}(P) \setminus \{(b, n) \mid (b, n) \in \text{edge}(P)\}$.

Note that $\text{src}(P) \setminus \{b\}$ and $\text{edge}(P) \setminus \{(b, n) \mid (b, n) \in \text{edge}(P)\}$ mean the multiset subtractions.

Definition 8 (context) Let P be a program and $b \in \text{src}(P)$ where b is an input prefix, a τ -prefix or a replication and the content of b is $\mathbf{0}$. A *simple first-order context* is the graph $P[\]$ such that the contents $\mathbf{0}$ of b is replaced with a hole “[]”. We call a simple context as a τ -context, an input context or a replication context if the hole is the contents of a τ -prefix, of an input prefix or of a replication respectively.

Let P be a program such that $b \in \text{src}(P)$ and b is an application $(x)\mathbf{0}\langle Q \rangle$. An *application context* $P[\]$ is the graph obtained by replacing the behavior b with $(x)[\]\langle Q \rangle$. A *simple context* is a simple first-order context or an application context.

A *context* is a simple context or the graph $P[Q[_]]$ that is obtained by replacing the hole of $P[\]$ with $Q[\]$ for a simple context $P[\]$ and a context $Q[\]$ (with some renaming of the names occur in Q if necessary).

For a context $P[\]$ and a program Q , $P[Q]$ is the program obtained by replacing the hole in $P[\]$ by Q (with some renaming of the names occur in Q if necessary).

2.2 Operational Semantics

Definition 9 (substitution) Let p be a behavior, an object or a program and o be an object. For a name a , we assume that $a \in \text{fn}(p)$. The mapping $[o/a]$ defined as follows is a *substitution*.

for a name c , $c[o/a] = \begin{cases} o & \text{if } c = a \\ c & \text{otherwise} \end{cases}$

for behaviors,

- $((x)P)[o/a] = (x)(P[o/a])$,
- $(o_1\langle o_2 \rangle)[o/a] = o_1[o/a]\langle o_2[o/a] \rangle$,
- $(!P)[o/a] = !(P[o/a])$,
- $(c(x).P)[o/a] = c(x).(P[o/a])$ and
- $(\tau.P)[o/a] = \tau.(P[o/a])$

and for a program P and $a \in \text{fn}(P)$, $P[o/a] = P'$ where P' is a program such that

- $\text{src}(P') = \{b[o/a] \mid b \in \text{src}(P)\}$,
- $\text{snk}(P') = \text{snk}(P)$ and
- $\text{edge}(P') = \{(b[o/a], n) \mid (b, n) \in \text{edge}(P)\}$.

For the cases of abstraction and input prefix, note that we can assume $x \neq a$ because $a \in \text{fn}((x)P)$ or $\in \text{fn}(c(x).P)$ without losing generality. (We can rename x if necessary.)

Definition 10 Let p be a local program or a local object. A substitution $[a/x]$ is *acceptable* for p if for any input prefix $c(y).Q$ occurring in p , $x \neq c$.

In the rest of this paper, we consider acceptable substitutions only for a program or an abstraction. For the detail, see [7].

Definition 11 (action) For a name a and an object o , an *input action* is a tuple $a(o)$ and an *output action* is a tuple $a\langle o \rangle$. An *action* is a *silent action* τ , an output action or an input action.

Definition 12 (labeled transition) For an action α , $\xrightarrow{\alpha}$ is the least binary relation on normal programs that satisfies the following rules.

input

If $b \in \text{src}(P)$ and $b = a(x).Q$, then

$$P \xrightarrow{a(o)} (P \setminus b) \parallel \nu\{n \mid (b, n) \in \text{edge}(P)\} \nu M(Q[o/x])$$

for an object o and a set of names M such that $\text{fn}(o) \cap \text{snk}(P) \subset M \subset \text{fn}(o) \setminus \text{fn}(P)$.

-conversion

If $b \in \text{src}(P)$ and $b = (y)Q\langle o \rangle$, then

$$P \xrightarrow{\tau} (P \setminus b) \parallel \nu\{n \mid (b, n) \in \text{edge}(P)\} (Q[o/y]).$$

τ -action

If $b \in \text{src}(P)$ and $b = \tau.Q$, then

$$P \xrightarrow{\tau} (P \setminus b) \parallel \nu\{n \mid (b, n) \in \text{edge}(P)\} (Q).$$

replication 1

$P \xrightarrow{\alpha} P'$ if $!Q = b \in \text{src}(P)$, and $P \parallel \nu\{n \mid (b, n) \in \text{edge}(P)\} Q' \xrightarrow{\alpha} P'$, where Q' is a program obtained from Q by renaming all names in $\text{snk}(R)$ to distinct fresh names that do not occur elsewhere in P

nor programs executed in parallel with P , for all R 's where each R is a program that occur in Q (including Q itself).

replication 2

$P \xrightarrow{\tau} P'$ if $!Q = b \in \text{src}(P)$ and $P \parallel \nu\{n \mid (b, n) \in \text{edge}(P)\} (Q'_1 \parallel Q'_2) \xrightarrow{\tau} P'$, where each $Q'_i (i = 1, 2)$ is a program obtained from Q by renaming all names in $\text{snk}(R)$ to distinct fresh names that do not occur elsewhere in P nor programs executed in parallel with P , for all R 's where each R is a program that occur in Q (including Q itself).

output

If $b \in \text{src}(P)$, $b = a\langle v \rangle$ then, $P \xrightarrow{a\langle v \rangle} P \setminus b$.

communication

If $b_1, b_2 \in \text{src}(P)$, $b_1 = a\langle o \rangle$, $b_2 = a\langle x \rangle.Q$ then,

$$P \xrightarrow{\tau} ((P \setminus b_1) \setminus b_2) \parallel \nu\{n \mid (b_2, n) \in \text{edge}(P)\} \nu(\text{fn}(o) \cap \text{snk}(P))(Q[o/x]).$$

The set of rules for a first-order program P is defined as the subset of the above rules that the object that occurs in **input**, **output** or **communication** rule is a name and **-conversion** rule is eliminated.

We can show that for any programs P, P' and any action such that $P \xrightarrow{\alpha} P'$, if P is local then P' is local and if P is normal then P' is normal.

Weak bisimulation equivalence is defined as usual also.

We denote $Q \stackrel{\hat{\alpha}}{\approx} Q'$ if and only if $Q \xrightarrow{\tau} \xrightarrow{\tau} \xrightarrow{\alpha} \xrightarrow{\tau} \xrightarrow{\tau} Q'$ or $Q' \xrightarrow{\tau} \xrightarrow{\tau} \xrightarrow{\alpha} \xrightarrow{\tau} \xrightarrow{\tau} Q$ and $Q = Q'$.

Definition 13 (weak bisimulation equivalence) A binary relation \mathcal{R} on normal programs is a *weak bisimulation* if: for any $(P, Q) \in \mathcal{R}$ (or $(Q, P) \in \mathcal{R}$), for any α and P' if $P \xrightarrow{\alpha} P'$ then there exists Q' such that $Q \stackrel{\hat{\alpha}}{\approx} Q'$ and $(P', Q') \in \mathcal{R}$ ($(Q', P') \in \mathcal{R}$ respectively). *Weak bisimulation equivalence* \approx is defined as follows: $P \approx Q$ iff $(P, Q) \in \mathcal{R}$ for some weak bisimulation \mathcal{R} .

The following proposition is straightforward from the definition.

Proposition 1 If $\text{src}(P_1) = \text{src}(P_2)$ then $P_1 \approx P_2$.

3. Scope Equivalence

Definition 14 For a program P and a name n such that $n, P/n$ is the program defined as follows:

$$\begin{aligned} \text{src}(P/n) &= \{b \mid b \in \text{src}(P), (b, n) \in \text{edge}(P)\}, \\ \text{snk}(P/n) &= \text{snk}(P) \setminus \{n\} \end{aligned}$$

and

$$\text{edge}(P/n) = \{(b, a) \mid b \in \text{src}(P/n), a \in \text{snk}(P/n), (b, a) \in \text{edge}(P)\}.$$

Intuitively P/n is the subsystem of P that consists of behaviors which are in the scope of n . Let P be an example of Fig. 1, P/a_1 is the subgraph of Fig. 1, obtained by removing the node of b_3 (and the edge from b_3 to a_2) and a_1 (and the edges to a_1) as shown in Fig. 2.

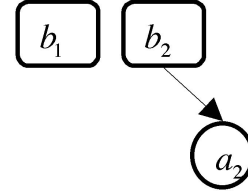


Fig 2. The graph P/a_1

Definition 15 (scope bisimulation) A binary relation \mathcal{R} on programs is *scope bisimulation* if for any $(P, Q) \in \mathcal{R}$,

- 1) $P = \mathbf{0}$ iff $Q = \mathbf{0}$,
- 2) P/n is an empty graph iff Q/n is an empty for any $n \in \text{snk}(P) \cap \text{snk}(Q)$,
- 3) $P/n \approx Q/n$ for any $n \in \text{snk}(P) \cap \text{snk}(Q)$ and
- 4) \mathcal{R} is a weak bisimulation.

It is easy to show that the union of all scope bisimulations is a scope bisimulation and it is the unique largest scope bisimulation.

Definition 16 (scope equivalence) The largest scope bisimulation is *scope equivalence* and denoted as \approx .

It is easy to show from the definition that \approx is an equivalence relation. The motivation and the background of the definition of \approx is reported in [4], [6], [7]. The following results for first-order programs can be shown as [6].

Proposition 2 If P and Q are first-order and $P \approx Q$ then

- 1) $P \parallel R \approx Q \parallel R$ for any first-order program R ,
- 2) $R[P] \approx R[Q]$ for any first-order τ -context $R[]$,
- 3) $R[P] \approx R[Q]$ for any first-order replication context $R[]$ and
- 4) $R[P] \approx R[Q]$ for any first-order input context $R[]$.

From **Proposition 2**, we have the following result.

Theorem 1 For any P and Q such that $P \approx Q$ and for any first-order context $R[]$, $R[P] \approx R[Q]$.

4. The Higher-Order Model and The First-Order Model

We can show that both of strong bisimulation equivalence and weak bisimulation equivalence are congruent wrt

input prefix context and application context also[8], [9]. Unfortunately, it is not the case for strong scope equivalence for higher-order programs[7]. We can also show that “ \approx ” is not congruent wrt input context nor application context. The essential problem is that “ \approx ” is not congruent wrt substitutions of abstractions as the following counter example shows.

Example 1 (i) Let P be a graph such that $\text{src}(P) = \{b_1, b_2\}$, $\text{edge}(P) = \{(b_1, n_1), (b_2, n_2)\}$ and $\text{snk}(P) = \{n_1, n_2\}$ and Q be a graph such that $\text{src}(Q) = \{b\}$, $\text{edge}(Q) = \{(b, n_1), (b, n_2)\}$ and $\text{snk}(Q) = \{n_1, n_2\}$ where both of b and $b_i (i = 1, 2)$ are $!x\langle a \rangle$ as Fig. 3. Note that $n_j (j = 1, 2)$ does not occur in b nor $b_i (i = 1, 2)$.

Lemma 1 Let P and Q be as **Example 1 (i)**. Then we have $P \approx Q$.

proof: (outline) **Definition 15**, 1 is obvious as neither P nor Q is an empty graph. For $n_j (j = 1, 2)$, both of P/n_j and Q/n_j are not \emptyset , so **Definition 15**, 2. holds. For 3. P/n_j is the graph such that $\text{src}(P/n_j) = \{b_j\}$ and Q/n_j is the graph such that $\text{src}(Q/n_j) = \{b\}$. As $b_i = b = !x\langle a \rangle$, $\text{src}(P/n_j) = \text{src}(Q/n_j)$. From **Proposition 1**, $P/n_j \approx Q/n_j$. For 3., it is easy to show that the relation $\{(P, Q)\}$ is a bisimulation because $P \xrightarrow{x\langle a \rangle} P$ is the only transition for P and $Q \xrightarrow{x\langle a \rangle} Q$ is the only transition for Q respectively.

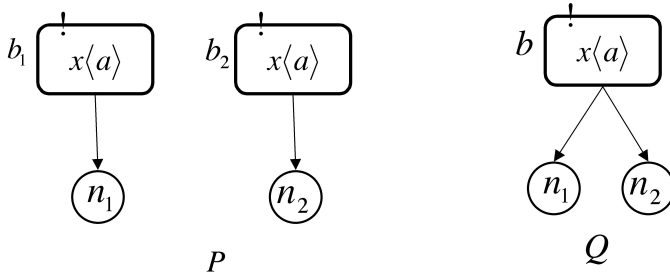


Fig. 3. Graph P and Q

Example 1 (ii) Let P and Q be as **Example 1(i)**. Now, let o be an abstraction $:(y)c(u).d(v).R$ where R is a program. $P[o/x]$ is the graph such that $\text{src}(P) = \{b_1[o/x], b_2[o/x]\}$, $\text{snk}(P) = \{n_1, n_2\}$ and $\text{edge}(P) = \{(b_1[o/x], n_1), (b_2[o/x], n_2)\}$ as the top of Fig. 4 and $Q[o/x]$ (Fig. 5. top) is a graph such that $\text{src}(Q) = \{b[o/x]\}$, $\text{snk}(Q) = \{n_1, n_2\}$ and $\text{edge}(Q) = \{(b[o/x], n_1), (b[o/x], n_2)\}$ where $b[o/x]$ and $b_i[o/x] (i = 1, 2)$ are $!(y)c(u).d(v).R\langle a \rangle$.

Note that the object o in the counter example is an abstraction. This happens only in the case of higher-order substitution. In fact, scope equivalence is congruent wrt substitution for the first-order case[6].

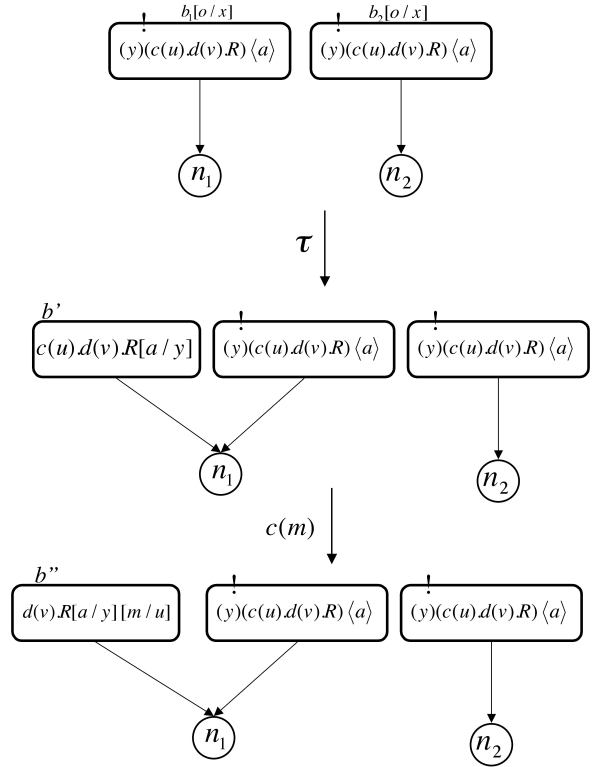


Fig. 4. Transition of $P[o/x]$

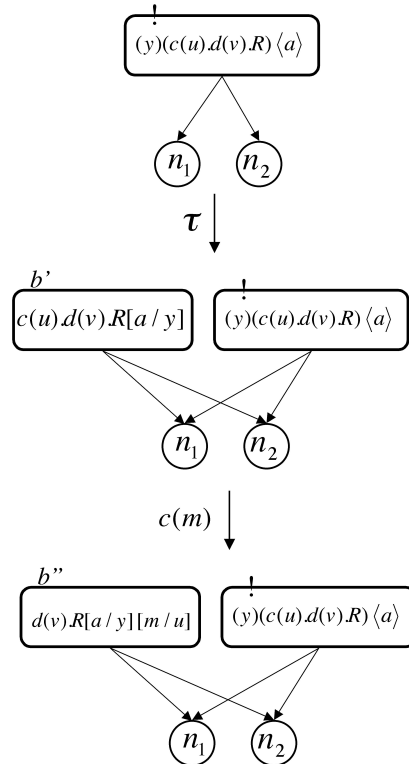


Fig. 5. Transition of $Q[o/x]$

Lemma 2 Let $P[o/x]$ and $Q[o/x]$ be as **Example 1 (ii)**.

Then, $P[o/x] \not\approx Q[o/x]$.

proof: See **Appendix**.

From **Lemma 1** and **2**, we have the following proposition.

Proposition 3 There exist P and Q such that $P \approx Q$ and $P[o/x] \not\approx Q[o/x]$ for some object o .

Proposition 4 There exist P and Q such that $P \approx Q$ but $I[P] \not\approx I[Q]$ for some input context $I[_]$.

proof: (outline) Let P and Q be as **Example 1** (i) and $I[_]$ be an input context with a behavior $m(x)._$. Consider the case of $I[P] \xrightarrow{m(o)}$ and $I[Q] \xrightarrow{m(o)}$ for o of **Example 1** (ii).

Proposition 5 There exist P and Q such that $P \approx Q$ but $A[P] \not\approx A[Q]$ for some application context $A[_]$.

proof: (outline) Let P, Q and o be as **Example 1** (ii) and $A[_]$ be an application context with a behavior $(x)_ \langle o \rangle$.

We denote a mapping from higher-order graphs into first-order graphs as $\llbracket _ \rrbracket$.

Definition 17 (full abstractness) Let \mathcal{R} be a binary relation on the set of programs. A mapping $\llbracket _ \rrbracket$ from higher-order programs into first-order programs is *full abstract* wrt \mathcal{R} if for any higher-order programs P and Q , $\llbracket P \rrbracket \mathcal{R} \llbracket Q \rrbracket$ iff $P \mathcal{R} Q$.

Definition 18 (homomorphism) A mapping $\llbracket _ \rrbracket$ is *homomorphic wrt input context* if for any input-context $R[_]$ and for any higher-order program P , $\llbracket R[P] \rrbracket = \llbracket R \rrbracket \llbracket P \rrbracket$.

Theorem 2 There is no mapping from higher-order programs into first-order programs which is full abstract wrt \approx and homomorphic wrt input context.

proof:(outline) Assume that there exists a mapping $\llbracket _ \rrbracket$ that is homomorphic wrt input context. Let P and Q be programs as **Example 1**. From **lemma 1**, $P \approx Q$. Now we assume $\llbracket P \rrbracket \approx \llbracket Q \rrbracket$ otherwise $\llbracket _ \rrbracket$ is not full abstract.

Let $R[_]$ be an input context $m(x)._$ as in the proof of **Proposition 4**. Then $\llbracket R[P] \rrbracket = \llbracket R \rrbracket \llbracket P \rrbracket$ and $\llbracket R[Q] \rrbracket = \llbracket R \rrbracket \llbracket Q \rrbracket$ because $\llbracket _ \rrbracket$ is homomorphic wrt input context. Now $\llbracket R \rrbracket[_]$ is a first-order context and $\llbracket P \rrbracket$ and both of $\llbracket Q \rrbracket$ are first-order programs. As \approx is a congruent relation wrt for any first-order context from **Theorem 1** and we assumed $\llbracket P \rrbracket \approx \llbracket Q \rrbracket$, then $\llbracket R \rrbracket \llbracket P \rrbracket \approx \llbracket R \rrbracket \llbracket Q \rrbracket$. Namely $\llbracket R[P] \rrbracket \approx \llbracket R[Q] \rrbracket$.

On the other hand, from the proof of **Proposition 4**, $R[P] \not\approx R[Q]$. Thus $\llbracket _ \rrbracket$ is not full abstract wrt \approx .

We can define the notion of *homomorphism wrt application context* similarly. Then we can also show that there is no compilation mapping that is full abstract wrt \approx and homomorphic wrt application context by the similar

argument with **Proposition 5**.

5. Conclusion

This paper presented the result that any compilation mapping from the higher-order model into the first-order model that is homomorphic wrt input context and full abstract wrt scope equivalence does not exist. We will study about this problem from the following approaches as future work.

The first approach is revision of the definition of scope equivalence. The definition of “ \approx ” is based on the idea that two process are equivalent if the components that know the name are equivalent for each name. This idea is implemented as the **Definition 15**, 3. One alternative idea for the third condition is $P/N \approx Q/N$ for each subset N of common private names instead of $P/n \approx Q/n$. P and Q in **Example 1** are not equivalent based on this definition. We should study if this alternative definition works well or not as an equivalence of programs.

The second one is reconsideration of modeling of higher-order communication. In our model, a tuple of a process variable that receive higher-order and an argument term has the same form as an output message. This idea is from LHO_π [11]. One of the main reason why LHO_π adopts this approach is type theoretical convenience. As we saw in **Lemma 2**, this identification of output messages and process variables arises the problem on the congruence of the scope equivalence. We should reconsider about the modeling of higher-order communication.

References

- [1] Ehrig, H. and B. König, Deriving Bisimulation Congruences in the DPO Approach to Graph Rewriting with Borrowed Contexts, *Mathematical Structures in Computer Science*, vol.16, no.6, pp. 1133-1163, (2006)
- [2] König, B., A Graph Rewriting Semantics for the Polyadic π -Calculus, *Proc. of GT-VMT '00*, pp. 451-458 (2000)
- [3] Milner, R., *The Space and Motion of Communicating Systems*, Cambridge University Press (2009)
- [4] Murakami M., A Formal Model of Concurrent Systems Based on Bipartite Directed Acyclic Graph, *Science of Computer Programming*, Elsevier, 61 pp. 38-47 (2006)
- [5] Murakami, M., A Graph Rewriting Model of Concurrent Programs with Higher-Order Communication, *Proc. of TMFCS 2008*, pp.80-87 (2008)
- [6] Murakami, M., Congruence Results of Scope Equivalence for a Graph Rewriting Model of Concurrent Programs, *Proc. of ICTAC2008, LNCS 5160*, pp. 243-257 (2008)
- [7] Murakami, M., On Congruence Property of Scope Equivalence for Concurrent Programs with Higher-Order Communication, *Proc of CPA2009*, IOS Press, pp. 49-66 (2009)
- [8] Murakami, M., Congruence Results of Behavioral Equivalence for A Graph Rewriting Model of Concurrent Programs with Higher-Order Communication, *Journal of Networking Technology*, Vol. 1, No.3, pp. 106-112 (2010)
- [9] Murakami, M. Congruence Results of Weak Equivalence for A Graph Rewriting Model of Concurrent Programs with Higher-Order Communication, *to appear in Proc. of ICCSEA 2011*
- [10] Sangiorgi, D. and D. Walker, *The π -calculus, A Theory of Mobile Processes*, Cambridge University Press, (2001)
- [11] Sangiorgi, D. Asynchronous Process Calculi: The First- and Higher-order Paradigms, *Theoretical Computer Science*, 253, pp. 311-350 (2001)

[12] V. Sassone and P. Sobociński, Reactive systems over cospans, Proc. of LICS '05 IEEE, pp. 311-320, (2005)

Appendix: Proof of lemma 2 (outline)

We show that for any relation \mathcal{R} , if $(P[o/x], Q[o/x]) \in \mathcal{R}$, then it is not a scope bisimulation. If \mathcal{R} is a scope bisimulation, \mathcal{R} is a weak bisimulation from **Definition 15**. Then for any $P[o/x]'$ such that $P[o/x] \xrightarrow{\alpha} P[o/x]'$, there exists $Q[o/x]'$ such that $Q[o/x] \xrightarrow{\hat{\alpha}} Q[o/x]'$ and $(P[o/x]', Q[o/x]') \in \mathcal{R}$.

From **replication 1** and **-conversion**, we have $P[o/x]'$ for τ such that: $\text{src}(P[o/x]') = \{b'\} \cup \text{src}(P[o/x])$ where $b' = c(u).d(v).R$, $\text{snk}(P[o/x]') = \text{snk}(P[o/x])$ and $\text{edge}(P[o/x]') = \text{edge}(P[o/x]) \cup \{(b', n_1)\}$ (Fig. 4. middle).

On the other hand, any $Q[o/x]'$ such that $Q[o/x] \xrightarrow{\hat{\alpha}} Q[o/x]'$ has a form such that $\text{src}(Q[o/x]') = \{b'_1, \dots, b'_h\} \cup \text{src}(Q[o/x])$ for some $h(0 \leq h)$, $b'_k = c(u).d(v).R$ for $1 \leq k \leq h$, $\text{snk}(Q[o/x]') = \text{snk}(Q[o/x])$ and $\text{edge}(Q[o/x]') = \text{edge}(Q[o/x]) \cup \bigcup_{1 \leq k \leq h} \{(b'_k, n_1), (b'_k, n_2)\}$ by h applications of **replication 1** and **-conversion**. As the following discussion is similar for any h , we consider the case of $h = 1$ (Fig. 5. middle).

If \mathcal{R} is a scope bisimulation, there exists $Q[o/x]''$ such that $Q[o/x]' \xrightarrow{\hat{c}^{(m)}} Q[o/x]''$ and $(P[o/x]'', Q[o/x]'') \in \mathcal{R}$ for any $P[o/x]'$ $\xrightarrow{c^{(m)}} P[o/x]''$. Let $P[o/x]''$ be a graph such that: $\text{src}(P[o/x]'') = \{b''\} \cup \text{src}(P[o/x]')$ where $b'' = d(v).R[m/u]$, $\text{snk}(P[o/x]'') = \text{snk}(P[o/x]')$ and $\text{edge}(P[o/x]'') = \text{edge}(P[o/x]) \cup \{(b'', n_1)\}$ obtained by applying **input rule** (Fig. 4. bottom).

The only transition of $Q[o/x]'$ that has the form of $Q[o/x]' \xrightarrow{\hat{c}^{(m)}} Q[o/x]''$ makes $\text{src}(Q[o/x]'') = \{b''\} \cup \text{src}(Q[o/x]')$ where $b'' = d(v).R[m/u]$, $\text{snk}(Q[o/x]'') = \text{snk}(Q[o/x])$ and $\text{edge}(Q[o/x]'') = \text{edge}(Q[o/x]') \cup \{(b'', n_1), (b'', n_2)\}$ (Fig. 5. bottom. It denotes the case of $\xrightarrow{c^{(m)}}$ without any τ -transitions.).

Then $(P[o/x]'', Q[o/x]'')$ is in \mathcal{R} if \mathcal{R} is a bisimulation. However, $(P[o/x]'', Q[o/x]'')$ does not satisfy the condition 3. of **Definition 15** because $P[o/x]''/n_2 \not\approx Q[o/x]''/n_2$ (Fig. 6). Even if $Q[o/x]'$ makes any number of τ -transitions, it does not make difference. Thus \mathcal{R} cannot be a scope bisimulation.

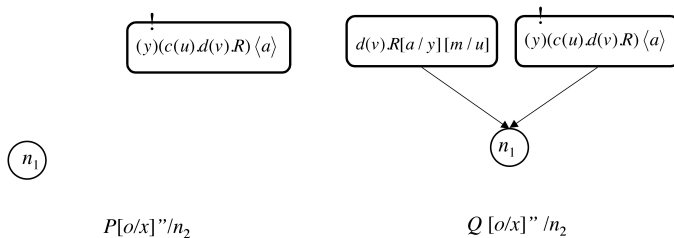


Fig. 6 $P[o/x]''/n_2$ and $Q[o/x]''/n_2$

Adding Autonomy into Object

Jamil Ahmed¹ and Sheng Yu¹

¹Department of Computer Science, University of Western Ontario, London, Ontario, Canada

Abstract - Real world objects can be classified into two kinds according to their behavior (1)autonomous objects (2)dependent objects. An object can behave both ways as well. Dependent objects are those objects which are of no use unless exploited by an external entity. Once they are created or instantiated, they keep waiting for the driver class to invoke their functions for their utilization. Example of dependent objects include a car, a calculator, a word processing application etc. Autonomous Objects are those objects which when created or instantiate, then they know by their self what they are supposed to do and then they readily start performing their task (set of methods) with possibly no external interaction or invocation. We emphasize that autonomy of object intuitively needs to have these two properties (1)Object runs its method(s) itself as soon as it is created. (2)More than one copy of object can be running simultaneously. Example of autonomous objects include a clock, a car set at cruise control, an Operating system kernel that always keeps active, a virus scan utility that always keeps active, Graphical Actors(simulation of humans) in game programming, an automatic robot, etc. We have established object calculus of autonomous object definition & object creation which incorporates the intuitive properties of autonomous objects as well. Our proposed calculus is based on the same structures as that of Abadi & Cardelli [1].

Keywords: Object Oriented Programming, Autonomy, Concurrency, Multithreading, Object Calculus.

1 Introduction

Contemporary object oriented programming languages do not yet explicitly provide any feature of “autonomy” for objects. We propose that an object in Object Oriented Programming can be defined as autonomous object. Adding the feature of autonomy to object has its own intuitive effects and introduces new abstraction to some programming languages contemporary features, while reducing the complexity of those features. Autonomous objects provide much more intuitive mechanism of programming for any autonomous computing e.g autonomous vehicles and robots, Graphical Actors(simulation of humans) in game programming, a virus scan utility that remains active all the time etc. In order to mechanize autonomous behavior as natural and intuitive, we introduce two components: (1) A compulsory run() method that will get invoked by default to start the function of autonomous object as soon as object is created. (2)When an autonomous object is created, it is by default created in its own separate and new thread. All our code examples in Figures are analogous to ‘java’ syntax.

Autonomy of objects also provide an abstraction to an important contemporary programming language feature, making Object oriented programming closer to natural way of programming and hiding much of the complexities of that language features. We propose that following feature gets new abstraction by virtue of the notion of “autonomy”.

- Concurrency (Multithreading)

As we argue that Autonomous object provide concurrency which is more intuitive and close to the concurrency of real world objects because with autonomous object we do not need to explicitly care about threads just like real world objects.

1.1 Autonomous Object Definition

We introduce a special method “run()” as a mandatory method of object definition for the object which is supposed to behave like autonomous object. This method is supposed to be invoked by constructor method of autonomous object by default. The purpose of “run()” method is to define in its body that what operations this autonomous object must start performing right after its creation. We give structure of autonomous object definition code below.

```

Class auto_class{
    .....
    Public void run()
    {..... }
    .....
    public auto_class ()
    {.....} //constructor is optional as
    usual
    .....
}

```

Fig. (A)

1.2 Autonomous Object Creation

We introduce the keyword “auto” to be used in conjunction with autonomous object creation. This “auto” keyword will force the object created as autonomous objects. Whenever an object is created with “auto” keyword, the compiler expects that the object must have special method “run()” defined in its definition. An attempt to create autonomous object using “auto” keyword will generate compile time error if the object definition doesn’t have “run()” method defined. We give autonomous object creation code below.


```

Class Driver
{
  main(){
  .....
  auto_class auto objA =new auto_class ();
  ..... }
}

```

Fig (B)

If the object definition have “run()” method defined but the object is not created with “auto” keyword then there will be no compiler error. Object creation without “auto” keyword will lead to usual object i.e non-autonomous or dependent object creation. The object created without “auto” keyword will not invoke the “run()” method, if there is any.

2 Concurrency by Virtue of Autonomy

Although the feature of concurrency is already provided by contemporary programming languages but this feature is provided by introducing additional and distinct entity rather than a built-in feature of object which makes it language based feature rather than a built counterpart of object. Those distinct entity (e.g thread) are then applied on objects and this is how objects can exploit the feature of concurrency so far. Although concurrency with the help of distinct entity like thread also gives a certain level of autonomy to Objects but as we propose autonomy as a built in feature of objects which cause concurrency to be a rather naturally associated and inherent to autonomous objects. Consequently the concurrency feature of programming languages will be under the hood of autonomous objects. Once we have autonomous objects, these independent autonomous objects will be well suited to inherently have concurrency capabilities.

Hence , notion of autonomous object provide new abstraction to thread such that each new instance of autonomous object will be created in its new and separate thread. This notion helps us to get rid of explicitly thinking in terms of thread and creating threads on our own. All we need to think about is Autonomous object. We won't need to know new language based ways (e.g thread libraries) and constructs or syntax for implementing and exploiting concurrency as we do by now in contemporary programming languages. With autonomous object, creating new thread and its handling won't be the responsibility of programmer any more.

This new abstraction will also be conducive in hiding many contemporary issues of multithreading (explicitly creating and destroying thread, races between the threads, deadlocks etc). It means multithreading will then become an inherent part of autonomous objects. All the principles and practices of concurrent programming (such as races between the threads, locks, deadlocks etc) will remain intact. The only differences will come up is that the thread management will be taken care of by autonomous object. Hence, the difference

will appear in the view point by achieving higher abstraction to concurrent programming.

When an autonomous object is created, it is by default created in its own separate and new thread. The keyword “auto” instructs the compiler that the objects created will run in separate thread. Hence, this reserve word causes the compiler to create a new thread by default so that this object is run on top of that thread.

The object created without auto keyword will not invoke the “run()” method and a separate thread will not be created.

3 Architecture of Autonomous Object and Concurrency

We propose architecture of Autonomous object and its concurrency in Fig (C). We introduce a new built in class of object oriented system in the compiler called “Autonomous” class. “Autonomous” class works in collaboration with built in “thread” class. Whenever an object is created with “auto” keyword, it gets inherited by the built in “Autonomous” class by default. The “auto” keyword invokes its “run()” method from within its constructor. It also enforces to inherit the autonomous object from a special class “Autonomous”. This “Autonomous” class in turn create a new thread object within it, using “Has a” inheritance, on top of which a newly created autonomous object will run. The “run()” method of autonomous object will by default invoke the “super.run()” instruction to override the “run()” method of “Autonomous” class.

Each autonomous object, when created, by default creates a “thread” object internally and hides thread level details inside it, thus providing a single and higher abstract level of concurrency. In other words we can say each autonomous object is created on top of a thread object to ensure autonomy. This thread is called primary thread of the object. Figure (C) gives our proposed architecture for any object definition say “auto_class” when an object of this class is created as an autonomous object.

To exploit autonomous object we show a driver class with “main()” function in which we can create one or more object as autonomous object. Each of those objects when created will get functional in separate threads synchronously. Compiler will take care of thread creation responsibility. The “driver” class and “auto_class” are user defined whereas the “autonomous” class and “thread” class are built in class which gets associated with the “auto_class” by the compiler.

By virtue of autonomy we have introduced a new abstraction for multithreading. Now we can exploit more than one autonomous objects to perform their operation concurrently. When two instances of same autonomous object are created, it is equivalent to two threads performing an operation concurrently.

In section 6 we show examples of contemporary code of multithreading (Example2-code#1) and equivalent code proposed by us (Example2-code#2) according to the notion of concurrency by autonomous objects.

```

Class Thread
{ ....
    run(){....}
    Synchronized(Object obj){....} //this
    //method will lock the object's methods to
    //be called by two threads at the same time
    isAlive(){....} //determines whether a
    //thread is still running
    Join(){....} //causes the main thread //(from
    where it is invoked) to wait until //the child
    thread terminates and "joins" main thread.
    ....
}
Class Autonomous
{
    ...
    Public Thread thread;
    Autonomous()
    { thread=new Thread(); }
    ...
    run()
    { thread.run(); .... }
    ...
}
Class auto_class
{
    ...
    run() // super.run() is invoked by default
    { ....}
    auto_class () // constructor
    { ..... }
    ...
}
Class driver
{
    ...
    auto_class auto objA=new auto_class ();
    //creating objA with "auto" keyword will
    //force "auto_class" to get implicitly
    //inherited from "Autonomous" class.
    ....
}

```

Fig (C)

4 Motivation of Concurrency by Autonomous Objects

Autonomous Objects will be the primary and base entity for concurrency instead of thread. Autonomous object notion provides higher abstraction to the widely varying language constructs and libraries of thread. Since concurrency is now represented by Objects at the higher abstract level, it will be very intuitive to define the soundness of Object calculi for autonomous object with built in concurrency feature. We won't need to introduce a separate entity, within the calculus, to represent threads unlike most of the alternate object calculus which introduces new constructs within calculus to incorporate thread and concurrency as in [3] , [4].

Objects, as an abstract entity for concurrency, represent more natural point of view for multithreading much closer to the concurrency point of view of real world objects as illustrated ahead by some multithreading scenario.

5 Single Threaded Autonomous Object

Example 1 illustrates a single threaded autonomous object. Main thread of driver class creates only one object with "auto" keyword to get only one new thread for autonomous object.

5.1 Example 1

In this example we have given object definition of an Autonomous "AutoPrinter" object. As soon as an autonomous object is created, the autonomous printer object is supposed to start printing autonomously without any external request. Within run() method we have defined the startprinting() method. When object is created using "auto" keyword then AutoPrinter object gets implicitly inherited by built in class "Autonomous", a new thread is created on top of which this object gets functional and "run()" method is invoked, by default, implicitly and synchronously from within the constructor of AutoPrinter. The implicit call to run() is taken care of by the compiler.

```

Example 1
class AutoPrinter{
    String name;
    int i=0;
    AutoPrinter () {
        System.out.println("Auto. Thread started ");
    }
    public void run() {
        StartPrinting();
    }
    Public StartPrinting(){
        While(i<100){
            System.out.println("Print in progress");
        }
    }
}
class Driver{
    public static void main (String args[]){
        System.out.println("Main thread started");
        AutoPrinter auto objA =new AutoPrinter ();
        System.out.println("Main thread terminated");
    }
}

```

Fig (D)

6 Synchronization

While exploiting multithreading, there are times, when more than one thread share the same resource. More than one thread can invoke the same method of that resource at the same time. Obviously only one of them should be allowed to

access the resource/method at one time. In this case we need to synchronize the threads.

Example2, code#2 illustrates a multithreaded autonomous object as two object are created with “auto” keyword. In this Example, we show by comparison that how autonomous object serves perfectly well in code#2 as an alternate of contemporary multithreading technique in code#1. Code#2 hides all thread level management code so that we can best appreciate the simplicity of code#2 and realize the abstraction of “concurrency by virtue of Autonomy”.

6.1 Example 2

```

Example 2—Code#1
class Printer {
    void Printlist (String s) {
        System.out.print ("printing long list for"+s);
    }
    try {
        Thread.sleep (1000);
    } catch (InterruptedException e) {
        System.out.println ("Interrupted");
    }
    System.out.print ("printing long list Ends for"+s);
}
}
class CompThread implements Runnable {
    String s1;
    Printer p1;
    Thread t;
    public CompThread (Printer p2, String s2) {
        p1= p2;
        s1= s2;
        t = new Thread(this);
        t.start();
    }
    public void run() {
        synchronized(p1){
            p1.Printlist(s1);
        }
    }
}
class Driver{
    public static void main (String args[]) {
        Printer p3 = new Printer();
        CompThread name1 = new CompThread (p3, "Bob");
        CompThread name2 = new CompThread (p3, "Mary");
        try {
            name1.t.join();
            name2.t.join();
        } catch (InterruptedException e ) {
            System.out.println( "Interrupted");
        }
    }
}

```

Fig (E)

Fig (E) gives a scenario using contemporary techniques of concurrency in java. The method “Printlist” of printer is

shared by two computer threads. We have explicitly synchronized the call to this method so that both threads do not intermingle their execution of this method. Synchronization statement clocks the invocation of this method by other thread as long as the execution of this method by first thread is under process.

Fig (F) gives alternate solution of this problem by exploiting the inherent power of autonomous object of our proposal. In code#1 we have to explicitly care about creating threads and using shared resource within the threads and then synchronizing. Where as in code #2 we can simply define the shared printer as an autonomous object and every time a new autonomous object is created, compiler will itself take care of the new thread creation issues.

This example best realizes the significance of abstraction provided by the notion of Autonomy. We can see that we gain same multithreading in code#2 as in code#1 but we don't even need to explicitly think about thread creation in code#2. All thread creation, for the sake of implementation, is done internally under the abstraction of autonomous object.

```

Example 2—Code#2
class AutoPrinter {
    public string pname;
    public void run() {
        thread.synchronized(){
            this.Printlist(pname);
        }
    }
    public AutoPrinter (string nm){    pname=nm;    }
    public void Printlist (String s) {
        system.out.print ("printing long list"+s);
        try {
            Thread.sleep (1000);
        } catch (InterruptedException e) {
            System.out.println ("Interrupted");
        }
        System.out.print ("printing long list Ends");
    }
}
class Demo{
    public static void main (String args[]) {
        try {
            AutoPrinter auto p1 =new AutoPrinter("Bob");
            AutoPrinter auto p2 =new AutoPrinter("Mary");
        } catch (InterruptedException e ) {
            System.out.println( "Interrupted");
        }
    }
}

```

Fig (F)

7 Calculus

A class is an object definition used to generate object. Pre-methods are the method definitions which becomes methods once embedded into objects as mentioned in [1]. A class is a collection of pre-methods together with a method called “new” for generating new objects. Class in the terminology of calculus is written as below:

$$c \triangleq [new = \bar{\sigma}(z)[l_i = \bar{\sigma}(s)z. l_{i(s)}^{ic\ 1\dots n}], \\ l_i = \lambda(s)b_i^{ic\ 1\dots n}]$$

The method $new = \bar{\sigma}(z)[l_i = \bar{\sigma}(s)z. l_{i(s)}^{ic\ 1\dots n}]$, applies the pre-methods of class to the self of the object, thereby converting the pre-methods into methods.

Given any class “c”, the invocation $c.new$ produces an object “o” as below and given in [1].

$$o \triangleq c.new = [l_i = \bar{\sigma}(x_i)b_i^{ic\ 1\dots n}]$$

7.1 Calculus for Autonomous Object

In our setting, as we have defined in section 3, in order to make a class behave as autonomous, it must be inherit from a parent “Autonomous” class. In our calculus we call it “c_super_auto” class and formally given as below

$$c_super_auto \triangleq [new = \bar{\sigma}(z)[l_i = \bar{\sigma}(s)z.l_{i(s)}^{ic\ 1\dots n}, \\ \bar{\sigma}(s)z.run(s)], thread=b, \\ run=\lambda(s)(s.thread:=s.thread.new), \\ l_i = \lambda(s)b_i^{ic\ 1\dots n}]$$

- $thread=b$ stands for $thread= \bar{\sigma}(s)b$, for an unused s because $thread=b$ is a field.
- $run=\lambda(s)(s.thread:=s.thread.new)$, A new instance of thread is created, so that each autonomous object can run in this new and separate thread.
- $(s.thread:=s.thread.new)$ is the body of run method.
- new method not only applies the pre-methods of class to the self of the object but also invoke the run method.
- run is also a special method similar to new method. A new thread instance is created within run method. Hence a new thread instance is created before new method is returned.
- $l_i = \lambda(s)b_i^{ic\ 1\dots n}$ represents all pre-methods of “c_super_auto” class.

In order to make a class behave as autonomous, it must be inherit from a parent “c_super_auto” class. As soon as object of a class is created with “auto” keyword, the class by default gets inherited from “c_super_auto” class.

Compiler is supposed to enforce this by default inheritance. In our calculus, we call the inherited class “c_auto” and formally given as below:

$$c_auto \triangleq [new = \bar{\sigma}(z)[l_i = \bar{\sigma}(s)z. l_{i(s)}^{ic\ 1\dots n+m}, \\ \bar{\sigma}(s)z.run(s)], \\ run = \lambda(s)c_super_auto.run(b_r)(s), \\ l_i = c_super_auto.l_j^{jc\ 1\dots n}, \\ l_k = \lambda(s)b_k^{kc\ n+1\dots n+m}]$$

- “c_auto” as an inherited class can reuse all the pre-methods of “c_super_auto”.
- $(c_super_auto.l_j^{jc\ 1\dots n})$ are the pre-methods of “c_super_auto” inherited into “c_auto”.
- $l_k = \lambda(s)b_k^{kc\ n+1\dots n+m}$ are more pre-methods peculiar to “c_auto”.
- $run = \lambda(s) c_super_auto.run(b_r)(s)$ shows that run is the inherited but over ridden pre-method which also invoke its parent’s run pre-method.
- $c_super_auto.run(b_r)(s)$ is the body of run pre-method of “c_auto”.
- $c_super_auto.run$ is the part of run pre-method body which is invoking the run method of parent.
- (b_r) is the remaining body of run pre-method which represent any custom user defined code.
- (s) is the self parameter of “c_auto” also passed on to $c_super_auto.run$

In our calculus an autonomous object “ao” is created from “c_auto” class is formally given as below:

$$ao \triangleq c_auto.new = [b_r \{ \{x < ao\} \}, l_i = \bar{\sigma}(x_i)b_i^{ic\ 1\dots n+m}]$$

- $b_r \{ \{x < ao\} \}$ shows that run method is invoked from within the $c_auto.new$ i.e as soon as “ao” is created.
- $l_i = \bar{\sigma}(x_i)b_i^{ic\ 1\dots n+m}$ shows the methods embedded into “ao” corresponds to the pre-methods of “c_auto” class. These methods include the ‘m’ pre-methods of c_auto as well as ‘n’ pre-methods of c_super_auto .

8 Conclusion

Our proposed autonomous object notion is compatible with all contemporary Object Oriented Programming techniques. It is not new programming paradigm. According to our proposed syntax when an Autonomous object is defined, its object can still be created as a usual, as a non-

autonomous object, without any extra care. Autonomous object provides better abstraction over thread and concurrency and is also sound in Object calculus as we have shown.

9 References

- [1] M.Abadi and L.Cardeli, "A Theory of Objects", Springer, New York, 1996. Chapter 6.
- [2] Michael Papathomas, Dimitri Konstantas, "Integration concurrency and Object Oriented Programming. An Evaluation of Hybrid"
- [3] Andrew D. Gordon, Paul D. Hankin, University of Cambridge Computer Laboratory, Cambridge, UK, "A Concurrent Object Calculus: Reduction and Typing".
- [4] Alan Jeffrey, DePaul University, "A Distributed object calculus", December 1999, Proc. FOOL 2000
- [5] Suresh Jagannathan!, Jan Vitek, Adam Welc, Antony Hosking, April 2005, Dept. of Comp. Sci., Purdue University, USA "A transactional object calculus".
- [6] Jonathan Aldrich Joshua Sunshine Darpan Saini Zachary Sparks, School of Comp. Sci., Carnegie Mellon University, OOPSLA 2009, "Typestate-Oriented Programming".
- [7] Haitong Wu, Sheng Yu, Dept. of Comp. Sci, Univ. of Western Ontario, "Adding states into Object Types".
- [8] Haitong Wu, Sheng Yu, Dept. of Comp. Sci, Univ. of Western Ontario, 2006 Elsevier, "Type Theory and Language Constructs for Objects with States".

Type Disjointness in the Language of Effective Definitions

Jarred Blount and J. Nelson Rushton

Dept of Computer Science, Texas Tech University

Abstract - This paper presents portions of the LED type system. LED types include integer, boolean, variable, set, tuple, function, and union types as well as user-defined recursive types. To accommodate overloading of function symbols in LED program, while ensuring deterministic evaluation of LED expressions, definitions of function symbols must not be ambiguous. The concept of type disjointness is utilized to determine if the declared domains of LED definitions are disjoint. An algorithm is presented for determining if two types are disjoint, that is, have any inhabitants in common. The algorithm is shown to be sound and complete a restricted form of the LED type system.

Keywords: functional programming, type safety

1 Introduction

The Language of Effective Definitions (LED) is a formal language for defining computable functions. It allows recursive definitions employing rational arithmetic, finite sets and tuples, and quantification and set comprehension restricted to finite sets. Since these are well known operations and their LED syntax is visually similar to their traditional informal syntax, we will not define the language formally here. For current purposes, an *informal* reading of function definitions is identical to its formal reading. For complete semantics see (Rushton and Blount 2011).

In this paper, the goal is to develop an algorithm that will be used to allow function symbols to be overloaded for arguments of different types, while guaranteeing at compile time that definitions are not ambiguous. For expressivity, we include variable (unknown or undetermined), set, tuple, function, and union types as well as user defined recursive types. The restrictions we are assuming at this time are that recursive type are explicitly defined by type rules, and hence are regular. We also prohibit mutually recursive types. This paper only introduces an overview of planned type system and one statically checkable type relationship, *disjointness*.

Related worked includes many systems that include one some of the features our system, and exclude others. Union types in programming languages, while having long been played a role in program analysis (Palsberg and Pavlopoulou, 1999), have been featured in few programming languages

(notably algol 68; cf. van Wijngaarden et al., 1978). More recently, they have being applied in the context of type systems for “semi-structured” database formats such as XML (Buneman and Pierce, 1998; Hosoya, Vouillon, and Pierce, 1991). Tuple and function types have long been featured in the simply typed lambda calculus. Basic properties of recursive types are were established in (MacQueen, Plotkin, and Sethi, 1986).

This paper is organized as follows, section 2 introduces the LED types along with some example LED expressions of each type, then more presented in detail in section 3. Section 4 presents the semantics of the LED type system. Section 5 presents the algorithm for determining type disjointness, and its proof of correctness wrt to a restricted form of the LED type system.

2 Simple Examples

The simplest of LED expression are the atomic expressions symbols, strings of digits, and decimal fractions. For example, the LED symbols `true`, and `false` are of type `boolean`. The LED numerals `0` and `256` are of type `integer`, and the LED decimal fractions `3.14` and `1.23` are `scalar`.

Second, are the literals for sets and tuple, following the normal conventions, `{1,2,3,52}` and is a set of integers denoted by `{integer}`, and `{}` is of type `{X}`, meaning the empty set is an object or inhabitant of any set type. The tuple `(1,2)` is a 2-tuple of integers denoted by `integer*integer`, and the tuple `(1.2,2.3,3,true)` is a 4-tuple denoted by `(scalar, scalar, integer, boolean)`.

The built-in function symbol, `+` is a function from a 2-tuple, or pair, of `integer` to an `integer` denoted by `integer*integer->integer`, but is also overloaded in that `+` is also a function from a 2-tuple of `scalar` to a `scalar` denoted by `scalar*scalar->scalar`.

User defined types can given by type rules. For example a user defined type `number`, could be defined by a type rule `number := integer | scalar`, intuitively meaning that any object of type `number` is exactly those objects that are either `integer` or `scalar`. Lists of `integer` can be described using the LED tuple operator. For example,

$(1, (2.3, (5, \text{empty})))$ is a valid LED tuple expression, and is an `intlist` using the type rules `intlist ::= empty | integer * intlist`, and built-in type `empty`.

3 Definitions and Syntax

A *type lexicon* is a 4-tuple $(B, D, V, <, disj)$ where B, D, V are disjoint finite sets of symbols, and $<$ and $disj$ are binary relations on B for *subtype* and *disjoint*. B, D , and V are the *built-in*, *defined*, and *variable* type symbols, respectively, of $(B, D, V, <, disj)$. The operators $\{\}$, \times , $\>$, and $|$ are called the *set*, *tuple*, *function* and *union type constructors* respectively. For any positive integer i , T_i is a meta-variable for any type, t_i is a meta-variable for any type symbol, and L is a meta-variable for a type lexicon. When no ambiguity will be introduced “over type lexicon L ” will be omitted from subsequent occurrences of a definition.

A *type* over a type lexicon L is one of the following:

- T_i where T_i is a type symbol of L
- $\{T_i\}$ where T_i is a type
- $T_1 \times \dots \times T_n$ where $n > 2$
- $T_1 | \dots | T_n$ where $n > 2$

Several later definitions will make use of view that types are abstract syntax trees. In this view, leaf nodes are labeled by type symbols, and non-leaf nodes will be labeled by type constructors or type symbols. If the root node of, (the abstract tree representation), of type T , is R , then T is called a *R type*. Likewise, if the root node of, (the abstract tree representation), of type T , is *not* R , then T is called a *non-R type*. For example, the types `int`, `int|bool`, $\{\text{int}\}$, `int` \times `int` are a built-in type, set type, union type, tuple type respectively. If the root is labeled by a type constructor, the type is a *constructed type*.

A *type rule* over L is an expression of the form $T_1 ::= T_2$, where T_1 is a defined type symbol, and T_2 is either a type or LED symbol. T_1 and T_2 are called the head and body of $T_1 ::= T_2$ respectively. A type rule is $T_1 ::= T_2$ *well-founded* if

- (1) T_2 is a union type, and
- (2) There is at least one child of T_2 containing only built-in type symbols, non-union type constructors, and
- (3) Each occurrence of T_1 in T_2 occurs within a set or tuple type constructor.

For example, the type rules $p ::= \{\} | \{p\}$, and $pp ::= \{\} | \{pp\}$ are well-founded. Examples of non simply well-founded type rules include $q ::= \{q\}$, which violates condition (1), $q ::= \{q\} | q \times \text{int}$, which violates condition (2), and $q ::= q | q \times \text{int}$ which violates condition (3). A *grammar* over L is a finite set G of type rules over L , such that each defined symbol of L occurs as the left side of

one rule in G . We consider grammars containing only well-founded rules.

);

4 Semantics

A *literal* is one of the following:

- (1) a symbol (string or non-white-space characters not beginning with a digit and containing no left or right parenthesis),
- (2) a numeral,
- (3) a decimal fraction,
- (4) (l_1, \dots, l_n) where $n > 1$ and each l_i is a literal, or
- (5) $\{l_1, \dots, l_n\}$ where $n > 1$ and each l_i is a literal, or

Cases (1), (2), and (3) are called *atomic literals*. Cases (4) and (5) are *set* and *tuple literals* respectively, and are *compound literals* collectively. For any positive integer j , l_j is a meta-variable for an arbitrary literal. Equality is defined in the obvious way for all literals.

Given a type lexicon $L = (B, D, <, disj)$, a *model* of L maps each built-in type symbol T_1 of L to a set of atomic literals $M(T_1)$ such that for any type T_2 of L :

1. $M(T_1)$ is a subset of $M(T_2)$ whenever $T_1 < T_2$, and
2. $M(T_1)$ is disjoint from $M(T_2)$ whenever $disj(T_1, T_2)$.

Given a lexicon L , a model M of L , and a grammar G over L . $inhab(L, M, G)$ is the smallest set S of type judgments $l : T$ (where l is a literal and T is a type over L) such that:

1. If $l \in M(T)$ then $l : T \in S$
2. If $l_i : T \in S$ for all $1 \leq i \leq n$, then $\{l_1, \dots, l_n\} : T \in S$
3. If $l_i : T_i \in S$ for all $1 \leq i \leq n$, then $(l_1, \dots, l_n) : T_1 \times \dots \times T_n \in S$
4. If $l : T_i \in S$ for some $1 \leq i \leq n$ then $l : T_1 | \dots | T_n \in S$
5. If $T_1 ::= T_2$ is a rule in G , and $l : T_2 \in S$ then $l : T_1 \in S$.

Definition 1. Disjoint types

Given a type lexicon L , grammar G over L , and model M of L , types T_1 and T_2 over L are *disjoint* if $\forall x, y .$ if $x : T_1 \in inhab(L, M, G)$ and $y : T_2 \in inhab(L, M, G)$ then $x \neq y$.

5 Type Disjointness algorithm

Given a type theory L with lexicon $(B, D, <, disj)$ grammar G over L , we give an algorithm by defining an piecewise effective binary predicate D over types of L .

Definition 2. Algorithm D

$D(T_1, T_2)$ if any of the following:

When either T_1 or T_2 is a built-in type

- (1) $disj(T_1, T_2)$, or
- (2) T_1 is a built-in type, and T_2 is a set or tuple, or
- (3) T_2 is a built-in type, and T_1 is a set or tuple,

When either T_1 or T_2 is a union type

- (4) T_1 is a union type, and for each child of T_i of T_1 $D(T_1, T_2)$, or
- (5) T_2 is a union type, and for each child of T_i of T_2 $D(T_1, T_1)$, or

When both T_1 and T_2 are set or tuple types

- (6) If both T_1 and T_2 are tuple types, and every pair of corresponding children, $D(T_{1i}, T_{2i})$, or
- (7) If both T_1 and T_2 are set types with children T_i and T_j , respectively, and $D(T_i, T_j)$,
- (8) If T_1 is a set type and T_2 is a tuple type.
- (9) If T_2 is a set type and T_1 is a tuple type.

When either T_1 or T_2 are defined types

- (10) If T_1 is defined by $T_1 ::= T_3$, and $D(T_3, T_2)$, or
- (11) If T_2 is defined by $T_2 ::= T_3$, and $D(T_1, T_3)$, or
- (12) If T_1 and T_2 are distinct defined type symbols defined by $T_1 ::= T_3$ and $T_2 ::= T_4$ and $D(T_3, T_4)$,

The following theorem is the main result of this paper, and says that types T_1 and T_2 are disjoint if and only if $C(T_1, T_2)$.

Theorem 1. Given a type lexicon L , with no defined type symbols, an empty grammar G over L , a model M of L , and types T_1 and T_2 over L , $D(T_1, T_2)$ if and only if T_1 and T_2 are disjoint.

Proof of Theorem 1 (only if)

Suppose $D(T_1, T_2)$, the proof proceeds by strong induction over the sum of the heights of the (syntax tree representation of) T_1 and T_2 . Both the base case and induction step will then proceed by cases, having one case for each branch of the algorithm. The cases for branches (10), (11), (12) are vacuously true, ie. they never apply, because the grammar G is empty.

In the base case T_1 and T_2 are built-in type symbols, hence only branch (1) of the disjoint algorithm need be examined. From the definition of the algorithm D follows $\text{disj}(T_1, T_2)$, it follows from the definition of model, that $M(T_1)$ and $M(T_2)$ are pair-wise disjoint sets. Consequently, T_1 and T_2 are disjoint.

The induction step will proceed by cases, one case for each branch of the definition of the disjoint algorithm. Notice that in the cases for branches (2), (3), (7), and (9) that no recursive call is made. We appeal directly to the definition of LED equality and definition of literal.

Suppose that case (2) applies, it can be shown that no atomic literal is equal to any tuple literal or set literal. A similarly argument will be used in case (3). In the cases (7) and (9), it can be shown that that no set literal is equal to any tuple literal, and so T_1 and T_2 are disjoint.

The cases for the remaining branches (4), (5), (6), (7), will require the induction hypothesis. In each of these branches D is applied to a single child, or children of either T_1 and T_2 , so T_1 and T_2 are disjoint by the induction hypothesis and definition of disjoint.

Proof of Theorem 1 (if)

Suppose T_1 and T_2 are disjoint. The proof proceeds by strong induction over the sum of the heights of the (syntax tree representation of) T_1 and T_2 . Both the base case and induction step will then proceed by cases of possible combinations of labels of the root of T_1 and T_2 . While there are at least 16 cases, $\{ \text{built-in, set, tuple, union} \}^2$. All built-in type symbols can be handled by a single meta-variable for an arbitrary built-in type symbol.

For the base case, the roots of T_1 and T_2 are labeled by built-in type symbols, and so $\text{disj}(T_1, T_2)$ is true by definition of a model. Hence $D(T_1, T_2)$ by branch (1) of the definition of D .

For the induction step, the proof will proceed with the following cases.

Suppose the root of T_1 is labeled by a built-in type symbol and T_2 is labeled by set or tuple, then $D(T_1, T_2)$ is true by branch (3) of the definition of D .

Suppose the root of T_2 is labeled by a built-in type symbol and T_1 is labeled by set or tuple, then $D(T_1, T_2)$ is true by branch (2) of the definition of D .

Suppose the root of T_1 is labeled by set and T_2 is labeled by tuple, then $D(T_1, T_2)$ is true by branch (8) of the definition of D .

Suppose the root of T_2 is labeled by set and T_1 is labeled by tuple, then $D(T_1, T_2)$ is true by branch (9) of the definition of D .

Suppose that both T_1 or T_2 are labeled with set, and the child of T_1 is T_{1c} and child of T_2 is T_{2c} . It can be shown that T_{1c} and T_{2c} are disjoint by the definition of LED equality between sets and the definition of disjoint types. $D(T_{1c}, T_{2c})$ follows using the induction hypothesis. A similar argument is used if both T_1 and T_2 are labeled with tuple.

Suppose that T_1 are labeled by union. For every child T_{1c} of T_1 , T_{1c} and T_2 are disjoint by the definition of disjoint types. $D(T_{1c}, T_{2c})$ follows using the induction hypothesis. A similar argument is used if T_2 is labeled with union.

Definition 2. Given a type lexicon L , and a grammar G over L , G is *stratified* if there exists a function f from type symbols of L to non-negative integers such that:

1. $f(T) = 0$ if T is a built-in type symbol, and
2. $f(T_1) > f(T_2)$ if $T_1 ::= T_3 \in G$, T_2 is a leaf of T_3 .

Conjecture 1. Given a type lexicon L , with stratified grammar G over L , a model M of L , and types T_1 and T_2 over L , $D(T_1, T_2)$ if and only if T_1 and T_2 are disjoint.

6 Conclusion and future work

The main theorem is critical to planned future work, in particular a proof of conjecture 1, that is, to extend algorithm D to user defined recursive types, and then incorporate function types, extend LED type system semantics to include function types, show LED to be type-safe. Well-typed LED will contain unambiguous function definitions that are correct wrt their declared signatures. Finally, we work to include type inference (and not require explicit type signatures), and parameterized user defined types such as `list(int)`, lists of integers.

7 References

- [1] Buneman, Peter and Benjamin Pierce. Union types for semistructured data. In *Internet Programming Languages*. Springer-Verlag, September 1998. Proceedings of the International Database Programming Languages Workshop. LNCS 1686.
- [2] Jim, Trevor and Jens Palsberg. Type inference in systems of recursive types with subtyping. Manuscript, 1999.
- [3] MacQueen, David, Gordon Plotkin, and Ravi Sethi. An ideal model for recursive polymorphic types. *Information and Control*, 71: 95–130, 1986.
- [4] Rushton, J., Blount, J. The Language of Effective Definitions. *International Conference on Frontiers in Education: Computer Science and Computer Engineering*.
- [5] van Wijngaarden, Adriaan, B. J. Mailloux, J. E. L. Peck, C. H. A. Koster, M. Sintzoff, C. H. Lindsey, L. G. L. T. Meertens, and R. G. Fisker. Revised report on the algorithmic language ALGOL 68. *Acta Informatica*, 5 (1–3): 1–236, 1975.

Performance Evaluation of the EXT4 File System

A Comparative Study Against EXT3, ReiserFS and JFS

Alireza Ghobadi¹, Amir Hesam Yaribakht¹, Sanam Maham², Mohammad Hossein Ghods²
¹Faculty of Information Technology Multimedia University Cyberjaya 63000 Selangor, Malaysia
²SOHA Sdn. Bhd., AB1 MSC Center, Cyberjaya, Selangor, Malaysia

Abstract— *Base on several definition of File System, File System is a system which is responsible to handle files and managing data in any operating system [1]. According to these definitions, choose a File System to managing data on your system, is one of the consideration for anybody who use a computer. Anybody can fill this problem when you have a lot of files with a large portion. File System types can be classified into disk File Systems, network File Systems and special purpose File Systems[2][3][4.] The purpose of this Paper is “comparison Performance evaluation among four File Systems”. According to above goals, there are several file systems on Linux operation system. These file system are EXT2, EXT3, EXT4, ReiserFS, JFS, and etc. In this paper, we are neither define several file systems on Linux nor compare them and tested by IOzone and Postmark benchmark tools [5][6]. The performance result has shown base on read, re-read, write and re-write of file for IOzone benchmark tool and create, read, append, delete for Postmark benchmark tool. We have chosen IOzone bench mark for our experiments as it is preferred for operating system evaluation [7].*

Keywords- Performance Evaluation, File System, EXT4, EXT3, ReiserFS

1 Introduction

Base on several definition of file system, File system is a system which is responsible to handle files and managing data in any operating system [1]. According to these definitions, choose a good file system with the best performance is one of the considerations [1].

File system types can be classified into disk, network and special purpose. A disk file system designed for the file storage on a data storage device. Normally disk drive, directly or indirectly connected to the computer or any computer device.

The principal aims of file system are to address scalability, performance, reliability, capabilities and robustness. On the other hand, the most popular Linux file system due to its reliability, rich feature set, relatively good performance, and strong compatibility between versions of file system.

According to above goals, there is several file system on Linux operating system. These file system are EXT2, EXT3, EXT4, ReiserFS, JFS, and etc [7][8][9][10][11][12].

In this paper, we are neither define several operating systems on Linux nor compare them and tested by IOzone and Postmark benchmark tools. We will be showing performance of them based on read, re-read, write and re-write of file.

We have chosen IOzone and Postmark benchmark tools for our experiments as it is preferred for operating system evaluation [13][14].

1.1 Instructions for authors

This paper, compares and evaluates EXT4 against EXT3 and ReiserFS, JFS file systems on Linux. The paper studies on these four file system in section 2, 3, 4. Related work in section 5. Then, it focuses on performance evaluation tools in section 6. Implementation of benchmark tools in section 7. Results and discussion in section 8, and finally, conclusion and future work in section 9.

2 Extent file systems

Extent file system was earlier file system that developed on Linux by Rmy Card, Laboratoire. Until now Extended has four generations and implement on several version of Linux [10].

EXT2 designed by Wayne Davidson and Stephen Tweedie and Theodore. They extended from EXT file system which designed by Remy Card and implement on the standard Linux file system [15][16].

EXT3 is one of the traditional Unix-derived file systems. It used a indirect block mapping scheme to keep track of each block. It the same data structures and supports journaling. EXT3, which just added some features to EXT2 while keeping the on-line format and approach of EXT2 [9].

EXT4 was developed by Theodore Ts'o, who was, at the time In 2006. The uber Linux developer, which developed the EXT3 maintainer, began work on EXT4., EXT4 changed a deep code change and the data structures. These changes used

to make a better file system, faster, reliable, more features, and better code. The most important and hard working on EXT4, added new features such as, Extents, journaling check summing, block allocation, delayed allocation, faster fsck, on-line defragmentation, and larger directory sizes (up to 64,000 files) [12].

3 ReiserFS

ReiserFS was developed as a part of the standard Linux kernel by Hans Reiser. [2] It is available on the most version of Linux operating system. ReiserFS supports metadata journaling. The ReiserFS has excellent performance for small-files. ReiserFS Developed based on B* Balanced Trees to organize directories, files, and data. B* provides fast directory lookups and fast deletes operations. Other performance features include support for sparse files and dynamic disk inode allocation [2].

4 Journalin File Systems (JFS)

JFS introduced by IBM as UNIX file system with the initial release of AIX Version 3.1. It has now introduced a second file system that is to run on AIX systems called Enhanced Journal File System (JFS2). JFS2 is available in AIX Version 5.0. The JFS Open Source code on originated [17].

JFS is modified primarily for the high throughput and reliability requirements of servers. JFS uses extent-based addressing structures, along with clustered block allocation policies. It is make compact, efficient, and scalable structures for mapping logical offsets within files to physical addresses on disk. An extent is a sequence of contiguous blocks allocated to a file as a unit. The addressing structure is a B+Tree populated with extent descriptors, rooted in I-node and keyed by logical offset within the file [17].

JFS supports block sizes of 512, 1024, 2048, and 4096 bytes on a per-file system basis. Smaller block sizes reduce the amount of internal fragmentation. However, small blocks can increase path length since block allocation activities may occur more often than if a large block size was used. The default block size is 4096 bytes [18].

JFS supports both sparse (which allow data to be written to random locations within a file without instantiating others unwritten file blocks.) and dense files, on a per-file system basis. [18]

5 Related Work

Some researcher studied on Extents file systems.[12][13][14] Avantika Mathur have worked on EXT3. The purpose of their research was to provide branch of EXT4 from EXT3.[4] They compared EXT3, EXT4; XFS file systems with three tools as FFSB, IOzone and Postmark. With FFSB tool they test these file systems base on throughput (MB/Sec) and CPU percent usage. [13] [14]

This test has shown that XFS has higher throughput (MB/Sec) than EXT3 and EXT4, but it has lower CPU percent

usage than EXT3 and EXT4. EXT4 has higher throughput (MB/Sec) performance than EXT3, but it has lower CPU percent usage than EXT3 [9]. By IOzone tool they test these file systems base of six operations as write, re-write, read, re-read, random write and random read [11]. In this test shows that in Write, re-write, random write and random read, EXT4 has higher throughput (KB/Sec) performance than XFS and in general EXT4 has higher throughput (KB/Sec) performance in all six operations than EXT3[9].

The test also shows that in read and re-read operations XFS has higher throughput (KB/Sec) performance than EXT3 and EXT4. Also observe that in re-write, random write, random read, XFS has higher throughput (KB/Sec) performance than EXT3. In write operation EXT3 has higher throughput (KB/Sec) performance than XFS. With Postmark tool they test these file systems based on two operations as read and write. The test result not only shows that EXT4 has higher throughput (MB/Sec) performance than EXT3 and XFS, but also EXT3 has higher throughput (MB/Sec) performance than XFS. In their comparison they find that EXT4 has a good improvement of EXT3 and has become an enterprise-ready solution, with a good balance of scalability, reliability, performance and stability. [2]

Other researcher named, Ricardo Galli works on journal file systems available for Linux as EXT3, ReiserFS, XFS and JFS and they introduce to the basic concepts of file systems, buffer-cache, and page-cache carried out in the Linux kernel. [17]Their performance result shows that XFS, ReiserFS and EXT3 have demonstrated that they are excellent and reliable file systems. In this research, they achieved (i) EXT3 is going to be the standard file system for Linux operating system, specially Red Hat, (ii) JFS is a valid alternative for migrating AIX and OS/2 installation to Linux.

(iii) In all journal file systems, ReiserFS is the only file system which has standard Linux tree since 2.4.1 which SuSE supports it. (iv) XFS is being used in large servers (especially in the Hollywood industry). It is due mainly to the influence of SGI market. (v) JFS has gotten the worst results (when tested by any benchmarks) not only on performance, but also for stability issue in the Linux port.

Dr. Oliver Diedrich has a well done study on EXT3 and EXT4 file systems. He compares the structure of EXT3 and EXT4 file systems base on large volumes, huge files and extent trees. He evaluates a performance of these two file systems based on creation (based on time and write speed) and deletion (based on time) of eight 1 GB files and 10000 random read and write operations in 8 GB. He did not mention what tools he used in his test, the performance with large files between EXT3 and EXT4. The tests shows that in creation of eight 1 GB files, time in EXT4 improved 6.9% and write speed also improved 7.0% than EXT3. In deletion of eight 1 GB files, time improved 97.2% than EXT3. And among 10000 random read and write operations in 8 GB, EXT4 improved 10.9% than EXT3[11].

6 Performance Evaluation Tools

Benchmark is a tool for performance evaluation. There are several benchmarks for file system available [13]

IOzone is one of the famous benchmark tools on file system to generate and measures a selection of file operations. It has been runs for test many operating systems. The IOzone tests file I/O performance. I/O performance tests based on Read, re-read, read backwards, read strided, write, re-write, fread, fwrite, random read/write, pread/pwrite variants, aio_read, aio_write, mmap. It is useful for file system analysis of a vendor's computer platform. [13]

Another famous benchmark is The Postmark. It is responsible to creating a massive bulk of alternatively modifying files and calculating the transaction rates for a workload approximating a large Internet electronic mail server [14]. Postmark operation, produces random text files. The text files size categorizes from low bound to high bound. The size is configurable between low and high bound The text file pool is of configurable size and can be located on any accessible file system. [14] Once the bulk has been created consists of a pair of smaller transactions (i) Create file or Delete file and (ii) Read file or Append file operation.

According to comparison of these benchmark tools, the achievement the IOzone performance shows that, this tool is more suitable for experimental result due to the performance of IOzone is higher than Postmark tool.

7 Implementation

File system benchmarking requires careful setup. An issue one must often contend with is how to defeat the effects of the file system buffer cache. Without careful experimental design, all of the file system requests could be satisfied in the cache and no disk activity would occur. A usual way to avoid this problem is to use a total file size that exceeds the amount of main memory available on the system.

Another approach is to use a file-access mode that bypasses the file system buffer cache, such as `O_DIRECT`. We chose to not use `O_DIRECT` for this paper

The second issue that one must address is estimating the accuracy of the results of the test. In our experience, file system benchmarks are notorious for being non repeatable, bimodal, and full of hysteresis effects, making it a challenge to get consistent results.

In this paper we have *IOzone and Postmark* benchmarks using below system:

- The small system that is Intel(R) processor Core(TM) Duo 2.20 GHz with 4GB of memory and a 320 GB SATA disk. For the experiments of this paper, this machine was booted with 4 GB of RAM.
- Nowadays, mentioned machine provides a sampling used to run Linux.
- For using IOzone and Postmark benchmarks firstly we should install them in the system.

- Now searching is in process and gives the latest model of the searched benchmark.

8 Results and Discussion

This section is a description of test result. Each graph defines system's structure.

8.1 IOzone result

Read graph shows that EXT4 has not good performance in those file size less than 100 MB. EXT4 is Extent base allocation. It is block allocation and it use contiguous allocation to allocate the file in blocks of disk. Because of this type of allocation, EXT4 has overhead and performance is not good on small file. In other hand, small file size because it is contiguous when file finish there are some space in the contiguous file that still empty. The other reason is the overhead for read the file system should refer to directory that has file name. Address of file and length of the file and if the file be small, this cause overhead happened. [13]

Allocation features use for large file size. Figure 4 shows that the performance of EXT4 is higher than other file systems. For the large file, EXT4's performance is higher than other file systems because of block allocation that fill the contiguous block in disk.

EXT3 allocates blocks for a file one at a time (typically using 4KB blocks). For very large files, the associated function that doses the allocation will have to be called thousands of times. EXT4 uses "multi-block allocation". It allows multiple blocks (hence the name) to be allocated during one function call. This can greatly improve the performance of EXT4 relative to EXT3, particularly for large files. [12]

JFS dynamically allocate space for disk I-nodes as required, freeing the space when it is no longer needed. Two different directory organizations are provided. (i) The first organization is used for small directories and stores the directory contents within the directory's I-node. This eliminates need for separate directory block I/O as well as need to allocate separate storage. [18] By using directory's I-node can eliminate separate directory block I/O and allocate separate storage. (ii) Organization is used for larger directories and represents each directory as a B+Tree keyed on name. It provides faster directory lookup, insertion, and deletion capabilities.

Because of these reasons JFS in small file has better performance than EXT4 but for file size from 100MB and above JFS has lower performance than EXT4. [12]

ReiserFS uses B* Balanced Trees to organize directories, files, and data. This provides fast directory lookups. [2] So it has better performance on read operation in small file size generally less and equal than 20MB file size.

First graph shows result base on read feature. Figure1 shows that experience by IOzone benchmark.

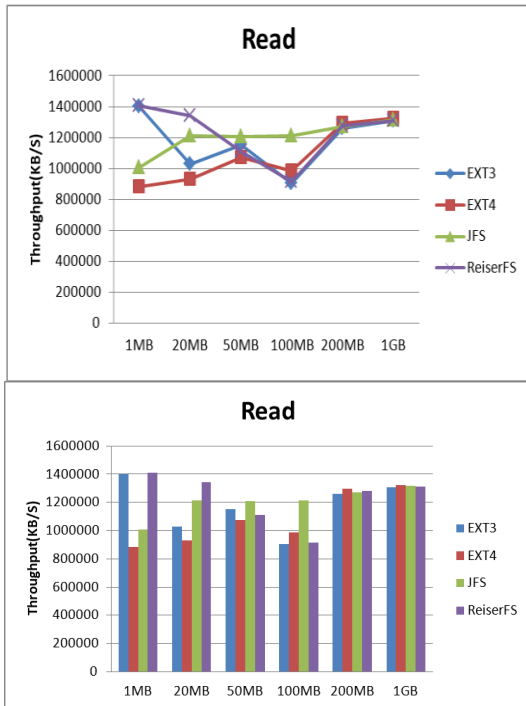


Figure 1: IOzone test for Read feature

According to above explanation, Figure1 shows that EXT4 has weakness in small file. Also this graph shows that ReiserFS and EXT3 have a good performance with small file. But with increasing file size, EXT4 performance increase as well. Also this graph shows that JFS more stable than other file system.

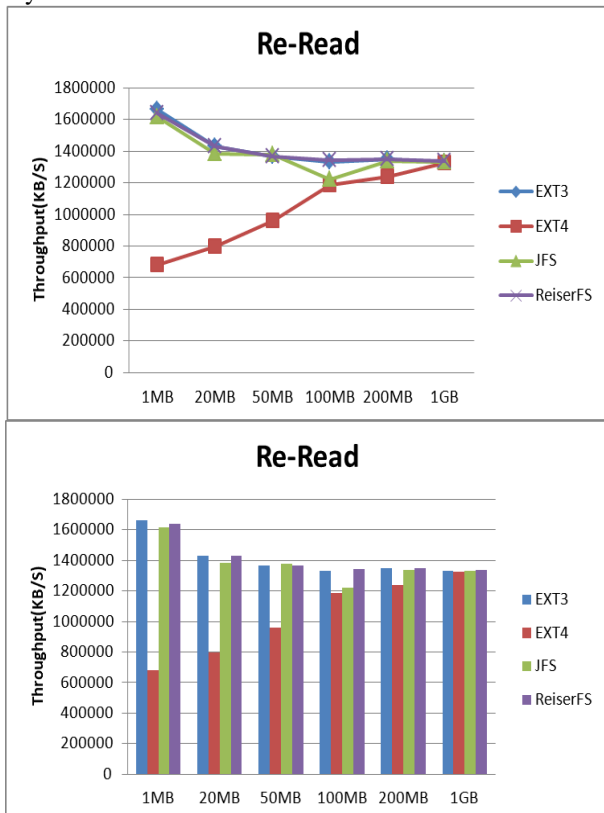


Figure 2: IOzone test for Re-Read feature

Figure2 shows that EXT3, ReiserFS, and JFS has a good performance when file size is small. But with increasing file size, JFS and EXT3 performance decrease slowly and EXT4 performance increase. Figure2 Shows with large file size JFS has higher performance.

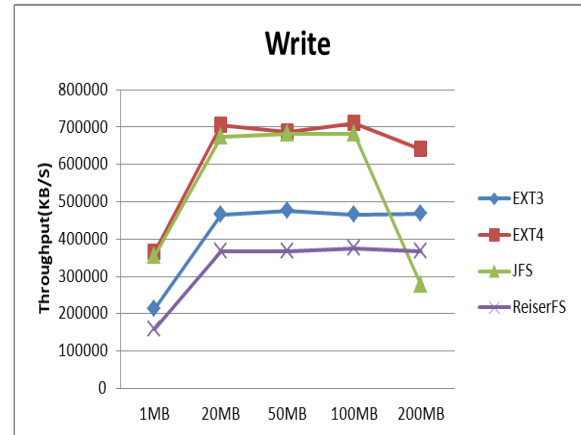


Figure3: IOzone test for Write feature

Figure3 shows write performance in these four file system. Figure3 shows that EXT4 and JFS has a good performance with small file size. ReiserFS has the lowest performance. But with increasing file size the EXT4 performance is not change too much, but JFS performance decreasing too much. In large file size JSF has the worst performance.

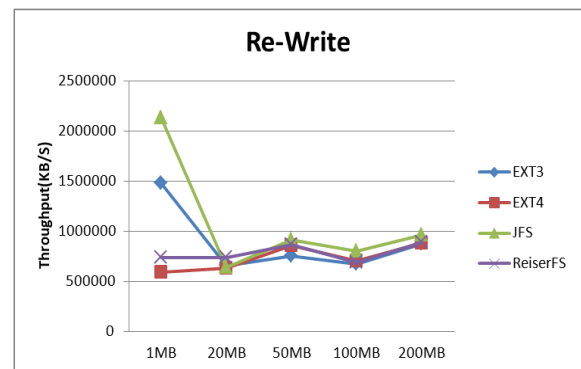


Figure 4: IOzone test for Re-Write feature

Figure4 shows Re-write feature that tested by four file system. For this feature, JFS has the best performance base on small file size and large file size. This graph shows that EXT4 performance slowly increase and JFS performance deeply decrease.



Figure5: All feature performance for small file size (1MB).

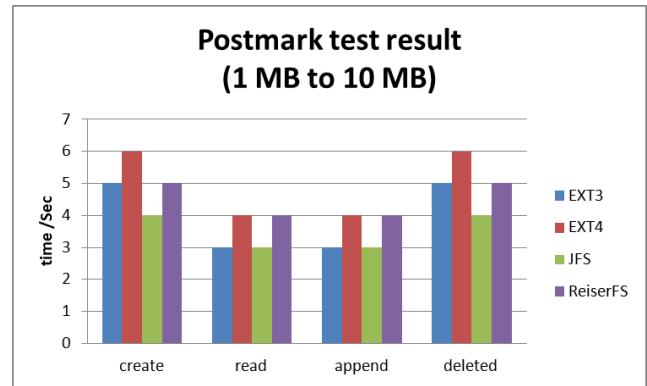


Figure7: Postmark test result base on creation, append, deletion, and read.



Figure6: All feature performance for large file size (100MB).

Figure 5 and Figure 6 shows other perspective of performance of these four file system. With comparison four feature performance in small file size; JFS has a good performance in Re-Write, and Re-Read. Also the worth performance in small file size belong to EXT4. Comparison between EXT3 and EXT4 which are using extent, EXT3 is much better than EXT4.

Also according to large file size, EXT4 mostly has a better performance than others. Specially, on their write file. JFS also has a good performance on read file.

Postmark result

According to postmark description, run postmark on four file systems. (EXT4, JFS, EXT3, and ReiserFS) Postmark used maximum create, read, append, and delete files base on Table 1.

Table1: Four file systems comparison table with postmark.

1MB-10MB	Create	Read	Append	Delete
Maximum Byte	749	247	253	749
EXT3	5	3	3	5
EXT4	6	4	4	6
JFS	4	3	3	4
ReiserFS	5	4	4	5

} Byte/Sec

Figure7 shows result of postmark testing. This graph shows that EXT4 is the fastest file system in all of the execution part. Also ReiserFS are in the second level.

1. CONCLUSION AND FUTURE WORK

Choose a file system to managing data on your system, is one of the consideration for anybody who use a computer. Anybody can fill this problem when you have a lot of files with a large portion [20].

This project define base on famous file system comparison on Linux. Project starts which a research on file system on Linux and then find some research on these file systems.

There is some evaluator tools look like FFSB, IOzone, and Postmark. Benchmark which using IOZONE and POSTMORK which is more famous. These tools are choosing to check which one is more reliable for evaluation for my project. Some of the Research objectives in this project are carrying out as follow (i) Assess different type of FS on Linux. Assess different type of file system performance evaluator tools (ii) Categorizes file system base in performance.

The result shows that EXT4 which is using extend have a performance on large file size but it is not suitable for small file size. In other word, with increase file size, the EXT4 performance increase. Other file systems (which is JFS and using journaling) also has a good performance (especially in write and Re-write file). But JFS is not update any more during 2 years. EXT3 also has a reliable performance on small file size.

9 Reference

- [1] Bryant, R., Forester, R., & Hawkes, J. (n.d.). File system Performance and Scalability in Linux 2.4.17. Proceedings of the FREENIX Track:2002 USENIX Annual Technical Conference. USENIX.
- [2] Sun Microsystems (2004). File System Performance: The Solaris™ OS, UFS, Linux EXT3, and ReiserFS. Aug.
- [3] How to Find the Block Size. (2005, Aug 18). Retrieved Oct 17, 2009, from LINFO: http://www.linfo.org/get_block_size.html

- [4] Terminal Window Definition. (2005, May 1). Retrieved Nov 9, 2009, from The Linux Information Project: http://www.linfo.org/terminal_window.html
- [5] Inode Definition. (2006, Sep 15). Retrieved Jan 5, 2010, from The Linux Information Project: <http://www.linfo.org/inode.html>
- [6] Boyne, J. (2005). Disc and Volume Size Limits.
- [7] Diedrich, O. (2009, May 29). The Ext4 Linux File System. Retrieved Dec 20, 2009, from the Open: <http://www.h-online.com/open/features/The-Ext4-Linux-file-system-746579.html>
- [8] Henson, V., Brown, Z., Ts'o, T., & van de Ven, A. (2006). Reducing fsck time for EXT2 file systems. Linux Symposium, p. 395.
- [9] Ts'o, T. (2002). Planned extensions to the Linux EXT2/EXT3 File system. USENIX 2002 Annual Technical Conference, Freenix Track , pp. 235–244 .
- [10] Tweedie, S. (98). Journaling the Linux EXT2fs File system. LinuxExpo.
- [11] Y. Ts'o , T., & Stephen, T. (2002, June 10). Planned extensions to the Linux EXT2/EXT3 File system . USENIX Association.
- [12] Layton, J. (2009, March 28). EXT4 File System: Introduction and Benchmarks. Retrieved Dec 29, 2009, from Linux mag: <http://www.linux-mag.com/id/7271/1>
- [13] Norcott., W. (98). IOzone File system Benchmark. Retrieved Sep 15, 2009, from IOzone: http://www.IOzone.org/docs/IOzone_msword_98.pdf
- [14] Katcher, J. (97, 10 8). Postmark: A New File System Benchmark. Retrieved Dec 1, 2009, from <http://www.netapp.com/technology/level3/3022.html>: <http://communities.netapp.com/servlet/JiveServlet/download/2609-1551/Katcher97-postmark-netapp->
- [15] Y. Ts'o , T., & Stephen, T. (2002, June 10). Planned Extensions to the Linux Ext2/Ext3 Filesystem . USENIX Association.
- [16] Tweedie, S. (98). Journaling the Linux ext2fs Filesystem. LinuxExpo.
- [17] Galli Granada, P. (2002, Jan 1). Journal File Systems in Linux. p. 6.
- [18] Steve. (2000, Jan 1). JFS overview . Best works in the Software Solutions & Strategy Division of IBM in Austin.
- [19] Norcott., W. (98). Iozone Filesystem Benchmark. Retrieved Sep 15, 2009, from iozone: http://www.iozone.org/docs/IOzone_msword_98.pdf
- [20] Stepohen, S. (2010). Novell makes file storage software shift. Retrieved Dec 23, 2009, from

SESSION
QUANTUM COMPUTING + AUTOMATA

Chair(s)

TBA

On The Power Of Distributed Bottom-up Tree Automata

Kamala Krithivasan¹ and Ajeesh Ramanujan¹

¹Department of Computer Science and Engineering
Indian Institute of Technology Madras, Chennai - 36
kamala@iitm.ac.in, ajeeshramanujan@yahoo.com

Abstract—Tree automata have been defined to accept trees. Different types of acceptance like bottom-up, top-down, tree walking have been considered in the literature. In this paper, we consider bottom-up tree automata and discuss the sequential distributed version of this model. Generally, this type of distribution is called cooperative distributed automata or the blackboard model. We define the traditional five modes of cooperation, viz. **-mode*, *t-mode*, $= k$, $\geq k$, $\leq k$ ($k \geq 1$) modes on bottom-up tree automata. We discuss the accepting power of cooperative distributed tree automata under these modes of cooperation. We find that the **-mode* does not increase the power, whereas the other modes increase the power. We discuss a few results comparing the acceptance power under different modes of cooperation.

Keywords: Tree Automata, ranked alphabet, distributed nondeterministic tree automata, modes of cooperation

1. Introduction

Finite tree automata are generalizations of word automata. While a word automaton accepts a word, a tree automaton accepts a tree. The theory of tree automata arises as a straight forward extension of the theory of finite automata [6]. Tree automata were introduced in [4], [5] and [12] to solve certain decision problems in logic. Since then they were successfully applied to many other decision problems in logic and term rewriting, see e.g. [1]. Even though the two models are used in different settings they are closely related to each other since a finite automaton can be seen as a special case of a finite tree automaton. Trees appear in many areas of computer science and engineering and tree automata are used in applications such as XML manipulation, natural language processing, and formal verification and logic design.

According to the manner in which the automaton runs on the input tree, finite tree automata can be either bottom-up or top-down. A top-down tree automaton starts its computation at the root of the tree and then simultaneously works down the paths of the tree level by level. The tree automaton accepts the tree if such a run can be defined. A bottom-up tree automaton starts its computation in the leaves of the input tree and works its way up towards the root.

A finite tree automaton can be either deterministic or non-deterministic. This is an important issue since deterministic top-down automata are strictly less expressive than non-deterministic top-down automata. For the bottom-up case,

deterministic bottom-up tree automata are just as powerful, from the point of view of language equivalence, as non-deterministic bottom-up tree automata. Non-deterministic top-down tree automata are equivalent to non-deterministic bottom-up tree automata [1].

In the last few years distributed and parallel computing has played an important role in Computer Science. Modelling these concepts using formal models has given rise to the concept of grammar systems and distributed automata. Grammar systems can be sequential or parallel. A co-operating distributed (CD) grammar system is sequential. Here, all grammars work on one sentential form. At any instant only one grammar is active. This is called a blackboard model. Suppose a problem is to be solved in a class. The teacher asks one student to start working on the problem on the blackboard. The student writes a few steps, then goes back. Another student comes and continues working on the problem. On his return, a third student comes and continues. The process continues till the problem is solved. Now, the question arises: at what time does one student return and the next one starts? There may be several ways for defining this. Correspondingly, in the CD grammar system, there are different modes of co-operation. The student may return when he is not able to proceed further (terminating mode); he may return at any time (**-mode*); he may return after doing k -steps ($= k$ -mode); he may return after doing k or less steps (\leq -mode); he may return after doing k or more steps (\geq -mode).

In this paper, we consider bottom-up tree automata and discuss the sequential distributed version of this model. We define the traditional five modes of cooperation, viz. **-mode*, *t-mode*, $= k$, $\geq k$, $\leq k$ ($k \geq 1$) modes on bottom-up tree automata. We discuss the accepting power of cooperative distributed tree automata under these modes of cooperation. We find that the **-mode* does not increase the power, whereas the other modes increase the power. We discuss a few results comparing the acceptance power under different modes of cooperation.

In the next section we give basic definitions needed for the paper. Section 3 contains the definition of cooperative distributed tree automata and some results about their accepting power. The paper concludes with a note in section 4.

2. Basic Definitions

Let N be the set of positive integers. Then the set of finite strings over N is denoted by N^* . The empty string is denoted by ϵ . A *ranked alphabet* Σ is a finite set of symbols together with a function $Rank : \Sigma \rightarrow N$. For $f \in \Sigma$, the value $Rank(f)$ is called the rank of f . For any $n \geq 0$, we denote by Σ_n the set of all symbols of rank n . Elements of rank $0, 1, \dots, n$ are respectively called constants, unary, \dots , n -ary symbols.

A *tree* t over an alphabet Σ is a partial mapping $t : N^* \rightarrow \Sigma$ that satisfies the following conditions:

- $dom(t)$ is a finite, prefix-closed subset of N^* , and
- for each $p \in dom(t)$, if $Rank(t(p)) = n > 0$, then $\{i | pi \in dom(t)\} = \{1, 2, \dots, n\}$.

Each $p \in dom(t)$ is called a *node* of t . The node with domain element ϵ is the *root*. For a node p , we define the i^{th} *child* of p to be the node pi , and we define the i^{th} *subtree* of p to be the tree t' such that $t'(p') = t(pip')$ for all $p' \in dom(t')$. A *leaf* of t is a node p which does not have any children, i.e. there is no $i \in N$ with $pi \in dom(t)$. We denote by $T(\Sigma)$ the set of all trees over the alphabet Σ . The *size* of a tree t is the number of elements in $dom(t)$. The *height* of a tree t is $\max\{|w| : w \in dom(t)\}$. Given a finite tree t , the *frontier* of t is the set $\{p \in dom(t) | \text{for all } n \in N, pn \notin dom(t)\}$. A tree with root a and subtrees t_1, t_2, \dots, t_r is represented by $a(t_1, t_2, \dots, t_r)$.

Example 1: Let $\Sigma = \{a, b, c, g, f\}, f \in \Sigma_2, g \in \Sigma_1, a, b \in \Sigma_0$. A tree over Σ and its diagrammatic representation is shown in Figure 1

Let t be the tree $f(g(a)f(bc))$.
 $dom(t) = \{\epsilon, 1, 11, 2, 21, 22\}$.
 $size(t) = 6$.
 $height(t) = 2$.
 $frontier(t) = \{11, 21, 22\}$.

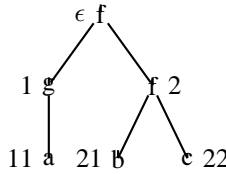


Fig. 1: A tree and its diagrammatic representation

A *nondeterministic finite tree automata* (NFTA) over an alphabet Σ is a tuple $A = (Q, \Sigma, Q_f, \Delta)$ where,

- Q is a finite set of states,
- Σ is a ranked input alphabet,
- $Q_f \subseteq Q$ is a set of final states,
- Δ is a finite set of transition rules.

Each transition rule is a triple of the form $((q_1, q_2, \dots, q_n), f, q)$ where $q_1, q_2, \dots, q_n, q \in Q, f \in \Sigma_n$,

i.e. $Rank(f) = n$. We use $f(q_1, q_2, \dots, q_n) \rightarrow q$ to denote that $((q_1, q_2, \dots, q_n), f, q) \in \Delta$. If $Rank(f) = 0$, i.e. f is a constant, then we use rules of the form $f \rightarrow q$. The epsilon rules are denoted by rules of the form $q_i \rightarrow q_j$. A *run* of A over a tree $t \in T(\Sigma)$ is a mapping $r : dom(t) \rightarrow Q$ such that for each node $p \in dom(t)$ where $q = r(p)$, we have that if $q_i = r(pi)$ for $1 \leq i \leq n$ then Δ has the rule $t(p)(q_1, q_2, \dots, q_n) \rightarrow q$. A set $B = \{q_1, q_2, \dots, q_n\} \subseteq Q, n \geq 1$ with respect to a tree $t' \in T(\Sigma \cup Q)$ is said to be an *active state set* if every $q_i = r(pi), i \geq 0$ for some $p \in dom(t)$ and $t(p) \in \Sigma$.

An *instantaneous description* (ID) of a NFTA is a pair (B, t) , where $t \in T(\Sigma \cup Q)$ and B is a set of active state set with respect to t .

For two ID's $(B, t), (B', t')$ we write $(B, t) \vdash (B', t')$ if there is a rule of the form $a(q_1, q_2, \dots, q_n) \rightarrow q' \in \Delta$ such that t' is obtained from t by replacing a subtree of t of the form $a(t_1, t_2, \dots, t_n)$ by $q'(t_1, t_2, \dots, t_n)$, where $a \in \Sigma_n, n \geq 0, t_1, t_2, \dots, t_n \in T(Q), r(\text{root}(t_1)) = q_1, r(\text{root}(t_2)) = q_2, \dots, r(\text{root}(t_n)) = q_n, q_1, q_2, \dots, q_n \in B$ and B' is the set of active state set after performing the transition.

The initial ID is $(\phi, t), t \in T(\Sigma)$ and the final ID is $(\{q_f\}, t')$ for some $q_f \in Q_f, t' \in T(Q)$. The reflexive and transitive closure of \vdash is denoted by \vdash^* .

A run represents the effect of a sequence of ID's from the initial ID to a final ID.

For a NFTA $A, L(A) = \{t \in T(\Sigma) | (\phi, t) \vdash^* (\{q_f\}, t'), q_f \in Q_f, t' \in T(Q)\}$.

A set L of tree languages over Σ is *recognizable* if $L = L(A)$ for some NFTA A . Two NFTA are said to be *equivalent* if they recognize the same tree language.

We give an example to show that certain tree languages are not recognizable.

Example 2: Let $\Sigma = \{f, g, a\}$, where $Rank(f) = 2, Rank(g) = 1, Rank(a) = 0$. Consider the tree language $L = \{f(g^i(a), g^i(a)) | i > 0\}$. Let us suppose that L is recognizable by an automaton A having k states. Consider the tree $t = f(g^k(a), g^k(a))$. t belongs to L , therefore there is a successful run of A on t . As k is the cardinality of the state set, there are two distinct positions along the first branch of the tree labeled with the same state. Therefore, one could cut the first branch between these two positions leading to a term $t' = f(g^j(a), g^k(a))$ with $j < k$ such that a successful run of A can be defined on t' . This leads to a contradiction with $L(A) = L$.

The proof can be generalized into a theorem, similar to pumping lemma for recognizable string languages, to recognizable tree languages [1].

3. Distributed Nondeterministic Tree Automata (DNFTA)

In this section we define distributed nondeterministic tree automata (DNFTA), the different modes of acceptance

of DNTA and discuss the power of different modes of acceptance.

Definition 1: A DNTA is a 4-tuple $D = (K, \Sigma, F, \Delta)$ where,

- K is an n -tuple (K_1, K_2, \dots, K_n) where each K_i is a set of states of the i^{th} component;
- Σ is a finite set of ranked alphabet;
- $F \subseteq \bigcup_i K_i$ is the set of final states;
- Δ is a n -tuple $(\delta_1, \delta_2, \dots, \delta_n)$ of state transition function where each δ_i is a set of transition rules of the i^{th} component having the form $f(q_1, q_2, \dots, q_n) \rightarrow q$, $f \in \Sigma_n, q_1, q_2, \dots, q_n \in K_i, q \in \bigcup_i K_i$ or $q_i \rightarrow q_j$.

In the case of DNTA, we can consider many modes of acceptance depending upon the number of steps the system has to go through in each of the n components. The different modes of acceptance are $*$ -mode, t -mode, $\leq k$ -mode, $\geq k$ -mode, and $= k$ -mode, where k is a positive integer. Description of each of the above modes of acceptance is as follows:

t-mode acceptance: An automaton that has a leaf transition rule begins processing the input tree. Suppose that the system starts from the component i . The control stays in component i as long as it can follow the transition rules in component i . Otherwise, it transfers the control to some other component j , $j \neq i$ which has the transition function to proceed. If more than one component succeeds, then the selection of j is done nondeterministically. The process is repeated and we accept the tree if the system reaches any one of the final states. It does not matter which component the system is in while accepting.

Definition 2: The instantaneous description (ID) of a DNTA $D = (K, \Sigma, F, \Delta)$ working in t -mode is given by a triple (B, t, i) where $B \subseteq \bigcup_i K_i$ and it denotes the current active state set of the whole system, $t \in T(\Sigma \cup \bigcup_i K_i)$ and $i, 1 \leq i \leq n$ the index of the component in which the system is currently in.

The transition between the ID's is defined as follows:

- i) $(B, t, i) \vdash_t (B', t', i)$ if there is a rule of the form $a(q_1, q_2, \dots, q_n) \rightarrow q' \in \delta_i$ such that t' is obtained from t by replacing a subtree of t of the form $a(t_1, t_2, \dots, t_n)$ by $q'(t_1, t_2, \dots, t_n)$, where $a \in \Sigma_n, n \geq 0, t_1, t_2, \dots, t_n \in T(\bigcup_i K_i)$, $r(\text{root}(t_1)) = q_1, r(\text{root}(t_2)) = q_2, \dots, r(\text{root}(t_n)) = q_n, q_1, q_2, \dots, q_n \in B$ and B' is the set of active state set after performing the transition.
- ii) $(B, t, i) \vdash_t (B, t, j)$ iff component i does not have a transition to proceed and component j has a transition to proceed.

The reflexive and transitive closure of \vdash_t is denoted by \vdash_t^* .

Definition 3: The language accepted by a DNTA $D = (K, \Sigma, F, \Delta)$ working in t -mode is defined as follows:

$$L_t(D) = \{t \in T(\Sigma) \mid (\phi, t, i) \vdash_t^* (\{q_f\}, t', j), t' \in T(\bigcup_i K_i), \text{ for some } q_f \in F, 1 \leq i, j \leq n\}.$$

We now give an example of a distributed bottom up tree automata working in t -mode.

Example 3: Consider the language

$$L_1 = \{a(bd(g^j d)^i(f), ce(h^k e)^l(f)), i, j, k, l \geq 1, |i - l| \leq 1\} \text{ over } \Sigma = \{a, b, c, d, e, f, g, h\}, a \in \Sigma_2, b, c, d, e, g, h \in \Sigma_1, f \in \Sigma_0.$$

We define a distributed tree automaton

$$D_1 = (K, \Sigma, \{q_a\}, \Delta) \text{ working in } t\text{-mode as follows.}$$

The components are defined as follows

- Component 1
 - $K_1 = \{q_f, q_g, q_1, q_2\}$,
 - $\delta_1 = \{d(q_f) \rightarrow q_g, g(q_g) \rightarrow q_1, g(q_1) \rightarrow q_1, q_2 \rightarrow q_f\}$
- Component 2
 - $K_2 = \{q_f, q_e, q_1, q_2\}$,
 - $\delta_2 = \{e(q_f) \rightarrow q_e, h(q_e) \rightarrow q_2, h(q_2) \rightarrow q_2, q_1 \rightarrow q_f\}$
- Component 3
 - $K_3 = \{q_f, q_a, q_b, q_c, q_d, q_e, q_1, q_2\}$,
 - $\delta_3 = \{f \rightarrow q_f, b(q_g) \rightarrow q_b, c(q_e) \rightarrow q_c, a(q_b, q_c) \rightarrow q_a\}$

The processing starts in component 3, with the two leaves using the rule $f \rightarrow q_f$. As further processing is not possible in component 3, processing continues with 2 or 1. Then it alternates between 1 and 2 processing d 's, g 's, e 's and f 's. Finally when the labels are b and c , processing takes the tree to q_b and q_c and in component 3 state q_a is reached by the root.

Theorem 1: There exists a language accepted by a DNTA working in t -mode which is not recognizable.

Proof: Consider the tree language L_1 . Let us suppose that L_1 is recognizable by an automaton A having k states. Consider the tree $t = a(bd(gd)^k(f), ce(he)^k(f)), k > 0$. t belongs to L_1 , therefore there is a successful run of A on t . As k is the cardinality of the state set, there are two distinct positions along the first branch of the tree labeled with the same state. Therefore, one could cut the first branch between these two positions leading to a term $t' = a(bd(gd)^j(f), ce(he)^k(f))$ with $j < k$ such that a successful run of A can be defined on t' . This leads to a contradiction with $L(A) = L_1$. So L_1 is not recognizable. ■

**-mode acceptance:* An automaton that has a leaf transition rule begins processing the input tree. Suppose that the system starts from the component i . Unlike the termination mode, the automaton can transfer the control to any of the components at any time i.e., if there is some $j, j \neq i$ such that the next move is possible then the system can transfer the control to the component j . The selection of j is done nondeterministically if there is more than one j .

The ID and the language accepted by the system in $*$ mode, $L_*(D)$ is defined as follows.

Definition 4: The instantaneous description(ID) of a DNTA $D = (K, \Sigma, F, \Delta)$ working in $*$ -mode is given by a triple (B, t, i) where $B \subseteq \bigcup_i K_i$ and it denotes the current active state set of the whole system, $t \in T(\Sigma \cup \bigcup_i K_i)$ and $i, 1 \leq i \leq n$ the index of the component in which the system is currently in.

The transition between the ID's is defined as follows:

- i) $(B, t, i) \vdash_* (B', t', i)$ if there is a rule of the form $a(q_1, q_2, \dots, q_n) \rightarrow q' \in \delta_i$ such that t' is obtained from t by replacing a subtree of t of the form $a(t_1, t_2, \dots, t_n)$ by $q'(t_1, t_2, \dots, t_n)$, where $a \in \Sigma_n, n \geq 0, t_1, t_2, \dots, t_n \in T(\bigcup_i K_i)$, $r(\text{root}(t_1)) = q_1, r(\text{root}(t_2)) = q_2, \dots$, $r(\text{root}(t_n)) = q_n, q_1, q_2, \dots, q_n \in B$ and B' is the set of active state set after performing the transition.
- ii) $(B, t, i) \vdash_* (B, t, j)$ iff component j has a transition to proceed.

The reflexive and transitive closure of \vdash_* is denoted by \vdash_{*} .

Definition 5: The language accepted by a DNTA $D = (K, \Sigma, F, \Delta)$ working in $*$ -mode is defined as follows: $L_*(D) = \{t \in T(\Sigma) \mid (\phi, t, i) \vdash_{*} (\{q_f\}, t', j), t' \in T(\bigcup_i K_i), \text{ for some } q_f \in F, 1 \leq i, j \leq n\}$

We give an example of a distributed bottom up tree automata working in $*$ -mode.

Example 4: Consider the language

$L_2 = \{a(b^i(d), c^j(d)), i, j \geq 1\}$ over $\Sigma = \{a, b, c, d\}$, $a \in \Sigma_2, b, c \in \Sigma_1, d \in \Sigma_0$. We define a distributed tree automaton $D_2 = (K, \Sigma, \{q_f\}, \Delta)$ as follows.

The components are defined as follows

- Component 1
 - $K_1 = \{q_b, q_d\}$
 - $\delta_1 = \{b(q_d) \rightarrow q_b, b(q_b) \rightarrow q_b\}$
- Component 2
 - $K_2 = \{q_d, q_c\}$
 - $\delta_2 = \{c(q_d) \rightarrow q_c, c(q_c) \rightarrow q_c\}$
- Component 3
 - $K_3 = \{q_f, q_b, q_c, q_d\}$
 - $\delta_3 = \{d \rightarrow q_d, a(q_b, q_c) \rightarrow q_f\}$

Processing starts in component 3, with the two leaves using the rule $d \rightarrow q_d$. As further processing is not possible in component 3, processing continues with components 1 or 2. Then it alternates between components 1 and 2 processing b 's and c 's. When all the b 's and c 's are exhausted the automaton moves to component 3 and reaches the final state by using rule $a(q_b, q_c) \rightarrow q_f$. The processing of any tree in L_2 uses component 3 two times, in the first and the last step.

Theorem 2: For any DNTA D working in $*$ -mode, $L_*(D)$ is recognizable.

Proof: Let $D = (K, \Sigma, F, \Delta)$ be a DNTA working in $*$ -mode where, $\Delta = (\delta_1, \delta_2, \dots, \delta_n)$ and the

components have states K_1, K_2, \dots, K_n . Define a NFTA $N = (K', \Sigma, F', \delta)$ where,

$$K' = \{[q, i] \mid q \in \bigcup_i K_i, 1 \leq i \leq n\}$$

$$F' = \{[q_f, i] \mid q_f \in F, 1 \leq i \leq n\}$$

δ contains the following transitions

for each $a(q_1, q_2, \dots, q_r) \rightarrow q \in \delta_i, r \geq 0, q_1, q_2, \dots, q_r \in K_i, 1 \leq i \leq n, a \in \Sigma$,
 $\{a([q_1, i_1], [q_2, i_2], \dots, [q_r, i_r]) \rightarrow [q, j]\} \in \delta$,
 $1 \leq j \leq n, q \in K_j, 1 \leq i_1, i_2, \dots, i_r \leq n$.

If $q_s \rightarrow q_t$ is a rule in the i^{th} component and $q_t \in K_i$, then add $[q_s, i] \rightarrow [q_t, j], 1 \leq j \leq n$ to δ .

If a tree t is accepted by a DNTA, then there is a sequence of ID's $(\phi, t) \vdash (B_1, t_1) \vdash \dots \vdash (\{q_f\}, t_r)$ leading to acceptance. The corresponding sequence of ID's for the NFTA is as follows: $(\phi, t, i_0) \vdash (B_1, t_1, i_1) \vdash \dots \vdash (\{q_f\}, t_r, i_r)$, $1 \leq i_j \leq n$. Similarly, if there is a sequence of ID's leading to acceptance in NFTA, then there is a corresponding sequence of ID's leading to acceptance in the DNTA. This construction of NFTA shows that $L_*(D) = L(N)$ and so $L_*(D)$ is recognizable. ■

= k -mode ($\leq k$ -mode, $\geq k$ -mode) acceptance : An automaton that has a leaf transition rule begins processing the input tree. Suppose that the system starts from the component i . The automaton transfers the control to another component $j, j \neq i$ only after the completion of exactly $k(k' (k' \leq k), (k' \geq k))$ number of steps in the component i . The selection of j is done nondeterministically if there is more than one j .

Definition 6: The instantaneous description(ID) of a DNTA $D = (K, \Sigma, F, \Delta)$ working in $= k$ -mode, $\leq k$ -mode, $\geq k$ -mode is given by a 4-tuple (B, t, i, j) where $B \subseteq \bigcup_i K_i$ and it denotes the current active state set of the whole system, $t \in T(\Sigma \cup \bigcup_i K_i)$, i the index of the component in which the system is currently in, $1 \leq i \leq n$, $j \geq 0$ denotes the number of steps for which the system has been in the i^{th} component.

The system accepts the tree only if the DNTA is in the final state in some component i after processing the tree and provided it has completed k -steps in the component i in the case of $= k$ -mode of acceptance (it has completed some $k' (k' \leq k)$ steps in the component i in the case of $\leq k$ -mode acceptance or it has completed some $k' (k' \geq k)$ steps in the component i in the case of $\geq k$ -mode of acceptance. The language accepted by the respective modes are denoted as $L_{=k}, L_{\leq k}, L_{\geq k}$.

We give an example of a distributed bottom-up tree automata working in $= 2$ -mode.

Example 5: Consider the language

$L_4 = \{b(a(b^{2i}(d), c^{2j}(d)), i, j \geq 1, i = j \text{ or } i = j + 1 \text{ or } j = i + 1)\}$ over $\Sigma = \{a, b, c, d\}, a \in \Sigma_2, b, c \in \Sigma_1, d \in \Sigma_0$.

We define a distributed tree automaton

$D_4 = (K, \Sigma, \{q_f\}, \Delta)$ working in = 2-mode as follows.

The components are defined as follows

- Component 1
 - $K_1 = \{q_b, q_d\}$,
 - $\delta_1 = \{b(q_d) \rightarrow q_b, b(q_b) \rightarrow q_b\}$
- Component 2
 - $K_2 = \{q_d, q_c\}$,
 - $\delta_2 = \{c(q_d) \rightarrow q_c, c(q_c) \rightarrow q_c\}$
- Component 3
 - $K_3 = \{q_f, q_a, q_b, q_c, q_d\}$,
 - $\delta_3 = \{d \rightarrow q_d, a(q_b, q_c) \rightarrow q_a, b(q_a) \rightarrow q_f\}$

Component 3 starts the processing, active for the first two steps, then the system switches between component 1 and 2 and ends the processing with component 3 for the last 2 steps. Using the technique used in example 2 we can show that L_4 is not recognizable.

Similarly we can find languages for = k -mode for $k \geq 3$.

Theorem 3: There exists a language accepted by a DNTA working in = k -mode, $k \geq 1$ which is not recognizable.

Proof: For $k = 2$, example 5 prove the result. For $k > 2$ consider the language

$$L_5 = \{a_{k-1}a_{k-2} \cdots a_1a_0(b^{ki}(e^{k-2}(d)), c^{kj}(g)), i, j \geq 1, k > 2, i = j \text{ or } i = j + 1 \text{ or } j = i + 1\} \text{ over}$$

$$\Sigma = \{b, c, d, e, g, a_0, a_1, a_2, \dots, a_{k-1}\}, a_0 \in \Sigma_2, b, c, e, a_1, \dots, a_{k-1} \in \Sigma_1, d, g \in \Sigma_0.$$

Constructing a DNTA for L_5 is similar to the construction in example 5. It is not difficult to see that L_5 can be accepted by a DNTA working in = k -mode with 3 components. Using the technique used in example 2 we can show that L_5 is not recognizable. ■

Theorem 4: There exists a language accepted by a DNTA working in $\geq k$ -mode, $k \geq 1$ which is not recognizable.

Proof: Consider the language

$$L_6 = \{f^n a_{k-1} a_{k-2} \cdots a_1 a_0 (b^{ki}(e^{k-2}(d)), c^{kj}(g)), i, j, n \geq 1, k > 2, i = j \text{ or } i = j + 1 \text{ or } j = i + 1\} \text{ over}$$

$$\Sigma = \{b, c, d, e, f, g, a_0, a_1, a_2, \dots, a_{k-1}\}, a_0 \in \Sigma_2, b, c, e, f, a_1, \dots, a_{k-1} \in \Sigma_1, d, g \in \Sigma_0.$$

Constructing a DNTA for L_6 is similar to the construction in example 5. It is not difficult to see that L_6 can be accepted by a DNTA working in $\geq k$ -mode with 3 components. Using the technique used in example 2 we can show that L_6 is not recognizable. For $k = 2$, example similar to 5 can be provided. ■

Theorem 5: There exists a language accepted by a DNTA working in $\leq k$ -mode, which is not recognizable.

Proof: Consider the language

$$L_7 = \{g(a^m(e), b^n(e)), m \geq 3, \frac{m+5}{8} \leq n \leq \frac{m+3}{2}\} \text{ over}$$

$$\Sigma = \{g, a, b, e\}, g \in \Sigma_2, a, b \in \Sigma_1, e \in \Sigma_0.$$

We define a distributed tree automaton

$D_7 = (K, \Sigma, \{q_f\}, \Delta)$ working in ≤ 2 -mode as follows.

The components are defined as follows

- Component 1

- $K_1 = \{q_{11}, q_{12}, q_{21}, q_{22}, q_{23}\}$,
- $\delta_1 = \{a(q_{11}) \rightarrow q_{12}, a(q_{12}) \rightarrow q_{12}, e \rightarrow q_{11}, g(q_{12}, q_{21}) \rightarrow q_f, g(q_{12}, q_{22}) \rightarrow q_f, g(q_{12}, q_{23}) \rightarrow q_f\}$

- Component 2

- $K_2 = \{q_{11}, q_{21}, q_{22}, q_{23}\}$,
- $\delta_2 = \{e \rightarrow q_{11}, b(q_{11}) \rightarrow q_{21}, q_{21} \rightarrow q_{22}, q_{22} \rightarrow q_{23}, q_{23} \rightarrow q_{11}\}$

Using the technique used in example 2 we can show that L_7 is not recognizable. ■

Theorem 6: For any recognizable language L , there is a DNTA D working in = 1-mode with two components.

Proof: Let $A = (Q, \Sigma, Q_f, \Delta)$ be a NFTA recognizing L . We construct a distributed tree automaton $D = (K, \Sigma, Q_f, \Delta')$ working in = 1-mode as follows.

The components are defined as follows

- Component 1
 - $K_1 = Q$,
 - $\delta_1 = \Delta$
- Component 2
 - $K_2 = Q$,
 - $\delta_2 = \Delta$

The construction shows that any recognizable language can be recognized by a DNTA working in = 1-mode with two components. ■

Theorem 7: For any recognizable language L , there is a DNTA D working in t -mode with two components.

Proof: Let $A = (Q, \Sigma, Q_f, \Delta)$ be a NFTA recognizing L . We construct a distributed tree automaton $D = (K, \Sigma, Q_f, \Delta')$ working in t -mode as follows.

The components are defined as follows

- Component 1
 - $K_1 = Q \cup \{q' | q \in Q\}$
 - δ_1 contains the following transitions for each $a(q_1, q_2, \dots, q_n) \rightarrow q \in \Delta, n \geq 0, q_1, q_2, \dots, q_n \in K_1, a' \in \Sigma \cup \{\epsilon\}$
 $a(q_1, q_2, \dots, q_n) \rightarrow q' \in \delta_1, q \in K_1$.
- Component 2
 - $K_2 = Q \cup \{q' | q \in Q\}$
 - δ_2 contains the following transitions
 $\forall q' \in K_2, q' \rightarrow q \in \delta_2, q \in K_2$.

The construction shows that any recognizable language can be recognized by a DNTA working in t -mode with two components. ■

Theorem 8: For any DNTA working in $*$ -mode, there is a DNTA working in = 1-mode with two components.

Proof: From theorem 2 we know that any DNTA working in $*$ -mode is recognizable. The theorem follows from the result of theorem 6. ■

We conjecture the following.

Conjecture 1: Any DNTA D working in $= k$ mode with 2 components is recognizable.

Conjecture 2: For any DNTA working in $= k$ -mode, there is a DNTA working in $= 1$ -mode.

4. Conclusion

In this paper we have defined cooperative distributed tree automata and the languages accepted under $*, t, = k, \leq k, \geq k$ (where k is an integer ≥ 1) modes. We showed that the power of tree automata is not increased by the $*$ mode of cooperation, whereas under the other modes, the power is increased. We have proved some results comparing their acceptance power. Other comparisons and decidability issues are being pursued. We are also looking into other application areas like representation of XML schemas and in syntactic pattern recognition.

The application of variable arity trees in representing XML schemas is considered in Murata [9]. The inference of such tree grammars is considered in [10]. Whether distributed tree automata (may be for variable arity trees) will be a better model for representing of XML schemas in an application which can be explored. Distributed version of automata for variable arity trees and other models of tree automata like top-down acceptance and tree walking automata may be more helpful in the above process.

References

- [1] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Draft book; available electronically on <http://www.grappa.univ-lille3.fr/tata>, 2008.
- [2] E. Csuhaj-Varju, J. Dassow, J. Kleeman and Gh. Paun. Grammar Systems: A Grammatical Approach to Distribution and cooperation. Gordon and Breach, London, 1994.
- [3] J. Dassow, G. Paun and G. Rozenberg, Grammar Systems chapter in Handbook of Formal Languages Vol2. edited by G. Rozenberg and A. Salomaa., Springer, 1997.
- [4] J. E. Doner. Decidability of the weak-second order theory of two successors. *Notices of the American Mathematical Society*, 12:365-468, 1965.
- [5] J. E. Doner. Tree acceptors and some of their applications. *J. Comput. Syst. Sci.*, 4(5):406-451, 1970.
- [6] J. E. Hopcroft and J. D. Ullman. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 1979.
- [7] K. Krithivasan, M. Sakthi Balan and P. Harsha. Distributed Processing in Automata, *International Journal of Foundations of Computer Science*, 10(4): 443-464, 1999.
- [8] K. Krithivasan and R. Rama. Introduction to Formal Languages, Automata Theory and Computation. Pearson, 2009.
- [9] M. Murata, D. Lee, M. Mani and K. Kawaguchi. Taxonomy of XML Schema Languages using Formal Language Theory, *ACM Trans. Inter. Tech.*, 5(4):660-704, 2005.
- [10] Neetha Sebastian and K. Krithivasan. Learning Algorithms for Grammars of Variable Arity Trees, *International Conference of Machine Learning and Applications*, 98-103, 2007.
- [11] G. Paun. Grammar Systems: A Grammatical Approach to Distribution and Computation, *Lecture Notes in Computer Science*, 944:429-443, 1995.
- [12] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic, *Mathematical Systems Theory*, 2:57-82, 1968.

Quantum Algorithm for Decomposing Black-Box Finite Abelian Groups

Yong Zhang

Department of Computer Science, Kutztown University of Pennsylvania, Kutztown, PA 19530

Abstract—Cheung and Mosca [1] gave an efficient quantum algorithm for decomposing finite abelian groups in a unique-encoding group model. We present another efficient quantum algorithm for this problem in a more general model, the black-box group model, where each group element is not necessarily uniquely encoded.

Keywords: quantum computation, black-box group model

1. Introduction

Any finite abelian group G can be decomposed into the direct sum of cyclic subgroups of prime power order. However, given a set of generators for G , no efficient classical algorithm is known to find the decomposition of G , i.e., find generators for the cyclic subgroups of G . This problem is at least as hard as INTEGER FACTORIZATION – finding nontrivial factors of a given integer N is equivalent to finding the decomposition of the (finite abelian) group \mathbb{Z}_N^* , the multiplicative group of integers modulo N .

Decomposing finite abelian groups plays an important role in quantum computation, the study of the information processing tasks that can be accomplished using quantum mechanical systems. We call an algorithm that is defined over a traditional computational model a *classical algorithm* and an algorithm that is defined over a quantum computational model a *quantum algorithm*.

In 1994 Shor [2] presented polynomial-time quantum algorithms for two important problems INTEGER FACTORIZATION and DISCRETE LOGARITHM. No efficient classical algorithms are known for these two problems. These two problems are widely believed to be hard on classical computers; their hardness are the basic assumptions for several cryptosystems including the widely used RSA public-key cryptosystem. Shor's paper is the first illustration of the practical importance of quantum computation. A key component in Shor's algorithms is the efficient implementation of the Quantum Fourier Transform (QFT), which explores the underlying algebraic structure of the problems. From this perspective, Shor's quantum algorithms, together with several other quantum algorithms, can be further generalized

to a quantum algorithm for the HIDDEN SUBGROUP problem where the given group G is abelian. In the case when G is non-abelian, the HIDDEN SUBGROUP problem generalizes other well-known problems such as GRAPH ISOMORPHISM. However, the non-abelian case is much harder to solve and remains a major challenge in quantum computation.

Before one can efficiently implement QFT to solve the abelian HIDDEN SUBGROUP problem, the decomposition of the given abelian group G must be known. Cheung and Mosca [1] first studied the problem of decomposing finite abelian groups. They gave an efficient quantum algorithm for this problem. However, one of their assumptions is that each element of the input group G is *uniquely* represented by a binary string. In another word, their quantum algorithm only works for a unique-encoding group model.

In this paper we study the problem of decomposing finite abelian groups in a more general group model — the black-box group model. In the black-box group model elements of the input group G are not necessarily uniquely encoded. The black-box group model was first introduced by Babai and Szemerédi [3] as a general framework for studying algorithmic problems for finite groups. It is a widely used model in computational group theory and quantum computation [4], [5], [6], [7], [8], [9]. This non-unique encoding feature enables this model to handle factor groups [3]. A *factor group* (also known as quotient group) is a group obtained by identifying together elements of a larger group using an equivalence relation. In this paper we give an efficient quantum algorithm for decomposing finite abelian groups in the black-box group model.

2. Preliminaries

In this section we give a brief introduction of the fundamental results in group theory. We refer the readers to a classic book on group theory [10] for more details.

A set G is called a *group* if there is a binary operation \cdot defined on G such that:

- 1) for any $x, y \in G$, $x \cdot y \in G$.
- 2) for any $x, y, z \in G$, $(x \cdot y) \cdot z = x \cdot (y \cdot z)$.

- 3) there is an identity element $e \in G$ such that for any $x \in G$, $x \cdot e = e \cdot x = x$.
- 4) for any $x \in G$, there is a unique $x^{-1} \in G$ such that $x \cdot x^{-1} = x^{-1} \cdot x = e$.

The set of integers \mathbb{Z} together with the “+” operation is an example of a group. Usually if the binary operation \cdot is obvious from the context, we will just write xy instead of $x \cdot y$.

A group G is *abelian* if and only if for any $x, y \in G$, $xy = yx$. Otherwise, G is *nonabelian*. A group G is *cyclic* if there exist $a \in G$ such that $G = \{a^n \mid n \in \mathbb{Z}\}$. Then we say a is a *generator* of G . A *subgroup* H of a group G is a subset which is also a group under the same operation in G . If H is a subgroup of a group G , then a *right coset* of H is a subset S of G such that $\exists x \in G$ for which $S = Hx = \{yx \mid y \in H\}$. A *left coset* of H is defined similarly. The *order* of a group G , denoted by $|G|$, is the cardinality of the set G . The order of the element a is the smallest number n such that $a^n = e$, denoted by $\text{ord}(a)$. If such $n \in \mathbb{Z}$ exists, we say a has *finite order*. In fact, the subset $\{e, a, a^2, \dots, a^{n-1}\}$ forms a subgroup. We call this subgroup the *cyclic subgroup generated by a* and denote it by $\langle a \rangle$.

Lagrange’s Theorem states that if H is a subgroup of a group G , then $|H|$ divides $|G|$. We say $[G : H] = |G|/|H|$ is the *index* of H in G . Let G be a group, p be a prime number, and P be a subgroup of G . If $|P| = p^r$ for some $r \in \mathbb{Z}$, we say P is a *p -subgroup* of G . If furthermore p^r divides $|G|$ but p^{r+1} does not, then we say P is a *Sylow p -subgroup* of G . Let G_1, G_2 be groups such that $G_1 \cap G_2 = \{e\}$. The set $\{(a_1, a_2) \mid a_1 \in G_1, a_2 \in G_2\}$, denoted by $G_1 \oplus G_2$, is called the *direct sum* of G_1 and G_2 . $G_1 \oplus G_2$ is a group under the binary operation \cdot such that $(a_1, b_1) \cdot (a_2, b_2) = (a_1 a_2, b_1 b_2)$.

The fundamental theorem of finite abelian groups states the following.

Theorem 2.1: Given a set $\{g_1, \dots, g_n\}$ of generators of the finite abelian group G , find a set of elements $h_1, \dots, h_k \in G$ such that $G = \langle h_1 \rangle \oplus \dots \oplus \langle h_k \rangle$ and $\langle h_i \rangle$ is a cyclic group of prime power order for all $1 \leq i \leq k$.

Next we introduce the black-box group model. We fix the alphabet $\Sigma = \{0, 1\}$. A *group family* is a countable sequence $\mathcal{B} = \{B_m\}_{m \geq 1}$ of finite groups B_m , such that there exist polynomials p and q satisfying the following conditions. For each $m \geq 1$, elements of B_m are encoded as strings (not necessarily unique) in $\Sigma^{p(m)}$. The group operations (inverse, product and identity testing) of B_m are performed at unit cost by black-boxes (or group oracles). The order of B_m is computable in time bounded by $q(m)$, for each m . We refer

to the groups B_m of a group family and their subgroups (presented by generator sets) as *black-box groups*. Common examples of black-box groups are $\{S_n\}_{n \geq 1}$ where S_n is the permutation group on n elements, and $\{GL_n(q)\}_{n \geq 1}$ where $GL_n(q)$ is the group of $n \times n$ invertible matrices over the finite field F_q . Depending on whether the group elements are uniquely encoded, we have the *unique encoding model* and *non-unique encoding model*, the latter of which enables us to deal with factor groups [3]. In the non-unique encoding model an additional group oracle has to be provided to test if two strings represent the same group element.

3. The Algorithm

Our algorithm uses a divide-and-conquer approach. The algorithm first finds the Sylow p -subgroups of the given input group and then decomposes each Sylow p -subgroup. We start with two technical Lemmas. The first Lemma shows how to find a p -Sylow subgroup in quantum polynomial time.

Lemma 3.1: Let $\mathcal{B} = \{B_m\}_{m > 0}$ be a group family. Let $G < B_m$ be an abelian black-box group given by generating sets $S = \{g_1, \dots, g_s\}$. For any prime number p , the generating sets for the p -Sylow subgroup of G can be computed in quantum polynomial time.

Proof: Since G is abelian, for any prime p , there is an unique p -Sylow subgroup of G . Let n be the order of B_m . By our assumption for black-box model, we can efficiently compute n . Furthermore, we can use Shor’s algorithm to compute the prime factorization $p_1^{e_1} \dots p_r^{e_r}$ of n . If p is not a factor of n , then clearly the p -Sylow subgroup of G is trivial. If p is equal to p_k for some $1 \leq k \leq r$, then we compute the set $S_k = \{g'_1, \dots, g'_s\}$ where $g'_i = g_i^{n/p_k^{e_k}}$. Note that this can be done efficiently using modular exponentiation. We claim that S_k is the generating set for the p -Sylow subgroup. Clearly the order of g'_i is power of p for all i , so $\langle S_k \rangle$ is a p -subgroup of G . To show that $\langle S_k \rangle$ is indeed the p -Sylow subgroup it suffices to show that any $g_i \in S$ can be written as products of elements in $\langle S_1 \rangle, \dots, \langle S_r \rangle$, i.e., $G = \langle S_1 \rangle \oplus \dots \oplus \langle S_r \rangle$. Since $\sum_{l=1}^r n/p_l^{e_l}$ is coprime with n and thus the order of any elements in G , for any $g_i \in G$, $g_i^{\sum_{l=1}^r n/p_l^{e_l}}$, which is a product of elements in $\langle S_1 \rangle, \dots, \langle S_r \rangle$, generates the same cyclic subgroup that g_i generates. ■

Any finite abelian p -group can be expressed as a direct sum of m cyclic groups with order p^{e_1}, \dots, p^{e_m} and $e_1 \leq \dots \leq e_m$. We say that (e_1, \dots, e_m) is the *type* of the p -group. In the second lemma we describe a method to decompose a finite abelian p -group.

Lemma 3.2: Let G be a finite abelian p -group of type

(m_1, m_2, \dots, m_s) . Let g_1, \dots, g_i be elements of G of orders p^{m_1}, \dots, p^{m_i} and for any $j \neq k$ and $1 \leq j, k \leq i$ the cyclic groups $\langle g_j \rangle, \langle g_k \rangle$ have trivial intersection. Given $a \langle g_1, \dots, g_i \rangle$ as an element in the factor group $G/\langle g_1, \dots, g_i \rangle$ of order $p^{m_{i+1}}$ with $a^{p^{m_{i+1}}} = g_1^{x_1} \dots g_i^{x_i}$, we can efficiently find another element g_{i+1} of order $p^{m_{i+1}}$ where $\langle g_{i+1} \rangle$ and $\langle g_1, \dots, g_i \rangle$ have trivial intersection.

Proof: First we show that x_j is a multiple of $p^{m_{i+1}}$ for all $1 \leq j \leq i$.

$$\begin{aligned} a^{p^{m_i}} &= (a^{p^{m_{i+1}}})^{p^{m_i - m_{i+1}}} \\ &= (g_1^{x_1} \dots g_i^{x_i})^{p^{m_i - m_{i+1}}} \\ &= g_1^{x_1 p^{m_i - m_{i+1}}} \dots g_i^{x_i p^{m_i - m_{i+1}}}. \end{aligned}$$

But $a^{p^{m_i}}$ is clearly in $\langle g_1, \dots, g_{i-1} \rangle$, so $g_i^{x_i p^{m_i - m_{i+1}}}$ is also in $\langle g_1, \dots, g_{i-1} \rangle$, therefore x_i is a multiple of $p^{m_{i+1}}$. Similarly we have

$$\begin{aligned} a^{p^{m_{i-1}}} &= (a^{p^{m_{i+1}}})^{p^{m_{i-1} - m_{i+1}}} \\ &= g_1^{x_1 p^{m_{i-1} - m_{i+1}}} \dots g_i^{x_i p^{m_{i-1} - m_{i+1}}} \\ &= g_1^{x_1 p^{m_{i-1} - m_{i+1}}} \dots g_{i-1}^{x_{i-1} p^{m_{i-1} - m_{i+1}}}. \end{aligned}$$

By the same reasoning x_{i-1} is also a multiple of $p^{m_{i+1}}$. Clearly this inductive procedure can go down to $i = 1$. Thus x_j is a multiple of $p^{m_{i+1}}$ for all $1 \leq j \leq i$. Let $y_j = x_j/p^{m_{i+1}}$ for $1 \leq j \leq i$ and $g_{i+1} = a g_1^{-y_1} \dots g_i^{-y_i}$. Then

$$\begin{aligned} g_{i+1}^{p^{m_{i+1}}} &= (a g_1^{-y_1} \dots g_i^{-y_i})^{p^{m_{i+1}}} \\ &= a g_1^{-x_1} \dots g_i^{-x_i} \\ &= e \end{aligned}$$

It is also easy to verify that $\langle g_{i+1} \rangle$ and $\langle g_1, \dots, g_i \rangle$ have trivial intersection. ■

Now we describe the whole algorithm. Given a generating set $\{g_1, \dots, g_s\}$ of a finite abelian group $G \subseteq B_m$, we want to output a set of elements $\{d_1, \dots, d_l\}$ such that $G = \langle d_1 \rangle \oplus \dots \oplus \langle d_l \rangle$. The algorithm uses a divide-and-conquer approach. It first computes the generating set of each p -Sylow subgroup, and then convert each generating set into an "independent generating set". We say a generating set S of a group is *independent* if for any two element $s_i, s_j \in S$, $\langle s_i \rangle$ and $\langle s_j \rangle$ has trivial intersection. Note that in a p -group an independent generating set is exactly the decomposition of the p -group.

We first compute $|B_m|$. Recall that in the black-box model, $|B_m|$ is computable in time bounded by $q(m)$, for each m . In some cases, we will also obtain the prime factorization of $|B_m|$. If not, we can always use Shor's quantum algorithm for INTEGER FACTORIZATION to get the prime factorization $p_1^{e_1} \dots p_r^{e_r}$. For $1 \leq i \leq s$, compute the order of g_i . This can be done using Watrous's quantum

procedure for computing order of an group element in any solvable group [4]. Then, by Lemma 3.1 we can compute the generating set of each p_i -Sylow subgroup, $1 \leq i \leq r$.

Let X_i be the generating set for the p_i -Sylow subgroup. For each $1 \leq i \leq r$, we use Lemma 3.2 to compute an independent generating set S_i of the p_i -Sylow subgroup. We will construct S_i in steps. Initially S_i is empty. We add one element to S_i at each step. Suppose after the $(j-1)$ 'th step, $S_i = \{s_1, \dots, s_{j-1}\}$. At the j 'th step, first compute an element $h \in X_i$ such that $h \langle S_i \rangle$ has the maximum possible order in the factor group $\langle X_i \rangle / \langle S_i \rangle$. This can be done by the constructive group membership test described in [11], i.e., we will get x_1, \dots, x_{j-1} such that $h^{ord(h \langle S_i \rangle)} = \prod_{k=1}^{j-1} s_k^{x_k}$. By Lemma 3.2, we will add the element $s_j = h \prod_{k=1}^{j-1} s_k^{-x_k / ord(h \langle S_i \rangle)}$ to the set S_i . We then test if X_i is a subset of $\langle S_i \rangle$. If yes, we can stop and return S_i as the independent generating set. Otherwise, we will go to the $j+1$ 'th step.

Once we compute the independent generating set S_i for each p_i -Sylow subgroup, the decomposition of G is obtained as $\cup_{i=1}^r S_i$.

4. Discussion

In this paper we present an efficient quantum algorithm to decompose finite-abelian groups in a more general group model — black-box group model. Comparing to Cheung and Mosca's algorithm [1], our algorithm is conceptually simpler and only uses elementary results in group theory. Components of our algorithm may be used to construct quantum algorithms for HIDDEN SUBGROUP problem over certain non-abelian finite groups.

References

- [1] K. Cheung and M. Mosca, "Decomposing finite abelian groups," *Quantum Information and Computation*, vol. 1, no. 3, 2001.
- [2] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, pp. 1484–1509, 1997.
- [3] L. Babai and E. Szemerédi, "On the complexity of matrix group problems I," in *Proceedings of the 25th IEEE Symposium on Foundations of Computer Science*, 1984, pp. 229–240.
- [4] J. Watrous, "Quantum algorithms for solvable groups," in *Proceedings of the 33rd ACM Symposium on the Theory of Computing*, 2001, pp. 60–67.
- [5] —, "Succinct quantum proofs for properties of finite groups," in *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science*, 2000.
- [6] K. Friedl, G. Ivanyos, F. Magniez, M. Santha, and P. Sen, "Hidden translation and orbit coset in quantum computing," in *Proceedings of the 35th ACM Symposium on the Theory of Computing*, 2003, pp. 1–9.

- [7] S. Fenner and Y. Zhang, "Quantum algorithms for a set of group-theoretic problems," in *Proceedings of the Ninth IC-EATCS Italian Conference on Theoretical Computer Science, Siena, Italy, 2005*, pp. 215–227, lecture notes in computer science No. 3701.
- [8] L. Babai, R. Beals, and A. Seress, "Polynomial-time theory of matrix groups," in *Proceedings of the 41st annual ACM symposium on Theory of computing*, ser. STOC '09, 2009, pp. 55–64.
- [9] P. Holmes, S. Linton, E. O'Brien, A. Ryba, and R. Wilson, "Constructive membership in black-box groups," *Journal of Group Theory*, vol. 11, pp. 747–763, 2008.
- [10] W. Burnside, *Theory of Groups of Finite Order*. Dover Publications, Inc, 1955.
- [11] G. Ivanyos, F. Magniez, and M. Santha, "Efficient quantum algorithms for some instances of the non-abelian hidden subgroup problem," in *Proceedings of 13th ACM Symposium on Parallelism in Algorithms and Architectures*, 2001, pp. 263–270.

Tree Insertion Systems

Kamala Krithivasan¹, K. S Sunil¹

¹Department of CSE, Indian Institute of Technology Madras, Chennai - 36, Tamilnadu, India

Abstract— We define tree insertion systems which generates trees. The generative power is compared with the traditional generating and accepting devices for trees. Methods for conversion of tree insertion system to regular tree grammar and finite state bottom-up tree automata to tree insertion system are explained with suitable examples. An outline of proof by induction for the equivalence of tree insertion system and regular tree grammars is given. Some extensions of tree insertion system, where tree nodes can have variable, but fixed arity and their powers are also discussed. Its found that such an extended system is capable of generating parse trees of context-free grammars.

Keywords: Tree insertion system, Regular tree grammar, Non deterministic finite tree automata, Extended tree insertion system

1. Introduction

Insertion-deletion systems are one of the models studied inspired by biology [2]. The operation of insertion and deletion on strings have some relevances to some phenomena in human genetics [3]. A DNA strand can be inserted into/deleted from another strand. The idea of insertion-deletion has been extended to arrays also [5]. In this paper we consider the insertion systems for trees. Trees are important data structures and find use in many applications from the description of parse trees to representation of XML and DTD [6]. Considering insertion systems in trees can have profound applications in such areas [1].

An *insdel* system [2] is a construct $\gamma = (V, T, A, R)$, where V is an alphabet, $T \subseteq V$, A is a finite language over V , and R is a finite set of triples of the form $(u, \alpha/\beta, v)$, where $u, v \in V^*$, $(\alpha, \beta) \in (V^+ \times \{\lambda\}) \cup (\{\lambda\} \times V^+)$. The elements of T are *terminal symbols*, those of A are *axioms*, the triples in R are *insertion-deletion rules*. The meaning of $(u, \lambda/\beta, v)$ is that β can be inserted in between u and v ; the meaning of $(u, \alpha/\lambda, v)$ is that α can be deleted from the context (u, v) . Stated otherwise, $(u, \lambda/\beta, v)$ corresponding to the rewriting rule $uv \rightarrow u\beta v$, and $(u, \alpha/\lambda, v)$ corresponds to the rewriting rule $u\alpha v \rightarrow uv$.

Similarly a tree, say t can be inserted as a sub-tree of another tree, say T based on some context. The insertion may be *fixed arity* or *variable arity*. Here arity refers to the arity of nodes in tree T . In this paper we consider the insertion of trees into trees and call it as *tree insertion system*. The major focus in this paper is fixed arity insertion.

1.1 Basic Definitions

Let N be the set of positive integers. Then the set of finite strings over N is denoted by N^* . The empty string is denoted by ϵ . A *ranked alphabet* Σ is a finite set of symbols together with a function $Rank: \Sigma \rightarrow N$. For $f \in \Sigma$, the value $Rank(f)$ is called the rank of f . For every $n \geq 0$, we denote by Σ_n the set of all symbols of rank n . Elements of rank $0, 1, \dots, n$ are respectively called constants, unary, \dots , n -ary symbols.

A *tree* t [4] over an alphabet Σ is a partial mapping $t: N^* \rightarrow \Sigma$ that satisfies the following conditions:

- $dom(t)$ is a finite, prefix-closed subset of N^* , and
- for each $p \in dom(t)$, if $Rank(t(p)) = n > 0$, then $\{i | p.i \in dom(t)\} = \{1, 2, \dots, n\}$

Each $p \in dom(t)$ is called a *node* of t . The node with domain element ϵ is the *root*. For a node p , we define the i^{th} *child* of p to be the node $p.i$, and we define the i^{th} *subtree* of p to be the tree t' such that $t'(p) = t(p.i.p')$, $\forall p' \in dom(t)$. We denote by $T(\Sigma)$ the set of all trees over the alphabet Σ . The *size* of a tree is the number of elements in $dom(t)$. The *height* of a tree t is $max\{|w| : w \in dom(t)\}$. Given a finite tree t , the *frontier* of t is the set $\{p \in dom(t) | \forall n \in N, p.n \notin dom(t)\}$. A tree with root a and subtrees t_1, t_2, \dots, t_r is represented by $a(t_1, t_2, \dots, t_r)$.

Example 1: Let $\Sigma = \{a, b, c\}$, $b \in \Sigma_2$, $c \in \Sigma_1$, $a \in \Sigma_0$.

A tree over Σ and its diagrammatic representation is shown in Fig. 1

Let t be the tree $b(b(b(a, a), a), c(a))$.

$dom(t) = \{\epsilon, 1, 1.1, 1.1.1, 1.1.2, 1.2, 2, 2.1\}$.

$size(t) = 8$.

$height(t) = 3$.

$frontier(t) = \{1.1.1, 1.1.2, 1.2, 2.1\}$.

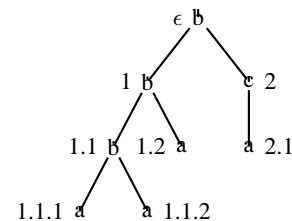


Fig. 1: A tree and its diagrammatic representation

A *Regular Tree Grammar (RTG)* [4] is a construct $G = (N, T, S, P)$ where,

- N is a finite set of non terminals,
- T is a finite set of terminals,
- $S \in N$ is the start symbol,
- P is a finite set of production rules.

Each rule in P is of the form $X \rightarrow x(X_1, X_2, \dots, X_p), p \geq 0$, where $x \in T$ and $X, X_1, X_2, \dots, X_p \in N$.

The *language generated* by an RTG G is represented by $L(G)$, is defined as the set of trees generated by G using productions rules in P , starting from S . A *regular tree language* is a language generated by a regular tree grammar.

Example 2: The language generated by the tree grammar $G_{r_2} = (N, T, S, P)$ where, $N = \{S, B, B', C, H\}$, $T = \{a_2, b_1, c_1, h_0\}$ and

$$P = \{S \rightarrow a(B, C), B \rightarrow b(B'), B' \rightarrow b(B)|b(H), \\ C \rightarrow c(C)|c(H), H \rightarrow h\}$$

is $L(G_{r_2}) = \{a(b^i(h), c^j(h)), i, j \geq 1, i \% 2 = 0\}$.

A *non deterministic bottom-up finite tree automata (NFTA)* [4] over an alphabet Σ is a tuple

$D = (Q, \Sigma, Q_f, \Delta)$ where,

- Q is a finite set of states,
- Σ is a ranked input alphabet,
- $Q_f \subseteq Q$ is a set of final states,
- Δ is a finite set of transition rules.

Each transition rule is a triple of the form $((q_1, q_2, \dots, q_n), f, q)$ where $q_1, q_2, \dots, q_n, q \in Q, f \in \Sigma_n$, i.e. $Rank(f) = n$. We use $f(q_1, q_2, \dots, q_n) \rightarrow q$ to denote that $((q_1, q_2, \dots, q_n), f, q) \in \Delta$. If $Rank(f) = 0$, i.e. f is a constant, then we use rules of the form $f \rightarrow q$. The epsilon rules are denoted by rules of the form $q_i \rightarrow q_j$. A *run* of A over a tree $t \in T(\Sigma)$ is a mapping $r: dom(t) \rightarrow Q$ such that for each node $p \in dom(t)$ where $q = r(p)$, we have that if $q_i = r(pi)$ for $1 \leq i \leq n$ then Δ has the rule $t(p)(q_1, q_2, \dots, q_n) \rightarrow q$.

Example 3: The tree automata $D_{r_4} = (Q, \Sigma, Q_f, \Delta)$ accepts trees with odd number of a 's over the alphabet $\Sigma = \{a_2, b_2, c_0\}$ where $Q = \{e, o\}, Q_f = \{o\}$ and Δ contains following transitions.

$$\begin{array}{lll} c \rightarrow e & b(e, e) \rightarrow e & b(o, o) \rightarrow e \\ a(e, o) \rightarrow e & a(o, e) \rightarrow e & b(e, o) \rightarrow o \\ b(o, e) \rightarrow o & a(e, e) \rightarrow o & a(o, o) \rightarrow o \end{array}$$

2. Definition

The *tree insertion system* is a tuple $\Gamma = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ where,

- Σ is a finite set of *ranked alphabets*.
- $\mathcal{A} = \{A_1 \cup A_2 \cup \dots \cup A_m\}$, where each $A_i, 1 \leq i \leq m$ is a finite set of axioms.

With each set A_i is associated a flag F_i which is a triple $[x_i, y_i, z_i]$, where $x_i, y_i, z_i \in \{-1, 0, 1, \dots, k\}$, for some fixed k , are integers, which plays some role in language generation unless $x_i = y_i = z_i = -1$. (For each insertion from A_i , the x_i value gets incremented if $x_i \leq y_i$ and the x_i value gets decremented if $x_i > y_i$. The x_i will be set to z_i if one insertion from A_i happens when $x_i = y_i$. The tree insertion system is said to be stable, if $x_i = y_i$ for all flags with $x_i \leq y_i$ initially and $x_i \neq y_i$ for all flags with $x_i > y_i$ initially.)

- $\mathcal{A}' \subseteq \mathcal{A}$ is a finite set of *initial axioms*.
- $R = \{r_1, r_2, \dots, r_n\}$ is a finite set of *insertion rules*

Each r_i , for $1 \leq i \leq n$ is of the form $(\chi, C_1, C_2, \dots, C_p)$ where,

- $\chi = (root, left, right)$ which represents a context.
 - * *root* is any node in the tree
 - * *left* is i^{th} child of *root*.
 - * *right* is $(i + 1)^{th}$ child of *root*. $0 \leq i \leq arity(root)$ and $p \leq arity(root)$ (- checks for the absence of a child).
- $C_i = (X, rt', k), 1 \leq i \leq p, X \in \mathcal{A}$
 - * *rt'* is the root of the tree to be attached.
 - * *k* is the position at which *rt'* is to get attached. $1 \leq k \leq arity(root)$ and it is between the nodes *left* and *right*.

As examples, $\chi = (a, b, c)$ denotes a node with label a having a node with label b as i^{th} child and a node with label c as $(i + 1)^{th}$ child for $1 \leq i < arity(a)$. $\chi = (a, -, -)$ denotes a leaf node with label a .

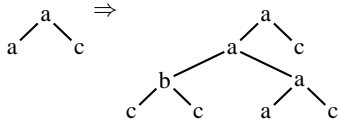
The *derivation step* is described as follows.

If $r = (\chi, C_1, C_2, \dots, C_p)$ is a rule with $\chi = (a, b, c)$ and $C_i = (X, d, k)$ where X is an axiom with trees having root with label d and t is a tree with root p (domain) having label a . Then $t \Rightarrow t'$ by rule r where $t'(p.k)$ is the tree with root label d , $t'(p.i)$ is tree with root label b and $t'(p.(i + p + 1))$ is tree with root label c .

\Rightarrow^* is the reflexive transitive closure of \Rightarrow .

The flag associated with X is also updated when r is applied. For each insertion from X , the x_i value gets incremented if $x_i \leq y_i$, the x_i value gets decremented if $x_i > y_i$ and the x_i will be set to z_i if $x_i = y_i$. We describe the derivation informally with an example.

Suppose $\begin{array}{c} a \\ / \quad \backslash \\ a \quad c \end{array}$ is a tree $A_1 = \left\{ \begin{array}{c} a \\ / \quad \backslash \\ a \quad c \end{array}, \begin{array}{c} b \\ / \quad \backslash \\ c \quad c \end{array} \right\}$ be the axiom and $r_1 = ((a, -, -), (A_1, b, 1), (A_1, a, 2))$ be the insertion rule. Then by using r_1



Here, at leaf node a , a subtree with root b is attached as the first child and another subtree with root a is attached as the second child.

The *language generated* by a tree insertion system Γ , represented by $L(\Gamma)$, is the set of trees, with each node having children exactly equal to its arity, derivable in Γ , when it is in *stable state*, from an initial axiom, using rules of Γ .

$$L(\Gamma) = \left\{ t \mid S \xrightarrow{*} t, S \text{ in some } A_i \in \mathcal{A}'. \text{ Each node of } t \right. \\ \left. \begin{array}{l} \text{has children exactly as its arity and} \\ \Gamma \text{ is in stable state} \end{array} \right\}$$

Example 4: $L_{r_2} = \{a(b^i(h), c^j(h)), i, j \geq 1, i \% 2 = 0\}$

$\Gamma_{r_2} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ where,

$\Sigma = \{a_2, b_1, c_1, h_0\}$, $\mathcal{A} = \{A_1, A_2\}$, $\mathcal{A}' = \{A_1\}$,

$F_1 = [-1, -1, -1]$, $F_2 = [-1, -1, -1]$, where

$$A_1 = \left\{ \begin{array}{c} a \\ / \quad \backslash \\ b \quad c \\ | \quad | \\ b \quad h \end{array} , \begin{array}{c} a \\ / \quad \backslash \\ b \quad c \\ | \quad | \\ b \quad h \end{array} \right\}, \quad A_2 = \left\{ \begin{array}{c} b \\ | \\ b \end{array} , \begin{array}{c} c \\ | \\ c \end{array} , \begin{array}{c} h \\ | \\ h \end{array} \right\}$$

$R = \{r_1, r_2, r_3, r_4\}$, where

$r_1 = (\chi_1, C_1)$, $r_2 = (\chi_2, C_2)$,

$r_3 = (\chi_1, C_3)$, $r_4 = (\chi_2, C_3)$,

$\chi_1 = (b, -)$, $\chi_2 = (c, -)$,

$C_1 = (A_2, b, 1)$, $C_2 = (A_2, c, 1)$, $C_3 = (A_2, h, 1)$

Example 5: $L_{r_4} = \{t \mid n_a(t) \% 2 \neq 0\}$

$\Gamma_{r_4} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ where,

$\Sigma = \{a_2, b_2, c_0\}$, $\mathcal{A} = \{A_1, A_2\}$, $\mathcal{A}' = \{A_1, A_2\}$

$F_1 = [-1, -1, -1]$, $F_2 = [0, 1, 2]$,

$$A_1 = \left\{ \begin{array}{c} b \\ / \quad \backslash \\ b \quad b \\ | \quad | \\ b \quad b \end{array} , \begin{array}{c} b \\ / \quad \backslash \\ b \quad c \\ | \quad | \\ b \quad c \end{array} , \begin{array}{c} b \\ / \quad \backslash \\ b \quad c \\ | \quad | \\ b \quad c \end{array} , \begin{array}{c} b, c \end{array} \right\}$$

$$A_2 = \left\{ \begin{array}{c} a \\ / \quad \backslash \\ c \quad a \\ | \quad | \\ b \quad a \end{array} , \begin{array}{c} a \\ / \quad \backslash \\ c \quad b \\ | \quad | \\ b \quad a \end{array} , \begin{array}{c} a \\ / \quad \backslash \\ b \quad a \\ | \quad | \\ b \quad a \end{array} , \begin{array}{c} a \\ / \quad \backslash \\ c \quad c \\ | \quad | \\ b \quad a \end{array} , \begin{array}{c} a \\ / \quad \backslash \\ b \quad a \\ | \quad | \\ b \quad a \end{array} , \begin{array}{c} a \\ / \quad \backslash \\ c \quad a \\ | \quad | \\ b \quad a \end{array} \right\}$$

$R = \{r_1, r_2\}$, where

$r_1 = (\chi_1, U_1, U_2)$, $r_2 = (\chi_2, U_1, U_2)$,

$\chi_1 = (a, -)$, $\chi_2 = (b, -)$,

$U_1 \in \{C_1, C_3\}$, $U_2 \in \{C_2, C_4\}$,

$C_1 = (A_1, V_1, 1)$, $C_2 = (A_1, V_1, 2)$,

$C_3 = (A_2, V_2, 1)$, $C_4 = (A_2, V_2, 2)$,

$V_1 \in \{c, b\}$, $V_2 \in \{a, b\}$

3. Equivalence with Regular Tree Grammar

Tree grammars are generating devices which is used for generating trees. The language generated by regular tree grammar is called *regular tree language*.

3.1 Insertion System to Regular Tree Grammar

3.1.1 Method of conversion

For a given tree insertion system $\Gamma = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ we can construct an equivalent regular tree grammar $G = (N, T, S, \bar{P})$.

First we consider the simple case where the flag

$$F_i = [-1, -1, -1], \forall A_i \in \mathcal{A}$$

- $T = \Sigma$
- N contains the start symbol S initially and more symbols of the form $M', M^{2'}, \dots, M^{k'}$ for some fixed k , are added to N as we proceed to define the rules.
- $\forall A_i \in \mathcal{A}'$, if $t \in A_i$ with root with label p having arity $m > 0$ and children with label p_1, p_2, \dots, p_m , then $S \rightarrow p(P_1, P_2, \dots, P_m)$ will be a production rule and if there is a node in t with label q and arity r having children with labels q_1, q_2, \dots, q_r , then $Q \rightarrow q(Y_1, Y_2, \dots, Y_r)$ will be a production rule where $Y_i = Q_i, \forall q_i$ with label not equal to q and $Y_i = Q'_i, \forall q_i$ with label equal to q . $Q'_i \rightarrow q(Y_1, Y_2, \dots, Y_r)$ with $Y_i = Q_i^{2'}, \forall Y_i = Q_i$.
- In general $Q_i^{n'} \rightarrow q(Y_1, Y_2, \dots, Y_r)$ with $Y_i = Q_i^{(n+1)'}, \forall Y_i = Q_i^{n'}$.
- If $m = 0$, $S \rightarrow P$ is a production rule.
- $\forall r_i \in R$, $r_i = (\chi, C_1, C_2, \dots, C_k)$ where $\chi = (p, left, right)$, $C_j = (X_j, rt_j, k')$, $\forall j \leq k$ with $rt_j \in \Sigma, X_j \in \mathcal{A}$, $P^{n'} \rightarrow p(RT_j)$, where RT_j is the non terminal corresponding to rt_j , is a production rule.
- $\forall t \in \mathcal{A} - \mathcal{A}'$ with root node having label p and arity 0, then $P \rightarrow p$ is a production rule.
- N will includes all such P_i , Q_i and Q'_i .

It can easily be proved by induction that $L(G) = L(\Gamma)$.

Example 6: $L_{r_2} = \{a(b^i(h), c^j(h)), i, j \geq 1, i \% 2 = 0\}$

$\Gamma_{r_2} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ where,

$\Sigma = \{a_2, b_1, c_1, h_0\}$, $\mathcal{A} = \{A_1, A_2\}$, $\mathcal{A}' = \{A_1\}$,

$F_1 = [-1, -1, -1]$, $F_2 = [-1, -1, -1]$,

$$A_1 = \left\{ \begin{array}{c} a \\ / \quad \backslash \\ b \quad c \\ | \quad | \\ b \quad h \end{array} , \begin{array}{c} a \\ / \quad \backslash \\ b \quad c \\ | \quad | \\ b \quad h \end{array} \right\}, \quad A_2 = \left\{ \begin{array}{c} b \\ | \\ b \end{array} , \begin{array}{c} c \\ | \\ c \end{array} , \begin{array}{c} h \\ | \\ h \end{array} \right\}$$

$R = \{r_1, r_2, r_3, r_4\}$, where

$r_1 = (\chi_1, C_1)$, $r_2 = (\chi_2, C_2)$,

$r_3 = (\chi_1, C_3)$, $r_4 = (\chi_2, C_3)$,

$\chi_1 = (b, -)$, $\chi_2 = (c, -)$,

$C_1 = (A_2, b, 1)$, $C_2 = (A_2, c, 1)$, $C_3 = (A_2, h, 1)$

The production rules are

$$\begin{aligned} \bar{P} = \{ & S \rightarrow a(B, C), B \rightarrow b(B'), B' \rightarrow b(B)|b(H), \\ & C \rightarrow c(C)|c(H), H \rightarrow h \} \end{aligned}$$

The regular tree grammar corresponding to Γ_{r_2} is $G_{r_2} = (\{S, B, B', C, H\}, \{a, b, c, h\}, S, \bar{P})$

The above case will not take care of languages like trees with node labels a, b , and c where number of a 's is odd. For such cases we give the construction below.

- $T = \Sigma$
- N contains the start symbol S initially and more symbols of the form $M', M^{2'}, \dots, M^{k'}$ for some fixed k , are added to N as we proceed to define the rules.
- Let $AJ \subseteq \mathcal{A}'$, where for each $A_i \in AJ$, $x_i = y_i = z_i$ in F_i and $AI = (\mathcal{A}' - AJ)$, where for each $A_i \in AI, n_i = |x_i - y_i|$. (This is used to take care of some constraint on the number of a particular terminal symbol $\sigma_k \in \Sigma$).
- $\forall A_i \in \mathcal{A}$, if there is a node with label p arity 0 is in A_i , then $P \rightarrow p$ is a production rule.
- $\forall A_i \in AJ$
If $\exists A_l \in AI$ with $(x_l \neq y_l)$ in F_l , $t \in A_i$ with root node with label p having arity m and children with label p_1, p_2, \dots, p_m , where for some $j \leq m$, p_j is the label of root node of some $t' \in A_l$, then from $S \rightarrow p(P_1, P_2, \dots, P_m)$ write j production rules, with $P_k = S, \forall k \leq j$.
If $m = 0$, $S \rightarrow P$ will be a production rule.
 $\forall A'_l \in AI$, if $(x'_l = y'_l)$ in F'_l then $S \rightarrow p(P_1, P_2, \dots, P_m)$ will also be a production rule.
 N will includes all such P_i .
- $\forall A_i \in AI$ with $n_i = 1$
If $t \in A_i$ with root with label p having arity m and children with label p_1, p_2, \dots, p_m , then $S \rightarrow p(P_1, P_2, \dots, P_m)$ will be a production rule and if there is a node in t with label q and arity r having children with labels q_1, q_2, \dots, q_r , then, $Q \rightarrow q(Q_1, Q_2, \dots, Q_r)$ will be a production rule.
If $m = 0$, $S \rightarrow P$ will be a production rule.
 $\forall r_i \in R$, $r_i = (\chi, C_1, C_2, \dots, C_{k'})$ where, $\chi = (p, left, right)$, $C_j = (X_j, rt_j, k'')$, $\forall j \leq k'$ with $rt_j \in \Sigma, X_j \in \mathcal{A}$
If $\forall j \leq k', X_j \notin AI, P \rightarrow p(RT_1, RT_2, \dots, RT_k)$ will be a production rule,
If $\exists X_j \in AI, j \leq k$,
– $P \rightarrow p(Y_1, Y_2, \dots, Y_k)$ will be a production rule with $Y_j = RT_j, \forall j$ where $X_j \notin AI$ and $Y_j = RT'_j, \forall j$ where $X_j \in AI$.
If $rt_j = \sigma_j$ where $X_j \in AI$ then $RT'_{\sigma_k} \rightarrow rt_j(Y_1, Y_2, \dots, Y_{p'})$ will be a production rule with $Y_{p''} = S$ for some $p'' \leq p'$, where p' is the arity of rt_j .

If $rt_j \neq \sigma_k$ where $X_j \in AI$ then $RT'_j \rightarrow rt_j(Y_1, Y_2, \dots, Y_{p'})$ will be a production rule $Y_{p''} = RT'_{\sigma_k}$ for some $p'' \leq p'$.

N will includes all such P_i, Q_i, RT'_i and Y_i .

- $\forall A_i \in AI$ with $n_i \neq 1$
If $t \in A_i$ with root with label $p \neq \sigma_i$ having arity m and children with label p_1, p_2, \dots, p_m , with some $p_j = \sigma_k, j \leq m$ then $S \rightarrow p(P_1, P_2, \dots, P_m)$, where both $P_j = P_{\sigma_k}$ and $P_j = S$ are production rules. Then $P_{\sigma_k} \rightarrow S$ and $P_{\sigma_k} \rightarrow p_{\sigma_k}(Y_1, Y_2, \dots, Y_{m'})$ will be production rules where m' is the arity of node with label p_{σ_k} and for some $m'' \leq m'$, $Y_{m''} = P'_{\sigma_k}$. $P'_{\sigma_k} \rightarrow p_{\sigma_k}(Y_1, Y_2, \dots, Y_{m'})$ and for some $m'' \leq m', Y_{m''} = P^{2'}_{\sigma_k}$. In general $P^{q'}_{\sigma_k} \rightarrow p_{\sigma_k}(Y_1, Y_2, \dots, Y_{m'})$ and for some $m'' \leq m'$, $Y_{m''} = P^{(q+1)'}_{\sigma_k}$ and $P^{z'_i}_{\sigma_k} \rightarrow p_{\sigma_k}(Y_1, Y_2, \dots, Y_{m'})$ and for some $m'' \leq m', Y_{m''} = S$.
If $m = 0$, $S \rightarrow P$ will be a production rule.
If $t \in A_i$ with root node with label $p = \sigma_k$ having arity m and children with label p_1, p_2, \dots, p_m , with some $p_j, j \leq m$ is the label of root node of some $t' \in AI$, then $S \rightarrow p(P_1, P_2, \dots, P_m)$ will be a production rule with $P_j = P'_j$ and $P'_j \rightarrow p_j(Y_1, Y_2, \dots, Y_{m'})$ where m' is the arity of node with label p_j and for some $m'' \leq m'$, $Y_{m''} = P'_{\sigma_k}$.
 $\forall r_i \in R$, $r_i = (\chi, C_1, C_2, \dots, C'_k)$ where $\chi = (p, left, right)$, $C_j = (X_j, rt_j, k'')$, $\forall j \leq k'$ with $rt_j \in \Sigma, X_j \in \mathcal{A}$
If $\forall j, X_j \in AJ$, with $p \neq \sigma_k$ then $P \rightarrow p(RT_1, RT_2, \dots, RT_m)$ will be a production rule where m is the arity of node with label p .
If $\forall j, X_j \in AI$, if $p \neq \sigma_k$ then $P \rightarrow p(RT'_1, RT'_2, \dots, RT'_m)$ will be a production rule where m is the arity of node with label p .
If $p = \sigma_k$ then $P \rightarrow p(Y_1, Y_2, \dots, Y_k)$ is a production rule with $Y_j = RT_j, \forall j$ where $X_j \in AJ$ $Y_j = RT'_j, \forall j$ where $X_j \in AI$.
If $rt_j = \sigma_k$ where $X_j \in AI$ then $RT'_{\sigma_k} \rightarrow rt_j(Y_1, Y_2, \dots, Y_{p'})$ will be a production rule with $Y_{p''} = RT'_{\sigma_k}$ for some $p'' \leq p'$, where p' is the arity of rt_j .
In general, for some $p'' \leq p$, $RT^{(z_j-1)'}_{\sigma_k} \rightarrow rt_j(Y_1, Y_2, \dots, Y_{p'})$, with $Y_{p''} = RT^{(z_i)'}_{\sigma_k}$, and $RT^{z'_i}_{\sigma_k} \rightarrow rt_j(Y_1, Y_2, \dots, Y_{p'})$, with $Y_{p''} = S$.
If $rt_j \neq \sigma_k$ where $X_j \in AI$ then $RT'_j \rightarrow rt_j(Y_1, Y_2, \dots, Y_{p'})$ will be a production rule $Y_{p''} = RT'_{\sigma_k}$ for some $p'' \leq p'$.
 N will includes all such P'_i 's, RT'_i 's and Y'_i 's.
- If $\forall k, Y_k = RT'_j$ for some rt_j in $P \rightarrow p(Y_1, Y_2, \dots, Y_r)$, then $P \rightarrow p(RT_1, RT_2, \dots, RT_k)$ will be a production

rule.

It can easily be proved by induction that $L(G) = L(\Gamma)$.

Example 7: $L_{r_4} = \{t | n_a(t) \% 2 \neq 0\}$

$$\Gamma_{r_4} = (\Sigma, \mathcal{A}, \mathcal{A}', R) \text{ where,}$$

$$\Sigma = \{a_2, b_2, c_0\}, \mathcal{A} = \{A_1, A_2\}, \mathcal{A}' = \{A_1, A_2\}$$

$$F_1 = [-1, -1, -1], F_2 = [0, 1, 0],$$

$$A_1 = \left\{ \begin{array}{c} b \quad b \quad b \quad b \quad b \quad b \\ \swarrow \quad \downarrow \quad \swarrow \quad \downarrow \quad \swarrow \quad \downarrow \\ b \quad b \quad b \quad c \quad c \quad b \end{array} \right\}$$

$$A_2 = \left\{ \begin{array}{c} a \quad a \quad a \quad a \quad a \quad a \\ \swarrow \quad \downarrow \quad \swarrow \quad \downarrow \quad \swarrow \quad \downarrow \\ c \quad c \quad b \quad b \quad b \quad c \quad c \quad b \quad a \quad a \quad a \\ \swarrow \quad \downarrow \quad \swarrow \quad \downarrow \quad \swarrow \quad \downarrow \quad \swarrow \quad \downarrow \\ a \quad c \quad b \quad a \quad a \quad b \quad c \quad a \end{array} \right\}$$

$$R = \{r_1, r_2\}, \text{ where}$$

$$r_1 = (\chi_1, U_1, U_2), r_2 = (\chi_2, U_1, U_2),$$

$$\chi_1 = (a, -, -), \chi_2 = (b, -, -),$$

$$U_1 \in \{C_1, C_3\}, U_2 \in \{C_2, C_4\},$$

$$C_1 = (A_1, V_1, 1), C_2 = (A_1, V_1, 2),$$

$$C_3 = (A_2, V_2, 1), C_4 = (A_2, V_2, 2)$$

$$V_1 \in \{c, b\}, V_2 \in \{a, b\}$$

The production rules are

$$P = \{S \rightarrow a(C, C)|a(B, B)|a(B, C)|a(C, B)|a(A, A)|A$$

$$b(A, C)|b(B, A)|b(A, B)|b(C, A)|$$

$$b(S, B)|b(B, S)|b(S, C)|b(C, S)\}$$

$$A \rightarrow a(C, C)|a(C, B)|a(B, C)|a(B, B)|a(C, A')|$$

$$a(C, B')|a(B, A')|a(B, B')|a(A', C)|a(A', B)|$$

$$a(B', C)|a(B', B)|a(A', A')|a(A', B')|$$

$$a(B', A')|a(B', B')|a(A, A)|a(B, B)\}$$

$$B \rightarrow b(C, C)|b(C, B)|b(B, C)|b(B, B)|b(C, A')|$$

$$b(C, B')|b(B, A')|b(B, B')|b(A', C)|b(A', B)|$$

$$b(B', C)|b(B', B)|b(A', A')|b(A', B')|$$

$$b(B', A')|b(B', B')|b(A, A)|b(B, B)\}$$

$$A' \rightarrow a(S, C)|a(S, B)|a(C, S)|a(B, S)|$$

$$a(S, A')|a(A', S)|a(S, B')|a(B', S)\}$$

$$B' \rightarrow b(A', C)|b(A', B)|b(C, A')|b(B, A')|$$

$$b(A', A')|b(A, A)|b(B', A')|b(A', B')\}$$

$$C \rightarrow c$$

$$\}$$

The regular tree grammar corresponding to Γ_{r_4} is
 $G_{r_4} = (\{S, A, A', B, B', C\}, \{a, b, c\}, S, P)$

Result 1: Given a tree insertion system, we can construct an equivalent regular tree grammar.

3.2 Finite state Bottom-up Tree Automata to Insertion System

Tree automata are accepting devices for trees. Finite tree automata are generalizations of word automata. While a word automaton accepts a word, a tree automaton accepts a tree. Finite tree automata can be either bottom-up or top-down [4]. A *top-down tree automaton* starts its computation

at the root of the tree and then simultaneously works down the paths of the tree level by level.

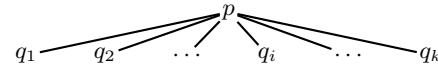
Since there exists a bottom-up finite tree automata for accepting a regular tree language, it is enough to simulate that automata using the tree insertion system, to show the equivalence of tree insertion system and regular tree grammars. A *bottom-up tree automaton* starts its computation in the leaves of the input tree and works its way up towards the root.

3.2.1 Method of conversion

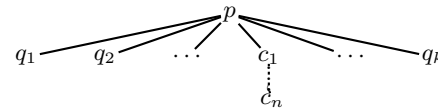
For a given deterministic bottom-up tree automata $D = (\Sigma', Q, Q_f, \Delta)$ we can construct a tree insertion system $\Gamma = (\Sigma, \mathcal{A}, \mathcal{A}', R)$, where

- $\Sigma = \Sigma'$
- If transitions are non-recursive

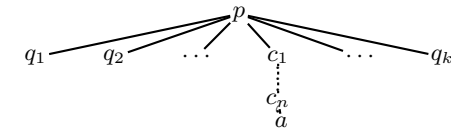
For each transition of the form $p(q_1, q_2, \dots, q_k) \rightarrow q_g$, where $q_1, q_2, \dots, q_k, q_g \in Q$, $p \in \Sigma'$, k is the arity of node with label p and $q_g \in Q_f$, will be in \mathcal{A}'



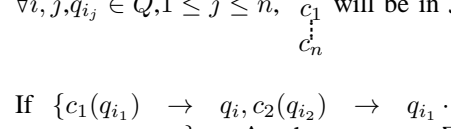
$\forall q_i, 1 \leq i \leq k$, if $\{c_1(q_{i_1}) \rightarrow q_i, c_2(q_{i_2}) \rightarrow q_{i_1} \dots c_n(q_{i_n}) \rightarrow q_{i_{n-1}}\} \in \Delta$ where $q_{i_n} = q_i$, $c_i \in \Sigma', 1 \leq i \leq n$ and $\forall i, j, q_{i_j} \in Q, 1 \leq j \leq n$, will be in \mathcal{A}'



If $\{c_1(q_{i_1}) \rightarrow q_i, c_2(q_{i_2}) \rightarrow q_{i_1} \dots c_n(q_{i_n}) \rightarrow q_{i_{n-1}}, a \rightarrow q_{i_n}\} \in \Delta$ where $a, c_i \in \Sigma', 1 \leq i \leq n$ and $\forall i, j, q_{i_j} \in Q, 1 \leq j \leq n$, will be in \mathcal{A}'



For each transition of the form $p(q_1, q_2, \dots, q_k) \rightarrow q_g$, where $q_1, q_2, \dots, q_k, q_g \in Q$, $p \in \Sigma'$, k is the arity of node with label p and $q_g \in Q_f$ $\forall q_i, 1 \leq i \leq k$ if $\{c_1(q_{i_1}) \rightarrow q_i, c_2(q_{i_2}) \rightarrow q_{i_1} \dots c_n(q_{i_n}) \rightarrow q_{i_{n-1}}\} \in \Delta$ where $q_{i_n} = q_i$, $c_i \in \Sigma', 1 \leq i \leq n$ and $\forall i, j, q_{i_j} \in Q, 1 \leq j \leq n$, c_1 will be in $\mathcal{A} - \mathcal{A}'$



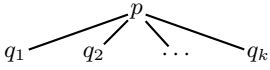
If $\{c_1(q_{i_1}) \rightarrow q_i, c_2(q_{i_2}) \rightarrow q_{i_1} \dots c_n(q_{i_n}) \rightarrow q_{i_{n-1}}, a \rightarrow q_{i_n}\} \in \Delta$ where $a, c_i \in \Sigma', 1 \leq i \leq n$ and $\forall i, j, q_{i_j} \in Q, 1 \leq j \leq n$, c_1 will be in $\mathcal{A} - \mathcal{A}'$

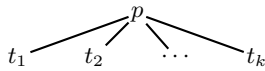
Set flag $F_i = [x_i, y_i, z_i]$ associated with each $A_i \in \mathcal{A}$ as $[-1, -1, -1]$.

- If transitions are recursive

Let AF and AN are two temporary axiom sets
For each transition of the form $p(q_1, q_2, \dots, q_k) \rightarrow q_g$,
where $q_1, q_2, \dots, q_k, q_g \in Q$, $p \in \Sigma'$, k is the arity of
node with label p , will be in

AF if $q_g \in Q_f$ and it will be in AN if $q_g \notin Q_f$.

Let $t =$  and

$t' =$ 

$\forall t \in AF$, if all $q_1, q_2, \dots, q_k \in Q_f$, then p will be in AF .

$\forall t \in AN$, if all $q_1, q_2, \dots, q_k \notin Q_f$, then p will be in AN

$\forall t \in AF$, t' will be in AF with t_i having the label of
root node of some $t'' \in AF$, $\forall q_i \in Q_f$.

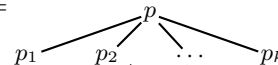
$\forall t \in AN$, t' will be in AN with t_i having the label of
root node of some $t'' \in AN$, $\forall q_i \notin Q_f$.

$\forall t \in AF$, t' will be in AF with t_i having the label of
root node of some $t'' \in AN$, $\forall q_i \notin Q_f$.

$\forall t \in AN$, t' will be in AN with t_i having the label of
root node of some $t'' \in AF$, $\forall q_i \in Q_f$.

Set flag $F_{AN} = [x, y, z]$ associated with AN as
 $[-1, -1, -1]$.

Set flag $F_{AF} = [x, y, z]$ associated with an axiom AF ,
which has a constraint on $\sigma_i \in \Sigma$, as follows.

- $x = 0$.
- If $\exists t =$  $\in AF$, $k \geq 0$, if
 $p, p_1, p_2, \dots, p_k \neq \sigma_i$ then $y_i = 0$.

- Else if $\exists t \in AF$, with n nodes of t have label σ_i ,
then $y_i = z_i = n$.

- If $t' =$  $\in AF$, $k' \geq 0$

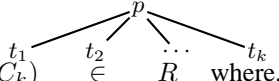
having $m \geq n$ nodes of t are with label σ_i .

- * $z = (m - n) - 1$, if $y = 0$.
- * $z = (m - 1)$, if $y \neq 0$.

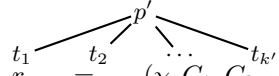
Now $\mathcal{A}' = AF$ and $\mathcal{A} = AF \cup AN$

- For each transitions of the form $a \rightarrow q_a \in \Delta$ where,
 $a \in \Sigma'$, $q_a \in Q$, a will be in \mathcal{A}' , if $q_a \in Q_f$ and a
will be in $\mathcal{A} - \mathcal{A}'$, if $q_a \notin Q_f$

- $\forall t \in \mathcal{A}$ which is of the form


 $r = (\chi, C_1, C_2, \dots, C_k) \in R$ where,
 $\chi = (p, -, \dots, -)$ and $\forall i \leq k$, $C_i = (X, a, i)$ where,
 $X \in \mathcal{A}$, for some $t' \in X$, t' has a as root.

And $\forall t_i, 1 \leq i \leq k$ which is of the form


 $r = (\chi, C_1, C_2, \dots, C_{k'}) \in R$ where,
 $\chi = (p', -, \dots, -)$ and $\forall i \leq k'$, $C_i = (X, a, i)$
where, $X \in \mathcal{A}$, for some $t' \in X$, t' has a as root.

Example 8: $L_{r_5} = \{a(b^i(h), c^j(h)), i, j \geq 1, i \% 3 = 0 \text{ and } j \% 2 = 0\}$ Consider the tree automata
 $D_{r_5} = (Q, \Sigma', Q_f, \Delta)$ which accepts the language
 L_{r_5} over the alphabet $\Sigma' = \{a_2, b_1, c_1, h_0\}$ where
 $Q = \{q_a, q_b, q_{b_1}, q_{b_2}, q_c, q_{c_1}, q_h\}$, $Q_f = \{q_a\}$ and Δ
contains following transitions.

$h \rightarrow q_h$ $b(q_h) \rightarrow q_{b_1}$ $b(q_{b_1}) \rightarrow q_{b_2}$
 $b(q_{b_2}) \rightarrow q_b$ $b(q_b) \rightarrow q_{b_1}$ $c(q_h) \rightarrow q_{c_1}$
 $c(q_{c_1}) \rightarrow q_c$ $c(q_c) \rightarrow q_{c_1}$ $a(q_b, q_c) \rightarrow q_a$

$A_1 = \left\{ \begin{array}{c} a \\ \left| \begin{array}{cc} b & c \\ \left| \begin{array}{cc} b & c \\ \left| \begin{array}{cc} b & h \end{array} \end{array} \end{array} \right. \end{array} \right. \right\},$

$A_2 = \left\{ \begin{array}{c} c \\ \left| \begin{array}{cccc} c & c & b & b & h \\ \left| \begin{array}{cccc} c & c & b & b & h \\ \left| \begin{array}{cccc} h & b & b & h \end{array} \end{array} \right. \end{array} \right. \right\}$

$R = \{r_1, r_2, r_3\}$, where
 $r_1 = (\chi_1, C_1, C_2)$, $r_2 = (\chi_2, U_1)$, $r_3 = (\chi_3, U_2)$,
 $\chi_1 = (a, -, -)$, $\chi_2 = (b, -)$, $\chi_3 = (c, -)$,
 $U_1 \in \{C_1, C_4\}$, $U_2 \in \{C_3, C_4\}$, $C_1 = (A_2, b, 1)$,
 $C_2 = (A_2, c, 2)$, $C_3 = (A_2, c, 1)$, $C_4 = (A_2, h, 1)$

So the tree insertion system equivalent to D_{r_5} is
 $\Gamma_{r_5} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ where,

$\Sigma = \{a_2, b_1, c_1, h_0\}$, $\mathcal{A} = \{A_1, A_2\}$, $\mathcal{A}' = \{A_1\}$,
 $F_1 = [-1, -1, -1]$, $F_2 = [-1, -1, -1]$

Example 9: $L_{r_4} = \{t | n_a(t) \% 2 \neq 0\}$

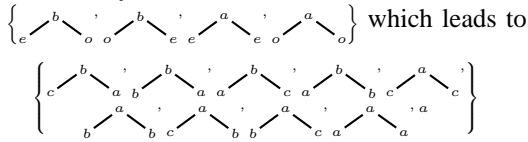
Consider the tree automata $D_{r_4} = (Q, \Sigma', Q_f, \Delta)$ which
accepts the language L_{r_4} over the alphabet $\Sigma' = \{a_2, b_2, c_0\}$
where $Q = \{e, o\}$, $Q_f = \{o\}$ and Δ contains following
transitions.

$c \rightarrow e$ $b(e, e) \rightarrow e$ $b(o, o) \rightarrow e$
 $a(e, o) \rightarrow e$ $a(o, e) \rightarrow e$ $b(e, o) \rightarrow o$
 $b(o, e) \rightarrow o$ $a(e, e) \rightarrow o$ $a(o, o) \rightarrow o$

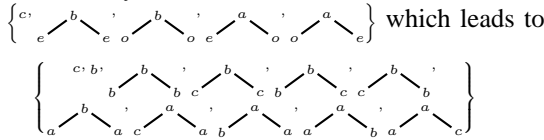
Here the transition shows a recursive behaviour and

so it is not possible to generate all axioms for this language. Generate temporary axioms: AF , from final state transitions and AN , from non-final state transitions.

AF initially contains

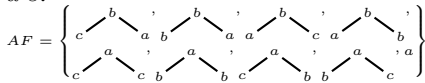


AN initially contains



The two axiom sets AF and AN differs in the number of a 's. So AF has some constraint on a . Set the flag associated with AF . Here $x = 0$. Since all axioms in AF contains a , $n = 1$ and one of the axiom contains 3 a 's, $m = 3$. So $y = 1$ and $z = 2$

In AF , split the axioms so that each axiom contains single a 's.



Now let $A_1 = AN$ and $A_2 = AF$.

As both a and b can lead to final state directly, the insertion can be start from either A_1 or A_2 . Hence $\mathcal{A} = \{A_1, A_2\}$, $\mathcal{A}' = \{A_1, A_2\}$, $F_1 = [-1, -1, -1]$, $F_2 = [0, 1, 2]$.

Create rules for generating all trees both in A_1 and in A_2 .

$R = \{r_1, r_2\}$, where

$$r_1 = (\chi_1, U_1, U_2), r_2 = (\chi_2, U_1, U_2),$$

$$\chi_1 = (a, -, -), \chi_2 = (b, -, -), U_1 \in \{C_1, C_3\},$$

$$U_2 \in \{C_2, C_4\}, C_1 = (A_1, V_1, 1), C_2 = (A_1, V_1, 2),$$

$$C_3 = (A_2, V_2, 1), C_4 = (A_2, V_2, 2), V_1 \in \{c, b\}, V_2 \in \{a, b\}$$

So the tree insertion system equivalent to D_{r_4} is $\Gamma_{r_4} = (\Sigma, \mathcal{A}, \mathcal{A}', R)$ where, $\Sigma = \{a_2, b_2, c_0\}$, $\mathcal{A}, \mathcal{A}'$ and R as given above.

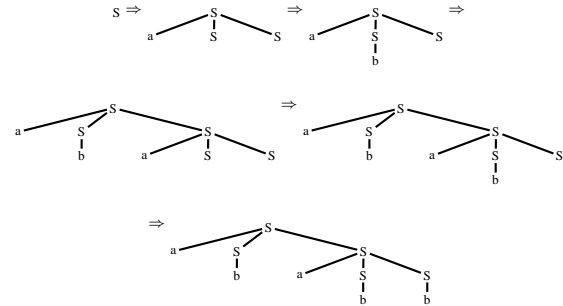
Result 2: Given a finite bottom-up tree automata, we can construct an equivalent tree insertion system.

Theorem 1: From Result 1 and Result 2, the generative powers of tree insertion systems and regular tree grammars are equal.

4. Extended Systems

An *extended tree insertion system* can be defined by defining two types of alphabets T and N with $\Sigma = T \cup N$ and additional restriction that the symbols in T have arity 0 and each symbol from N has a finite number of arities. With this extended definition, the parse trees of a context-free grammar can be generated.

As an example, consider the CFG $G = (\{S\}, \{a, b\}, \{S \rightarrow aSS, S \rightarrow b\}, S)$. The parse tree of G can be generated by the extended tree insertion system $(\Sigma, \mathcal{A}, \mathcal{A}', R)$ with $\Sigma = N \cup T$, $N = \{S\}$, $T = \{a, b\}$, arity of S is $\{1, 3\}$. Axioms are $A_1 = \{S\}$ and $A_2 = \{a, S, b\}$ with flags $F_1 = [-1, -1, -1], F_2 = [-1, -1, -1]$, where $\mathcal{A}' = \{A_1\}$. The rules are $r_1 = (\chi_1, C_1, C_2, C_3)$ and $r_2 = (\chi_1, C_4)$, where $\chi_1 = (S, -, -)$, $C_1 = (A_2, a, 1)$, $C_2 = (A_1, S, 2)$, $C_3 = (A_1, S, 3)$, $C_4 = (A_2, b, 1)$. As an example of derivation



5. Conclusion

In this paper we defined tree insertion systems and showed that the generative power is the same as that of non deterministic bottom-up tree automata (or equivalently regular tree grammars). By defining an extended system, the power slightly increases. It should be noted that the insertion definition given here is a restricted one with the inserted tree occupying lower levels only. It may be interesting to widen the definition in such a way that a subtree is inserted into a tree by cutting the tree at an internal node and inserting the subtree with the cut subtree attaching to a leaf of the inserted tree. Exploring the generative power of such systems with different constraints is being explored. It would also be interesting to consider *deletion system* also.

References

- [1] Akihiro Takaharai., Takashi Yokomori, On the Computational power of insertion-deletion systems, Natural Computing 2. pp. 321-336. Kluwer Academic Publishers,(2003)
- [2] Gheorghe Paun., Grzegorz Rozenberg., Arto Saloma., DNA Computing, New Computing Paradigms. Springer, (1998)
- [3] Lila Kari., Gheorghe Paun., Thierrin,G., Yu,S., At the crooroads of DNA computing and formal languages: Characterizing RE using insertion-deletion systems. Proc. of 3rd DIMACS Workshop on DNA based Computing, Philadelphia, pp. 318-333, (1997)
- [4] Hubert Comon., Max Dauchet Remi Gilleron.,Florent Jacquemard Denis Lugiez., Christof Loding., Sophie Tison Marc Tommasi., Tree Automata Techniques and Applications. Draft book; available electronically on <http://www.grappa.univ-lille3.fr/tata>, 2008.
- [5] M B Siddardha and K. Krithivasan, Ambiguity in Insertion-Deletion Kolam Array Grammers , Journal of Combinatorics, Information and System Sciences, Vol. 33,Nos. 3-4, pages 323-337, 2008.
- [6] M. Murata, D. Lee, M. Mani and K. Kawaguchi. Taxonomy of XML Schema Languages using Formal Language Theory, *ACM Trans. Inter. Tech.*, 5(4):660-704, 2005.

Avalanche in States; Combination of Sandpile and Cellular Automata for Generate Random Numbers

Seyed Morteza Hosseini¹, Hossein Karimi², Majid Vafaei Jahan³

^{1,2,3}Department of Software Engineering, Mashhad Branch - Islamic Azad University, Mashhad, Iran

Abstract- Cellular Automata (CA) is a self organizing structure with complex behavior which can be used in pseudo-random numbers generation (PRNG). Pure CA has a simple structure but has no ability to produce long sequences of random numbers. In order to rectify this problem, programmable CA (PCA), using stimulating factor or combination of different self organizing Criticality phenomenon can be used. In this paper, a PCA by using Sandpile model is proposed. The Sandpile is a complex system operating at a critical state between chaos and order. This state is known as Self-Organized Criticality (SOC) and is characterized by displaying scale invariant behavior. In the precise case of the Sandpile Model, by randomly and continuously dropping "sand grains" on top of a two dimensional grid lattice, a power-law relationship between the frequency and size of sand "avalanches" is observed. The avalanche behavior and the pure CA behavior are combined in a novel method which can be used as the pseudo-random number generator.

Keywords: Random Number Generator, Self-Organizing Criticality, Sandpile Model, Cellular Automata.

1 Introduction

Random number generators (RNGs) play an important role in several computational fields, including Monte Carlo techniques [1], Brownian dynamics [2], stochastic optimization methods [2, 3] and key-based cryptography [4]. It is usual to use mathematical or even evolutionary methods to construct RNGs that yields high quality generators. The quality of generators that determined by statistical tests have a great important role; for example, in cryptography, low quality of RNGs causes easily breaking the encrypted context [4]. In solving optimization problems, as shown in [5], performance and speed of algorithms directly depend on quality of used RNGs. Because of simple structure of CA and its complex behavior and high ability to be used parallel, CA has a well ability in generating random numbers. But one of the major problems of CA is its bounded generated random number sequence because of the self-organizing ability

and generate frequent numbers with specific rules. Hence, a strategy for increasing the complexity of behavior and the implementation of CA to present a better random number sequence is required. In this paper, a new method for stimulating CA and increasing the complexity of CA's behavior based on Sandpile model has been presented. Sandpile model, because of existence of avalanche phenomenon has a non-equilibrium behavior. Hybridizing this model with cellular automata causes a random behavior that it leads to generate a qualified sequence of random numbers.

Herein a two dimensional $n \times m$ CA and combination of 8 rules has been used. The obtained results show that the sequence of generated numbers by CA passed all parts of diehard test suite, entropy and chi-square and other static tests. Some of the other of advantages of this method are uniform distribution of generated numbers by CA, high ability of parallel processing and also the sensitivity to bit changes in particular applications such as cryptography. This paper is organized as following: in following section related works discussed. Section 3, contains the basic concepts of CA and Sandpile model. In section 4 the proposed RNG algorithm and its behavior are discussed. In section 5, the experimental results are illustrated and finally, in section 6, the conclusion and future works are discussed.

2 Related Works

The first work to apply CA as RNG was done by Wolfram in 1986. His work shows the ability of CA to generate random bits [6, 7]. Basic researches on CA are on producing RNG by one dimensional CA with 3 neighbors [7]. Other researches are focused on increasing CA's complexity with combinations of controllable cells [4, 8] or increasing CA's complexity with increasing dimensionality. RNG are produced by using one dimensional CA studied in [9, 10, 11, 12, 13] and two dimensional CA in [14, 15, 16] and three dimensional CA in [17]. Hortensius proposed the first non-uniform CA or programmable CA (PCA) by using of the combination of two rules, 90 and 150 in 1989[9]. PCA is a non-uniform CA that allows different rules to be used at different time steps on the same cell. He also represented another

generator using the combination of rules 30 and 45 in [10] that its output bits have more dependencies to each other rather than rules 90 and 150.

Recently, extensive studies have been done on PCA for generating random numbers [11, 15, 16, 18, 19]. First works on two dimensional CA represented by Chaudhuri et al. in 1994 [14]. Their results show that produced generator using this CA works better rather than one dimensional CA with the same size. In [20, 21] all 256 (simple) elementary cellular automata were investigated (including those with rules given 90 and 150). It was found that CA with nonlinear rules 45 (or its equivalent rules 75, 89 or 101) exhibit chaotic (or pseudo-random) behaviors similar to those obtained in LFSRs.

3 Cellular Automata and Sandpile Model

3.1 Cellular Automata

A cellular automaton (CA), introduced by Von Neumann in 1940s, is a dynamic system in which its time, space and states are all discrete. The CA evolves deterministically in discrete time steps and each cell takes its value from a finite set S, called the State Set. A CA is named Boolean if $s = \{0, 1\}$. The $i - th$ cell is denoted by $\langle i \rangle$ and the state of cell $\langle i \rangle$ at time t is denoted by a_i^t . For each cell $\langle i \rangle$, called central cell, a symmetric neighborhood of radius r is defined by (1):

$$v_i = \{ \langle i - r \rangle, \dots, \langle i \rangle, \dots, \langle i + r \rangle \} \tag{1}$$

the value of each cell $\langle i \rangle$ is updated by a local transition function f_i -called rule- which for a symmetric neighborhood with radius r is defined as follows (2):

$$a_i^{t+1} = f(a_{i-r}^t, \dots, a_i^t, \dots, a_{i+r}^t) \tag{2}$$

or equivalently by (3):

$$a_i^{t+1} = f(v_i^t) \tag{3}$$

Such that v_i^t is as follows (4):

$$v_i^t = f(a_{i-r}^t, \dots, a_i^t, \dots, a_{i+r}^t) \tag{4}$$

To represent a symmetric rule of radius r for a Boolean CA, a binary string of length L is used, where $L = 2^{2r+1}$, Table 1 Shows the rule 90 of radius one ($r=1$).

Table 1. The Rule Representation Of Boolean Symmetric Rule 90 Of Radius One

Neighborhood Number	7	6	5	4	3	2	1	0
v_i^t	111	110	101	100	011	010	001	000
$f(v_i^t)$	0	1	0	1	1	0	1	0

If all CA cells obey the same rule, then the CA is said to be a uniform CA; otherwise, it is a non-uniform CA[22]; in addition, a CA is said to be a CA with periodic boundary condition if the extreme cells are adjacent to

each other else it called null-boundary CA. If a CA rule involves only XOR logic, it is called a linear rule; rules involving XNOR logic are referred to complemented rules. A CA with all cells having linear rules is called linear CA, whereas a CA having a combination of linear and complemented rules is called an additive CA [23]. Nandi et al. presented a programmable CA (PCA) in 1994 [23]. A CA is said to be a PCA if it uses a control CA to determine the rules of each cell. A control CA is essentially just another basic CA which is usually of uniform nature. The rule function used by each cell changes with time and is decided by the control CA. PCA is, in fact, a non-uniform CA because all its cells collectively use different rule functions. A PCA may use m -bit control CA, where $m \geq 1$. For each cell, there are 2^m rules to choose from, thereby, allowing less probability of correlations among the cells. Compared to uniform CA, PCA allows several control lines per cell. Through these control lines, different rules can be applied to the same cell at different time steps according to the rule control signals. Fig. 1, shows a PCA cell structure.

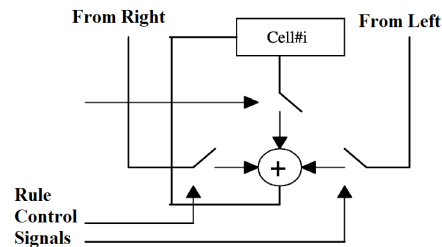


Fig. 1. A PCA cell structure

As Illustrated in Fig. 1, control signals select cell's rule. In this paper a two dimensional Sandpile model and two dimensional PCA with non-periodic boundary condition is considered. Each PCA cell's state can be a number as 0, 1, 2, 3. Herein, applied rules are the same rules that were used in elementary CA.

3.2 Sandpile Model

In 1987, Bak at al. [24] identified the SOC phenomenon associated with dynamical systems. The first system were SOC was observed was named after its inspiration as the Sandpile model, and consists of a cellular automata where at each cell of the lattice, there is a value which corresponds to the slope of the pile. Grains of sand are randomly “thrown” into the lattice where they pile up and increment the values of the cells. When a value exceeds a specific threshold, an avalanche takes place and four grains belonging to that cell are distributed by the neighboring sites (*von Neumann* neighborhood). If one of those sites also exceeds the threshold value(z_c), the avalanche continues, and the grains are also sent to the adjacent cells. The procedure of the Sandpile model is shown in fig. 2.

With these settings, and depending on the state of the lattice and the position of the new grain, a grain may cause rather different responses. It may not cause any change in the system if it falls in a cell with its value bellow threshold (other than increasing the sand on the

cell, of course) and it may generate large avalanches of sand that will strongly redefine the shape of the pile.

Procedure Sandpile

Consider a lattice (x,y) and a function $z(x,y)$ which represents the number of grains in the cells.

Starting with a flat surface $z(x,y) = 0$ for all x and y :

Add a grain of sand: $z(x,y) = z(x,y) + 1$

if $z(x,y) > z_c$ then an avalanche occurs

$$z(x,y) = z(x,y) - z_c$$

$$z(x \pm 1, y) = z(x \pm 1, y) + 1$$

$$z(x, y \pm 1) = z(x, y \pm 1) + 1$$

if $z(x, y \pm 1) = 4$ or $z(x \pm 1, y) = 4$

Update z recursively

Fig. 2. 2D Bak-Tang-Wisenfeld Sandpile Model

4 Proposed RNG Based On Combination of Sandpile and PCA

4.1 Proposed RNG

In this scheme a two dimensional $n \times m$ PCA with Null boundary condition is used to generate random numbers by using 8 rules: 153, 30, 90, 165, 86, 105, 110, 150. According to [25], generated numbers by these rules have the best results in different tests such as entropy, chi-square and diehard. The Boolean expression of each CA rule is shown in Table 2.

Table 2: The detail and Boolean expression of each CA Rule

Rule Name	Possible Input Configuration	Boolean Representation
	111110101100011010001000	
101	0 1 1 0 0 1 0 1	$[x_{i-1} \text{ nor } x_{i+1}] \text{ or } [(x_i \text{ xor } x_{i+1}) \text{ and } x_{i-1}]$
105	0 1 1 0 1 0 0 1	$\text{Not}[x_{i-1} \text{ xor } x_i \text{ xor } x_{i+1}]$
86	0 1 0 1 0 1 1 0	$[x_{i-1} \text{ nor } x_i] \text{ xor } [\text{not}(x_{i+1})]$
165	1 0 1 0 0 1 0 1	$[x_{i-1}] \text{ xnor } [x_{i+1}]$
90	0 1 0 1 1 0 1 0	$[x_{i-1}] \text{ xor } [x_{i+1}]$
30	0 0 0 1 1 1 1 0	$[x_{i-1}] \text{ xor } [x_i \text{ or } x_{i+1}]$
153	1 0 0 1 1 0 0 1	$[x_i] \text{ xnor } [x_{i+1}]$
150	1 0 0 1 0 1 1 0	$[x_{i-1}] \text{ xor } [x_i] \text{ xor } [x_{i+1}]$

In this paper, for Sandpile implementation, the threshold value is considered equal to 4 and each cell has four nearest neighbors: up, down, left and right (Von-Neuman neighborhood). The value of each CA's cell is an integer between 0 and 3. For converting these numbers to 0 and 1(binary state), their residual over 2 is used. Hence, numbers 0 and 2 (even numbers) are delegated to 0 and numbers 1 and 3 (odd numbers) are delegated to 1. In order to generate random numbers, the CA was initialized by random numbers between 0 and 3. At each time step, there are two steps that are discussed in the following:

Firstly, a $n \times m$ CA is set to the binary state and each row divides into 4-cell's parts (each part has four cells). In each word the first two cells show the number of time that Sandpile run on each cell (φ) and the second two cells show the cell that action should be run on it (α).

Each word (φ, α) , is extracted from the word in the previous row. Because of increasing φ has no tangible effect on the quality of the generated random numbers and only increases processing time, φ is restricted to the maximum equal to 2. If the first two cell of each word is equal to 0, 1 or 2 the Sandpile action is run once; else if it is equal to 3 the Sandpile action is run twice. Fig. 3, show the hardware schema process of way of selection for performing sandpile action and the number of sandpile actions for a 4 cell section in row $i+1$.

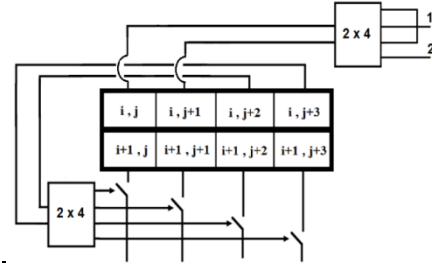
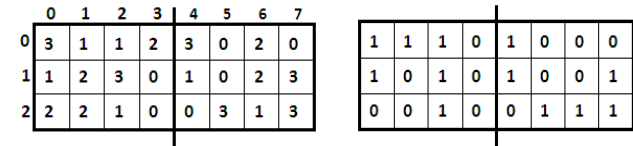


Fig. 3. Proposed CCA with selection of a cell for running Sandpile action.

For example, Fig. 4(a), shows the three rows of CA; and Fig. 4(b), shows its binary state. These figures show the number of Sandpile runs and the cells that Sandpile applied on them.



(a) three rows of CA (b) binary state of three rows
Fig. 4. An example of CA and its binary state

For determination of (φ, α) , in the first word of the second row, the corresponding data in the previous row i.e. first row was used. The value of the first two bits of the first word of first row is $(11)_2$ that imply number 3. So as mentioned, the number of Sandpile run in first word of the second row would be equal to 3. The value of the second two bits of the first word of the first row is $(10)_2$ that shows cell 2 is selected. So Sandpile is run twice on cell 2 of the first word of second row i.e. the cell $[1, 2]$. Because the considered CA is periodic, previous row of the first row is the last row (seventh row). It is repeated for second word of second row similarly. The second word of the first row, which determine the number of Sandpile run is $(10)_2 = 2$ and the next word which determine the cell that the Sandpile applied it was $(00)_2 = 0$, Thus on the zero cell of the second word of the second row i.e. cell $[1, 4]$, the Sandpile was performed once. For all rows, these data inferred synchronically.

In the second step, the CA has been updated by the eight mentioned rules. This step comprised three parts. In the first part, a rule for each cell, according to the Table 3

synchronously determined. The number of rules which used in this paper is specified. To determine a rule for cell $[i, j]$, the procedure is shown as following: for cells $[i-1, j-1]$, $[i-1, j]$, $[i-1, j+1]$ and $[i+1, j-1]$, $[i+1, j]$, $[i+1, j+1]$, the XOR operator was used on them correspondingly (i.e. the XOR operator applied on cells $[i-1, j-1]$ and $[i+1, j-1]$ and so forth) and generate an integer between 0 and 7. Fig. 5, shows the PCA Structure and hardware presentation of determined rules 90/ 150/ 165/ 105/ 101/ 86/ 30/153 for cell $[i, j]$.

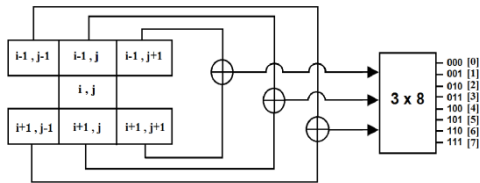


Fig. 5. Proposed PCA with Determined rules for cell $[i, j]$

The obtained number is the rule number that must be applied on the cell $[i, j]$. For instance, if after performing Sandpile action, the values of CA will be Fig. 4(b), the number of determined rule for cell $[1, 1]$ is equal to $111 \oplus 001 = 110$, i.e. which is the 6th rule. In other words, according to Table 3 in performing rule section on CA, the rule which should be applied on the cell $[1, 1]$ is 153.

Table 3. CA rules lookup Table

0 (000)	1 (001)	2 (010)	3 (011)	4 (100)	5 (101)	6 (110)	7 (111)
101	105	86	165	90	30	153	150

In the second part, determined rules were applied. Using rules also is synchronously and the rules would be applied with respect to the rows. For example, in Fig. 4(b), the result after performing rule 153 on cell $[1, 1]$ is equal to $Rule_{153}(101) = 0$. In the third part, for updating the CA, the value of each CA cell should be added to the value that obtained from the used rule on that cell, which will be 0 or 1, and since all values must be an integer between 0 and 3, their integer residual of them by 4 were calculated.

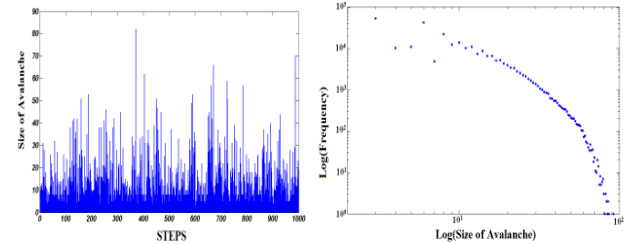
4.2 How Use of Sandpile Model Results in Random State in Cellular Automata?

Role of sandpile model in this model is to actuate and produce the maximum disturbance in cellular automata for preventing from cycle formation and reaching the maximum entropy in cellular automata. As it was stated, The Sandpile is a complex system operating at a critical state between chaos and order and a power-law relationship between the frequency and size of sand "avalanches" is observed. In a system exhibiting critical behavior, A small perturbation in one given location of the system may generate a small effect on its neighbourhoods or a chain reaction that affects all the constituents of the system. The statistical distributions

describing the response of the system exhibiting SOC are given by power laws in the form

$$P(s) \sim s^{-\tau} \tag{5}$$

where s is the number of constituents of the system affected by the perturbation, d is the duration of the chain reaction(lifetime), and τ are constants. Large avalanches are very rare while small ones appear very often. Without any fine-tuning of parameters, the system evolves to a non-equilibrium critical state. Fig. 6 shows a distribution of avalanches created by our sandpile model with a dimension of 12×12 , which has been running for 100000 steps.



(a) Size of avalanches over time (steps); right: Log-log transformation of the size of avalanches in relation to their frequency of occurrence

Fig. 6. Power law number output of the sandpile model.

As it is shown in figure 6, behavior of applied sandpile model in the proposed generator follows power law and the number of avalanche occurrences is inversely proportional with the size of avalanche. The average occurred state change in cellular automata was measured equal to %47.83 after performing sandpile on rows.

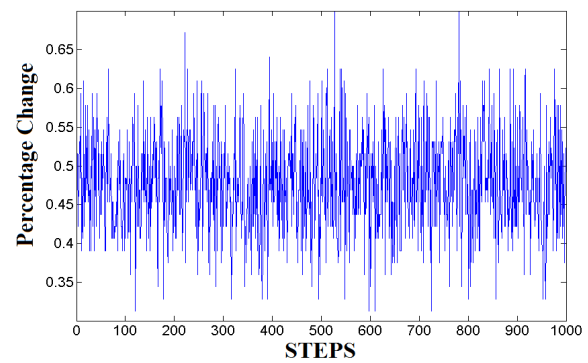


Fig. 7. Percentage Changes in Cells After Applying Sandpile

Fig. 7 shows percentage changes in cells after 1000 times sandpile performance. Combining this model with cellular automata, in addition to disturbing cells state, causes a severe mutation in values of cellular automata because of creating huge avalanches and prevents from short period length sequences and leads to the maximum entropy in cells values. Average of Percentage changes after performing rules of cellular automata and performing sandpile action on cells is %50.1.

5 Experimental Results

For analyze the proposed generator, the generated bits sequence divided into 4-bit parts and so, different tests such as entropy, chi-square and the changes sensitivity

and other tests on the obtained numbers were assessed which are between 0 and 15. To perform all tests which will be presented in following sections, an 8×8 cellular automata is applied with random initial values.

5.1 Several Basic Statistical Tests For PRNG

Let $s = s_0, s_1, s_2, \dots, s_{n-1}$ be a binary sequence of length n . This subsection presents several basic statistical tests that are commonly used for determining whether the binary sequence s possesses some specific characteristics that a truly random sequence would be likely to exhibit. It is emphasized again that the outcome of each test is not definite, but rather probabilistic.

5.1.1 Frequency Test (Monobit Test)

The purpose of this test is to determine whether the number of 0's and 1's in s are approximately the same, as would be expected for a random sequence. Let n_0, n_1 denote the number of 0's and 1's in s , respectively. The statistic used is:

$$x1 = \frac{(n_0 - n_1)^2}{n} \tag{6}$$

which approximately follows a χ^2 distribution with 1 degree of freedom if $n \geq 10$. For a significance level of $\alpha = 0.05$, the threshold values for this test is 3.8415 [26].

5.1.2 Serial Test (Two-Bit Test)

The purpose of this test is to determine whether the number of occurrences of 00, 01, 10, and 11 as subsequences of s are approximately the same, as would be expected for a random sequence. Let n_0, n_1 denote the number of 0's and 1's in s , respectively, and let $n00, n01, n10, n11$ denote the number of occurrences of 00, 01, 10, 11 in s , respectively. Note that $n00 + n01 + n10 + n11 = (n - 1)$ since the subsequences are allowed to overlap. The statistic used is:

$$\frac{4}{n-1}(n00^2 + n01^2 + n10^2 + n11^2) - \frac{2}{n}(n_0^2 + n_1^2) + 1 \tag{7}$$

which approximately follows a χ^2 distribution with 2 degrees of freedom if $n \geq 21$. For a significance level of $\alpha = 0.05$, the threshold values for this test is 5.9915 [26].

5.1.3 Poker Test

Let m be a positive integer such that $\lfloor \frac{n}{m} \rfloor \geq 5$. (2^m) and let $k = \lfloor \frac{n}{m} \rfloor$. Divide the sequence s into k non-overlapping parts each of length m , and let n_i be the number of occurrences of the i^{th} type of sequence of length $m, 1 \leq i \leq 2^m$. The poker test determines whether the sequences of length m each appear approximately the same number of times in s , as would be expected for a random sequence. The statistic used is:

$$x3 = \frac{2^m}{k} \left(\sum_{i=1}^{2^m} n_i^2 \right) - k \tag{8}$$

Which approximately follows a χ^2 distribution with $2^m - 1$ degrees of freedom. Note that the poker test is a generalization of the frequency test: setting $m = 1$ in the poker test yields the frequency test. For a significance

level of $\alpha = 0.05$, the threshold values for this test is 14.0671 [26].

5.1.4 Autocorrelation Test

The purpose of this test is to check for correlations between the sequence s and (non-cyclic) shifted versions of it. Let d be a fixed integer, $1 \leq d \leq \lfloor n/2 \rfloor$. The number of bits in s not equal to their d -shifts is $A(d) = \sum_{i=0}^{n-d-1} s_i \oplus s_{i+d}$ where \oplus denotes the XOR operator. The statistic used is:

$$x_5 = 2 \left(A(d) - \frac{n-d}{2} \right) / \sqrt{n-d} \tag{9}$$

Which approximately follows an $N(0; 1)$ distribution if $n - d \geq 10$. Since small values of $A(d)$ are as unexpected as large values of $A(d)$, a two-sided test should be used. For a significance level of $\alpha = 0.05$, the threshold values for this test is 1.96 [26]. In Table 4, values of discussed tests are presented for the proposed generator. For this, a sequence of random numbers is generated with 1 million bits and discussed tests are implemented on it. This procedure is repeated 100 times and its average is given, too..

Table 4. Values of 4 basic statistical test

TESTS				Pass
Frequency	Serial	Poker	Autocorrelation	
0.459	2.533	8.851	0.312	4/4

As it is shown in Table 4, generator is able to pass all tests.

5.2 ENT Test

The ENT test is useful for evaluating pseudorandom number generators for encryption and statistical sampling applications, compression algorithms, and other applications where the information density of a file is of interest [27]. The ENT test is a collective term for three tests, known as the Entropy test, Chi-square test, and Serial correlation coefficient (SCC) test. Table 5 shows values of this test for the proposed generator. In entropy test, its maximum value is 4 and Chi-Square test with freedom degree 4 and precision of 0.1 is used. For doing these tests, a sequence of length 2^{17} is used.

Table 5. ENT Test

TESTS		
Entropy	Chi-Square	SCC
3.9999	3.0185	0.00008

As it is represented in above table, generated sequence is able to pass all tests successfully and with good result.

5.3 PRNG Quality Evaluation

To compare how our PRNG performs against several different PRNGs, we use Diehard test suite [28]. For this reason, the proposed generator based on obtained score from DIEHRD test is compared with other generators. we used Johnson's scoring scheme [29]: we initialized ($a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7$) with 32 different random values obtained from <http://randomnumber.org>, got 32 different 10MB files, and then assigned scores based on the results of the Diehard tests. The PRNGs we have compared to

ours are of several different kinds: Linear Congruential Generators (rand [30], rand1k [31], pm [32]), Multiply with Carry Generators (mother [33]), Additive and Subtractive Generators (add [30], sub [32]), Compound Generators (shsub [30], shpm [32], shlec [32]), Feedback Shift Register Generators (tgfsr [34], fsr [35]), and Tausworthe Generators (tauss [36]).

Each of the Diehard tests produces one or more p-values. We categorize them as good, suspect or rejected. We classify a p-value as rejected if $p \geq 0.998$, and as suspect if $0.95 \leq p < 0.998$; all other p-values are considered to be good. We assign two points for every rejection, one point for every suspect classification, and no points for the rest. Finally, we add up these points to produce a global Diehard score for each PRNG, and compute the average over the 32 evaluations: low scores indicate good PRNG quality. The information relating to the different PRNGs was taken from [31, 37]. The results are presented in Table 6. We note that our PRNG is outstandingly better than the rest of the analyzed PRNGs: the lowest scores correspond to shsub (17.125) and fsr (17.90625), significantly greater than our PRNG (12.718750). On the other hand, the average scores increase up to 50.59375 (pm), 66.53125 (rand), and even 291.78125 (rand1k).

Table 6. PRNG Diehard Scores

PRNG	Total Score	Mean
Rand	2129	66.531250
rand1k	9337	291.78125
Pm	1619	50.593750
Mother	602	18.812500
Add	577	18.031250
Sub	655	20.468750
Shsub	548	17.125000
shpm	799	24.968750
shlec	751	23.468750
fsr	573	17.906250
tgfsr	584	18.250000
tauss	935	29.218750
Proposed PRNG	407	12.718750

5.4 Avalanche Effect

Bit change sensitivity analysis is used to analyze the RNGs that are used in cryptography. One of the desired properties in each cryptography algorithm is that a small change in plaintext or key yields salient changes in ciphertext. In special case, changing one bit in key or plaintext should change in half of the ciphertext. This property is known as avalanche and represented by Fiestel in 1973 [16].

As mentioned before, the proposed generator could be used to generate key in cryptography. To generate a unique key in both encoding and decoding, the initial values must be available. Thus, for high security in cryptography, generated bits stream must have too much dependency to this parameter. As mentioned before, a 8×8 CA with an integer between 0 and 3 as the value of each cell was used. Then two bits are needed to determine the value of each cell and 128 bits could determine the

value of cells. Two analysis of this section, 128 bits randomly generated to determine the initial state and the generated bits sequence generated from one cell. Then one of these 128 bits has been inverted and the sequence of generated bits with the new initial state of the same cell generated. At last, the both sequences have been compared with each other. Fig. 8, shows the changes percent of generated sequences from one specific cell with two initial states that differ only in the *ith* bit (horizontal axis).

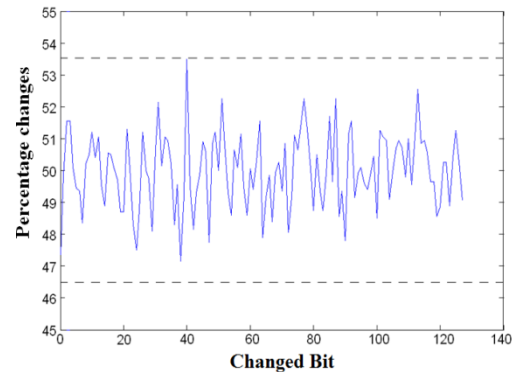


Fig. 8. changes percentage between generated data from two keys that only differ in one bit

6 Conclusion

In this paper, a new PCA for generating random numbers using CA and Sandpile model presented. This method, have the high performance in all tests and could be used in cryptography. Because of avalanche and self organizing properties of Sandpile model, it has a complex behavior and could be used as a convenient factor in stimulating of CA to generate a high quality sequence of random numbers. In each step, first the Sandpile process applied on the four neighbors of each cell of the two dimensional automata and then the CA has been updated by the synthetic of 8 rules 165, 105, 90, 150, 153, 101, 30, 86. The results of applied tests on the generated numbers show that this generator has the maximum entropy and since passing the chi-square and diehard tests. This generator also has a convenient speed and holds the ability of parallelism of CA.

Reference

- [1] J. Gentle, "Random number generation and Monte Carlo methods," *Springer New York* 2003, 2th edition, 2004, ISBN-10: 0387001786
- [2] A. Reese, "Random number generators in genetic algorithms for unconstrained and constrained optimization," *Nonlinear Analysis: Theory, Methods & Applications*, Vol. 71, pp: 679 - 692, 2009.
- [3] P.L.Ecuyer, "Random numbers for simulation," *Communications of the ACM*, Vol. 33, pp:85-97, 1990.
- [4] M. Tomassini, M. Sipper, and M. Perrenoud, "On the generation of high quality random numbers by two-dimensional Cellular Automata," *IEEE Transactions on Computers*, Vol. 49, pp: 1146 -1151, 2000.

- [5] J.Carmelo, A. Bastos-Filho, D. Jaulio, D. Andrade, R. Marcelo, S. Pita, D. Ramos, "Impact of the Quality of Random Numbers Generators on the Performance of Particle Swarm Optimization," *IEEE International Conference on Systems, Man and Cybernetics*, pp: 4988–4993, 2009.
- [6] S. Wolfram, "Cryptography with cellular automata," in *Proc. CRYPTO 85 Advances in Cryptography*, Vol. 218, pp: 429–432, 1985.
- [7] S. Wolfram, "Theory and Applications of Cellular Automata," *River Edge, NJ: World Scientific*, pp:1983–1986, 1986.
- [8] S.-U. Guan and S. Zhang, "A family of controllable cellular automata for pseudorandom number generation," *International Journal of Modern Physics C*, Vol. 13, Issue 8, pp:1047–1073 2002.
- [9] P. D. Hortensius, R. D. Mcleod, and H. C. Card, "Parallel random number generation for VLSI system using cellular automata," *IEEE Transactions on Computers*, Vol. 38, pp: 1466–1473, 1989.
- [10] P. D. Hortensius, R. D. Mcleod, W. Pries, D. M. Miller, and H. C. Card, "Cellular automata-based pseudorandom number generators for built-in self-test," *IEEE Transactions on Computers*, Vol. 8, pp: 842–859, 1989.
- [11] P. Angheliescu "Encryption Algorithm using Programmable Cellular Automata", World Congress on Internet Security (WorldCIS), pp: 233 – 239, 2011
- [12] S.H. Shin, K.Y. Yoo, "Analysis of 2-State, 3-Neighborhood Cellular Automata Rules for Cryptographic Pseudorandom Number Generation "International Conference on Computational Science and Engineering, CSE '09, pp: 399 – 404, 2009.
- [13] X.Xuwen, L.Yuanxiang, X.Zhuliang, W.Rong, "Data Encryption Based on Multi-Granularity Reversible Cellular Automata", International Conference on Computational Intelligence and Security, 2009. CIS '09, pp: 192 – 196, 2009.
- [14] D. R. Chowdhury, I. S. Gupta, and P. P. Chaudhuri, "A class of two-dimensional cellular automata and applications in random pattern testing," *Journal of Electronic Testing: Theory and Applications*, Vol. 5, pp: 65–80, 1994.
- [15] B.H.Kang, D.H.Lee, C.P.Hong, "High-Performance Pseudorandom Number Generator Using Two-Dimensional Cellular Automata", 4th IEEE International Symposium on Electronic Design, Test and Applications, pp:597 - 602, 2008
- [16] B.H.Kang, D.H.Lee, C.P.Hong, "Pseudorandom Number Generation Using Cellular Automata", Novel Algorithms and Techniques In Telecommunications, Automation and Industrial Electronics, pp:401-404, 2008.
- [17] S.H.Shin, G.D.Park, K.Y.Yoo, "A Virtual Three-Dimension Cellular Automata Pseudorandom Number Generator Based on the Moore Neighborhood Method", 4th International Conference on Intelligent Computing, ICIC 2008, pp: 174-181, 2008.
- [18] A. Ray, D. Das, "Encryption Algorithm for Block Ciphers Based on Programmable Cellular Automata", Information Processing and Management, Vol.70, pp:269-275, 2010.
- [19] N.S.Maiti, S.Ghosh, B.K.Shikdar, P.P.Chaudhuri, "Programmable Cellular Automata (PCA) Based Advanced Encryption Standard (AES) Hardware", 9th International Conference on Cellular Automata for Research and Industry, ACRI, pp:271-274, 2010.
- [20] R. Dogaru, I. Dogaru, and H. Kim, "Binary chaos synchronization in elementary cellular automata", *Int. J. Bifurcation and Chaos*, 19, 2009.
- [21] R. Dogaru, I. Dogaru, H.Kim, "Synchronization in elementary cellular automata", *Proceedings of the 10th International Workshop on Multimedia Signal Processing and Transmission (MSPT'08)*, July 21-22, pp. 35-40, 2008.
- [22] I.Kokolakis, I.Andreadis, and P. Tsalids, "Comparison between cellular automata and linear feedback shift registers based pseudo-random number generators," *Microprocessors and Microsystems*, Vol. 20, pp: 643–658, 1997.
- [23] S. Nandi, B. K. Kar, and P. P. Chowdhuri, "Theory and applications of cellular automata in cryptography," *IEEE Transactions on Computers*, Vol. 43, pp:1346–1357, 1994.
- [24] P.Bak, C.Tang, K.Wiesenfeld, "Self-organized criticality: an explanation of 1/f noise", *Physical Review of Letters*, pp: 381-384, 1987
- [25] F. Seredynski, P. Bouvry, and A.Y. Zomaya, "Cellular automata computations and secret key cryptography," *Parallel Computing*, Vol.30, pp: 753-766, 2004.
- [26] Alfred J. Menezes, Paul C. van Oorschot, Scott A.Vanstone "Handbook of Applied Cryptograph", CRC Press; 1 edition, pp:181-183, 1996, ISBN-10: 0849385237.
- [27] ENT Test Suite, [http:// www.fourmilab.ch/random](http://www.fourmilab.ch/random)
- [28] G.Marsaglia, Diehard test, <http://stat.fsu.edu/~geo/diehard.html>, 1998.
- [29] B.C. Johnson. Radix-b extensions to some common empirical tests for PRNGs. *ACM Trans. on Modeling and Comp. Sim.*, 6(4):261–273, 1996.
- [30] D.E. Knuth. *The Art of Computer Programming, Volume 2*, Addison-Wesley, 3rd edition, 1998.
- [31] M.M. Meysenburg and J.A. Foster. Randomness and GA performance, revisited. In *Proc. of GECCO '99*, volume 1, pages 425–432. Morgan Kaufmann, 1999.
- [32] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, 1992.
- [33] G. Marsaglia. Yet another RNG. Posted to sci.stat.math, 1994.
- [34] M. Matsumoto and Y. Kurita. Twisted GFSR generators. *ACM Trans. on Modeling and Comp. Sim.*, 2(3):179–194, 1992.
- [35] B. Schneier. *Applied Cryptography*. John Wiley, 1994.
- [36] S. Tezuka and P. L'Ecuyer. Efficient and portable combined Tausworthe Random Number Generators. *ACM Trans. on Modeling and Comp. Sim.*, 1(2):99–112, 1991.
- [37] M.M. Meysenburg and J.A. Foster. The quality of PRNGs and simple genetic algorithm performance. In *Proc. of the 7th Int. Conference on Genetic Algorithms*, pp: 276–281, 1997.
- [38] W. Stallings, "Cryptography and Network Security," *Prentice Hall, 3th Edition*, 2002, ISBN-10: 0130914290.

The Role of Orthomodularity in the Marsden-Herman Theorem in Quantum Logic

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA

Abstract

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of quantum-mechanical behaviors and operations. In much the same way that the structure of conventional propositional (Boolean) logic (BL) is the logic of the description of the behavior of classical physical systems and is isomorphic to a Boolean algebra (BA), so also the algebra, $C(H)$, of closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space is a logic of the descriptions of the behavior of quantum mechanical systems and is a model of an ortholattice (OL). An OL can thus be thought of as a kind of "quantum logic" (QL). $C(H)$ is also a model of an orthomodular lattice, which is an OL conjoined with the orthomodularity axiom (OMA). Because the propositions of a QL are not in general commutative, quantum logicians have paid much attention to "quasi"-commutative theorems, one of the better known of which is the Marsden-Herman theorem (MHT). In a QL, the non-commutativity of (certain) observables can be captured as the failure of the (Boolean) distributive law. Informally, the MHT states that if there is a cyclic chain of commuting elements in an orthomodular lattice, a strong version of the distributive law holds. Here I provide an automated deduction of the MHT that uses the OMA and show that OMA is required by the MHT.

Keywords: automated deduction, quantum computing, orthomodular lattice, Hilbert space

1.0 Introduction

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of the description of quantum-mechanical behaviors ([1], [17], [18], [20]). In much the same way that conventional propositional (Boolean) logic (BL,[12]) is the logical structure of description of the behavior of classical physical systems (e.g. "the measurements of the position and momentum of particle P are commutative") and is isomorphic to a Boolean lattice ([10], [11], [19]), so also the algebra, $C(H)$, of the closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space H ([1], [4], [6], [9], [13]) is a logic of the descriptions of the behavior of quantum mechanical systems (e.g., "the measurements of the position and momentum of particle P are *not* commutative") and is a model ([10]) of an ortholattice (OL; [8]). An OL can thus be thought of as a kind of "quantum logic" (QL; [19]). $C(H)$ is also a model of (i.e., isomorphic to a set of sentences which hold in) an orthomodular lattice (OML; [7], [8]), which is an OL conjoined with the orthomodularity axiom (OMA; see Figure 1). The rationalization of the OMA as a claim proper to physics has proven problematic ([13], Section 5-6), motivating the question of whether the OMA is required in an adequate characterization of QL. Thus formulated, the question suggests that the OMA is specific to an OML, and that as a consequence, banning the OMA from QL yields a "truer" quantum logic.

Lattice axioms

$$\begin{aligned}
x &= c(c(x)) && (\text{AxLat1}) \\
x \vee y &= y \vee x && (\text{AxLat2}) \\
(x \vee y) \vee z &= x \vee (y \vee z) && (\text{AxLat3}) \\
(x \wedge y) \wedge z &= x \wedge (y \wedge z) && (\text{AxLat4}) \\
x \vee (x \wedge y) &= x && (\text{AxLat5}) \\
x \wedge (x \vee y) &= x && (\text{AxLat6})
\end{aligned}$$

Ortholattice axioms

$$\begin{aligned}
c(x) \wedge x &= 0 && (\text{AxOL1}) \\
c(x) \vee x &= 1 && (\text{AxOL2}) \\
x \wedge y &= c(c(x) \vee c(y)) && (\text{AxOL3})
\end{aligned}$$

Orthomodularity axiom

$$y \vee (c(y) \wedge (x \vee y)) = x \vee y \quad (\text{AxOM})$$

where

x, y are variables ranging over lattice nodes
 \wedge is lattice meet
 \vee is lattice join
 $c(x)$ is the orthocomplement of x
 $=$ is equivalence ([12])
 1 is the maximum lattice element ($= x \vee c(x)$)
 0 is the minimum lattice element ($= c(1)$)

Figure 1. Lattice, ortholattice, and orthomodularity axioms.

In QL, the non-commutativity of (certain) observables can be captured as the failure of the distributive law ($x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$). (This is a lattice-theoretic way of representing non-commutativity; a physicist would likely say that non-commutativity is fundamental and the failure of distributivity is derivative. The two representations are formally equivalent.) A QL (without AxOM), in fact, can be thought of as a BL in which the

distribution law does not hold. Because of the fundamental role that non-commutativity plays in QL, quantum logicians have paid much attention to "quasi"-commutative theorems, which help to ground a large class of equivalence representations in quantum logic, and are thus of potential interest in optimizing quantum circuit design. Among the better known of the quasi-commutative theorems is the Marsden-Herman Theorem (MHT, [8]), shown in Figure 2

If $u, z, w,$ and x are elements of an orthomodular lattice and

$$C(u, z) \ \& \ C(z, w) \ \& \ C(w, x) \ \& \ C(x, u)$$

then

$$\begin{aligned}
((u \vee z) \wedge (w \vee x)) &= \\
&((u \wedge w) \vee (u \wedge x)) \vee ((z \wedge w) \vee (z \wedge x)).
\end{aligned}$$

where $C(x,y)$, "x commutes with y", is defined as

$$C(x,y) \leftrightarrow (x = ((x \wedge y) \vee (x \wedge c(y))))$$

\leftrightarrow means "if and only if"

Figure 2. The Marsden-Herman Theorem.

Informally stated, the MHT says that if there is a four-element cyclic commutative chain of elements in an orthomodular lattice, then a strong distribution law holds for those elements.

executed in that framework on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 (clocked @ 2.33 GHz) and 8.00 GB RAM, running under the *Windows Vista Home Premium (SP2)/Cygwin* operating environment.

2.0 Method

The OML axiomatizations of Megill, Pavičić, and Horner ([5], [14], [15], [16], [21], [22]) were implemented in a *prover9* ([2]) script ([3]) configured to derive the MHT, and to show that the MHT requires the orthomodularity axiom, (AxOM), then

3.0 Results

Figure 3 shows the proof of the MHT produced by [3] on the platform described in Section 2.0.

```

===== PROOF =====
% Proof 1 at 59.44 (+ 0.37) seconds.
% Length of proof is 81.
% Level of proof is 13.

2 C(x,y) <-> x = (x ^ y) v (x ^ c(y)) # label("Df--commutes") # label(non_clause).
[assumption].
3 C(u,z) & C(z,w) & C(w,x) & C(x,u) # label(non_clause). [assumption].
4 (u v z) ^ (w v x) = ((u ^ w) v (u ^ x)) v ((z ^ w) v (z ^ x)) # label(non_clause) #
label(goal). [goal].
8 -C(x,y) | (x ^ y) v (x ^ c(y)) = x # label("Df--commutes"). [clausify(2)].
9 C(x,y). [clausify(3)].
13 x = c(c(x)) # label("AxLat1"). [assumption].
14 c(c(x)) = x. [copy(13),flip(a)].
15 x v y = y v x # label("AxLat2"). [assumption].
16 (x v y) v z = x v (y v z) # label("AxLat3"). [assumption].
18 x v (x ^ y) = x # label("AxLat5"). [assumption].
19 x ^ (x v y) = x. [assumption].
20 c(x) ^ x = 0 # label("AxOL1"). [assumption].
21 c(x) v x = 1 # label("AxOL2"). [assumption].
22 x v c(x) = 1. [copy(21),rewrite([15(2)])].
23 x ^ y = c(c(x) v c(y)) # label("AxOL3"). [assumption].
24 x v (c(x) ^ (y v x)) = y v x # label("AxOM"). [assumption].
25 x v c(x v c(y v x)) = y v x. [copy(24),rewrite([23(3),14(2)])].
36 ((c1 ^ c3) v (c1 ^ c4)) v ((c2 ^ c3) v (c2 ^ c4)) != (c1 v c2) ^ (c3 v c4).
[deny(4)].
37 c(c(c1 v c2) v c(c3 v c4)) != c(c(c1) v c(c3)) v (c(c(c1) v c(c4)) v (c(c(c2) v c(c3))
v c(c(c2) v c(c4))))).
[copy(36),rewrite([23(3),23(9),23(16),23(22),16(27),23(34)]),flip(a)].
38 (x ^ y) v (x ^ c(y)) = x. [resolve(9,a,8,a)].

```

```

39 c(c(x) v y) v c(c(x) v c(y)) = x. [copy(38),rewrite([23(1),23(6),14(7),15(8)])].
40 c(1) = 0. [back_rewrite(20),rewrite([23(2),14(2),22(2)])].
41 c(c(x) v c(y v y)) = x. [back_rewrite(19),rewrite([23(2)])].
42 x v c(c(x) v c(y)) = x. [back_rewrite(18),rewrite([23(1)])].
43 x v (y v z) = y v (x v z). [para(15(a,1),16(a,1,1)),rewrite([16(2)])].
44 c(c(c1 v c2) v c(c3 v c4)) != c(c(c1) v c(c3)) v (c(c(c2) v c(c3)) v (c(c(c1) v c(c4))
v c(c(c2) v c(c4))))). [back_rewrite(37),rewrite([45(36)])].
45 x v (c(x v c(y v x)) v z) = y v (x v z).
[para(25(a,1),16(a,1,1)),rewrite([16(2)]),flip(a)].
46 x v (y v c(x v (y v c(z v (x v y)))))) = z v (x v y).
[para(25(a,1),16(a,1,1)),rewrite([16(7)]),flip(a)].
47 x v c(x v c(y v (z v x))) = y v (z v x).
[para(16(a,1),25(a,1,2,1,2,1)),rewrite([16(8)])].
48 c(x v y) v c(x v c(y)) = c(x). [para(14(a,1),39(a,1,1,1,1)),rewrite([14(4)])].
49 0 v c(c(x) v c(x)) = x. [para(22(a,1),39(a,1,1,1,1)),rewrite([40(2),14(4)])].
50 c(x) v c(x v y) = c(x). [para(41(a,1),14(a,1,1)),flip(a)].
51 c(x v x) = c(x). [para(41(a,1),41(a,1,1,2)),rewrite([14(2)])].
52 0 v x = x. [back_rewrite(60),rewrite([73(5),14(3)])].
53 x v c(c(x) v y) = x. [para(14(a,1),42(a,1,2,1,2))].
54 x v c(x v c(x)) = x. [para(25(a,1),42(a,1,2,1))].
55 x v x = x. [para(40(a,1),42(a,1,2,1,2)),rewrite([15(3),75(3),14(2)])].
56 x v (y v c(x v c(z v x))) = y v (z v x). [para(25(a,1),45(a,1,2)),flip(a)].
57 x v (y v x) = y v x. [para(82(a,1),16(a,2,2)),rewrite([15(2)])].
58 c(x) v (c(c(x) v z) v z) = x v z. [para(76(a,1),16(a,1,1)),flip(a)].
59 x v (y v c(c(x) v z)) = y v x. [para(76(a,1),45(a,1,2)),flip(a)].
60 c(x) v c(y v x) = c(x). [para(14(a,1),81(a,1,2,1,2))].
61 x v (c(y v c(x)) v z) = x v z. [para(81(a,1),16(a,1,1)),flip(a)].
62 c(x) v (c(x v y) v z) = c(x) v z. [para(66(a,1),16(a,1,1)),flip(a)].
63 c(x v y) v c(x v (y v z)) = c(x v y). [para(16(a,1),66(a,1,2,1))].
64 c(c(x) v y) v (z v x) = z v x.
[para(76(a,1),52(a,1,2,2,1,2,2,1,2)),rewrite([108(10),85(9),76(9)])].
65 c(x) v (c(y v x) v z) = c(x) v z. [para(112(a,1),16(a,1,1)),flip(a)].
66 c(x) v (y v c(z v x)) = y v c(x). [para(112(a,1),45(a,1,2)),flip(a)].
67 c(x v y) v (c(y v c(x v y)) v z) = c(y) v z.
[para(112(a,1),51(a,1,2,1,1,2,1)),rewrite([14(6),15(5),185(13)])].
68 x v c(x v c(y v (z v (u v x)))) = y v (z v (u v x)).
[para(16(a,1),53(a,1,2,1,2,1,2)),rewrite([16(9)])].
69 c(x v y) v c(y v c(x)) = c(y). [para(15(a,1),56(a,1,1,1))].
70 c(x v y) v c(c(y) v x) = c(x). [para(15(a,1),56(a,1,2,1))].
71 x v c(x v c(y)) = x v y.
[para(56(a,1),56(a,1,1,1)),rewrite([14(2),14(6),15(5),16(5),112(4),14(7)])].
72 x v (y v (z v (u v x))) = y v (z v (u v x)). [back_rewrite(202),rewrite([253(7)])].
73 x v c(x v y) = x v c(y). [para(14(a,1),253(a,1,2,1,2))].
74 x v c(y v x) = x v c(y). [para(15(a,1),257(a,1,2,1))].
75 x v (c(x v y) v z) = x v (c(y) v z).
[para(257(a,1),16(a,1,1)),rewrite([16(3)]),flip(a)].
76 x v (y v c(x v z)) = y v (x v c(z)). [para(257(a,1),45(a,1,2)),flip(a)].
77 c(x v y) v (c(y v c(x)) v z) = c(y) v z. [back_rewrite(190),rewrite([305(5)])].
78 c(x v y) v c(c(z v c(x)) v y) = c(c(z v c(x)) v y).
[para(113(a,1),112(a,1,2,1)),rewrite([15(8)])].
79 c(x v y) v c(c(x) v y) = c(y). [para(15(a,1),236(a,1,2,1))].
80 c(c(x v y) v z) = c(c(x) v z) v c(x v (c(y) v z)).
[para(129(a,1),236(a,1,1,1)),rewrite([14(8),15(7),306(7)]),flip(a)].
81 c(c(x) v y) v c(x v (z v y)) = c(c(x) v y) v c(z v y).
[back_rewrite(401),rewrite([491(7),14(7),188(10),491(11),14(11)]),flip(a)].
82 c(c(c1) v c(c3 v c4)) v c(c(c2) v c(c3 v c4)) != c(c(c1) v c(c3)) v (c(c(c2) v c(c3))
v (c(c(c1) v c(c4)) v c(c(c2) v c(c4))))). [back_rewrite(46),rewrite([491(10),500(19)])].
83 c(x v y) v (z v c(c(y) v x)) = z v c(x). [para(237(a,1),45(a,1,2)),flip(a)].
84 c(x v c(y v z)) = c(x v (y v c(z))) v c(x v c(y)).
[para(110(a,1),237(a,1,2,1)),rewrite([14(2),15(4),308(4),14(10)]),flip(a)].
85 c(c(c1) v c(c3)) v (c(c(c2) v c(c3)) v (c(c3 v (c(c1) v c(c4))) v c(c3 v (c(c2) v
c(c4)))))) != c(c(c1) v c(c3)) v (c(c(c2) v c(c3)) v (c(c(c1) v c(c4)) v c(c(c2) v
c(c4))))).
[back_rewrite(507),rewrite([516(8),45(7),15(15),516(23),45(22),15(30),45(31),16(30),45(31)
)])].
86 c(x v y) v c(y v x) = c(x v y). [para(93(a,1),130(a,1,2,1))].
87 c(x v c(y)) v (c(x v (z v c(y))) v c(x v c(z))) = c(x v (z v c(y))) v c(x v c(z)).
[para(236(a,1),130(a,1,2,1,2)),rewrite([516(4),15(12),516(16)])].
88 c(x v y) v (c(c(x) v y) v z) = c(y) v z. [para(481(a,1),16(a,1,1)),flip(a)].
89 c(x v y) v (c(x v (z v y)) v u) = c(x v y) v u. [para(45(a,1),185(a,1,2,1,1))].

```

```

664 c(x v (y v c(z))) v c(x v c(y)) = c(x v c(z)) v c(x v c(y)).
[back_rewrite(549),rewrite([656(12)]),flip(a)].
677 c(x v c(y v z)) = c(x v c(z)) v c(x v c(y)). [back_rewrite(516),rewrite([664(12)])].
680 c(x) v (y v (z v c(u v x))) = y v (z v c(x)).
[para(16(a,1),188(a,1,2)),rewrite([16(9)])].
814 c(x v (y v z)) v c(y v x) = c(y v x).
[para(542(a,1),174(a,1,2)),rewrite([14(3),16(2),542(11)])].
1741 c(x v (y v z)) v (c(y v x) v u) = c(y v x) v u. [para(814(a,1),16(a,1,1)),flip(a)].
2626 c(x v c(y)) v (c(y v (x v z)) v u) = c(x v c(y)) v (c(x v z) v u).
[para(512(a,1),322(a,1,1,1)),rewrite([14(10),15(9),16(9),110(8),677(13),14(13),16(15)])].
2886 c(c(x) v y) v (z v (u v c(x v y))) = z v (u v c(y)).
[para(563(a,1),255(a,1)),rewrite([680(6)]),flip(a)].
26019 c(x v c(y)) v (z v c(y v (x v u))) = c(x v c(y)) v (z v c(x v u)).
[para(2886(a,1),1741(a,1)),flip(a)].
26036 c(c(c1) v c(c3)) v (c(c(c2) v c(c3)) v (c(c3 v (c(c1) v c(c4))) v c(c(c2) v
c(c4)))) != c(c(c1) v c(c3)) v (c(c(c2) v c(c3)) v (c(c(c1) v c(c4)) v c(c(c2) v
c(c4))))). [back_rewrite(525),rewrite([26019(30)])].
26094 $F. [para(45(a,1),26036(a,1)),rewrite([2626(28),45(27)]),xx(a)].

===== end of proof =====

```

Figure 3. Summary of a *prover9* ([2]) proof of the Marsden-Herman Theorem ([8]). The proof assumes the inference rules of *prover9*. The general form of a line in this proof is “*line_number conclusion [derivation]*”, where *line_number* is a unique identifier of a line in the proof, and *conclusion* is the result of applying the *prover9* inference rules (such as *paramodulation*, *copying*, and *rewriting*), noted in square brackets (denoting the *derivation*), to the lines cited in those brackets. Note that some of “logical” proof lines in the above have been transformed to two text lines, with the *derivation* appearing on a text line following a text line containing the first part of that logical line. The detailed syntax and semantics of these notations can be found in [2]. All *prover9* proofs are by default proofs by contradiction. Note that this proof uses the orthomodularity axiom, AxOM.

The total time to produce the proof in Figure 3 on the platform described in Section 2.0 was ~1 minute.

Not only does the proof in Figure 3 use the modularity axiom AxOM, AxOM is *required* by the MHT, as Figure 4 shows.

```

===== PROOF =====

% Proof 1 at 0.08 (+ 0.03) seconds: "AxOM".
% Length of proof is 48.
% Level of proof is 11.

2 C(x,y) <-> x = (x ^ y) v (x ^ c(y)) # label("Df--commutes") # label(non_clause).
[assumption].
3 C(u,z) & C(z,w) & C(w,x) & C(x,u) -> (u v z) ^ (w v x) = ((u ^ w) v (u ^ x)) v ((z ^ w)
v (z ^ x)) # label("Marsden-Herman Theorem") # label(non_clause). [assumption].
4 y v (c(y) ^ (x v y)) = x v y # label("AxOM") # label(non_clause) # label(goal).
[goal].
7 x = c(c(x)) # label("AxLat1"). [assumption].
8 c(c(x)) = x. [copy(7),flip(a)].
9 x v y = y v x # label("AxLat2"). [assumption].
10 (x v y) v z = x v (y v z) # label("AxLat3"). [assumption].
12 x v (x ^ y) = x # label("AxLat5"). [assumption].
13 x ^ (x v y) = x. [assumption].
14 c(x) ^ x = 0 # label("AxOL1"). [assumption].
15 c(x) v x = 1 # label("AxOL2"). [assumption].

```

```

16 x v c(x) = 1. [copy(15),rewrite([9(2)])].
17 x ^ y = c(c(x) v c(y)) # label("AxOL3"). [assumption].
28 -C(x,y) | (x ^ y) v (x ^ c(y)) = x # label("Df--commutes"). [clausify(2)].
29 -C(x,y) | c(c(x) v y) v c(c(x) v c(y)) = x.
[copy(28),rewrite([17(2),17(7),8(8),9(9)])].
30 C(x,y) | (x ^ y) v (x ^ c(y)) != x # label("Df--commutes"). [clausify(2)].
31 C(x,y) | c(c(x) v y) v c(c(x) v c(y)) != x.
[copy(30),rewrite([17(2),17(7),8(8),9(9)])].
32 -C(x,y) | -C(y,z) | -C(z,u) | -C(u,x) | ((x ^ z) v (x ^ u)) v ((y ^ z) v (y ^ u)) = (x
v y) ^ (z v u) # label("Marsden-Herman Theorem"). [clausify(3)].
33 -C(x,y) | -C(y,z) | -C(z,u) | -C(u,x) | c(c(x v y) v c(z v u)) = c(c(x) v c(z)) v
(c(c(x) v c(u)) v (c(c(y) v c(z)) v c(c(y) v c(u))))).
[copy(32),rewrite([17(5),17(9),17(14),17(18),10(23),17(26)]),flip(e)].
34 c1 v (c(c1) ^ (c2 v c1)) != c2 v c1 # label("AxOM") # answer("AxOM"). [deny(4)].
35 c1 v c(c1 v c(c1 v c2)) != c1 v c2 # answer("AxOM").
[copy(34),rewrite([9(6),17(7),8(4),9(12)])].
36 c(1) = 0. [back_rewrite(14),rewrite([17(2),8(2),16(2)])].
37 c(c(x) v c(x v y)) = x. [back_rewrite(13),rewrite([17(2)])].
38 x v c(c(x) v c(y)) = x. [back_rewrite(12),rewrite([17(1)])].
40 x v (y v z) = y v (x v z). [para(9(a,1),10(a,1,1)),rewrite([10(2)])].
42 x v (c(x) v y) = 1 v y. [para(16(a,1),10(a,1,1)),flip(a)].
49 C(x,c(x)) | 0 v c(c(x) v c(x)) != x.
[para(16(a,1),31(b,1,2,1)),rewrite([36(8),9(8)])].
50 c(0) = 1. [para(36(a,1),8(a,1,1))].
53 C(x,1) | c(0 v c(x)) v c(1 v c(x)) != x.
[para(36(a,1),31(b,1,2,1,2)),rewrite([9(5),9(9),9(11)])].
54 c(x) v c(x v y) = c(x). [para(37(a,1),8(a,1,1)),flip(a)].
58 c(0 v c(x)) = x. [para(16(a,1),37(a,1,1,2,1)),rewrite([36(3),9(3)])].
60 C(x,x v y) | c(1 v y) v x != x. [para(37(a,1),31(b,1,2)),rewrite([40(5),42(5)])].
61 1 v x = 1. [para(36(a,1),37(a,1,1,1)),rewrite([58(6)])].
62 c(x v x) = c(x). [para(37(a,1),37(a,1,1,2)),rewrite([8(2)])].
63 C(x,1) | x v 0 != x. [back_rewrite(53),rewrite([58(6),61(5),36(4)])].
65 C(x,x v y) | 0 v x != x. [back_rewrite(60),rewrite([61(4),36(4)])].
68 C(x,c(x)) | 0 v x != x. [back_rewrite(49),rewrite([62(7),8(5)])].
74 x v 0 = x. [para(16(a,1),38(a,1,2,1)),rewrite([36(2)])].
76 x v x = x. [para(36(a,1),38(a,1,2,1,2)),rewrite([9(3),58(4)])].
78 C(x,1). [back_rewrite(63),rewrite([74(4)]),xx(b)].
81 0 v x = x. [hyper(29,a,78,a),rewrite([9(3),61(3),36(2),36(4),9(4),58(5)])].
83 C(x,c(x)). [back_rewrite(68),rewrite([81(4)]),xx(b)].
84 C(x,x v y). [back_rewrite(65),rewrite([81(4)]),xx(b)].
88 C(0,x).
[para(50(a,1),31(b,1,1,1,1)),rewrite([61(4),36(4),50(5),61(6),36(5),76(5)]),xx(b)].
89 C(x,0).
[para(50(a,1),31(b,1,2,1,2)),rewrite([9(5),81(5),8(4),9(5),61(5),36(4),74(4)]),xx(b)].
95 C(c(x),x). [para(8(a,1),83(a,2))].
112 x v c(x v c(x v y)) = x v y.
[hyper(33,a,89,a,b,88,a,c,95,a,d,84,a),rewrite([9(3),81(3),9(4),16(4),36(4),9(4),81(4),8(
3),8(5),9(4),9(9),54(9),8(7),50(7),8(8),61(7),36(7),50(8),61(9),36(8),76(8),74(7),9(6)]),
flip(a)].
113 $F # answer("AxOM"). [resolve(112,a,35,a)].

```

===== end of proof =====

Figure 4. Summary of *prover9* proof showing the MHT requires the orthomodularity axiom, AxOM.

The total time to complete the proof shown in Figure 4 on the platform described in Section 2.0 was 0.11 seconds.

4.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, and the late John K. Prentice, whose passion for foundations of physics inspired those of us privileged to have known him. For any infelicities that remain, I am solely responsible.

5.0 References

- [1] von Neumann J. *Mathematical Foundations of Quantum Mechanics*. 1936. Translated by R. T. Beyer. Princeton. 1983.
- [2] McCune WW. *prover9 and mace4*. URL <http://www.cs.unm.edu/~mccune/prover9/>. 2009.
- [3] Horner JK. MHT *prover9* scripts. 2011. Available from the author on request.
- [4] Dalla Chiara ML and Giuntini R. *Quantum Logics*. URL <http://xxx.lanl.gov/abs/quant-ph/0101028>. 2004.
- [5] Megill ND and Pavičić M. Orthomodular lattices and quantum algebra. *International Journal of Theoretical Physics* 40 (2001), pp. 1387-1410.
- [6] Akhiezer NI and Glazman IM. *Theory of Linear Operators in Hilbert Space. Volume I*. Translated by M. Nestell. Frederick Ungar. 1961.
- [7] Holland, Jr. SS Orthomodularity in infinite dimensions: a theorem of M. Solèr. *Bulletin of the American Mathematical Society* 32 (1995), pp. 205-234.
- [8] Marsden EL and Herman LM. *A condition for distribution in orthomodular lattices*. Kansas State University Technical Report #40. 1974.
- [9] Knuth DE and Bendix PB. Simple word problems in universal algebras. In J. Leech, ed. *Computational Problems in Abstract Algebra*. Pergamon Press. 1970. pp. 263-297.
- [10] Chang CC and Keisler HJ. *Model Theory*. North-Holland. 1990. pp. 38-39.
- [11] Birkhoff G. *Lattice Theory*. Third Edition. American Mathematical Society. 1967.
- [12] Church A. *Introduction to Mathematical Logic. Volume I*. Princeton. 1956.
- [13] Jauch J. *Foundations of Quantum Mechanics*. Addison-Wesley. 1968.
- [14] Megill ND. *Metamath*. URL <http://us.metamath.org/qlegif/mmql.html#unify>. 2004.
- [15] Horner JK. An automated deduction system for orthomodular lattice theory. *Proceedings of the 2005 International Conference on Artificial Intelligence*. CSREA Press. 2005. pp. 260-265.
- [16] Horner JK. An automated equational logic deduction of join elimination in orthomodular lattice theory. *Proceedings of the 2007 International Conference on Artificial Intelligence*. CSREA Press. 2007. pp. 481-488.
- [17] Messiah A. *Quantum Mechanics*. Dover. 1958.
- [18] Horner JK. Using automated theorem-provers to aid the design of efficient compilers for quantum computing. *Los Alamos National Laboratory Quantum Institute Workshop*. December 9–10, 2002. URL http://www.lanl.gov/science/centers/quantum/qls_pdfs/horner.pdf.
- [19] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-243.
- [20] Nielsen MA and Chuang L. *Quantum Computation and Quantum Information*. Cambridge. 2000.
- [21] Pavičić M and Megill N. Quantum and classical implicational algebras with primitive implication. *International Journal of Theoretical Physics* 37 (1998), 2091-2098. <ftp://m3k.grad.hr/pavicic/quantum-logic/1998-int-j-theor-phys-2.ps.gz>.
- [22] Horner JK. An automated deduction of the relative strength of orthomodular and weakly orthomodular lattice theory. *Proceedings of the 2009 International Conference on Artificial Intelligence*. CSREA Press. 2009. pp. 525-530.

Requiring Multiplicity for Nondeterministic Automata

Jeffrey W. Holcomb^{1,2}

¹Holcomb Technologies, Irving, Texas, USA

²Computer Science and Engineering, Southern Methodist University, Dallas, Texas USA

Abstract—This paper is the second in a series of papers to formalize the proof presented in [1] that the set difference between NP and P is not the empty set \emptyset . The critical difference between the proof presented in [1] and previous attempts at this problem is that the proof in [1] focuses on the nature of the answers to our set of problems belonging to P and NP instead of the set of problem directly. In a previous work [2] we presented a formalization of deterministic automata in terms of [1]. Here, we will begin the formalization of nondeterministic automata in terms of the structures required by [1]. Specifically, we will demonstrate that, for nondeterministic automata, at least one of the stage games for our Bayesian/Markov game will need to be non-cooperative.

Key words: Automata and formal languages; Complexity Theory; Game Theory

1 Introduction

Earlier this year our group proposed redirecting the question of whether or not $|\text{NP/P}| > 0$ from focusing on the problems being studied to focusing on the nature of the answers to said problems [1]. Specifically, and bearing in mind the structural constraints of deterministic automata, can deterministic automaton generate truly stochastic answers [1], [2]?

Many talented researchers have contributed greatly to our understanding of this problem [3], [4], [5], [6], [7], and many attempts have been made to solve this problem [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22]; most recently a proof proposed by Vinay Deolalikar. The question regarding the emptiness of $|\text{NP/P}|$ has been one of the most elusive questions in mathematics and computer science since it was first discussed in a 1965 paper by Jack Edmonds [23]; however until recently, the focus has been on the problems belonging to P and NP: not the answers to said problems. It is our hope that a change in focus will finally bring an end to this long debate.

In this paper we continue to formalize our high level proof [1]. Specifically, we will begin to formalize the structural properties of nondeterministic automata. Because of the flexibility of nondeterministic automata, this is actually a much simpler task than our formalization of deterministic automata [2]. First we will discuss the structure of the stage games G^τ for G_B and then the structure for our component

Markov games with respect to single automaton/algorithm implementations.

This paper is divided into six sections: Introduction, Definitions, Background, Research, Conclusion, and Future Research. Our Definitions section covers the definitions specifically relevant to this paper, the Background section covers work done by previous researchers that is relevant to this paper, the Research section presents the research done by our group, the Conclusion section is a brief synopsis of the conclusions drawn by our group, and the Future Research section outlines the work that must still be completed.

2 Definitions

Definition 1 A **stochastic answer** \mathcal{A}_{sto} to a problem P is an answer that has both the internal quality of randomness and the external quality of randomness in occurrence such that a problem instance $\iota \in P$ cannot be mapped directly to a single given answer $\mathcal{A}[1]$.

Definition 2 A **non-stochastic answer** $\mathcal{A}_{non-sto}$ to a problem P is an answer that can be determined directly from the supplied problem instance ι such that $\iota \models \mathcal{A}[1]$.

3 Background

The question regarding the emptiness of $|\text{NP/P}|$ has been recognized by the Clay Mathematics Institute as one of the top seven hardest questions in mathematics, and has been an open question since it was first investigated by Jack Edmonds [23] in 1965. Many researchers have contributed greatly to our understanding of this problem [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22]; however, our approach differs substantially from previous attempts to answer this question. Specifically, most researchers begin their investigation by analyzing the various problems that belong to P and NP; whereas, we began our investigation by looking for methodologies to model system equilibria. Secondly, after a bit of analysis we came to the conclusion that it would be more beneficial to investigate the characteristics of the answers to problems belonging to P and NP. This led us to the formulation of stochastic answers their application to the question of whether or not $|\text{NP/P}| = \emptyset$ [1]. In [1], we presented a high level outline for the arguments that we will be utilizing in our proof. We then began to

formalize this proof in [2] with respect to deterministic automata.

All of the main structural elements utilized by our proof are taken from temporal logic [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35] and extensive form games. The arguments presented by our proof are with respect to Bayes-Nash equilibria [36] over our temporal logic structures. For our research, we are mainly interested in the game theoretic formulations for the various forms of temporal logic [26], [32], [33], [37], [38], [39], [40], [41].

4 Research

4.1 Model Structure

The purpose of this section is to provide an overview of the structure utilized by the arguments in this paper to model automata. For our arguments, we are not concerned about specific implementations or applications of automata, but are concerned with the natural structure of automata in general. As such, we will consider an automaton \mathcal{M} as a generic automaton that is capable of solving any given solvable problem P for an answer \mathcal{A} .

Since P must be expressed in a manner that is understandable to \mathcal{M} , we will consider an instance of P as an acceptable word \mathbf{t} : where P itself is a set of related words. In this manner, the set \mathbf{i} of all problems P is also the complete set of acceptable words \mathbf{t} for our generic automaton \mathcal{M} : finite and infinite.

We define \mathcal{M} formally according to the quintuple $(Q, \Sigma, \delta, q^0, F)$: where Q is a finite set of states for \mathcal{M} , Σ is \mathcal{M} 's input alphabet; δ is a state transition function for \mathcal{M} ; q^0 is the initial state, or set of initial states, for \mathcal{M} ; and F is the set of final, or accepting, states for \mathcal{M} such that $F \subseteq Q$. For nondeterministic automata \mathcal{M} our transition function will be defined as $\delta: Q \times \Sigma \rightarrow 2^Q$ [34].

Since we have not yet applied a specific problem P to \mathcal{M} we are not concerned about our acceptance condition for \mathcal{M} ; however, we are concerned about the set \mathbf{i} of acceptable words \mathbf{t} . From \mathbf{i} we construct our Σ -tree \mathbf{t} for \mathcal{M} ; where \mathbf{t} is defined according to the pair (ν, \mathbf{i}) such that $\nu: \mathbf{i} \rightarrow \Sigma$. We then say that a run r on \mathbf{t} is performed by \mathcal{M} if r maps \mathbf{i} to Q [25], [28]; where r is a single instance of \mathcal{M} and the set \mathbf{r} of runs r forms a Q -tree \mathbf{q} in like manner as our Σ -tree \mathbf{t} ¹.

¹ We define a Q -tree \mathbf{q} as a pair (Ξ, \mathbf{r}) where \mathbf{r} is our set of runs r , $\Xi: \mathbf{r} \rightarrow Q$, and $r: \mathbf{i} \rightarrow Q$. We define a \mathbf{q} -tree in this manner rather than as was done by Pnueli and Rosner in [28] because we find it more intuitive to our adaptation for a Bayesian game. That is, a run r can now be interpreted directly as a sequence of state-input pairs that are easily reinterpreted as pure strategies for our Bayesian game. Ultimately, defining our Q -tree this way enables us to define the probability distribution for our Bayesian game on top of our Q -tree instead of as an integral part of our Q -tree.

With our Σ -tree and our Q -tree defined and characterized, we can now define our high level game structure. We define our high level game G_B as a Bayesian game over \mathbf{r} , and according to the quadruple $(N, \mathbf{G}, \mathcal{G}, \Pi)$; where N is the set of n players for G_B , \mathbf{G} is a set of component games for G_B , \mathcal{G} is a common prior over all component games that participate in G_B , and Π is a tuple of partitions π_α of G_B for each agent $i \in N$. We define a partitioning

$$\Pi: P \times \mathbf{r} \rightarrow 2^{2^P} \quad (1)$$

of G_B as a Bayes-Nash equilibrium \mathcal{A}^2 for G_B with respect to P , and an ordering of the pure strategies of G_B into partitions π_α that define the component games for our current instance of G_B ; where each component game is defined as a Markov game G_M over the sub-tree defined by the set of pure strategies $r_{\alpha,a} \in \pi_\alpha$; and where our set of Markov games, for an instance of G_B , exist in a one-to-one correspondence with our set of partitions π_α and are viewed as concurrent game structures with respect to a given problem P . Here, we will view each partition π_α as an independent algorithm implementable on \mathcal{M} . For our purposes, N will always be defined as the set $\{time, memory\}$, regardless of our instances of \mathcal{M} . Initially, and before we apply P to \mathcal{M} , \mathcal{G} will be defined according to the total likelihood that our players i will play a pure strategy r_a in response to any given problem instance \mathbf{t}_a ; however, after we apply P to \mathcal{M} , thereby imparting a partitioning $\Pi(P, \mathbf{r})$, but before we apply a specific problem instance P_α to \mathcal{M} we will induce a re-evaluation of \mathcal{G} such that \mathcal{G} will now only provide support to partitions that may be used in response to P . As such, the individual probabilities $c_{i,\alpha,a}$ associated with each individual pure strategy $r_{\alpha,a}$ will now be proportional to the likelihood that a given pure strategy $r_{\alpha,a}$ will be played in response to a given problem instance \mathbf{t}_a with respect to P .

4.2 Nondeterministic Games

There is a wide variety of nondeterministic behavior with respect to automata: some of which originating from within the automaton and some of which originates from outside the automaton. This includes indeterminism with respect to environmental input [26], [32], [39], indeterminism with respect to a guessing module [6], indeterminism with respect to circuit error [4], [7], and really any other source of indeterminacy with respect to our circuit behavior. Our goal here is to construct a formalization for a model of automata that is capable of accurately incorporating all of these forms of nondeterministic behavior. This goal is what has led us to the formulation of nondeterministic games.

By nondeterministic game we mean a Bayesian game that is being utilized to model a nondeterministic automaton,

² The set of all Bayes-Nash equilibria for G_B will be denoted as \mathcal{A}

or algorithm, that can be utilized to solve a nondeterministic problem $P \in \text{NP}^3$. Since we know that G_B is partitioned into Markov subgames G_M , where each subgame is defined over a partition π_α that represents an independent algorithm that can be implemented on \mathcal{M} , let us first consider how we can implement a nondeterministic algorithm as a Markov subgame G_M . The implementation of a nondeterministic game G_M is almost identical to that of deterministic games discussed in [2]. The main difference between our nondeterministic games and our deterministic games is that our nondeterministic games G_M do not have an equality restriction on their mixed strategies, see [2], and therefore their mixed strategies are not required to be reducible to a single pure strategy $r_{\alpha,a}$: or a^τ for a stage game G^τ .

Axiom 1: A nondeterministic stage game $Q_{\text{non-stabilizing}}^\tau$ that contains a nondeterministic state transition defined by the transition function $\delta: Q \times \Sigma \rightarrow 2^Q$ for an algorithm π_α defined over a finite, nondeterministic automaton \mathcal{M} must contain at least one Nash equilibrium S^τ such that there exists a positive probability $c_{i,\alpha,a}^\tau \times c_{j,\alpha,a}^\tau$ that the players $i, j \in N$ will choose disparate actions $a_{i,\alpha,a}^\tau \neq a_{j,\alpha,a}^\tau$, for at least two players i and j ; where $c_{i,\alpha,a}^\tau \times c_{j,\alpha,a}^\tau > 0$ even after the value for t^τ is known.

Proof:

As shown by the proof of Axiom 4 and Corollary 1 in [2], if after the value for t^τ is known, $a_{i,\alpha,a}^\tau = a_{j,\alpha,a}^\tau$ for all actions a^τ and all pairs of players i and j for all stage games G^τ across all runs $r_{\alpha,a}$ then all transitions a^τ are predictable, and therefore determinable based upon the input t^τ . This essentially means that if only a single transition a_a^τ can be performed from q^τ in response to a given input t^τ , and therefore $a_a^\tau = a_{i,\alpha,a}^\tau = a_{j,\alpha,a}^\tau$, then a_a^τ is determinable based upon t^τ . However, if multiple transitions $a_a^\tau, a_b^\tau, \dots$ can be performed in response to the input t^τ then t^τ will determine

³ We should note that we do recognize a difference between a nondeterministic automata and a completely stochastic game. This difference is simply the understanding that the various runs $r_{\alpha,a}$ in an algorithm π_α will have at least a minimal, recognizable relationship between them whereas a completely and purely stochastic game will have no discernable means, throughout the entirety of the game, to determine either which action a_a^τ will be performed or which state $q_a^{\tau+1}$ will be transitioned to at any given time instance τ . This acknowledgement, or assumption, is important because it allows us to continue to sort, or partition, the pure strategies $r_{\alpha,a}$ in G_B into independent algorithms π_α based upon the similarities, or relationships, between our pure strategies π_α . These relationships can either be established across observed structural relationships between our pure strategies π_α , upon a desired structure for the Σ -tree that will represent our desired algorithm π_α , a desired objective \mathcal{A} to a problem P , or any other method for establishing relationships between our pure strategies $r_{\alpha,a}$ for G_B .

the set of transitions a_a^τ that can be performed from q^τ . This set of transitions a_a^τ enables \mathcal{M} to choose a next state during G^τ via some other factor than our input value t^τ , such as random behavior. As such, in order for G^τ to be nondeterministic there must exist a Nash equilibrium point S^τ for G^τ such that $a_i^\tau \neq a_j^\tau$ for the actions of at least two players $i, j \in N$.

Let us assume to the contrary that a) a^τ is not determinable based upon t^τ and that b) \mathcal{M} can behave nondeterministically when $a_i^\tau = a_j^\tau$ for all players i and j and all stage games G^τ .

- If a^τ is not determinable based upon t^τ then there exists a transition a_a^τ , or set a^τ of transitions $(a_a^\tau, \dots, a_b^\tau)$, that is independent of t^τ . Since our transition function δ is defined with respect to both q^τ and t^τ , as $\delta(q^\tau, t^\tau)$, the existence of a_a^τ or a^τ is in contradiction to the definition of \mathcal{M} and therefore Axiom 1 is correct.
- If \mathcal{M} can behave nondeterministically when all players i agree upon a pure strategy a^τ then \mathcal{M} can behave independently of a^τ . This is in contradiction to the definition of \mathcal{M} and the definition of a^τ . As such, Axiom 1 must be correct.

The probability distribution for s_τ is popularly modeled as an error function [26], [32], [39], [42]: which is closely in line with the existence of non-stabilizing circuits [34]. However, to appreciate how this error function behaves we need to know more about its implementation on the Q -tree upon which G_B is defined. Specifically, it is a rather simple analogy to argue that our non-reducible mixed strategy s_τ , with respect to its associated stage game G^τ , is representative of a non-stabilizing circuit in that we do not know how the subsequent stabilizable circuitry $q^{\tau+1}$ will interpret the values produced by q^τ until after $q^{\tau+1}$ has performed its associated operations on said values. This is reflected in our model in that we will not be able to determine which pure strategy $r_{\alpha,a}$ will be instantiated by \mathcal{M} until after we have reached $G^{\tau+1}$ or $G^{\tau+2}$: assuming that our model is capable of perfect recall⁴.

This source of instability within our stage games G^τ obliges us to analyze the word structure for the Q -tree automaton that G_M , via G_B , is defined over. Primarily, unlike deterministic games which can be reduced to simple coordination games when the value for t_a is known, our sources of instability, such as non-stabilizing circuits, prevent us from being able to reduce the tree structure for G_M to a single pure strategy even when the value for t_a is known.

⁴ In truth, even if our model does not have perfect recall, we can still theoretically back-compute $r_{\alpha,a}$ via the methods outlined in [45], [44].

Corollary 1: A nondeterministic algorithm π_α instantiated on an automaton \mathcal{M} can be defined via a Markov game G_M such that a) the pure strategies $r_{\alpha,a}$ in G_M represent the instances of \mathcal{M} over a given input sequence \mathbf{i}_a , and where at least one of the stage games G^τ is a nondeterministic stage game; and b) any given possible instance of π_α on \mathcal{M} can be defined by a pure strategy $r_{\alpha,a} \in \pi_\alpha$.

Proof:

- We know from Axiom 1 and [2] that if all of the stage games in G_M are deterministic then the outcome of any given instance $r_{\alpha,a}$ of G_M will be completely determinable based upon its associated input sequence \mathbf{i}_a . As such, in order for G_M to be nondeterministic there must exist at least one nondeterministic stage game $q_{non-stabilizing}^\tau$ in G_M . To assume the contrary of this would be in contradiction to the definition of a nondeterministic automaton.
- We know that G_M represents all possible ways in which we can instantiate an algorithm π_α on \mathcal{M} over an input vector \mathbf{i} , or set of input sequences \mathbf{i}_a . As such, if \mathcal{M} performs a transition a^τ from our set A of actions in response to an input $i^\tau \in \mathbf{i}_a$ then a^τ must exist in G_M ; for all transitions a^τ performed during the instantiation of π_α in response to the input sequence \mathbf{i}_a . The sequence of transitions a^τ that are performed by \mathcal{M} during an instantiation of π_α define a single path, or pure strategy $r_{\alpha,a}$ through the tree structure of G_B . Since all of the actions a^τ that are performed during our instantiation of π_α must exist in G_M , $r_{\alpha,a}$ must also exist in G_M : and therefore $r_{\alpha,a} \in \pi_\alpha$.
- Let us assume to the contrary that there can exist an instance of π_α on \mathcal{M} that cannot be defined by a pure strategy $r_{\alpha,a} \in \pi_\alpha$. If this were true then there must exist an action a^τ that exist in G_M but that does not exist in π_α . Since G_M is defined by π_α , this would be in contradiction to the definition of G_M .

It should be noted that the nondeterministic algorithm π_α described in Corollary 1 is not a concurrent process and therefore only involves a single algorithm being performed on a single automaton \mathcal{M} . This is similar to the nondeterministic automaton described by Garey and Johnson in [6] that utilized a guessing module: where our guessing module is represented by a stage game G^τ , representative of a state $q_{non-stabilizing}^\tau$, during which at least two players i and j disagree upon which transition a_a^τ should be performed. This model for nondeterministic algorithms is common in modern interpretations of nondeterministic automata that incorporate an error function for our non-stabilizing circuit components $q_{non-stabilizing}^\tau$ [26], [32], [39], [42]. However, this does not

exclude us from consideration of the concurrent models for nondeterministic automata [24], [26], [28], [35], [40], and [43]. In concurrent models, our nondeterministic automaton simultaneously runs multiple concurrent, deterministic algorithms not knowing which algorithm will run to completion first. Similarly, we could state that we concurrently run multiple deterministic automata without knowing how each, individual automaton will contribute to the various parts or components of the final answer. We can directly interpret this traditional model for nondeterministic automata as concurrently providing support to multiple partitions of G_B . From an implementation standpoint, we can consider this to represent either a single physical automaton \mathcal{M} that concurrently implements multiple algorithms, each represented by a separate partition π_α , or a set \mathcal{M} of concurrent automata \mathcal{M} that are each, individually complete with respect to P^5 .

5 Conclusion

In this paper we formalized the structural nature of nondeterministic automata, as modeled by a Bayesian game, with respect to our overall proof that $NP/P \neq \emptyset$ [1]. During this formalization we demonstrated that nondeterministic stage games requires a non-reducible mixed strategy and that nondeterministic Markov games require non-reducible mixed strategies. The vast majority of the details presented here had been demonstrated previous by many very talented researchers [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [37], [38], [39], [40], [41] but not in the context of our high level proof that $NP/P \neq \emptyset$. Between [2], Axiom 1, and Corollary 1 it should begin to become somewhat evident as to why deterministic automata cannot, by themselves, generate stochastic answers. That is, deterministic automata lack the multiplicity of runs, or actions, with respect to at least one input sequence that is inherent in nondeterministic automata.

That said, Axiom 1 and Corollary 1 is only half way through our formalization of nondeterministic automata. That is, we still need to characterize the use case of an implementation involving multiple, concurrent deterministic automata running to solve the same problem. Additionally, we will have to show how these two perspectives on nondeterministic automata are equivalent. Both of these will need to be demonstrated before we can finalize the formalization of the high level proof presented in [1].

6 Future Research

At this point we have formalized deterministic automata [2] but our formalization of nondeterministic automata is still incomplete. As such, we will need to complete our formalization of nondeterministic automata. Once we have fully completed our formalization of nondeterministic

⁵ Though it is not required, we will always assume here that each $\mathcal{M} \in \mathcal{M}$ is a universal automaton that is complete with respect to our \mathcal{Q} -tree q .

automata in the context of [1] then we can finally pull everything together and present the formal concluding arguments for [1].

For more information on the work presented here please see the author's website at www.holcombtechnologies.com/dissertation.aspx.

7 References

- [1] J. Holcomb, "Stochastic Answers and the Question of Whether or Not PSPACE is a Proper Subset of NPSPACE"; Proceedings of Intellectbase International Consortium, Intellectbase International Consortium, Vol. 15, 2011, pp. 142-148.
- [2] J. Holcomb, "Deterministic Automata on Bayesian Games"; unpublished.
- [3] S. Arora, "The Approximability of NP-Hard Problems"; Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 1998, pp. 337-348.
- [4] R. Impagliazzo, "Can every Randomized Algorithm be Derandomized"; Proceedings of the 38th Annual ACM Symposium on Theory of Computing, ACM, 2006, pp. 373-374.
- [5] D. S. Johnson, "The NP-Completeness Column: An Ongoing Guide--The Tale of the Second Prover"; Journal of Algorithms, Vol. 13, Issue 3, pp. 502-524, 1992.
- [6] M. R. Garey, and D. S. Johnson, Computers and Intractability, A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, 1979.
- [7] P. B. Miltersen, Handbook of Randomized Computing, Kluwer, 2001, pp. 843-941.
- [8] B. S. Anand, "Why Brouwer was Justified in his Objection to Hilbert's Unqualified Interpretation of Quantification"; Proceedings of the 2008 International Conference on Foundations of Computer Science, CSREA Press, 2008, pp. 166-169.
- [9] N. Argall, "P=NP - An Impossible Question"; unpublished, 2003.
- [10] G. Bolotashvili, "Solution of the Linear Ordering Problem (NP=P)"; Computing Research Repository (CoRR)informal publication, 2003.
- [11] F. Capasso, "A Polynomial-time Heuristic for Circuit-SAT"; informal publication, 2005.
- [12] M. Diaby, "On the Equality of Complexity Classes P and NP: Linear Programming Formulation of the Quadratic Assignment Problem"; Proceedings of the International MultiConference of Engineers and Computer Scientists (IMECS), Newswood Limited, 2006, pp. 774-779.
- [13] M. Diaby, "A Linear Programming Formulation of the Traveling Salesman Problem"; Proceedings of the 11th WSEAS International Conference on Applied Mathematics, World Scientific and Engineering Academy and Society (WSEAS), Issue 6, 2007, pp. 97-102.
- [14] M. Diaby, "Linear Programming Formulation of the Vertex Colouring Problem"; Int. J. Mathematics in Operational Research, Vol. 2, Issue 3, pp. 259-289, 2010.
- [15] M. Diaby, "Linear Programming Formulation of the Set Partitioning Problem"; Int. J. Operational Research, Vol. 8, Issue 4, pp. 399-427, 2010.
- [16] S. Gram, "Redundancy, Obscurity, Self-Containment & Independence"; 3rd International Conference on Information and Communications Security, Springer, Vol. 2229, 2001, pp. 495-501.
- [17] S. Gubin, "A Polynomial Time Algorithm for the Traveling Salesman Problem"; CoRR, informal publication, 2008.
- [18] A. D. Plotnikov, "Polynomial Time Partition of a Graph into Cliques"; South West Journal of Pure and Applied Mathematics (SWJPAM), Vol. 1, pp. 16-29, November 1996.
- [19] K. Riaz, and M. S. Khiyal, "Finding Hamiltonian Cycle in Polynomial Time"; Information Technology Journal, Vol. 5, pp. 851-859, 2006.
- [20] C. B. Romero, "The Complexity of The NP-Class"; CoRR, informal publication, 2010.
- [21] C. Sauerbier, "A Polynomial Time (Heuristic) SAT Algorithm"; CoRR, informal publication, 2002.
- [22] G. Zhu, "The Complexity of HCP in Digraphs with Degree Bound Two"; CoRR, informal publication, 2007.
- [23] J. Edmonds, "Paths, Trees, and Flowers"; Canadian Journal of Mathematics, Vol. 17, pp. 449-467, 1965.
- [24] A. Pnueli, "The Temporal Logic of Programs"; Proc. 18th Ann. Symp. Foundations of Computer Science (FOCS'77), IEEE, 1977, pp. 46-57.
- [25] M. O. Rabin, "Decidability of Second-Order Theories and Automata on Infinite Trees"; Bulletin of the American Mathematical Society, Vol. 74, Issue 5, pp. 1025-1029, 1968.
- [26] R. Alur, A. T. Henzinger, and O. Kupferman, "Alternating-Time Temporal Logic"; Journal of the ACM, Vol. 49, Issue 5, pp. 672-713, September 2002.

- [27] O. Kupferman, M. Y. Vardi, and P. Wolper, "An Automata-Theoretic Approach to Branching-Time Model Checking"; *Journal of the ACM*, Vol. 47, Issue 2, pp. 312-360, 2000.
- [28] A. Pnueli, and R. Rosner, "On the Synthesis of a Reactive Module"; *Proceedings of the 16th ACM Symposium on Principles of Programming Languages (SIGPLAN-POPL)*, ACM, 1989, pp. 179-190.
- [29] E. A. Emerson, and C. Jutla, "Tree Automata, mu-calculus and Determinacy"; *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, IEEE, 1991, pp. 368-377.
- [30] M. De Wulf, L. Doyen, N. Maquet, and J. Raskin, "Antichains: Alternative Algorithms for LTL Satisfiability and Model-Checking"; *Tools and Algorithms for the Construction and Analysis of Systems*, 14th International Conference, Springer, Vol. 4963, 2008, pp. 63-77.
- [31] M. De Wulf, L. Doyen, T. A. Henzinger, and J. Raskin, "Antichains: A New Algorithm for Checking Universality of Finite Automata"; *Computer Aided Verification*, 18th International Conference, Springer, Vol. 4144, 2006, pp. 17-30.
- [32] F. Horn, W. Thomas, and N. Wallmeier, "Optimal Strategy Synthesis in Request-Response Games"; *Automated Technology for Verification and Analysis*, 6th International Symposium, (ATVA '08), Springer, Vol. 5311, 2008, pp. 361-373.
- [33] K. Chatterjee, T. A. Henzinger, and F. Horn, "Finitary Winning in Omega-Regular Games"; *ACM Trans. Comput. Log.*, Vol. 11, pp. 257-271, 2009.
- [34] S. Safra, "On the Complexity of Omega-Automata"; *29th Annual Symposium on Foundations of Computer Science (FOCS)*, IEEE, 1988, pp. 319-327.
- [35] M. Y. Vardi, "Automatic Verification of Probabilistic Concurrent Finite-State Programs"; *Proc. 26th IEEE Symp. on Foundations of Computer Science*, 1985, pp. 327-338.
- [36] K. Leyton-Brown, and Y. Shoham, *Essentials of Game Theory*, a Concise, Multidisciplinary Introduction, Morgan & Claypool Publishers, 2008.
- [37] E. Filiot, N. Jin, and J. Raskin, "An Antichain Algorithm for LTL Realizability"; *Proceedings of the 21st International Conference Computer Aided Verification (CAV '09)*, Springer-Verlag, Vol. 5643, 2009, pp. 263-277.
- [38] B. Jobstmann, and R. Bloem, "Optimizations for LTL synthesis"; *Formal Methods in Computer-Aided Design*, 6th International Conference (FMCAD), IEEE Computer Society, 2006, pp. 117-124.
- [39] K. Chatterjee, L. Doyen, T. A. Henzinger, and J. Raskin, "Algorithms for Omega-Regular Games with Imperfect Information"; *Lecture Notes in Computer Science*, Vol. 4207, Issue 3:4, pp. 287-302, September 2006.
- [40] T. Agotnes, V. Goranko, and W. Jamroga, "Alternating-Time Temporal Logics with Irrevocable Strategies"; *Proceedings of the 11th Conference on Theoretical Aspects of Rationality and Knowledge (TARK '07)*, ACM, 2007, pp. 15-24.
- [41] K. Chatterjee, T. A. Henzinger, and N. Piterman, "Strategy Logic"; *CONCUR 2007 - Concurrency Theory*, 18th International Conference, Vol. 4703, 2007, pp. 59-73.
- [42] R. Shaltiel, "Typically-Correct Derandomization"; *SIGACT News*, Vol. 41, Issue 2, pp. 57-72, June 2010.
- [43] E. W. Weisstein, *CRC Concise Encyclopedia of Mathematics*, Chapman & Hall, 1999.
- [44] G. H. Mealy, "A Method for Synthesizing Sequential Circuits"; *Bell System Technical Journal*, Vol. 34, Issue 5, pp. 1045-1079, September 1955.
- [45] E. F. Moore, "Gedanken-Experiments on Sequential Machines"; *Journal of Symbolic Logic*, pp. 129-153, 1956.

An Automated Deduction of the Gudder-Schelp-Beran Theorem from Ortholattice Theory

Jack K. Horner
P. O. Box 266
Los Alamos, New Mexico 87544 USA

Abstract

The optimization of quantum computing circuitry and compilers at some level must be expressed in terms of quantum-mechanical behaviors and operations. In much the same way that the structure of conventional propositional (Boolean) logic (BL) is the logic of the description of the behavior of classical physical systems and is isomorphic to a Boolean algebra (BA), so also the algebra, $C(H)$, of closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space is a logic of the descriptions of the behavior of quantum mechanical systems and is a model of an ortholattice (OL). An OL can thus be thought of as a kind of "quantum logic" (QL). $C(H)$ is also a model of an orthomodular lattice, which is an OL conjoined with the orthomodularity axiom (OMA). Because the propositions of a QL are not in general commutative, quantum logicians have paid much attention to "quasi"-commutative theorems, including the so-called "exchange" theorems, one of the best known of which is the Gudder-Schelp-Beran (GSB) theorem. Here I show that, contrary to apparently universal practice, the GSB can be proved without using the orthomodularity assumption, and thus holds even in ortholattices proper. This result appears to be novel.

Keywords: automated deduction, quantum computing, orthomodular lattice, Hilbert space

1.0 Introduction

The optimization of quantum computing circuitry and compilers at some level must

be expressed in terms of the description of quantum-mechanical behaviors ([1], [17], [18], [20]). In much the same way that conventional propositional (Boolean) logic (BL,[12]) is the logical structure of description of the behavior of classical physical systems (e.g. "the measurements of the position and momentum of particle P are commutative") and is isomorphic to a Boolean lattice ([10], [11], [19]), so also the algebra, $C(H)$, of the closed linear subspaces of (equivalently, the system of linear operators on (observables in)) a Hilbert space H ([1], [4], [6], [9], [13]) is a logic of the descriptions of the behavior of quantum mechanical systems (e.g., "the measurements of the position and momentum of particle P are *not* commutative") and is a model ([10]) of an ortholattice (OL; [8]). An OL can thus be thought of as a kind of "quantum logic" (QL; [19]). $C(H)$ is also a model of (i.e., isomorphic to a set of sentences which hold in) an orthomodular lattice (OML; [7], [8]), which is an OL conjoined with the orthomodularity axiom (OMA; see Figure 1). The rationalization of the OMA as a claim proper to physics has proven problematic ([13], Section 5-6), motivating the question of whether the OMA is required in an adequate characterization of QL. Thus formulated, the question suggests that the OMA is specific to an OML, and that as a consequence, banning the OMA from QL yields a "truer" quantum logic.

Lattice axioms

$$\begin{aligned}
x &= c(c(x)) && (\text{AxLat1}) \\
x \vee y &= y \vee x && (\text{AxLat2}) \\
(x \vee y) \vee z &= x \vee (y \vee z) && (\text{AxLat3}) \\
(x \wedge y) \wedge z &= x \wedge (y \wedge z) && (\text{AxLat4}) \\
x \vee (x \wedge y) &= x && (\text{AxLat5}) \\
x \wedge (x \vee y) &= x && (\text{AxLat6})
\end{aligned}$$

Ortholattice axioms

$$\begin{aligned}
c(x) \wedge x &= 0 && (\text{AxOL1}) \\
c(x) \vee x &= 1 && (\text{AxOL2}) \\
x \wedge y &= c(c(x) \vee c(y)) && (\text{AxOL3})
\end{aligned}$$

Orthomodularity axiom (OMA)

$$y \vee (c(y) \wedge (x \vee y)) = x \vee y \quad (\text{AxOM})$$

where

x, y are variables ranging over lattice nodes
 \wedge is lattice meet
 \vee is lattice join
 $c(x)$ is the orthocomplement of x
 $=$ is equivalence ([12])
 1 is the maximum lattice element ($= x \vee c(x)$)
 0 is the minimum lattice element ($= c(1)$)

Figure 1. Lattice, ortholattice, and orthomodularity axioms.

In QL, the non-commutativity of (certain) observables can be captured as the failure of the distributive law ($x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$). (This is a lattice-theoretic way of representing non-commutativity; a physicist would likely say that non-commutativity is fundamental and the failure of distributivity is derivative. The two representations are formally equivalent.) A QL (without AxOM), in fact, can be thought of as a BL in which the distribution law does not hold. Because of the fundamental role that non-commutativity

plays in QL, quantum logicians have paid much attention to "quasi"-commutative theorems. In this family of "almost"-commutative theorems are the so-called "exchange" theorems, which help to ground a large class of equivalence representations in quantum logic, and are thus of potential interest in optimizing quantum circuit design. Among the best known of the exchange theorems is the Gudder-Schelp-Beran (GSB) theorem ([8], Theorem 4.2, p. 263), shown in Figure 2

If $x, y,$ and z are elements of an orthomodular lattice and

$$C(y, z) \quad (\text{Hypothesis 1})$$

and

$$C(x, (y \wedge z)), \quad (\text{Hypothesis 2})$$

then

```

C((x ^ y), z)      (i)
C((x ^ z), y)      (ii)
C((c(x) ^ y), z)   (iii)
C(c(x) ^ z, y)     (iv)
C(c(x) v c(y), z) (v)
C(c(x) v c(z), y) (vi)
C(x v c(y), z)     (vii)
C(x v c(z), y)     (viii)

```

where $C(x,y)$, "x commutes with y", is defined as

```

C(x,y) <-> (x = ((x ^ y) v (x ^ c(y))))
<-> means "if and only if"

```

Figure 2. The GSB theorem

2.0 Method

The OML axiomatizations of Megill, Pavičić, and Horner ([5], [14], [15], [16], [21], [22]) were implemented in a *prover9* ([2]) script ([3]) configured to derive the GSB theorem and executed in that framework on a Dell Inspiron 545 with an Intel Core2 Quad CPU Q8200 (clocked @ 2.33 GHz) and 8.00 GB RAM, running

under the *Windows Vista Home Premium (SP2)/Cygwin* operating environment.

3.0 Results

Figure 3 shows the proof of the GSB theorem produced by [3] on the platform described in Section 2.0.

```

===== PROOF =====

3 C(x ^ y,z) # label("Theorem 4.2(i)") # label(non_clause) #
label(goal). [goal].
23 x ^ y = c(c(x) v c(y)) # label("AxOL3"). [assumption].
40 C(x,y) # label("Hyp1, Beran Thm 4.2"). [assumption].
42 -C(c1 ^ c2,c3) # label("Theorem 4.2(i)") # answer("Theorem 4.2(i)").
[deny(3)].
43 -C(c(c(c1) v c(c2)),c3) # answer("Theorem 4.2(i)").
[copy(42),rewrite([23(3)])].
44 $F # answer("Theorem 4.2(i)"). [resolve(43,a,40,a)].

===== end of proof =====

===== PROOF =====

4 C(x ^ z,y) # label("Theorem 4.2(ii)") # label(non_clause) #
label(goal). [goal].
23 x ^ y = c(c(x) v c(y)) # label("AxOL3"). [assumption].
40 C(x,y) # label("Hyp1, Beran Thm 4.2"). [assumption].
45 -C(c4 ^ c5,c6) # label("Theorem 4.2(ii)") # answer("Theorem
4.2(ii)"). [deny(4)].
46 -C(c(c(c4) v c(c5)),c6) # answer("Theorem 4.2(ii)").
[copy(45),rewrite([23(3)])].

```

```

47 $F # answer("Theorem 4.2(ii)"). [resolve(46,a,40,a)].

===== end of proof =====

===== PROOF =====

5 C(c(x) ^ y,z) # label("Theorem 4.2(iii)") # label(non_clause) #
label(goal). [goal].
13 x = c(c(x)) # label("AxLat1"). [assumption].
14 c(c(x)) = x. [copy(13),flip(a)].
23 x ^ y = c(c(x) v c(y)) # label("AxOL3"). [assumption].
40 C(x,y) # label("Hyp1, Beran Thm 4.2"). [assumption].
48 -C(c(c7) ^ c8,c9) # label("Theorem 4.2(iii)") # answer("Theorem
4.2(iii)"). [deny(5)].
49 -C(c(c7 v c(c8)),c9) # answer("Theorem 4.2(iii)").
[copy(48),rewrite([23(4),14(3)])].
50 $F # answer("Theorem 4.2(iii)"). [resolve(49,a,40,a)].

===== end of proof =====

===== PROOF =====

6 C(c(x) ^ z,y) # label("Theorem 4.2(iv)") # label(non_clause) #
label(goal). [goal].
13 x = c(c(x)) # label("AxLat1"). [assumption].
14 c(c(x)) = x. [copy(13),flip(a)].
23 x ^ y = c(c(x) v c(y)) # label("AxOL3"). [assumption].
40 C(x,y) # label("Hyp1, Beran Thm 4.2"). [assumption].
51 -C(c(c10) ^ c11,c12) # label("Theorem 4.2(iv)") # answer("Theorem
4.2(iv)"). [deny(6)].
52 -C(c(c10 v c(c11)),c12) # answer("Theorem 4.2(iv)").
[copy(51),rewrite([23(4),14(3)])].
53 $F # answer("Theorem 4.2(iv)"). [resolve(52,a,40,a)].

===== end of proof =====

===== PROOF =====

7 C(c(x) v c(y),z) # label("Theorem 4.2(v)") # label(non_clause) #
label(goal). [goal].
40 C(x,y) # label("Hyp1, Beran Thm 4.2"). [assumption].
54 -C(c(c13) v c(c14),c15) # label("Theorem 4.2(v)") # answer("Theorem
4.2(v)"). [deny(7)].
55 $F # answer("Theorem 4.2(v)"). [resolve(54,a,40,a)].

===== end of proof =====

===== PROOF =====

8 C(c(x) v c(z),y) # label("Theorem 4.2(vi)") # label(non_clause) #
label(goal). [goal].
40 C(x,y) # label("Hyp1, Beran Thm 4.2"). [assumption].
56 -C(c(c16) v c(c17),c18) # label("Theorem 4.2(vi)") # answer("Theorem
4.2(vi)"). [deny(8)].
57 $F # answer("Theorem 4.2(vi)"). [resolve(56,a,40,a)].

```

```

===== end of proof =====
===== PROOF =====

9 C(x v c(y),z) # label("Theorem 4.2(vii)") # label(non_clause) #
label(goal). [goal].
40 C(x,y) # label("Hyp1, Beran Thm 4.2"). [assumption].
58 -C(c19 v c(c20),c21) # label("Theorem 4.2(vii)") # answer("Theorem
4.2(vii)"). [deny(9)].
59 $F # answer("Theorem 4.2(vii)"). [resolve(58,a,40,a)].

===== end of proof =====
===== PROOF =====

10 C(x v c(z),y) # label("Theorem 4.2(viii)") # label(non_clause) #
label(goal). [goal].
40 C(x,y) # label("Hyp1, Beran Thm 4.2"). [assumption].
60 -C(c22 v c(c23),c24) # label("Theorem 4.2(viii)") # answer("Theorem
4.2(viii)"). [deny(10)].
61 $F # answer("Theorem 4.2(viii)"). [resolve(60,a,40,a)].

===== end of proof =====

```

Figure 3. Summary of a *prover9* ([2]) proof of the Gudder-Schelp-Beran theorem ([8], p. 263). The proof assumes the inference rules of *prover9*. The general form of a line in this proof is “*line_number conclusion [derivation]*”, where *line_number* is a unique identifier of a line in the proof, and *conclusion* is the result of applying the *prover9* inference rules (such as hyperresolution, copying, and rewriting), noted in square brackets (denoting the *derivation*), to the lines cited in those brackets. Note that some of “logical” proof lines in the above have been transformed to two text lines, with the *derivation* appearing on a text line following a text line containing the first part of that logical line. The detailed syntax and semantics of these notations can be found in [2]. All *prover9* proofs are by default proofs by contradiction.

The total time to produce the proofs in Figure 3 on the platform described in Section 2.0 was ~0.4 seconds.

4.0 Discussion

Several features of the proofs in Section 3.0 are worth noting:

1. The proofs nowhere use the OMA, and thus are actually proofs that GSB holds without the modularity assumption (i.e., GSB holds for all ortholattices, not just orthomodular lattices. It appears that all of the published proofs to date of GSB use the OMA.)

2. None of the proofs use Hypothesis 2 of the GSB as formulated in Figure 2 (Hypothesis 2 can be derived from Hypothesis 1 by substituting x for y , and $y^{\wedge} z$ for z , in Hypothesis 1). Therefore, Hypothesis 2 is redundant.

3. The proofs of consequences (i) - (iv) use only ortholattice axiom 3 (AxOL3). This means that the GSB restricted to consequences (i)-(iv) holds in a subtheory of ortholattice theory.

4. The proofs of consequences (v) - (viii) use only the first hypothesis of GSB. This means that the GSB restricted to consequences (v)-(viii) holds in a subtheory of lattice theory. Note also that proof by contradiction in these cases results in very short proofs that do not need to use the definiens of the "commutes" relation.

5. The proofs in Section 3.0 deploy several inference rules (rewriting, copying, and hyperresolution) that are on the surface more powerful than the combination of condensed detachment and substitution per se, a behavior which puts the dependencies of the GSB consequents on its hypotheses in sharp relief. Each of *prover9*'s inference rules is derivable from the combination of condensed detachment and substitution alone, however, so the more inclusive set of inference rules used here can be invoked without loss of generality.

5.0 Acknowledgements

This work benefited from discussions with Tom Oberdan, Frank Pecchioni, Tony Pawlicki, George Hrabovsky of the Madison Area Science and Technology Institute for Scientific Computing, and the late John K. Prentice, whose passion for foundations of physics inspired those of us privileged to have known him. For any infelicities that remain, I am solely responsible.

6.0 References

- [1] von Neumann J. *Mathematical Foundations of Quantum Mechanics*. 1936. Translated by R. T. Beyer. Princeton. 1983.
- [2] McCune WW. *prover9 and mace4*. URL <http://www.cs.unm.edu/~mccune/prover9/>. 2009.
- [3] Horner JK. GSB prover9 script. 2011. Available from the author on request.
- [4] Dalla Chiara ML and Giuntini R. *Quantum Logics*. URL <http://xxx.lanl.gov/abs/quant-ph/0101028>. 2004.
- [5] Megill ND and Pavičić M. Orthomodular lattices and quantum algebra. *International Journal of Theoretical Physics* 40 (2001), pp. 1387-1410.
- [6] Akhiezer NI and Glazman IM. *Theory of Linear Operators in Hilbert Space. Volume I*. Translated by M. Nestell. Frederick Ungar. 1961.
- [7] Holland, Jr. SS Orthomodularity in infinite dimensions: a theorem of M. Solèr. *Bulletin of the American Mathematical Society* 32 (1995), pp. 205-234.
- [8] Beran L. *Orthomodular Lattices: Algebraic Approach*. D. Reidel. 1985.
- [9] Knuth DE and Bendix PB. Simple word problems in universal algebras. In J. Leech, ed. *Computational Problems in Abstract Algebra*. Pergamon Press. 1970. pp. 263-297.
- [10] Chang CC and Keisler HJ. *Model Theory*. North-Holland. 1990. pp. 38-39.
- [11] Birkhoff G. *Lattice Theory*. Third Edition. American Mathematical Society. 1967.
- [12] Church A. *Introduction to Mathematical Logic. Volume I*. Princeton. 1956.
- [13] Jauch J. *Foundations of Quantum Mechanics*. Addison-Wesley. 1968.
- [14] Megill ND. *Metamath*. URL <http://us.metamath.org/qlegif/mmql.html#unify>. 2004.

[15] Horner JK. An automated deduction system for orthomodular lattice theory. *Proceedings of the 2005 International Conference on Artificial Intelligence*. CSREA Press. 2005. pp. 260-265.

[16] Horner JK. An automated equational logic deduction of join elimination in orthomodular lattice theory. *Proceedings of the 2007 International Conference on Artificial Intelligence*. CSREA Press. 2007. pp. 481-488.

[17] Messiah A. *Quantum Mechanics*. Dover. 1958.

[18] Horner JK. Using automated theorem-provers to aid the design of efficient compilers for quantum computing. *Los Alamos National Laboratory Quantum Institute Workshop*. December 9–10, 2002. URL http://www.lanl.gov/science/centers/quantum/qls_pdfs/horner.pdf.

[19] Birkhoff G and von Neumann J. The logic of quantum mechanics. *Annals of Mathematics* 37 (1936), 823-243.

[20] Nielsen MA and Chuang L. *Quantum Computation and Quantum Information*. Cambridge. 2000.

[21] Pavičić M and Megill N. Quantum and classical implicational algebras with primitive implication. *International Journal of Theoretical Physics* 37 (1998), 2091-2098. <ftp://m3k.grad.hr/pavicic/quantum-logic/1998-int-j-theor-phys-2.ps.gz>.

[22] Horner JK. An automated deduction of the relative strength of orthomodular and weakly orthomodular lattice theory. *Proceedings of the 2009 International Conference on Artificial Intelligence*. CSREA Press. 2009. pp. 525-530.

SESSION

NOVEL ALGORITHMS + DNA BASED COMPUTING + GAME THEORY + TURING MACHINES

Chair(s)

Prof. Hamid R. Arabnia

Molecular solutions for the maximum K-Facility dispersion problem on DNA-based supercomputing

Nozar Safaei 2, Babak Dalvand 2, Saeed Safaei 1, Vahid Safaei

1-Department of Mathematics, Arak University, Arak, Markazi, Iran

2-Department of Mathematics, Islamic Azad University of Khoramabad, Lorestan, Iran

3-Department of Mechanics, Islamic Azad University of Khomini shahr, Esfahan, Iran

Abstract - In recent works for high-performance computing, computation with DNA molecules, i.e. DNA computing, has considerable attention as one of non-silicon-based computing. Watson–Crick complementarity and massive parallelism are two important features of DNA. Using the features, one can solve an NP-complete problem, which usually needs exponential time on a silicon-based computer, in a polynomial number of steps with DNA molecules. In this paper, we consider a procedure for maximum K-Facility dispersion problem in the Adleman–Lipton model. The procedure works in $O(n^2)$ steps for maximum K-Facility dispersion problem of a directed graph with n vertices.

Keywords: maximum K-Facility dispersion problem; Adleman–Lipton model, NP complete;

1 Introduction

In recent works for high-performance computing, computation with DNA molecules, i.e. DNA computing, has considerable attention as one of non-silicon-based computing. Watson–Crick complementarity and massive parallelism are two important features of DNA. Using the features, one can solve an NP-complete problem, which usually needs exponential time on a silicon-based computer, in a polynomial number of steps with DNA molecules. As the first work for DNA computing, Adleman (1994) presented an idea of solving the Hamiltonian path problem of size n in $O(n)$ steps using DNA molecules. Lipton (1995) demonstrated that Adleman's experiment could be used to determine the NP-complete satisfiability (SAT) problem (the first NP-complete problem). Ouyang et al. (1997) presented a molecule biology-based experimental solution to the maximal clique NP-complete problem. In recent years, lots of papers have occurred for designing DNA procedures and algorithms to solve various NP-complete problems. Moreover, procedures for primitive operations, such as logic or

arithmetic operations, have been also proposed so as to apply DNA computing on a wide range of problems (Frisco, 2002; Fujiwara et al., 2004; Guarnieri et al., 1996; Gupta et al., 1997; Hug and Schuler, 2001; and Kamio et al., 2003).

In this paper, the DNA operations proposed by Adleman (1994) and Lipton (1995) are used for figuring out solutions of maximum K-Facility dispersion problem.

Given a complete directed graph $G=(V,E)$ with costs on edge satisfying the triangle inequality and an integer k find a set $F \subseteq V, |F|=k$ so as to minimize

$$\min_{f_1, f_2 \in F} \{d(f_1, f_2)\}$$

The rest of this paper is organized as follows. In Section 2, the Adleman–Lipton model is introduced in detail. Section 3 we present a DNA algorithm for solving the maximum K-Facility dispersion problem and the complexity of the proposed algorithm is described. We give conclusions in Section 4.

2 The Adleman–Lipton model

Bio-molecular computers work at the molecular level. Because biological and mathematical operations have some similarities, DNA, the genetic material that encodes for living organisms, is stable and predictable in its reactions and can be used to encode information for mathematical systems.

A DNA (deoxyribonucleic acid) is a polymer which is strung together from monomers called deoxyribo-nucleotides (Păun et al., 1998). Distinct nucleotides are detected only with their bases. Those bases are, respectively, abbreviated as A (adenine), G (guanine), C (cytosine) and T (thymine). Two strands of DNA can form (under appropriate conditions) a double strand, if the respective bases are the Watson–Crick complements of each other – A matches T and C matches G; also 3' -end matches 5' -end, e.g. the singled strands 5'-

ACCGGATGTCA-3' and 3' -TGGCCTACAGT-5' can form a double strand. We also call the strand 3'-TGGCCTACAGT-5' as the complementary strand of 5'-ACCGGATGTCA-3' and simply denote 3'-TGGCCTACAGT-5' by $\overline{ACCGGATGTCA}$. The length of a single stranded DNA is the number of nucleotides comprising the single strand. Thus, if a single stranded DNA includes 20 nucleotides, it is called a 20 mer. The length of a double stranded DNA (where each nucleotide is base paired) is

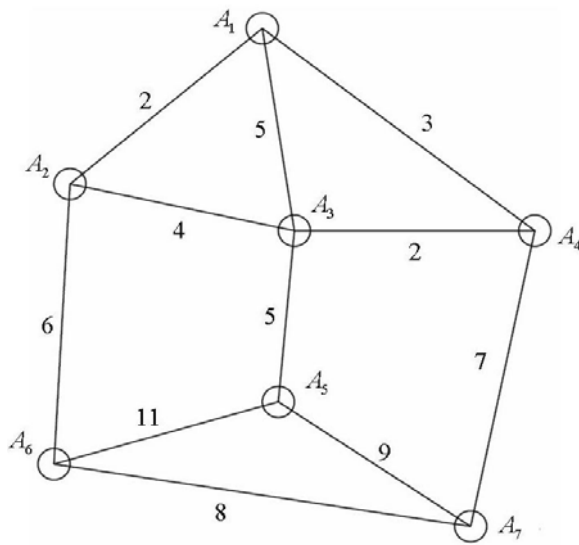


Fig. 1. Graph G.

counted in the number of base pairs. Thus, if we make a double stranded DNA from a single stranded 20 mer, then the length of the double stranded DNA is 20 base pairs, also written as 20 bp.

The Adleman–Lipton model: A (test) tube is a set of molecules of DNA (i.e. a multi-set of finite strings over the alphabet {A, C, G, T}). Given a tube, one can perform the following operations:

- (1) Merge (T_1, T_2): for two given test tubes T_1, T_2 it stores the union $T_1 \cup T_2$ in T_1 and leaves T_2 empty;
- (2) Copy (T_1, T_2): for a given test tube T_1 it produces a test tube T_2 with the same contents as T_1 ;
- (3) Detect (T): Given a test tube T it outputs “yes” if T contains at least one strand, otherwise, outputs “no”;
- (4) Separation (T_1, X, T_2): for a given test tube T_1 and a given set of strings X it removes all single strands containing a string in X from T_1 , and produces a test tube T_2 with the removed strands;

(5) Selection (T_1, L, T_2): for a given test tube T_1 and a given integer L it removes all strands with length L from T_1 , and produces a test tube T_2 with the removed strands;

(6) Cleavage ($T, \sigma_0\sigma_1$): for a given test tube T and a string of two (specified) symbols $\sigma_0\sigma_1$ it cuts each double strand containing $\left[\begin{smallmatrix} \sigma_0\sigma_1 \\ \sigma_0\sigma_1 \end{smallmatrix} \right]$ in T into two double strands as follows:

$$\left[\begin{smallmatrix} \alpha_0\sigma_0\sigma_1\beta_0 \\ \alpha_1\sigma_0\sigma_1\beta_1 \end{smallmatrix} \right] \Rightarrow \left[\begin{smallmatrix} \alpha_0\sigma_0 \\ \alpha_1\sigma_0 \end{smallmatrix} \right], \left[\begin{smallmatrix} \sigma_1\beta_0 \\ \sigma_1\beta_1 \end{smallmatrix} \right]$$

(7) Annealing (T): for a given test tube T it produces all feasible double strands in T . The produced double strands are still stored in T after Annealing;

(8) Denaturation (T): for a given test tube T it dissociates each double strand in T into two single strands;

(9) Discard (T): for a given test tube T it discards the tube T ;

(10) Append (T, Z): for a given test tube T and a given short DNA singled strand Z it appends Z onto the end of every strand in the tube T ;

(11) Read (T): for a given tube T , the operation is used to describe a single molecule, which is contained in the tube T . Even if T contains many different molecules each encoding a different set of bases, the operation can give an explicit description of exactly one of them.

Since these eleven manipulations are implemented with a constant number of biological steps for DNA strands (Păun et al., 1998), we assume that the complexity of each manipulation is $O(1)$ steps.

3 DNA algorithm for the maximum k -Facility dispersion problem

Let $G=(V,E)$ be a directed graph with the set of vertices being $V=\{A_k \mid k=1,2,\dots,n\}$ and the set of edges being $E=\{e_{ij} \mid \text{for some } 1 \leq i, j \leq n\}$. Let $|E|=d$. Then $d \leq n(n-1)$. Note that e_{ij} is in E if the vertices A_i and A_j are connected by an edge.

In the following, the symbols $0,1,\#,X,A_k,B_k (k=1,2,\dots,n)$ denote distinct DNA singled strands with same length, say 10-mer. And $\|\cdot\|$ denotes the length of the DNA singled strand. Obviously the length of the DNA singled strands greatly depends on the size of the problem involved in order

to distinguish all above symbols and to avoid hairpin formation (Li et al., 2003). The DNA singled strand Y_{ij} is used to denote the weights on the edges $e_{ij} \in E$ with $\|Y_{ij}\| = w_{ij}$ if the corresponding weight equals w_{ij} . Suppose that all weights in the given graph are commensurable, i.e., there exists a number y such that each weight is an integral multiple of y (here, take $y = 10$) in the following discussion.

Y_{ij} single strings represents the distance between vertices A_i and vertices A_j . Besides that, the single string B_i1A_i shows that the A_i vertices exist in the set whereas B_i0A_i illustrates that A_i vertices does not exist in the set.

If $i = j$ and there is no edge between A_i and A_j then the length of the strings representing this edge is considered to be 0 mer. Let

$$P = \{0, 1, X, A_1\#, \#B_n, A_k B_{k-1} \mid k = 2, 3, \dots, n\}$$

$$Q = \{\#, \overline{B_k 1 A_k}, \overline{B_k 0 A_k} \mid k = 2, 3, \dots, n\}$$

$$H = \{Y_{ij} \mid 1 \leq i, j \leq n, i \neq j\}$$

We design the following algorithm to solve the maximum K-Facility dispersion problem and give the corresponding DNA operations as follows:

3.1 Produce each possible subset of the set V

For a graph with n vertices, each possible subset of the set V of vertices is represented by an n -digit binary number. A bit set to 1 represents a vertex in the subset, and a bit set to 0 represents a vertex out of the subset. For example, the subset (A_6, A_5) in Fig. 1 is represented by the binary number 0110000. In this way, we transform all possible subsets of V in an n -vertex graph into an ensemble of all n -digit binary numbers. We call this the data pool.

- (1-1) Merge (P, Q);
- (1-2) Annealing (P);
- (1-3) Denaturation (P);
- (1-4) Separation (P, $\{A_1\#$ }, T_{tmp});
- (1-5) Discard (P);
- (1-6) Separation (T_{tmp} , $\{\#B_n\}$, P);

After above six steps of manipulation, singled strands in tube P will encode all 2^n partitions of V in the form of n -digit

ternary numbers. For example, for the graph in Fig. 1 with $n=7$ we have, e.g. the singled strand $\#B_7 0 A_7 B_6 1 A_6 B_5 0 A_5 B_4 0 A_4 B_3 1 A_3 B_2 1 A_2 B_1 0 A_1 \#$

which denotes the subset $\{A_6, A_3, A_2\}$ corresponding to the binary number 0100110. This operation can be finished in $O(1)$ steps since each manipulation above works in $O(1)$ steps.

Suppose that all edges cost of Fig.1 satisfying the triangle inequality.

3.2 Counting vertices of each subset

At first we want to counting vertices of each subset.

If a vertex exists in the set, the following algorithm will add a string X to its corresponding string.

- For $d = 1$ to $d = n$
- 2-1 separation (P, $\{B_d 1 A_d\}$, T_1)
- 2-2 Append (T_1 , X)
- 2-3 Merge (P, T_1)
- 2-4 Discard (T_1)
- End for

Time analysis of the above algorithm

Each of the above actions will conclude at $O(1)$. Therefore the algorithm will terminate at $O(n)$.

3.3 Finding sets of k vertices

Separation of all the strings representing subsets that contain k vertices. Therefore to find sets of k vertices, strings that contain $\underbrace{\{XX \dots XX\}}_{k \text{ times}}$ must be found.

- 3-1 Separation (P, $\underbrace{\{XX \dots XX\}}_{k \text{ times}}$, T_1)
- 3-2 Discard (P)
- 3-3 Separation (T_1 , $\underbrace{\{XX \dots XX\}}_{k+1 \text{ times}}$, T_2)
- 3-4 Merge (P, T_1)

Instruction (3-1) will separate all the strings containing $\underbrace{\{XX \dots XX\}}_{k \text{ times}}$, $\underbrace{\{XX \dots XX\}}_{k+1 \text{ times}}$, $\underbrace{\{XX \dots XX\}}_{k+2 \text{ times}}$, ... from tube P and will place them in tube T_1 .

Instruction (3-3) will separate all the strings containing $\{\underbrace{XX\dots XX}_{k+1 \text{ times}}, \{\underbrace{XX\dots XX}_{k+2 \text{ times}}, \{\underbrace{XX\dots XX}_{k+3 \text{ times}}, \dots$ from tube T_1 and will place them in tube T_2 .

Hence all the strings containing $\{\underbrace{XX\dots XX}_{k \text{ times}}\}$ will remain in T_1 tube.

Time analysis of the above algorithm:

Each of the above actions will conclude at $O(1)$. Therefore the algorithm will terminate at $O(n)$.

3.4 Step 4

Let the subset F correspond the n -digit binary number. $a_n \dots a_i \dots a_j \dots a_1$ For each pair (a_i, a_j) with $a_i = 1, a_j = 0$ or $a_i = 0, a_j = 1$ we append the singled strand $y_{i,j}$ or $y_{j,i}$ to the end of the singled strand which encode the n -digit binary number $a_n \dots a_i \dots a_j \dots a_1$. For example, the singled strands

$B_7 0A_7 B_6 1A_6 B_5 1A_5 B_4 0A_4 B_3 1A_3 B_2 1A_2 B_1 0A_1$ #
(representing the binary number 0110110 for the

graph in Fig. 1) is transformed into

$B_7 0A_7 B_6 1A_6 B_5 1A_5 B_4 0A_4 B_3 1A_3 B_2 1A_2 B_1 0A_1$ # $y_{6,5} y_{6,2}$
 $y_{3,2} y_{5,3}$

where the singled strands $y_{6,3}, y_{5,2}$ do not appear since there are not corresponding edges in the graph shown in Fig. 1

For $k = n$ to $k = 1$

(4 - 1) Separation $(P, \{B_k 1A_k\}, T_1)$

For $i = n$ to $i = 1$

(4 - 2) Separation $(T_1, \{B_i 1A_i\}, T_2)$

(4 - 3) Append $(T_2, y_{k,i})$

(4 - 4) Merge (T_1, T_2)

End For

(4 - 5) Merge (P, T_1)

End For

Time analysis of the above algorithm

Each of the above actions will conclude at $O(1)$. This algorithm consists of 2 for blocks, therefore the algorithm will terminate at $O(n^2)$.

3.5 Step 5

In the last stage, the set with the minimum distance must be recognized.

To do this $Y_{i,j}$ strings must be sorted based on their length.

These strings will be indexed from 1 to n^2 , in a way that the string with the minimum length will be indexed 1 and the string with the shortest length will be indexed n^2 .

For $d = 1$ to n^2

(5 - 1) Separation (P, Y_d, T_1)

(5 - 2) if detect(T) is yes,

then end for else continue the circulation.

Time analysis of the above algorithm

Each of the above actions will conclude at $O(1)$. This algorithm consists of 2 for blocks, therefore the algorithm will terminate at $O(n^2)$.

3.6 Giving the exact solutions

Finally the Read operation is applied to giving the exact solutions to the maximum K-Facility dispersion.

(6 - 1) Read (T).

4 Conclusion

As the first work for DNA computing, (Adleman, 1994) presented an idea to demonstrate that deoxyribonucleic acid (DNA) strands can be applied

to solving the Hamiltonian path NP-complete problem of size n in $O(n)$ steps using DNA molecules. Adleman's work shows that one can solve an NP-complete problem, which usually needs exponential time on a silicon-based computer, in a polynomial number of steps with DNA molecules. From then on, Lipton (1995) demonstrated that Adleman's experiment could be used to determine the NP-complete satisfiability (SAT) problem (the first NP-complete problem). Ouyang et al. (1997) showed that restriction enzymes could be used to solve the NP-complete clique problem. In recent years, lots of papers have occurred for designing DNA procedures and algorithms to solve various NP-complete problems. As Guo et al. (2005) pointed out, it is still important to design DNA procedures and algorithms for

solving various NP-complete problems since it is very difficult to use biological operations for replacing mathematical operations.

In this paper, we propose a procedure for maximum K-Facility dispersion NP-complete problems in the Adleman–Lipton model. The procedure works in $O(n^2)$ steps for maximum K-Facility dispersion problem of a directed graph with n vertices. All our results in this paper are based on a theoretical model. However, the proposed procedures can be implemented practically since every DNA manipulation used in this model has been already realized in lab level.

5 References:

- [1] Adleman, L.M., Molecular computation of solution to combinatorial problems. *Science* 266, 1021–1024, 1994.
- [2] Frisco, P., Parallel arithmetic with splicing. *Romanian J. Inf.Sci. Technol.* 2, 113–128, 2002.
- [3] Fujiwara, A., Matsumoto, K., Chen, Wei. Procedures for logic and arithmetic operations with DNA molecules. *Int. J. Found. Comput. Sci.* 15, 461–474, 2004.
- [4] Guarnieri, F., Fliss, M., Bancroft, C. Making DNA add. *Science* 273, 220–223, 1996..
- [5] Guo, M.Y., Chang, W.L., Ho, M., Lu, J., Cao, J.N. Is optimal solution of every NP-complete or NP-hard problem determined from its characteristic for DNA-Based computing. *BioSystem* 80, 71–82, 2005.
- [6] Gupta, V., Parthasarathy, S., Zaki, M.J. Arithmetic and logic operations with DNA. In: *Proceedings of Third DIMACSWorkshop on DNA-Based Computers*, 212–220, 1997.
- [7] Hug, H., Schuler, R. DNA-based parallel computation of simple arithmetic. In: *Proceedings of the Seventh International Meeting on DNA-based Computers*, 159–166, 2001.
- [8] Kamio, S., Takehara, A., Fujiwara, A. Procedures for computing the maximum with DNA strands. In: *Arabnia, Humid, R., Mun, Youngsong (Eds.), In: Proceedings of the International Conference on DNA-Based Computers*. 2003.
- [9] Li, D., Huang, H., Li, X., Li, X. Hairpin formation in DNA computation presents limits for large NP-complete problems. *BioSystem* 72, 203–207, 2003.
- [10] Lipton, R.J. DNA solution of HARD computational problems. *Science* 268, 542–545, 1995.
- [11] Ouyang, Q., Kaplan, Peter, D., Liu, S., Libchaber, A. DNA solution of the maximal clique problem. *Science* 278, 446–449, 1997.
- [12] Păun, G., Rozeberg, G., Salomaa, A. *DNA Computing*. Springer-Verlag. 1998.

An Inclusion-Exclusion Algorithm for the k -tour Problem

Haseeb Baluch and Andrzej Lingas

Department of Computer Science, Lund University, Sweden

Abstract—Consider an undirected graph G with n vertices, among them a distinguished vertex s called the origin, and nonnegative-integer edge weights in $\{1, \dots, M\}$.

The k -tour problem for G is to cover all vertices of G with cycles such that: each cycle passes through s and includes at most k other vertices, each vertex different from s is visited exactly once by the cycles, and the total weight of the cycles is minimal. This problem is a special case of the general vehicle routing problem and it is known to be NP-hard for $k \geq 3$.

We show that the k -tour problem for G can be solved in time $O(2^n n^7 k^2 M^2 (n \log n + \log M))$ and space $O(n^4 k M (n \log n + \log M))$.

Keywords: k -tour problem, vehicle routing problem, time complexity, space complexity

1. Introduction

The k -tour cover problem (k -TC) is a natural and well known generalization of the traveling salesperson problem (TSP) to include several tours [2], [3], [10], [13]. We are given an undirected graph (V, E) , a distinguished vertex $s \in V$ called the origin as well as a weight function defined on E . The weight of an edge (v, u) is interpreted as the distance between the sites corresponding to v and u , respectively. A k -tour is a cycle in the graph which includes the origin and at most k other vertices. The weight of a k -tour is the sum of weights of the edges included in it. The objective is to find a set of k -tours which visits each vertex in $V \setminus \{s\}$ exactly once and achieves the minimum total weight.

The k -TC problem corresponds to the so called *capacitated vehicle routing problem* well known in Operations Research [13]. The latter name reflects the standard application when the vertices in V model customer locations, and the origin s models a depot. A set of vehicles deployed at the depot has to serve all the customers under the constraint that each vehicle can serve at most k customers. The objective is to minimize the total distance run by the vehicles. The capacitated vehicle routing problem is one of the basic cases of a general vehicle routing problem studied very extensively in the literature (cf. [13]) since many years ago [7].

The k -TC problem for $k = n - 1$ is equivalent to the TSP problem at least in the metric case and hence it is NP-hard. In fact, the k -TC problem is known to be NP-hard for all $k \geq 3$ [2]. Both because of the hardness of k -TC and its applications, the mostly studied variants of k -TC are the metric ones, when the weight function satisfies the triangle

inequality, and in particular the two-dimensional Euclidean one, when the vertices are points in the plane and the weight of an edge is the Euclidean length of the straight-line segment connecting its endpoints.

While the general metric case of k -TC for $k \geq 3$ is known to be APX-complete [2], the approximability status of the two-dimensional Euclidean k -TC problem has not been resolved completely yet. The latter variant is known to admit the so called quasi-PTAS [8] and even PTAS for $k \leq 2^{\log^{o(1)} n}$ [1] (see also [3], [10]) or $k = \Omega(n)$ [3].

In this paper we focus on the exact complexity of the general, non-necessarily metric variant of k -TC. [1] (see also [3], [10]) There is an extensive literature on the exact complexity of the Hamiltonian cycle or path problem. For several decades, the best known upper-time bound was $2^n n^{O(1)}$ [11], where n is the number of vertices of the host graph. This upper time-bound is even achievable when only polynomial space is used [4], [12]. Very recently, Björklund has presented a novel Monte Carlo algorithm for Hamiltonicity detection in an undirected graph, running in time $O(1.657^n)$ [5]. In Operations Research, computational results on exact algorithms for the vehicle routing problem are also known [6].

A straightforward approach to the general k -TC follows from the observation that k -tours covering all vertices can be combined into a single tour visiting each vertex different from the origin exactly once. By enumerating sequences of vertices of length at most $2n - 2$, trying all its partitions into fragments of length at most $k + 1$, we can sieve out all reasonable feasible sets of k -tours covering all the vertices and choose among them the minimum one roughly in time $2^{O(n \log n)}$.

A better straightforward method is to enumerate all possible partitions of the input set of vertices different from s into two subsets whose cardinalities are different by at most k and apply the method recursively to each of the subsets. One returns a union of solutions to two such subsets achieving the minimum total weight. At the bottom of the recursion for instances on at most $3k$ vertices one can apply the aforementioned $2^{O(n \log n)}$ -time algorithm. This divide-and-conquer method takes $2^{2n+O(k \log n)}$ time.

Our contributions

Our algorithm for k -TC is based on the use of the principle of inclusion-exclusion to count the number of feasible solutions. This method was originally applied by Karp in [12] (rediscovered in [4]) in order to count the number of Hamiltonian cycles using the concept of walks

avoiding a subset of the vertex set. We rely on and introduce a generalization of the latter concept to include the so called k -walks avoiding a subset of the vertex set. In consequence, we can solve the k -tour problem in a graph with n vertices and nonnegative-integer edge weights in $\{1, \dots, M\}$ in time $O(2^n n^7 k M^2 (n \log n + \log M))$ and space $O(n^4 k M (n \log n + \log M))$.

In the next section, we introduce the concept of subset avoiding k -walks. In Section 3, we present our algorithm for the k -tour problem.

2. k -walks

Let $G = (V, E)$ be an undirected graph on n vertices with a distinguished origin vertex s , and nonnegative integer edge weights not exceeding M . For $k \in \{1, \dots, n\}$, define a k -walk in G as an alternating sequence of vertices and edges of G $x_0, e_1, x_1, \dots, e_l, x_l$ such that $x_0 = s$, $e_i = (x_{i-1}, x_i)$ for $i = 1, \dots, l$, and any maximal subsequence of consecutive vertices different from s contains at most k vertices. It follows in particular that x_{i-1}, x_i cannot be both equal to s for $i = 1, \dots, l$. The length of k -walk is the total number of its edges while its weight is the total weight of its edges. A k -walk avoids a set of vertices S if $x_0, \dots, x_l \notin S$. For a subset $S \subseteq V \setminus \{s\}$, $m \in \{1, \dots, 2n - 2\}$, $q \in \{\lceil n/k \rceil, \lceil n/k \rceil + 1, \dots, n - 1\}$, $l \in \{1, \dots, k\}$, $v \in V \setminus \{s\}$, $W \in \{0, 1, \dots, (2n - 2)M\}$, we define $k - WALK_v^{m,q,l}(S, W)$ as the set of all k -walks that start in the origin s , end in the vertex v , have length m , visit the origin vertex q times, have a maximal suffix of vertices different from s of length $l \leq k$, avoid all vertices in the set S and have total weight W .

For a given subset $S \subseteq V \setminus \{s\}$, given $m \in \{1, \dots, 2n - 2\}$, given $l \in \{0, 1, \dots, k\}$, given $q \in \{\lceil n/k \rceil, \lceil n/k \rceil + 1, \dots, n\}$, given $v \in V \setminus \{s\}$, given $W \in \{0, 1, \dots, (2n - 2)M\}$, we can compute the cardinalities $|k - WALK_v^{m,q,l}(S, W)|$ from the cardinalities $|k - WALK_{v'}^{m-1,q',l'}(S, W')|$ by the following recurrences, where $v' \notin S$, $q' \in \{q - 1, q\}$ and $W' \leq W$:

$$|k - WALK_v^{m,q,l}(S, W)| = \sum_{(v',v) \in E} |k - WALK_{v'}^{m-1,q',l-1}(S, W - \text{weight}((v',v)))|$$

$$|k - WALK_s^{m,q,0}(S, W)| = \sum_{r=1}^k \sum_{(v',s) \in E \& v' \notin S} |k - WALK_{v'}^{m-1,q-1,r}(S, W - \text{weight}((v',s)))|$$

For convention, at the bottom of the recursion, we set

$$|k - WALK_s^{0,1,0}(S, 0)| = 1$$

and if $q > 1$ or $l \neq 0$ or $W \neq 0$

$$|k - WALK_s^{0,q,l}(S, W)| = 0$$

The recurrences lead to the following algorithm for counting the cardinalities of sets of k -walks with different parameters.

Algorithm 1 Cardinalities of k -Walks

```

1: Input: integer  $k$ , an edge weighted graph  $G = (V, E)$ ,
   where for  $(x, y) \in E$  the weight of  $(x, y)$  is denoted
   by  $w(x, y)$ , a vertex  $s$  designated as the origin, a subset
    $S \subseteq V \setminus \{s\}$  and  $n = |V|$ .
2: Output: for all  $m \in \{1, \dots, 2n\}$ ,  $q \in \{\lceil n/k \rceil, \lceil n/k \rceil + 1, \dots, n\}$ ,  $l \in \{0, 1, \dots, k\}$ ,  $v \in V \setminus S$ ,
   and  $W \in \{0, 1, \dots, (2n - 2)M\}$ , the cardinalities
    $|k - Walk_s^{m,q,l}(S, W)|$ .
3: begin
4:  $|k - Walk_s^{0,0,0}(S, 0)| \leftarrow 0$ 
5: for all  $m \in \{1, \dots, 2n - 2\}$  do
6:   for all  $q \in \{\lceil n/k \rceil, \lceil n/k \rceil + 1, \dots, n\}$  do
7:     for all  $l \in \{0, 1, \dots, k\}$  do
8:       for all  $v \in V \setminus S$  do
9:         for all  $W \in \{0, 1, \dots, (2n - 2)M\}$  do
10:          if  $m = 0$  then
11:            if  $q = 1 \& l = 0 \& W = 0$  then
12:               $|k - Walk_s^{m,q,l}(S, 0)| \leftarrow 1$ 
13:            else
14:               $|k - Walk_s^{m,q,l}(S, W)| \leftarrow 0$ 
15:          else
16:            if  $l = 0 \& v = s$  then
17:               $|k - WALK_v^{m,q,l}(S, W)| \leftarrow$ 
18:                 $\sum_{r=1}^k \sum_{(v',s) \in E \& v' \notin S} |k -$ 
19:                   $WALK_{v'}^{m-1,q-1,r}(S, W -$ 
20:                     $\text{weight}((v',s)))|$ 
21:            else
22:               $|k - WALK_v^{m,q,l}(S, W)| \leftarrow 0$ 
23:          end

```

The cardinality of $k - Walk_s^{m,q,l}(S, W)$ can be an $O(n \lg n)$ bit number. In the computation there are $O(nk)$ additions of $O(n \lg n)$ bit numbers as well as subtractions of $O(\lg n + \lg M)$ bit numbers. Thus, it takes time $O(kn(n \lg n + \lg M))$ and space $O(n \lg n + \lg M)$. Hence, we obtain the following lemma.

Lemma 1. *For a given subset $S \subseteq V \setminus \{s\}$, all $m \in \{0, 1, \dots, 2n - 2\}$, all $l \in \{0, 1, \dots, k\}$, all $q \in \{\lceil n/k \rceil, \lceil n/k \rceil + 1, \dots, n\}$, all $v \in V \setminus \{s\}$, and all $W \in \{0, 1, \dots, (2n - 2)M\}$, one can compute the cardinalities $|k - WALK_v^{m,q,l}(S, W)|$ in time $O(n^5 k^2 M (n \lg n + \lg M))$ and space $O(n^4 k M (n \lg n + \lg M))$.*

By a closed k -walk, we shall mean a k -walk that starts and ends at the origin s . For short, we shall denote $k - WALK_s^{m,q,0}(S, W)$ by $k - CW^{m,q}(S, W)$.

Corollary 2. *For a given subset $S \subseteq V \setminus \{s\}$, all $q \in \{\lceil \frac{n}{k} \rceil, \lceil \frac{n}{k} \rceil + 1, \dots, n\}$, all $m \in \{0, 1, \dots, 2n - 2\}$ and all $W \in \{0, 1, \dots, (2n - 2)M\}$, we can compute the cardinalities $|k - CW^{m,q}(S, W)|$ in time $O(n^5 k^2 M(n \lg n + \lg M))$ and space $O(n^4 k M(n \lg n + \lg M))$.*

3. An exact algorithm for the k -tour problem

Our algorithm for the k -tour problem relies on the following lemma following from the inclusion-exclusion principle.

Lemma 3. *For $m \in \{0, 1, \dots, 2n - 2\}$, $q \in \{\lceil \frac{n}{k} \rceil, \lceil \frac{n}{k} \rceil + 1, \dots, n\}$ and $W \in \{0, 1, \dots, (2n - 2)M\}$, the number of closed k -walks that cover all the vertices in graph $G = (V, E)$, visit the origin q times, achieve the total length m and the total weight W is*

$$\sum_{S \subseteq V \setminus \{s\}} (-1)^{|S|} |k - CW^{m,q}(S, W)|.$$

Proof: To obtain the number of k -walks that correspond to the desired closed k -walks, we need to subtract from $|CW^{m,q}(\emptyset, W)|$ the number of closed k -walks of length m and weight W , visiting the origin q times, that avoid at least one vertex, that is, belong to $\bigcup_{v \in V \setminus \{s\}} CW^{m,q}(\{v\}, W)$. By the inclusion-exclusion principle

$$|k - CW^{m,q}(\emptyset, W)| - \left| \bigcup_{v \in V \setminus \{s\}} k - CW^{m,q}(\{v\}, W) \right| = \sum_{S \subseteq V \setminus \{s\}} (-1)^{|S|} |k - CW^{m,q}(S, W)|.$$

By combing Corollary 2 and Lemma 3, we can compute the number of closed k -walks covering all the n vertices in the input graph, achieving a given total length m and a given total weight W , and visiting the origin q times, in time $O(2^n n^5 k^2 M(n \log n + \log M))$ and space $O(n^4 k M(n \log n + \log M))$.

By performing the counts for each $m \in \{n, \dots, (2n - 2)\}$, $W \in \{n, (2n - 2)M\}$, and $q = m - (n - 1) + 1$, we can determine the minimum weight of a solution to the k -tour problem for the input graph G in time $O(2^n n^7 k M^2(n \log n + \log M))$ and space $O(n^4 k M(n \log n + \log M))$. By a standard backtracking, we can also determine a solution to the k -tour problem for G achieving the minimum weight within the same asymptotic time. Hence, we obtain our main result.

Theorem 4. *The k -tour problem for an undirected graph with n vertices and nonnegative integer weights in $\{1, \dots, M\}$ can be solved in time $O(2^n n^7 k^2 M^2(n \log n + \log M))$ and space $O(n^4 k M(n \log n + \log M))$.*

4. Final remarks

We assume a strict definition of k -TC that requires a set of k -tours to visit each vertex different from the origin exactly once. In the literature [2], [3], [10], [13] which is concerned

solely with the Euclidean and metric cases it is sufficient to require a set of k -tours to cover all the vertices. Because of the triangle inequality and the possibility of shortcutting, one can always trivially transform a set of k -tours to a not heavier one which satisfies the strict definition. Thus, the strict definition and the relaxed one which allows for visiting a vertex several times are essentially equivalent in the metric case. This is not the case in the general graph case. For instance, it is well known that generally TSP does not admit any reasonable approximation while the relaxed TSP can be trivially approximated within 2 by doubling the edges of a minimum spanning tree of the input graph.

Our exact method of solving k -TC can be easily adapted to the relaxed variant of k -TC, where each vertex different from the origin may be visited several times. The adaptation involves increasing the upper bound on the total length of k tours up to $(k + 1)(n - 1)$ which in turn adds an additional polynomial factor in k to the resulting time and space complexities.

Our method subsumes the aforementioned straightforward permutation and divide-and-conquer methods if the maximum edge weigh M satisfies $M \ll \sqrt{n!}$ and $M \ll 2^{n+O(k \lg n)}$, respectively.

References

- [1] A. Adamaszek, A. Czumaj and A. Lingas. PTAS for k -tour cover problem on the plane for moderately large values of k . Proc. ISAAC 2009, Lecture Notes in Computer Science, Springer Verlag, pp. 994-1003.
- [2] T. Asano, N. Katoh, H. Tamaki, and T. Tokuyama. Covering points in the plane by k -tours: a polynomial time approximation scheme for fixed k . IBM Tokyo Research Laboratory Research Report RT0162, 1996.
- [3] T. Asano, N. Katoh, H. Tamaki, and T. Tokuyama. Covering points in the plane by k -tours: Towards a polynomial time approximation scheme for general k . In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 275-283, 1997.
- [4] E.T. Bax. Inclusion and exclusion algorithm for the Hamiltonian path problem. *Information Processing Letters* 47(4), pp. 203-207, 1993.
- [5] A. Björklund. Determinant Sums for Undirected Hamiltonicity. In *Proceedings of the th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages , 2010.
- [6] N. Christofides, A. Mingozzi and P. Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming* 20 (1981) pp. 255-282, North-Holland Publishing Company.
- [7] G. B. Dantzig and R. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80-91, October 1959.
- [8] A. Das and C. Mathieu. A quasi-polynomial time approximation scheme for Euclidean capacitated vehicle routing. Proc. SODA 2010, pp. 390-403.
- [9] M.R. Garey and D.S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-completeness*. W.H. Freeman and Company, New York 1979.
- [10] M. Haimovich and A.H.G. Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of Operation Research*, 10(4):527-542, 1985.
- [11] M. Held and R. Karp. A dynamic programming approach to sequencing problems. *Journal of SIAM* 10, pp. 196-210.
- [12] R.M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters* 1(2), pp. 49-51, 1982.
- [13] P. Toth and D. Vigo. *The Vehicle Routing Problem*. SIAM, Philadelphia, 2001.

Sub-Exponential Algorithms for 0/1 Knapsack and Bin Packing

Thomas E. O'Neil

Computer Science Department
University of North Dakota
Grand Forks, ND, USA 58202-9015

Abstract - This paper presents simple algorithms for 0/1 Knapsack and Bin Packing with a fixed number of bins that achieve time complexity $p(n) \cdot 2^{O(\sqrt{x})}$ where x is the total bit length of a list of sizes for n objects. The algorithms are adaptations of a method that achieves a similar complexity for the Partition and Subset Sum problems. The method is shown to be general enough to be applied to other optimization or decision problem based on a list of numeric sizes or weights. This establishes that 0/1 Knapsack and Bin Packing have sub-exponential time complexity using input length as the complexity parameter. It also supports the expectation that all NP-complete problems with pseudo-polynomial time algorithms can be solved deterministically in sub-exponential time.

Keywords: 0/1 Knapsack, dynamic programming, Bin Packing, sub-exponential time, NP-complete problems.

1 Introduction

The comparative complexity of problems within the class NP-Complete has been a recurring theme in computer science research since the problems were defined and cataloged in the early years of the discipline [2]. In 1990, Stearns and Hunt [7] classified a problem to have power index i if the fastest algorithm that solves it requires $2^{O(n^i)}$ steps. Assuming that Satisfiability has power index 1, they argued that the Clique and Partition problems have power index one-half. Their analysis is based on two algorithms with time complexity $p(n) \cdot 2^{O(\sqrt{x})}$, where x is the length in bits of the input representations and $p(n)$ is a polynomial function of the number of graph edges (for Clique) or the number of integers in the input set (for Partition). These results were interpreted to provide strong evidence that Clique and Partition were easier problems than Satisfiability and most other NP-Complete problems.

In a subsequent study, Impagliazzo, Paturi, and Zane [3] presented another framework for comparison of NP-complete problems. Instead of adopting the power index terminology of Stearns and Hunt, they categorized problems based on weakly exponential ($2^{n^{o(1)}}$) or strongly

exponential ($2^{\Omega(n)}$) lower bounds (assuming that Satisfiability will one day be proven to be strongly exponential) and sub-exponential ($2^{o(n)}$) upper bounds. To avoid inconsistencies related to the characterization of input length, they defined a family of reductions (the Sub-Exponential Reduction Family) that would allow the complexity measure to be parameterized. This framework tolerated polynomial differences in the lengths of problem instances, and there was no complexity distinction among Clique, Independent Set, Vertex Cover or k -Sat. These conclusions are not consistent with those of Stearns and Hunt, where both Clique and Partition were easier than Satisfiability. It is clear that representations and complexity measures for problem instances play a critical role in complexity analysis.

In classical complexity theory, the complexity measure is the length of the input string. This parameter is formally determined, simply by counting the bits in the string. The advantage of using the formal measure is that it requires no semantic interpretation of the input string, and problems with vastly different semantics can be grouped together in formal complexity classes. Within the class NP-complete, we find that for many problems, the use of simple semantic complexity measures will not clash with detailed analysis based on the formal measure. This is generally true of strong NP-complete problems, where the objects in the input representing variables or nodes or set elements can be numbered (in binary). The numbers are just labels used for identification of the objects. There are other problems in the class, however, where the input contains a list of weights or values, and analysis based on semantic measures such as the number of objects versus the sum (or maximum) of the values can give radically different results: exponential time with one measure, polynomial time with the other. This collection of problems includes Partition, Subset Sum, 0/1 Knapsack, and Bin Packing, which we will refer to as the Subset Sum family. The safest approach to analysis of these problems is to use the formal complexity measure, which incorporates both relevant semantic parameters, and in this paper we show that the Subset Sum family of pseudo-polynomial-time problems is $2^{O(\sqrt{x})}$ (which is sub-exponential).

Stearns and Hunt [7] were apparently the first to demonstrate that an algorithm for the Partition problem exhibits sub-exponential time. The significance of this result was probably obscured by the claim in the same paper that the Clique problem is also sub-exponential, while its dual problem Independent Set remains strongly exponential. This apparent anomaly is a representation-dependent distinction, and it disappears when a symmetric representation for the problem instance is used [5]. The complexity distinction between Partition and Satisfiability, however, appears to have stronger credibility. In [6] it is shown that the sub-exponential upper bound for Partition is also valid for Subset Sum. The algorithm for Subset Sum is a variant of dynamic programming that is much simpler and more general than the backtracking/dynamic programming hybrid that Stearns and Hunt designed for Partition. In this paper, the sub-exponential Subset Sum algorithm is adapted to 0/1 Knapsack and Bin Packing with a fixed number of bins, establishing that these problems are also sub-exponential with respect to the formal complexity measure (total bit-length of input, denoted x). We also abstract from the previous methods a lemma that identifies the property of ordered sets of integers that is exploited to achieve sub-exponential time.

More recent complexity studies in the research literature for problems in the Subset Sum family do not typically use the input length as the complexity parameter. The current upper bound for both Subset Sum and 0/1 Knapsack is apparently $2^{O(n^2)}$ when the number of objects in the list is used as the complexity measure [8]. A lower bound of $\Omega(2^{n^2/\sqrt{n}})$ for Knapsack has also been demonstrated in [1]. The lower bound applies only to algorithms within a model defined generally enough to include most backtracking and dynamic programming approaches. The sub-exponential bounds derived here using the formal complexity measure complement rather than supersede the strongly exponential bounds derived using the number of objects in the input list (denoted n) as the complexity parameter.

2 Generalized Dynamic Programming

The Stearns and Hunt algorithm for Partition [7] combines backtracking with dynamic programming. Such hybrid approaches had been previously described in operations research literature (e.g. [4]). The input set is ordered and divided into a denser and a sparser subset. Backtracking is employed on the sparse subset, while dynamic programming is used for the dense subset. The results are combined to achieve time complexity $2^{O(\sqrt{x})}$, where x is the total length in bits of the input.

In this paper we employ a simpler algorithm that achieves the same goal. The approach was first developed for Subset Sum and Partition [6]. Similar to conventional

dynamic programming, it represents a breadth-first enumeration of partial solutions. The problem instance is a list of objects, each of which has a size. The algorithm maintains a pool of partial solutions as it processes each object. The list of objects is ordered by size, and the largest objects are processed first. In contrast with conventional dynamic programming, the pool of solutions is dynamically allocated (hence the acronym DDP, for dynamic dynamic programming). It first grows and then shrinks as more objects are processed. The entire pool of solutions is traversed for each object, updating each solution by possibly subtracting the current object's size from the solution's remaining capacity. Each solution is also evaluated relative to the sum of sizes of the objects yet to be processed. The sum of remaining sizes can be used to prune the pool of solutions depending on problem semantics. This pruning relative to the sum of sizes of the unprocessed objects places a sub-exponential upper bound on the number of partial solutions in the pool.

The time analysis of the DDP method relies on a simple lemma (abstracted from the analysis in [6]) that allows us to bound the k^{th} value in an ordered list as a function of its position in the list and the total bit-length of the entire list (see Lemma 1 below). Bounding the k^{th} value allows us to bound the sum of the first k values as well. This, in turn, leads to a bound on the length of the pool of partial solutions in DDP algorithms.

Lemma 1: Let L represent a list of n positive natural numbers in non-decreasing order, let $L[k]$ represent the k^{th} number in the list, let b_i be the bit length of the i^{th} number, and let b be total number of bits in the entire list:

$$b = \sum_{i=1}^n b_i = \sum_{i=1}^n 1 + \lceil \lg L[i] \rceil. \quad \text{Then } L[k] < 2^{(b-k+1)/(n-k+1)}.$$

Proof: An upper bound on the value of $L[k]$ for any list with total bit length b is obtained by reserving as few bits as possible for the smaller numbers in the list and as many bits as possible for $L[k]$ and the numbers that follow it. This is accomplished by setting $L[1]$ through $L[k-1]$ to 1 and distributing the remaining bits equally among the higher $n-k+1$ numbers. In that case, $L[k]$ has no more than $(b-k+1)/(n-k+1)$ bits, establishing $L[k] < 2^{(b-k+1)/(n-k+1)}$. \square

3 The Knapsack Problem

The 0/1 Knapsack problem is defined as follows: given a set of n objects S with sizes $s[1..n]$ and values $v[1..n]$, find a subset of objects with the highest value whose size is less than or equal to C , the capacity of the knapsack [2]. The problem can also be expressed as a decision problem, where we determine the existence of a subset whose value is greater than or equal to a target value V .

```

/* Given a set of  $n$  objects whose sizes are specified
in an array  $s[1..n]$  in non-decreasing order and whose
values are stored in an array  $v[1..n]$ , find the highest
valued subset whose total size is less than or equal to
capacity  $C$ . */
public int Knapsack()
1)  $bestval \leftarrow 0$ ;
2)  $sizeofrest \leftarrow \sum_{i=1}^n s[i]$  ;  $valueofrest \leftarrow \sum_{i=1}^n v[i]$  ;
3)  $Pool \leftarrow \{(C, 0)\}$ ;
4) for  $i \leftarrow 1$  to  $n$ 
5)    $size \leftarrow s[n - i + 1]$ ;  $value \leftarrow v[n - i + 1]$ ;
6)    $NewList \leftarrow \{\}$ ;
7)   for each sack in  $Pool$ 
8)     if ( $sack.capacity < size$ )
9)       continue;
10)    else if ( $sack.capacity > sizeofrest$ )
11)       $bestval \leftarrow \max(bestval,$ 
            $sack.value + valueofrest)$ 
12)      remove sack from  $Pool$ ;
13)    else
14)       $bestval \leftarrow \max(bestval,$ 
            $sack.value + value)$ ;
15)       $NewList.append((sack.capacity - size,$ 
            $sack.value + value))$ ;
    end for
16)    $sizeofrest \leftarrow sizeofrest - size$ ;
     $valueofrest \leftarrow valueofrest - value$ ;
17)    $Pool \leftarrow merge(Pool, NewList)$ ;
    end for
18) return  $bestval$ ;

```

Figure 1. The *Knapsack* algorithm.

3.1 The *Knapsack* algorithm

In adapting the DDP method to the *Knapsack* problem, we can iterate either the size or the value array as the control for the outer loop. Here we use the size array. The algorithm keeps a pool of (*capacity*, *value*) pairs representing partially filled knapsacks, initially containing an empty sack represented as (C , 0), where C is the capacity of the empty sack. For each object in S and for each sack currently in the pool, we add a new sack representing the current sack plus the current object. This is accomplished by subtracting the object size from the sack's remaining capacity and adding the object value to the sack's value.

Pseudo-code for the *Knapsack* algorithm is shown in Figure 1. Lines 1-3 initialize the global *Pool*, the *bestval* variable, and variables representing the cumulative size and value of the remaining objects. There is one iteration of the outer *for* loop (lines 4-17) for each object in the set $S = \{y_1, y_2, \dots, y_n\}$. The size array s , in which $s[i]$ is the size of

object y_i , is assumed to be in non-decreasing order, and the largest numbers are processed first, so object y_{n-i+1} is processed during the i^{th} iteration. The pool of partially filled sacks is updated by the inner *for* loop (lines 7-15). For each sack in the pool, $s[n-i+1]$ is subtracted from its capacity and $v[n-i+1]$ is added to its value, placing the new (*capacity*, *value*) on a second ordered sack list. The pool and the new sack list are merged in the last step of the outer loop (line 17). The best value for a filled sack is updated when appropriate in lines 11 and 14, whenever an updated sack is created. At completion of the outer loop, the best value is returned. The algorithm does not return the contents of the sack with the best value, but this could be accomplished by adding a reference to a subset object to the (*capacity*, *value*) pairs in the pool, increasing the time complexity by no more than a factor of n .

The inner loop has two conditions that moderate the length of the pool. Lines 8 and 9 skip sacks that can't hold the current object. Also, in lines 10-12, sacks with enough capacity to hold all remaining objects are removed from the pool after updating the *bestval* variable. If all remaining objects will fit in a sack, there is no process them one-by-one.

The outer loop also has logic to control the size of the pool. The last step in the outer loop is a sequential merge operation that adds the new partially filled sacks to the pool. If two sacks with the same capacity are encountered during the merge, only the sack with the higher value is added to the pool. Thus the capacities of all sacks in the pool are unique.

3.2 Time Analysis of *Knapsack*

The time analysis closely follows the method used for the Subset Sum algorithm in [6]. Let $S = \{y_1, y_2, \dots, y_n\}$ and assume the sizes are stored in non-decreasing order ($s[i] \leq s[i+1]$). The total number of steps is determined by the size of *Pool*. With each iteration of the outer *for* loop, *Pool* is traversed and possibly extended (requiring 2 passes – one by the inner *for* loop and the other by the sequential merge step). The total amount of work is closely estimated (within a factor of 2) by

$$\sum_{i=1}^n |Pool(i)| \quad (1)$$

where $|Pool(i)|$ is the length of *Pool* at the beginning of outer loop iteration i .

Since the merge operation eliminates duplication of capacities, we can describe length of $Pool(i)$ as at most $MaxC(i)$, the largest capacity of any sack on the list at the beginning of iteration i . The list is actually smaller than this, since all the capacities between zero and the maximum are not present. We also know that the length of the list can, at most, double with each loop iteration, so regardless

of the maximum value in the list, its length cannot exceed 2^i . This gives us

$$|Pool(i)| \leq \min(2^i, MaxC(i)). \quad (2)$$

The length of *Pool* will grow rapidly and later possibly shrink as i approaches n . Our goal is to find an upper bound for $MaxC(i)$. Initially $MaxC(1) = C$, which is the capacity of the empty sack. Only smaller-capacity sacks are added to the list, and eventually the larger-capacity sacks are removed when the condition in line 10 becomes true, so

$$MaxC(i) \leq \sum_{j=1}^{n-i+1} s[j] \leq (n-i+1) \cdot s[n-i+1]. \quad (3)$$

Bounding $MaxC(i)$ thus reduces to finding an upper bound for $s[n-i+1]$, and Lemma 1 is invoked for this purpose. To complete the analysis, we bound the step counts as a function of b , the total bit length of the size array s . We consider two cases.

Case 1. $n \leq \sqrt{b}$. Here we have

$$\sum_{i=1}^n |Pool(i)| \leq \sum_{i=1}^n \min(2^i, MaxC(i)) \leq n \cdot 2^{\sqrt{b}}. \quad (4)$$

Case 2. $n > \sqrt{b}$. In this case we split the summation at $i = \sqrt{b}$.

$$\sum_{i=1}^n |Pool(i)| \leq \sum_{i=1}^n \min(2^i, MaxC(i)) \quad (5)$$

$$\leq \sum_{i=1}^{\sqrt{b}-1} \min(2^i, MaxC(i)) + \sum_{i=\sqrt{b}}^n \min(2^i, MaxC(i)) \quad (6)$$

$$\leq (\sqrt{b}-1) \cdot 2^{\sqrt{b}-1} + \sum_{i=\sqrt{b}}^n \min(2^i, MaxC(i)) \quad (7)$$

$$\leq (\sqrt{b}-1) \cdot 2^{\sqrt{b}-1} + (n-\sqrt{b}+1) \cdot MaxC(\sqrt{b}) \quad (8)$$

$$\leq (\sqrt{b}-1) \cdot 2^{\sqrt{b}-1} + (n-\sqrt{b}+1) \cdot \sum_{j=1}^{n-\sqrt{b}+1} s[j]. \quad (9)$$

$$\leq (\sqrt{b}-1) \cdot 2^{\sqrt{b}-1} + (n-\sqrt{b}+1)^2 \cdot s[n-\sqrt{b}+1] \quad (10)$$

At this point, we employ Lemma 1 to compute the bound for $s[k]$ where $k = n - \sqrt{b} + 1$, and we continue by replacing $s[n - \sqrt{b} + 1]$ with $2^{\sqrt{b}+1}$:

$$< (\sqrt{b}-1) \cdot 2^{\sqrt{b}-1} + (n-\sqrt{b}+1)^2 \cdot 2^{\sqrt{b}+1} \quad (11)$$

$$< (\sqrt{b} + (n+1)^2) \cdot 2^{\sqrt{b}+1} \quad (12)$$

$$< (2n^2 + 6n + 2) \cdot 2^{\sqrt{b}}. \quad (13)$$

This establishes that the time complexity of *Knapsack* is $O(p(n)2^{\sqrt{b}})$ for a polynomial function $p(n)$. The argument b is the total bit length of the list of sizes. The entire input for the problem also includes the capacity C and a list of n values. We can't make any specific assumptions about the relative magnitudes of the sizes and values, but we are certain that if x is the total input length, then b will be smaller than x , and the $O(p(n)2^{\sqrt{b}})$ step count will also be $O(p(n)2^{\sqrt{x}})$.

4 The Bin Packing Problem

The Bin Packing problem is defined as follows: given a set of n objects S with sizes $s[1..n]$, determine whether the objects will fit into a fixed number of k bins, each with a capacity of B . The problem can also be expressed as an optimization problem in which the smallest B is determined [2]. When B is equal to the sum of all sizes divided by k , the problem represents a generalization of the Partition problem.

```
/* Given a set of  $n$  objects whose sizes are specified
in an array  $s[1..n]$  in non-decreasing order, determine
whether all objects can be stored in  $k$  bins, each with
capacity  $B$ .
*/
```

```
public boolean BinPack()
1) sizeofrest =  $\sum_{i=1}^n s[i]$  ;
2) Pool = {(B, B, ..., B)};
3) for  $i \leftarrow 1$  to  $n$ 
4)   nextsize  $\leftarrow s[n-i+1]$ ;
5)   NewList  $\leftarrow \{\}$ ;
6)   for each bintuple in Pool
7)     if (bintuple.capacity[1] < nextsize)
8)       continue;
9)     else if (bintuple.capacity[1] > sizeofrest)
10)      return true;
11)    else
12)      for  $j \leftarrow 1$  to  $k$ 
13)        newtuple  $\leftarrow$  update(bintuple,  $j$ , nextsize);
14)        if (newtuple != null)
15)          NewList.insert(newtuple);
16)        end for
17)      Pool  $\leftarrow$  NewList;
18)      sizeofrest  $\leftarrow$  sizeofrest - nextsize;
19)    end for
20)  return false;
```

Figure 2. The *BinPack* algorithm.

4.1 The *BinPack* Algorithm

When we adapt the DDP strategy to Bin Packing, we find a few significant differences from the *Knapsack* version. The *BinPack* algorithm is shown in Figure 2. The pool of partial solutions must be a list of k -tuples, where each component of a tuple is the remaining capacity of one of the bins (see line 2). Also, we are not searching for a subset. All the objects in the original set S must be included in the solution. This has implications for the logic in the nested loops of the algorithm. Any partial solution in the inner loop that cannot accommodate the next object can be

discarded (lines 7-8), and the pool of updated partial solutions created by the inner loop replaces the pool from the previous iteration of the outer loop (rather than merging with the previous pool; see line 16). We also find that the test enforcing the upper limit on the size of the pool (relative to the sum of the remaining object sizes) triggers early termination (lines 9-10). This version of the algorithm does not specify what objects are placed in what bins, but this information could be included by associating a reference to a size n object to each partial solution. This would increase the time complexity by no more than a factor of n .

4.2 Time Analysis of *BinPack*

The time analysis of *BinPack* follows the same general logic as the analysis for *Knapsack*. The major difference is the growth rate of the pool of partial solutions. While the pool can double in length with each iteration of the inner loop in *Knapsack*, it can increase in length by a factor of k in *BinPack*. Another significant difference is the cost of suppressing duplicates in the pool of partial solutions. We make the conservative assumption that the insertion of an updated partial solution in the pool takes linear time in the current length of the pool. We demonstrate below that in spite of these significant differences, the time complexity of the algorithm remains sub-exponential.

To proceed with the analysis, let $S = \{y_1, y_2, \dots, y_n\}$, and assume the sizes are stored in non-decreasing order ($s[i] \leq s[i+1]$). As with *Knapsack*, The total number of steps is closely related to the size of *Pool*. With each iteration of the outer *for* loop, *Pool* is traversed and replaced with an updated version (called *NewList*). Each insertion into *NewList* requires linear time. The total amount of work is therefore estimated as

$$\left(\sum_{i=1}^n |Pool(i)|^2\right) \quad (14)$$

where $|Pool(i)|$ is the length of *Pool* at the beginning of outer loop iteration i .

Since the insert operation of line 15 eliminates duplication of capacities, we can describe length of $Pool(i)$ as at most $MaxC(i)^k$. If $MaxC(i)$ is the largest capacity of any bin in any tuple on the list at the beginning of iteration i , the number of distinct tuples cannot exceed this quantity raised to the power k . This grossly overestimates the number of tuples, since the capacities within each tuple are in non-increasing order and since all the tuples have the same sum. It is an interesting counting problem to determine a tight upper bound for the number of tuples, but the loose bound is sufficient to establish the desired complexity result. We also know that the length of the list can, at most, grow by a factor of k with each loop iteration, so regardless of the maximum value in the list, its length cannot exceed k^i . This gives us

$$|Pool(i)| \leq \min(k^i, MaxC(i)^k). \quad (15)$$

Lines 9 and 10 assure us that the algorithm terminates if $MaxC(i)$ exceeds the sum of the remaining object sizes, so we have

$$MaxC(i) \leq \sum_{j=1}^{n-i+1} s[j] \leq (n-i+1) \cdot s[n-i+1]. \quad (16)$$

To complete the analysis, we bound the step counts as a function of x , the total bit length of the size array s . As before, we consider two cases.

Case 1. $n \leq \sqrt{x}$. Here we have

$$\sum_{i=1}^n |Pool(i)|^2 \leq \sum_{i=1}^n \min(k^i, MaxC(i)^k)^2 \quad (17)$$

$$\leq n \cdot (k^n)^2 \leq n \cdot k^{2\sqrt{x}} \leq n \cdot 2^{2\lg k \sqrt{x}}. \quad (18)$$

Case 2. $n > \sqrt{x}$. In this case we split the summation at $i = \sqrt{x}$.

$$\sum_{i=1}^n |Pool(i)|^2 \leq \sum_{i=1}^{\sqrt{x}} \min(k^i, MaxC(i)^k)^2 \quad (19)$$

$$\leq \sum_{i=1}^{\sqrt{x}-1} \min(k^i, MaxC(i)^k)^2 + \sum_{i=\sqrt{x}}^n \min(k^i, MaxC(i)^k)^2 \quad (20)$$

$$\leq (\sqrt{x}-1) \cdot k^{\sqrt{x}-1} + (n-\sqrt{x}+1) \cdot (MaxC(\sqrt{x})^k)^2 \quad (21)$$

Then by Lemma 1:

$$< (\sqrt{x}-1) \cdot k^{\sqrt{x}-1} + (n-\sqrt{x}+1) \cdot ((n-\sqrt{x}+1)2^{\sqrt{x}+1})^{2k} \quad (22)$$

and by algebraic simplification:

$$< (n + n^{2k+1}) 2^{2k(\sqrt{x}+1)} \quad (23)$$

Since k is a constant, this establishes that the time complexity of *BinPack* is $p(n) \cdot 2^{O(\sqrt{x})}$ for a polynomial function $p(n)$.

5 Conclusion

The algorithms in the previous sections demonstrate that dynamic programming with dynamic allocation (DDP) can be used to prove that 0/1 Knapsack and Bin Packing with a fixed number of bins have time complexity $p(n) \cdot 2^{O(\sqrt{x})}$ where x is the total bit length of n input numbers. This places these problems with Partition and Subset Sum in the subclass of NP-complete problems that have sub-exponential upper bounds on running time, when input length is used as the complexity parameter.

The Knapsack problem was formulated as an optimization problem above, while Bin Packing was presented as a decision problem. It is apparent that the *Knapsack* algorithm can be modified to solve the decision version of the problem without changing its time complexity. It is also possible to modify *BinPack* to find the smallest bin capacity needed to store all objects in k bins, as long as k is constant, without changing its time complexity. Given the simplicity and generality of Lemma

1, which provides the foundation for the time analyses, we expect that the DDP method can be applied to any NP-complete problem involving a list of weighted objects that has pseudo-polynomial time complexity.

References

- [1] M. Alekhovich, A. Borodin, J. Buresh-Oppenheim, R. Impagliazzo, A. Magen, and T. Pitassi, "Toward a Model for Backtracking and Dynamic Programming," *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, pp. 308-322 (2005).
- [2] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman Press, San Francisco, CA (1979).
- [3] R. Impagliazzo, R. Paturi, and F. Zane, "Which Problems Have Strongly Exponential Complexity?," *Journal of Computer and System Sciences* 63, pp. 512-530, Elsevier Science (2001).
- [4] S. Martello and P. Toth, "A mixture of dynamic programming and branch-and-bound for the subset sum problem," *Management Science* 30(6), pp. 765-771 (1984).
- [5] T. E. O'Neil, "The Importance of Symmetric Representation," *Proceedings of the 2009 International Conference on Foundations of Computer Science (FCS 2009)*, pp. 115-119, CSREA Press (2009).
- [6] T. E. O'Neil and S. Kerlin, "A Simple $2^{O(\sqrt{x})}$ Algorithm for Partition and Subset Sum," *Proceedings of the 2010 International Conference on Foundations of Computer Science (FCS 2010)*, pp. 55-58, CSREA Press (2010).
- [7] R. Stearns and H. Hunt, "Power Indices and Easier Hard Problems", *Mathematical Systems Theory* 23 (1990), pp. 209-225.
- [8] G. J. Woeginger, "Exact Algorithms for NP-Hard Problems: A Survey," *Lecture Notes in Computer Science* 2570, pp. 185-207, Springer-Verlag, Berlin (2003).

On the Computing Power of Statecharts

Hanlin Lu and Sheng Yu

Department of Computer Science, University of Western Ontario, London, Ontario, Canada, N6A5B7
{hlu47, syu}@csd.uwo.ca

Abstract—*Statecharts provide a practical and expressive visual formalism to describe reactive systems. They have been adopted by a number of object modeling techniques and languages, such as the UML state machines. Although Statecharts' semantics has attracted much attention, the computation power of Statecharts was seldom considered. In this paper, we study the computation power of Statecharts by linking them to Wegner's Interaction Machines.*

Keywords: statecharts, computing power, finite automata, turing machines, interaction machines, interactive turing machines

1. Introduction

Statechart [5] was introduced by David Harel in 1987 as one of the most popular state-based visual/graphical formalism for designing reactive systems. With the hierarchy, concurrency and many other features, the statechart was more practical and expressive than the classical state diagrams [1]. They have been adopted by a number of object modeling techniques and languages, including the Object Modeling Technique (OMT) [17] and the Unified Modeling Language (UML) [18]. Besides, there are many formalisms for statecharts supported by software such as STATEMATE [7], Rhapsody [6], etc.

An accurate and comprehensive description of statecharts will benefit their applications greatly. However, Statecharts' semantics was not defined precisely in Harel's original paper [5], as described in [19]. The study for the semantics of statecharts has began by Harel et al. [8] right after the publication of [5]. Since then, many related papers have been published, e.g., [16], [11]. During the last twenty years, it has attracted much attention to formalize Statecharts' semantics, especially for the UML State machines, and various kinds of formal approaches have been introduced, e.g. Petri-net, temporal logic, graph transformation systems, etc. In the survey [2] by Crane and Dingel in 2005, the semantics for the UML state machines were categorized into 26 kinds based on the formalization approach.

We find that although so many formal models have been applied to describe Statecharts, the computation power of Statecharts was seldom considered. And not much research, [3], [9], on the computation power of the models that has been used to formalize Statecharts has been done. On

the contrary, we can find descriptions like "Statecharts are extended finite automata" in many places to equate the computation power of statecharts to finite automata. It has been shown in [12] clearly that they are very different. We will show in this paper that the computation power of statecharts is more appropriately modeled by Interaction Machines [20].

In the next section, we will introduce the basic definitions. In Section 3, we will investigate the linkage between Statecharts and Interaction Machines. We will conclude our study in Section 4.

2. Preliminary

In this section, we will review the basic definitions for Statecharts and Interaction Machines.

2.1 Statecharts

There are already many models for formalizing the Statecharts' semantics. In this paper, however, we intend to study their computing power rather than any specific definition of semantics. Hence we need a brief yet powerful definition for Statecharts, which includes only the essential elements.

Definition 1 (Statecharts): A statechart is a 9-tuple,

$$(Q, v, E, A, C, \delta, \eta, s, T),$$

where Q is the nonempty finite set of states; v is an unbounded variable; E is the nonempty finite set of events; A is the finite set of actions, which is split into two subsets: the set of external actions A_{ex} , which is a finite set of symbols, and the set of internal actions A_{in} , which is a finite set of functions $p : v \rightarrow v$; C is the finite set of conditions, where each condition is a boolean-valued function of the form $q : v \rightarrow \{0, 1\}$; $\delta : Q \times E \times C^* \rightarrow Q$ is the transition function; $\eta : \delta \rightarrow A^*$ is the action function; s is the initial state; and $T \subseteq Q$ is the set of terminating states.

We use only one unbounded variable v , for simplicity, to represent all variables used in any instance of statecharts. Each state $q \in Q$ can be defined recursively as a statechart S_q of the next level. However, a multi-level statechart can always be transformed into an equivalent one-level statechart although the number of states can increase significantly.

This definition is not intended to include all the features of statecharts. For example, the "activities" [18] can be included in the definition, which are associated with states, the history pseudo states can be described by variables, etc.

2.2 Interaction Machines

Interaction Machines (IMs) [20] were introduced by Peter Wegner more than ten years ago. They were considered to be a further development of *Turing machines (TMs)* model. He claimed that *TMs* shut out the world during the computation and stop their computation after generating outputs, thus they can not model interaction behaviors [4], [20], [21].

IMs extend *TMs* with dynamic streams to record interaction histories. The behavior of an *IM* will depend on both the current input and its previous inputs. We consider that *IMs* are more accurate theoretical models for statecharts than *TMs* since *IMs* are better models for interactions.

Based on Wegner's *IMs*, Sheng Yu defined *Interactive Turing Machines (ITMs)* [22], which extend Multi-tape *TMs* with interaction tapes. The *Sequential ITMs (SITMs)* [22] provide detailed relations between *TMs*' computing and sequential interactions. In this paper, we use *SITMs* as the model for Statecharts. For simplicity, we define *SITMs* with only one work tape.

Definition 2 (Sequential Interactive Turing Machines): Formally, a *Sequential Interactive Turing machine (SITM)* M is an 11-tuple

$$(I, O, Q, \Gamma, \gamma, \omega, s, \#, \$, B, T),$$

where I is the finite input alphabet; O is the finite output alphabet; Q is the nonempty finite set of states; $\#, \$, B$ are three special symbols, where $\#$ is the delimiter preceding an input string and at the end of an output string, $\$$ is the other delimiter that is ending an input string and preceding an output string, B is the blank symbol of the interaction tape and k work tapes; Γ is the finite work-tape alphabet;

$$\gamma : Q \times (I \cup \{\#, \$\}) \times \Gamma \rightarrow Q \times \{R, S\} \times (\Gamma \times \{L, R, S\})$$

is the transition function of M in the reading phase (when the *SITM* can only read on the interaction tape);

$$\omega : Q \times \{B\} \times \Gamma \rightarrow Q \times (O \cup \{\#\} \times \{R, S\}) \times (\Gamma \times \{L, R, S\})$$

is the transition function of M in the writing phase, where L (left), R (right), and S (stationary) are directions of the movement of a head; $s \in Q$ is the starting state of M ; $T \subseteq Q$ is the set of terminating states.

3. Linkage between Statecharts and Interactive Turing Machines

In this section, we will use a general notion to study the linkage between Statecharts and Interactive Turing Machines.

3.1 Labelled Transition Systems

Basically, both Statecharts and Interactive Turing Machines satisfy the general notion, *Labelled transition systems (LTSs)* [15], which consist of only the states, transitions

and labels. Note that, in *LTSs*, none of the three elements are necessarily finite.

Definition 3 (Labelled transition system): A Labelled transition system (*LTS*) is a 3-tuple

$$(S, T, \{\overset{t}{\rightarrow} \mid t \in T\}),$$

where S is the set of states, T is the set of labels, $\overset{t}{\rightarrow} \subseteq S \times S$, where $t \in T$, is the set of labelled transitions.

Example 1: Let statechart S be the 9-tuple

$$(Q_s, v_s, E_s, A_s, C_s, \delta_s, \eta_s, s_s, T_s).$$

Then S can be expressed as

$$((Q_s, v_s), E_s, \overset{E_s}{\rightarrow} \subseteq (Q_s, v_s) \times (Q_s, v_s))$$

in terms of an *LTS*, where the set of states is the pair formed by S 's states Q_s and the variable v_s (we will call this pair S 's *configuration* in the following to avoid confusion with the *states* of S , Q_s); the labels is the set of events E_s of S ; the transitions are obtained by applying S 's transition function δ_s to its action functions η_s , though all external actions are not considered here since they do not directly affect the statechart's state transitions.

Example 2: Let an *SITM* M be the tuple

$$(I_m, O_m, Q_m, \Gamma_m, \gamma_m, \omega_m, s_m, \#, \$, B, T_m).$$

We can write an *LTS* L_m as the following, which is exactly M ,

$$((Q_m, \Gamma^*, P), I_m \cup \{\#, \$, B\} \cup \{\epsilon\}, \\ I_m \cup \{\#, \$, B\} \cup \{\epsilon\} \xrightarrow{\quad} \subseteq (Q_m, \Gamma^*, P) \times (Q_m, \Gamma^*, P)).$$

The state set of L_m is the tuple formed by M 's state Q_m , the set of possible strings on M 's work tape Γ^* and the set of all positions of the head, P , on work tape, to avoid confusion with M 's states Q_m , we call this tuple M 's *configuration* in the following; the labels of L_m contains all possible symbols on M 's interaction tape, $I_m \cup \{\#, \$, B\}$, and the empty symbol ϵ , since the head on the interaction tape is allowed to stay after each transition which means no new input will be read at the next step; the set of transitions is isomorphic to the set $\gamma_m \cup \omega_m$, though we do not consider the output that M writes on the interaction tapes for every transition.

3.2 Bisimulation and Observation Equivalence

Our method to show Statecharts are Interaction Machines is based on *bisimulation*, which has been studied by David Park [14] and Robin Milner [13].

Bisimulation is an equivalence relation describing whether two agents behave in the same way based on observation. The "agents" are computing systems that are identified by their *states*, e.g. Statecharts and *SITMs*. The relation is also known as the *weak bisimulation*, since agents' internal interactions that cannot be detected from outside are not required to match exactly.

To express a sequence of state transition behaviors for agents, we need the following definition.

Definition 4: Let $L = (S, T, \{\overset{t}{\rightarrow} \mid t \in T\})$ be an LTS. If $\{a_1, \dots, a_n\} \in T$, $\{P_0, \dots, P_n\} \in S$, we write

$$P_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} P_n, \quad (1)$$

if

$$P_0 \xrightarrow{a_1} P_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} P_n.$$

We also write

$$P(\overset{a_1}{\rightarrow})^* P', \quad (2)$$

if there is an arbitrary number of transitions labelled by a_1 such that

$$P \xrightarrow{a_1} \dots \xrightarrow{a_1} P'.$$

We may notice from Example 2, there can be transitions in SITMs that are triggered by nothing (an empty input ϵ) from outside. In [13], such input is considered as a *silent action*, by which the transitions are labelled can execute without awareness from outside.

Definition 5 (Bisimulation): A binary relation $R \subseteq \mathcal{P} \times \mathcal{P}$ over agents is a (weak) *bisimulation* if $(P, Q) \in R$ implies, for all $\alpha \in Act$,

- 1) Whenever $P \xrightarrow{\alpha} P'$ then, for some Q' ,

$$Q(\overset{\epsilon}{\rightarrow})^* \overset{\hat{\alpha}}{\rightarrow} (\overset{\epsilon}{\rightarrow})^* Q'$$

and $(P', Q') \in R$

- 2) Whenever $Q \xrightarrow{\alpha} Q'$ then, for some P' ,

$$P(\overset{\epsilon}{\rightarrow})^* \overset{\hat{\alpha}}{\rightarrow} (\overset{\epsilon}{\rightarrow})^* P'$$

and $(P', Q') \in R$

Where \mathcal{P} is the set of agents; Act is the set of labels;

$$\hat{\alpha} = \begin{cases} \alpha & \text{if } \alpha \text{ is non-silent, e.g. } \alpha \in Act \setminus \{\epsilon\}, \\ \epsilon & \text{if } \alpha = \epsilon. \end{cases}$$

Based on the above definitions, we can claim the following:

Theorem 1: For any Statechart S , there is a Sequential Interactive Turing Machine M such that M and S satisfy the bisimulation relation.

Proof: We prove this by providing a method to construct an Sequential Interactive Turing Machine (SITM) M for any given statechart S such that M bisimulates S . We may consider the statechart S in Example 1.

Let M 's input alphabet I_m formed by the all the elements of the events E_s of S such that

$$I_m = E_s.$$

Since M 's work tape and S 's variables are essentially equivalent storage models, there exists a bijection between the string on the work tape and the value of the variable,

$$f : \Gamma^* \rightarrow v_s.$$

However, at each computing step, the work tape is allowed to read/write only one symbol from a finite set, which is different from operating a variable, M may require a finite number of steps to modify its work tape in order to simulate the change of value of v_s in S .

Let M has the same number of states as S such that there is a bijection between Q_m and Q_s . Note that Q_m will be expanded later. Let the initial configuration of M , (s_m, ϵ, p_0) , simulates that of S , (s_s, val_0) , where $\epsilon \in \Gamma^*$ represents the value of the blank work tape and p_0 is the starting position of the work tape's head, $val_0 \in v_s$ is the initial value of v_s .

For every configuration $(q_s, val) \in (Q_s, v_s)$ of S , if and only if $\exists i \in E_s$ such that

$$(q_s, val) \xrightarrow{i} (q'_s, val'),$$

we construct a (finite sequence of) transition(s)

$$(q_m, t_m, p)(\overset{\epsilon}{\rightarrow})^* \xrightarrow{i} (\overset{\epsilon}{\rightarrow})^* (q'_m, t'_m, p'),$$

in M such that ϵ is the "silent action" of M , (q_m, t_m, p) simulates (q_s, val) and (q'_m, t'_m, p') simulates (q'_s, val') , where

$$(q_m, t_m, p), (q'_m, t'_m, p') \in (Q_m, \Gamma^*, P)$$

and

$$(q_s, val), (q'_s, val') \in (Q_s, v_s).$$

The set of states Q_m may be expanded after the construction of transitions.

Since all the configurations of M are constructed to simulate that of S , it can be verified that for every configurations of M , $(q_m, t_m, p) \in (Q_m, \Gamma^*, P)$, if there exists a transition

$$(q_m, t_m, p) \xrightarrow{i} (q'_m, t'_m, p'),$$

where $i \in I_m \cup \{\epsilon\}$ and (q_m, t_m, p) is simulating $(q_s, v_s) \in (Q_s, V_s)$, then

- 1) if $i \neq \epsilon$, S has the configuration (q'_s, v'_s) such that

$$(q_s, v_s) \xrightarrow{i} (q'_s, v'_s),$$

where $e_s \in E_s$ is identical to i_m , and (q'_s, v'_s) is simulating (q'_m, t'_m, p') ;

- 2) if $i = \epsilon$, S does not have any corresponding transition and (q_s, v_s) is simulating (q'_m, t'_m, p') .

■

Theorem 2: For any SITM N , there is a Statechart T such that T and N satisfy the bisimulation relation.

The proof for Theorem 2 has been omitted since it is similar to the proof for Theorem 1. Now we have shown that Statecharts and SITMs are equivalent models.

4. Conclusions

Although much research has been done to achieve a precise semantics for Statecharts, the issues on the computation power of statecharts have not gained much attention. In this paper, we use an approach which is different from other formalisms. We claim that Wegner's Interaction Machines are more precise models for Statecharts and we have shown that Statecharts and Interactive Turing Machines [22] are equivalent based on bisimulation. We believe that an accurate and comprehensive description of statecharts will benefit the applications of statecharts greatly, for example, providing a more accurate description for computing objects.

References

- [1] Taylor L. Booth, In book of *Sequential Machines and Automata Theory*. John Wiley & Sons, (1967).
- [2] Michelle L. Crane and Juergen Dingel On the semantics of UML State machines: Categorization and Comparison Technical Report, Queen's University, (2005).
- [3] Doron Drusinsky and David Harel. On the Power of Bounded Concurrency I: Finite Automata, *Journal of the ACM*, Vol. 41, 517–539 (1994).
- [4] Dina Goldin and Peter Wegner. The interactive nature of computing: refuting the strong Church-Turing thesis. *Minds and Machines*, 18(1):17–38 (2008).
- [5] David Harel. Statecharts: A visual formulation for complex systems. *Science of Computer Programming*, 8(3):231–274 (1987).
- [6] David Harel, Hillel Kugler. The Rhapsody Semantics of Statecharts (or, On the Executable Core of the UML) In *Integration of Software Specification Techniques for Application in Engineering*, LNCS 3147, 325-354 (2001)
- [7] David Harel, Hag Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, Vol. 16, No. 4, 403IC414 (1990)
- [8] D. Harel, A. Pnueli, J. P. Schmidt, and S. Sherman. On the formal semantics of statecharts. In *Proceedings of the Second IEEE Symposium on Logic in Computation*, IEEE, 54IC64 (1987).
- [9] Tirza Hirst and David Harel. On the power of bounded concurrency II: pushdown automata, *Journal of the ACM*, Vol. 41, 540–554 (1994).
- [10] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley (1979).
- [11] Cornelis Huizing, Rob Gerth and Willem P. de Roever. Modeling Statecharts Behaviour in a Fully Abstract Way. In *Proceedings of the 13th Colloquium on Trees in Algebra and Programming*, 271–294 (1988).
- [12] Hanlin Lu and Sheng Yu. Are Statecharts Finite Automata. In *Proceedings of the 14th International Conference on Implementation and Application of Automata*, LNCS 5642, Springer, 258-261 (2009).
- [13] Robin Milner. In the Book *Communication and Concurrency*. Prentice Hall, 88-128 (1989).
- [14] David Park. *Concurrency and automata on infinite sequences*. Theoretical Computer Science, LNCS 104, 167-183 (1981).
- [15] Gordon D. Plotkin. *A Structural Approach to Operational Semantics*, Report DAIMI-FN-19, Computer Science Dept, Århus University, Denmark (1981).
- [16] Amir Pnueli and M. Shalev. What is in a step: On the semantics of statecharts. In *TACS '91: Proceedings of the International Conference on Theoretical Aspects of Computer Software*, Springer-Verlag, 244-264 (1991).
- [17] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall (1991).
- [18] James Rumbaugh, Ivar Jacobson and Grady Booch. *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education (2004).
- [19] Michael von der Beeck A comparison of statecharts variants. in *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94)*, LNCS 863, Springer, 128-148(1994).
- [20] Peter Wegner. Why interaction is more powerful than algorithms. *Commun. ACM*, 40(5):80–91 (1997).
- [21] Peter Wegner. Interactive foundations of computing. *Theoretical Computer Science* 192(2):315–351 (1998).
- [22] Sheng Yu. The time dimension of computation models. In *Book of Where mathematics, computer science, linguistics and biology meet*, Chapter 14, 161-172 (2001).
- [23] Sheng Yu. Regular Languages. In *Handbook of Formal Languages*, vol. 1, edited by G. Rozenberg and A. Salomaa, Springer (1997).

Playing Challenging Iterated Two-Person Games Well: A Case Study on the Iterated Traveler's Dilemma

Philip Dasler, Predrag T. Tošić

Department of Computer Science, University of Houston, Houston, Texas, USA
{philip.dasler, pedja.tosic}@gmail.com

Abstract—We study an interesting 2-player game known as the Iterated Traveler's Dilemma, a non-zero sum game in which there is a large number of possible actions in each round and therefore an astronomic number of possible strategies overall. What makes the Iterated TD particularly interesting is that it defies the usual prescriptions of classical game theory insofar as what constitutes an "optimal" strategy. In particular, TD has a single Nash equilibrium, yet that equilibrium corresponds to a very low payoff, essentially minimizing social welfare. We propose a number of possible strategies for ITD and perform a thorough comparison via a round-robin tournament in the spirit of Axelrod's well-known work on the Prisoner's Dilemma. We motivate the choices of "players" that comprise our tournament and then analyze their performance with respect to several metrics. Finally, we share some interesting conclusions and outline directions for future work.

Keywords: game theory, two-person non-zero-sum games, bounded rationality, decision making under uncertainty, tournaments

1. Introduction

Theoretical computer science, mathematical economics and AI research communities have extensively studied strategic interactions among two or more autonomous agents from a game-theoretic standpoint. Game theory provides mathematical foundations for modeling interactions among, in general, *self-interested* autonomous agents that may need to combine competition and cooperation in non-trivial ways in order to meet their objectives [1–3]. A classical example of such interactions is the *iterated prisoner's dilemma* [4, 5], a two-person non-zero sum game that has been extensively studied by psychologists, sociologists, economists and political scientists, as well as mathematicians and computer scientists.

We have been studying an interesting and complex 2-player game known as the *Iterated Traveler's Dilemma* [6–8]. The *Traveler's Dilemma* (TD) is a non-zero sum game in which each player has a large number of possible actions. In the iterated context, this means many possible actions in each round and therefore (for games with many rounds) an astronomic number of possible strategies overall. What

makes Iterated TD particularly interesting, is that its structure defies the usual prescriptions of classical game theory insofar as what constitutes an "optimal" strategy. There are two fundamental problems to be addressed in this context. One is finding an optimal, or close to optimal, strategy from the standpoint of an individual intelligent agent. This is the "default" problem in classical game theory: finding the best "play" for each agent participating in the game. The second core problem is identifying *pairs of strategies* that would result in an overall desirable behavior, such as maximizing a joint utility function of some sort (i.e., appropriately defined "social welfare"). We have been investigating both these problems in the context of the Iterated Traveler's Dilemma, which has thus far received only modest attention by the computer science research communities. In this paper, we shed some light on the first problem above using a round-robin, many-round tournament and several different performance metrics. We also draw several interesting (and possibly controversial) conclusions based on our extensive experimentation and analyses.

The rest of this paper is organized as follows. We first describe the Traveler's Dilemma (TD) game and motivate why we find it interesting. We also briefly survey the prior art. We then describe the Iterated TD round-robin tournament that we have devised, implemented and experimented with. In that context, we focus on the actual strategies we have chosen as the participants in this tournament, and on why these strategies are good examples of the kinds of strategies one would expect to be "reasonable". We then describe several *metrics* that we have used as yardsticks of performance of the various strategies involved in our round-robin tournament. Next, we summarize our tournament results and discuss our main findings, both those that we expected and those that we honestly find fairly surprising. Finally, we draw conclusions based on our analytical and experimental results to date and outline some promising directions for future research.

2. Traveler's Dilemma (TD)

The Traveler's Dilemma was originally introduced by K. Basu in 1994 [9]. The motivation behind the game was to show the limitations of classical game theory [10], and in particular the notions of *individual rationality* that stem from game theoretic notions of "solution" or "optimal play"

based on well-known mathematical concepts such as *Nash equilibria* [3, 9, 11]. TD is defined with the following parable:

“An airline loses two suitcases belonging to two different travelers. Both suitcases happen to be identical and contain identical antiques. An airline manager tasked to settle the claims of both travelers explains that the airline is liable for a maximum of \$100 per suitcase, and in order to determine an honest appraised value of the antiques the manager separates both travelers so they can't confer, and asks them to write down the amount of their value at no less than \$2 and no larger than \$100. He also tells them that if both write down the same number, he will treat that number as the true dollar value of both suitcases and reimburse both travelers that amount. However, if one writes down a smaller number than the other, this smaller number will be taken as the true dollar value, and both travelers will receive that amount along with a bonus/malus: \$2 extra will be paid to the traveler who wrote down the lower value and a \$2 deduction will be taken from the person who wrote down the higher amount. The challenge is: what strategy should both travelers follow to decide the value they should write down?”

This game has several interesting properties. Perhaps the most striking among them is that its *unique* Nash equilibrium, the action pair $(p, q) = (\$2, \$2)$, is actually rather bad for both players. This choice of actions results in:

- very low payoff to each player individually (basically, only slightly above the absolutely worst possible, which is \$0); and, moreover,
- it minimizes *social welfare*, if we understand social welfare to simply be the sum of the two players' individual utilities.

Yet, it has been argued [7, 9, 12] that a *perfectly rational* player, according to classical game theory, would “reason through” and converge to choosing the lowest possible value, \$2. Given that the TD game is symmetric, each player would reason along the same lines and, once selecting \$2, not deviate from it, as *unilaterally* deviating from a Nash equilibrium is presumably bad “by definition”. However, the non-equilibrium pair of strategies (\$100, \$100) results in each player earning \$100, very near the best possible individual payoff. Hence, the early studies of TD concluded that this game demonstrates a woeful inadequacy of classical, Nash Equilibrium based notions of rational behavior. It has also been shown that humans (both game theory experts and laymen) tend to play far from the Nash equilibrium [6], and therefore fare much better than they would if they followed the classical approach.

In general, basing the notion of a “solution” to a game on *Nash equilibria* (NE) has been known to be tricky. Among other things, a game may fail to have any NE (in pure strategies), or it may have multiple Nash equilibria. TD is interesting precisely because it has a *unique* pure-strategy

Nash equilibrium, yet this NE results in nearly as low a payoff as one can get. The situation is further complicated by the fact that the game's only “stable” strategy pair is easily seen to be nowhere close to *Pareto optimal*; there are many obvious ways of making both players much better off than if they play the equilibrium strategies. In particular, while neither stable nor an equilibrium, (\$100, \$100) is the unique strategy pair that maximizes social welfare (understood as the sum of individual payoffs), and is, in particular, Pareto optimal. So, the fundamental question arises, how can agents learn to sufficiently trust each other so that they end up playing this optimal strategy pair in the Iterated TD or similar scenarios?

3. The Iterated TD Tournament

Our Iterated Traveler's Dilemma tournament is similar to Axelrod's Iterated Prisoner's Dilemma tournament [13]. In particular, it is a round-robin tournament where each agent plays N matches against each other agent and its own “twin”. A match consists of T rounds. In order to have statistically significant results (esp. given that many of our strategies involve randomization in some way), we have selected $N = 100$ and $T = 1000$. In each round, both agents must select a valid bid within the *action space*, defined as $A = \{2, 3, \dots, 100\}$.

The method in which an agent chooses its next action for all possible histories of previous rounds is known as a strategy. A *valid strategy* is a function S that maps some set of inputs to an action: $S : \cdot \rightarrow A$. In general, the input may include the entire history of prior play, or, in the case of *bounded rationality* models, an appropriate summary of the past histories.

We next define the participants in the tournament, that is, the set of strategies that play one-against-one matches with each other. Let C be the set of agents competing in the tournament: $C = \{c : (c \in S) \wedge (c \text{ is in the tournament})\}$.

We specify a pair of agents competing in a match as $(x, y) \in C$. While we refer to agents as *opponents* or *competitors*, this need not necessarily imply that the agents act as each other's adversaries.

We define agents' actions as follows:

x_t = the bid traveler x makes on round t .

x_{nt} = the bid traveler x makes on round t of match n .

We next define the *reward function* that describes agent payoffs. *Reward per round*, $R : A \times A \rightarrow \mathbb{Z} \in [0, 101]$, for action α against action β , where $\alpha, \beta \in A$, is defined as $R(\alpha, \beta) = \min(\alpha, \beta) + 2 \cdot \text{sgn}(\beta - \alpha)$, where $\text{sgn}(x)$ is the usual *sign* function. Therefore, the total reward $M : S \times S \rightarrow \mathbb{R}$ received by agent x in a match against y is defined as:

$$M(x, y) = \sum_{t=1}^T R(x_t, y_t)$$

Within a sequence of matches, the reward received by agent x in the n^{th} match against y shall be denoted as $M_n(x, y)$.

4. Strategies in the Tournament

In order to make a reasonable baseline comparison, we chose to utilize the same strategies used by [14], ranging from rather simplistic to relatively complex. What follows is a brief description of the 9 classes of strategies. For a more in depth description see [14].

Randoms: The first, and simplest, class of strategies play a random value, uniformly distributed across a given interval. We have implemented two instances using the following integer intervals: [2, 100] and [99, 100].

Simpletons: The second extremely simple class of strategies are agents that choose the same dollar amount in every round. The \$ values we used in the tournament were $x_t = 2$ (the lowest possible), $x_t = 51$ (“median”), $x_t = 99$ (one below maximal possible, resulting in maximal payoff should the opponent make the highest possible bid), and $x_t = 100$ (the highest possible).

Tit-for-Tat, in spirit: The next class of strategies are those that can be viewed as *Tit-for-Tat-in-spirit*, where Tit-for-Tat is the famously simple, yet effective, strategy for the iterated prisoner’s dilemma [4, 5, 13, 15]. The idea behind *Tit-for-Tat* is simple: cooperate on the first round, then do exactly what your opponent did on the previous round. In the iterated TD, each agent has many actions at his disposal, hence there are different ways of responding appropriately in a Tit-for-Tat manner. In general, playing high values can be considered as an approximate equivalent of “cooperating”, whereas playing low values is an analogue of “defecting”. Following this basic intuition, we have defined several Tit-for-Tat-like strategies for the iterated TD. These strategies can be roughly grouped into two categories. First, the simple Tit-for-Tat strategies bid some value ϵ below the bid made by the opponent in the last round, where $\epsilon \in \{1, 2\}$. Second, the predictive Tit-for-Tat strategies compare whether their last bid was lower than, equal to, or higher than that of their opponent. Then a bid is made similar to the simple TFT strategy, i.e. some value ϵ below the bid made by competitor c in the last round, where $c \in [x, y]$ and $\epsilon \in \{1, 2\}$. The key distinction is that a bid can be made relative to either the opponent’s last bid or the bid made by one’s own strategy itself. In essence, this strategy is predicting that the opponent may make a bid based on the agent’s own last move and, given that prediction, it attempts to “outsmart” the opponent.

Mixed: The mixed strategies probabilistically combine up to three strategies. For each mixed strategy, a strategy σ is selected from one of the other strategies defined in the competition (i.e., $\sigma \in C$) for each round according to a probability mass distribution. Once a strategy has been selected, the value that σ would bid at time step t is bid. We chose to use only mixtures of the TFT, Simpleton, and

Random strategies. This allowed for greater transparency when attempting to interpret and understand the causes of a particular strategy’s performance.

Buckets – Deterministic: These strategies keep a count of each bid by the opponent in an array of “buckets”. The bucket that is most full (i.e., the value bid most often) is used as the predicted value, with ties being broken by one of the following methods: the highest valued bucket wins, the lowest valued bucket wins, a random bucket wins, and the most recently tied-for-the-largest bucket wins. The strategy then bids the next lowest value below the predicted value. An instance of each tie breaking method competed in the tournament.

Buckets – Probability Mass Function based: As above, this strategy class counts instances of the opponent’s bids and uses them to predict the agent’s own next bid. Rather than picking the value most often bid, the buckets are used to define a probability mass function from which a prediction is randomly selected. Values in the buckets decay over time in order to emphasize newer data over old and we have set a retention rate ($0 \leq \gamma \leq 1$) to determine the rate of decay. We have entered into our tournament several instances of this strategy using the following rate of retention values γ : 0.8, 0.5, and 0.2. As above, the strategy bids the next lowest value below the predicted value. We observe that the “bucket” strategies based on probability mass buckets are quite similar to a learning model in [7].

Simple Trend: This strategy looks at the previous k time steps, creates a line of best fit on the rewards earned, and compares its slope to a threshold δ . If the trend has a positive slope greater than δ , then the agent will continue to play the same bid it has been as the rewards are increasing. If the slope is negative and $|slope| > \delta$, then the system is trending toward the Nash equilibrium and, thus, the smaller rewards. In this case, the agent will attempt to maximize social welfare and play 100. Otherwise, the system of bidding and payouts is relatively stable and the agent will play the “one under” strategy. We have implemented instances of this strategy with $\delta = 0.5$ and the following values of k : 3, 10, and 25. While our choice of δ intuitively makes sense, we admit that picking δ “half-way” between 0.0 and 1.0 is fairly arbitrary.

Q-learning: This strategy uses a learning rate α to emphasize new information and a discount rate γ to emphasize future gains. In particular, the learners in our tournament are simple implementations of *Q-learning* [16] as a way of predicting the best action at time $(t + 1)$ based on the action selections and payoffs at times $\{1, 2, \dots, t\}$. This is similar to the Friend-or-Foe Q-learning method [17] without the limitation of having to classify the allegiance of one’s opponent.

Due to scaling issues, our implementation of Q-learning does not capture the entire state/action space but rather divides it into a small number of meaningful classes, namely,

three states and three actions, as follows:

State: The opponent played higher, lower, or equal to our last bid.

Action: Play one higher than, one lower than, or equal to our previous bid.

Recall that *actions* are defined for a single round. Our implementation treats each state as a collection of moves by the opponent over the last k rounds. We have decided to use 5 as an arbitrary value for k as it allows us to capture some history without data sizes becoming unmanageable. We have implemented this basic Q-learning algorithm with the learning rates of 0.8, 0.5 and 0.2, and with discount rates of 0.0 and 0.9, for a total of 6 different variations of Q-learning.

Zeuthen Strategies: A Zeuthen Strategy [18] calculates the level of risk of each agent, and makes *concessions* accordingly. Risk is the ratio of loss from accepting the opponent's proposal to the loss of forcing the conflict deal (the deal made when no acceptable proposal can be found). While ITD is not a strict negotiation, we treat each bid as a proposal. If $x_t = i$, then X is proposing $(i, i + 1)$ be the next action pair. For the Traveler's Dilemma, we consider the conflict deal to be the Nash Equilibrium at $(\$2, \$2)$.

Given the proposals of each agent, a risk comparison is done. An agent will continue to make the same proposal while its risk is greater than or equal to its opponent's. Otherwise, the agent will make the *minimal sufficient concession*, i.e. the agent adjusts its proposal so that (i) the agent's risk is higher than that of its opponent and (ii) the opponent's utility increases as little as possible. Due to the peculiar structure of the Iterated TD game, it is possible for the "concession" to actually lead to a *loss of utility* for the opponent. We have therefore implemented two variations: one that allows a negative concession and one that does not.

5. Utility metrics

In order to classify a particular strategy as better than another, one needs to define the metric used to make this determination. Our experimentation and subsequent analysis were performed with respect to four distinct utility metrics. The first, U_1 , treats the actual dollar amount as the direct payoff to the agent. This is the most common metric in the game theory literature; prior art on Iterated TD generally considers only this metric or some variant of it. In contrast, U_2 is a "pairwise victory" metric: an agent strives to beat its opponent, regardless of the actual dollar amount it receives. Finally, we introduce two additional metrics, U_3 and U'_3 , that attempt to capture both the payoff (dollar amount) that an agent has achieved, and the "opportunity lost" due to not playing differently. In a sense, both of these metrics attempt to quantify how much an agent wins compared to how much an *omniscient agent* (i.e., one that always correctly predicts the other agent's bid) would be able to win. To be clear, the assumption here is one of omniscience, not omnipotence:

an "ideal" omniscient agent is still not able to actually force what the other agent does.

Total reward: U_1

This metric captures the overall utility rewarded to the agent. It is simply a sum of all money gained, normalized by the total number of rounds played and the maximum allowable reward. It is defined as follows:

$$U_1(x) = \frac{1}{|C|} \sum_{j \in C} \left[\frac{1}{\max(R)NT} \sum_{n=1}^N M_n(x, j) \right]$$

where:

- $\max(R)$ is the maximum possible reward given in one round;
- N is the number of matches played between each pair of competitors;
- T is the number of rounds to be played in each match; and
- $|C|$ is the number of competitors in the tournament.

Pairwise Victory Count : U_2

This metric captures the agent's ability to do better than its opponents on a match per match basis. The metric itself is essentially the difference between matches won and matches lost. The result is normalized and 0.5 is added in order to bring all values inside the range $[0.0, 1.0]$. It is defined as follows:

$$U_2(x) = 0.5 + \frac{1}{2|C|} \sum_{j \in C} \left[\frac{1}{N} \sum_{n=1}^N \text{sgn}(M_n(x, j) - M_n(j, x)) \right]$$

where:

- N is the number of matches played between each pair of competitors;
- $|C|$ is the number of competitors in the tournament.

The intent of this metric is to capture a strategy's ability to "outsmart" its opponent, regardless of the possibility of a Pyrrhic victory.

Perfect Score Proportion : U_3

This metric attempts to capture the level of optimality of an agent, where both the achieved payoff and the missed opportunity for yet higher payoff (based on what the opposing agent does) are taken into account. The metric captures these two aspects of performance by keeping a running total of the *lost reward* ratio. This is the ratio of the reward received to the best possible reward, given what the opponent has played. The resulting sum is then normalized by the total number of rounds played. The metric is formally defined as follows:

$$U_3 = \frac{1}{|C|} \sum_{j \in C} \left[\frac{1}{N} \sum_{m=1}^N \left(\frac{1}{T} \sum_{t=1}^T \frac{R(x_{mt}, j_{mt})}{R(\max(2, (j_{mt} - 1)), j_{mt})} \right) \right]$$

where:

- N is the number of matches played between each pair of competitors
- T is the number of rounds to be played in each match
- $|C|$ is the number of competitors in the tournament

During the course of our work, it has been observed that this metric tends to be biased in favor of strategies that *overbid*. When overbidding, the difference between the reward received and the optimal reward is at worst 4. Thus, regardless of by how much an agent overbids, the lost reward ratio remains relatively small.

Perfect Bid Proportion : U'_3

This metric is another attempt to capture the level of optimality of the agent, but without the overbid bias. It does so by keeping a running total of the bid imperfection ratio. This is the difference between the agent’s bid and the ideal bid, given what the opponent has played, divided by the greatest possible difference in bids. Since we want to look favorably upon a smaller difference, this value is subtracted from 1. This sum is then normalized using the total number of rounds played. The metric is defined as follows:

$$U'_3 = \frac{1}{|C|} \sum_{j \in C} \left[\frac{1}{N} \sum_{m=1}^N \left(\frac{1}{T} \sum_{t=1}^T \left[1 - \frac{|x_{mt} - (j_{mt} - 1)|}{\max(A) - \min(A)} \right] \right) \right]$$

where:

- N is the number of matches played between each pair of competitors;
- T is the number of rounds to be played in each match;
- $|C|$ is the number of competitors in the tournament.

6. Results and Analysis

The Traveler’s Dilemma Tournament that we have experimented with involves a total of 38 competitors (i.e., distinct strategies). Each competitor plays each other competitor (including its own “twin”) 100 times. Each match is played for 1000 rounds. No meta-knowledge or knowledge of the future is allowed: learning and adaptation of those agents whose strategies are adaptable takes place exclusively based on the previous rounds in a match against a given opponent without *a priori* knowledge of that opponent. For definitions of the shorthand notation used in the sequel, see [14]. Throughout the rest of the paper, we assume the default version of ITD: the space of allowable bids is the interval of integers [2, 100], granularity of bids is 1, and the Bonus/Malus is equal to 2.

[Note: due to space constraints, we do not include the tabulated tournament results with respect to metric U'_3 .]

The top three performers in our tournament, with respect to the earned dollar amount as the bottom line (metric U_1), are three “dumb” strategies that always bid very high. Interestingly enough, randomly alternating between the highest possible bid (\$100) and “one under” the highest bid (\$99) slightly outperforms both “always max. possible” and “always one under max. possible” strategies. We find it

Table 1: Ranking Based on U_1

0.760787	Random [99, 100]
0.758874	Always 100
0.754229	Always 99
0.754138	Zeuthen Strategy - Positive
0.744326	Mixed - L(y-g) E(x-g) H(x-g), 80%; (100, 20%)
0.703589	Simple Trend - K = 3, Eps = 0.5
0.681784	Mixed - TFT (y-1), 80%; (R[99, 100], 20%)
0.666224	Simple Trend - K = 10, Eps = 0.5
0.639572	Simple Trend - K = 25, Eps = 0.5
0.637088	Mixed - L(x) E(x) H(y-g), 80%; (100, 20%)
0.534378	Mixed - L(y-g) E(x-g) H(x-g), 80%; (100, 10%); (2, 10%)
0.498134	Q Learn - alpha= 0.2, discount= 0.0
0.497121	Q Learn - alpha= 0.5, discount= 0.0
0.496878	Q Learn - alpha= 0.5, discount= 0.9
0.495956	Q Learn - alpha= 0.2, discount= 0.9
0.493640	Q Learn - alpha= 0.8, discount= 0.0
0.493639	Buckets - (Fullest, Highest)
0.493300	Q Learn - alpha= 0.8, discount= 0.9
0.492662	TFT - Low(y-g) Equal(x-g) High(x-g)
0.452596	Zeuthen Strategy - Negative
0.413992	Buckets - PD, Retention = 0.5
0.413249	Always 51
0.412834	Buckets - PD, Retention = 0.2
0.408751	Buckets - PD, Retention = 0.8
0.406273	Buckets - (Fullest, Random)
0.390303	TFT - Simple (y-1)
0.387105	Buckets - (Fullest, Newest)
0.334967	Buckets - (Fullest, Lowest)
0.329227	TFT - Simple (y-2)
0.316201	Random [2, 100]
0.232063	Mixed - L(y-g) E(x-g) H(x-g), 80%; (2, 20%)
0.164531	Mixed - L(x) E(x) H(y-g), 80%; (100, 10%); (2, 10%)
0.136013	TFT - Low(x) Equal(x) High(y-g)
0.135321	TFT - Low(x) Equal(x-2g) High(y-g)
0.030905	TFT - Low(x-2g) Equal(x) High(y-g)
0.030182	TFT - Low(x-2g) Equal(x-2g) High(y-g)
0.026784	Mixed - L(x) E(x) H(y-g), 80%; (2, 20%)
0.024322	Always 2

somewhat surprising that the performance of Tit-for-Tat-based strategies varies so greatly depending on the details of bid prediction method and metric choice. So, while a relatively complex TFT-based strategy that, in particular, (i) makes a nontrivial model of the other agent’s behavior and (ii) “mixes in” some randomization, is among the top performers with respect to metric U_1 , other TFT-based strategies have fairly mediocre performance with respect to the same metric, and are, indeed, scattered all over the tournament table. In contrast, if metric U_3 is used, then the simplest, deterministic “one under what the opponent did on the previous round” TFT strategy, which is a direct analog of the famous TFT in Axelrod’s Iterated Prisoner’s Dilemma, is the top performer among all 38 strategies in the tournament – while more sophisticated TFT strategies, with considerably more complex models of the opponent’s behavior and/or randomization involved, show fairly average performance. Moreover, if U_3 is used as the yardstick, then (i) 3 out of the top 4 performers overall are TFT-based strategies, and (ii) all simplistic TFT strategies outperform all more sophisticated ones.

Not very surprisingly, the top (and bottom) performers with respect to metric U_1 and those with respect to U_2 are practically inverted; so, for example, the very best performer

Table 2: Ranking Based on U_2

0.984342	Always 2
0.924474	TFT - Low(x-2g) Equal(x-2g) High(y-g)
0.915263	TFT - Low(x-2g) Equal(x) High(y-g)
0.887500	Mixed - L(x) E(x) H(y-g), 80%; (2, 20%)
0.845132	TFT - Simple (y-2)
0.839868	TFT - Low(x) Equal(x-2g) High(y-g)
0.832368	TFT - Low(x) Equal(x) High(y-g)
0.791842	TFT - Simple (y-1)
0.727105	Buckets - PD, Retention = 0.2
0.681842	Mixed - L(x) E(x) H(y-g), 80%; (100, 20%)
0.669079	Mixed - L(y-g) E(x-g) H(x-g), 80%; (2, 20%)
0.653816	Buckets - PD, Retention = 0.5
0.629605	Mixed - L(x) E(x) H(y-g), 80%; (100, 10%); (2, 10%)
0.622632	TFT - Low(y-g) Equal(x-g) High(x-g)
0.616711	Buckets - PD, Retention = 0.8
0.558158	Mixed - TFT (y-1), 80%; (R[99, 100], 20%)
0.557368	Buckets - (Fullest, Newest)
0.539342	Simple Trend - K = 25, Eps = 0.5
0.528421	Buckets - (Fullest, Lowest)
0.491842	Random [2, 100]
0.483684	Simple Trend - K = 10, Eps = 0.5
0.480789	Buckets - (Fullest, Random)
0.463816	Buckets - (Fullest, Highest)
0.455000	Mixed - L(y-g) E(x-g) H(x-g), 80%; (100, 10%); (2, 10%)
0.407500	Simple Trend - K = 3, Eps = 0.5
0.303158	Mixed - L(y-g) E(x-g) H(x-g), 80%; (100, 20%)
0.260263	Q Learn - alpha= 0.5, discount= 0.0
0.255658	Q Learn - alpha= 0.8, discount= 0.0
0.253289	Q Learn - alpha= 0.8, discount= 0.9
0.252763	Q Learn - alpha= 0.2, discount= 0.0
0.249605	Q Learn - alpha= 0.5, discount= 0.9
0.247237	Q Learn - alpha= 0.2, discount= 0.9
0.200000	Always 51
0.183289	Zeuthen Strategy - Negative
0.092368	Always 99
0.066711	Random [99, 100]
0.040789	Zeuthen Strategy - Positive
0.013289	Always 100

with respect to U_2 is the strategy “always bid \$2” (which also happens to be the only non-dominated strategy in the classical game theoretic-sense). On the other hand, the three best performers with respect to U_1 are all among the four bottom performers with respect to U_2 , with the only strategy that *may* maximize social welfare (bidding \$100 against a collaborative opponent) falling at the rock bottom of the U_2 ranking. The main conclusion we draw from this performance inversion is that *when a two-player game has a structure that makes it very far from being zero-sum, the traditional precepts from classical game theory on what constitutes good strategies are more likely to fail*. This does not mean to suggest that classical game theory is useless; rather, we’d argue that the appropriate quantitative, mathematical models of rationality for zero-sum, or nearly zero-sum, encounters aren’t necessarily the most appropriate notions for games that are rather far from being zero-sum.

Returning to our tournament results, what we have found *very surprising* is the relative mediocrity of the learning based strategies: Q-learning based strategies did not excel with respect to any of the four metrics we studied. On the other hand, it should be noted that the adaptability of Q-learning based strategies apparently ensures that they do not do too badly overall, regardless of the choice of metric.

Table 3: Ranking Based on U_3

0.973118	Buckets - PD, Retention = 0.2
0.970587	Buckets - PD, Retention = 0.5
0.970356	TFT - Simple (y-1)
0.968923	Simple Trend - K = 10, Eps = 0.5
0.967860	TFT - Low(y-g) Equal(x-g) High(x-g)
0.965654	TFT - Simple (y-2)
0.962212	Simple Trend - K = 3, Eps = 0.5
0.959252	Buckets - PD, Retention = 0.8
0.955886	Simple Trend - K = 25, Eps = 0.5
0.953725	Buckets - (Fullest, Newest)
0.945405	Buckets - (Fullest, Random)
0.945222	Buckets - (Fullest, Lowest)
0.943694	Buckets - (Fullest, Highest)
0.919699	Mixed - TFT (y-1), 80%; (R[99, 100], 20%)
0.908562	Mixed - L(y-g) E(x-g) H(x-g), 80%; (2, 20%)
0.899511	Mixed - L(x) E(x) H(y-g), 80%; (100, 20%)
0.864914	TFT - Low(x) Equal(x) High(y-g)
0.863831	TFT - Low(x) Equal(x-2g) High(y-g)
0.823397	Mixed - L(y-g) E(x-g) H(x-g), 80%; (100, 20%)
0.822670	Random [99, 100]
0.820128	Always 99
0.818674	Always 100
0.817728	Zeuthen Strategy - Positive
0.803646	Q Learn - alpha= 0.2, discount= 0.9
0.801725	Q Learn - alpha= 0.5, discount= 0.0
0.801380	Q Learn - alpha= 0.2, discount= 0.0
0.800006	Q Learn - alpha= 0.5, discount= 0.9
0.798992	TFT - Low(x-2g) Equal(x) High(y-g)
0.798681	TFT - Low(x-2g) Equal(x-2g) High(y-g)
0.798417	Always 2
0.798402	Q Learn - alpha= 0.8, discount= 0.9
0.798277	Mixed - L(x) E(x) H(y-g), 80%; (2, 20%)
0.797728	Q Learn - alpha= 0.8, discount= 0.0
0.758573	Mixed - L(x) E(x) H(y-g), 80%; (100, 10%); (2, 10%)
0.751044	Mixed - L(y-g) E(x-g) H(x-g), 80%; (100, 10%); (2, 10%)
0.741721	Always 51
0.521901	Zeuthen Strategy - Negative
0.518840	Random [2, 100]

Furthermore, the choice of the learning rate seems to make very little difference: for each of the four metrics, all three Q-learning based strategies show similar performance, and hence end up ranked adjacently or almost adjacently.

Another interesting result is the performance of Zeuthen-based strategies. ITD as a strategic encounter is not of a negotiation nature, hence we have been criticized for even considering Zeuthen-like strategies as legitimate contenders in our tournament. However, excellent performance of the Zeuthen strategy with positive concessions only (at least w.r.t. the “bottom line” metric U_1) validates our approach. Interestingly enough, the same strategy does not perform particularly well w.r.t. metric U_3 . It is worth noting, however, that the only three strategies that outperform Zeuthen-positive with respect to U_1 perform similarly to Zeuthen-positive with respect to U_3 . Those strategies perform only slightly better than Zeuthen and way below the best performers with respect to U_3 , namely, the bucket-based, simplistic TFT-based, and simple-trend-based strategies.

Finally, given the performance of Zeuthen-negative (the variant allowing negative “concessions”) with respect to all metrics, it appears that “enticing” the opponent to behave differently indeed does not work when the “concessions” are not true concessions. Intuitively, this makes a perfect sense;

our simulation results provide an experimental validation of that common-sense intuition.

Due to space constraints, further analysis of our tournament results is left for the future work.

7. Summary and Future Work

We study the *Iterated Traveler's Dilemma* two-player game by designing, implementing and analyzing a round robin tournament with 38 distinct participating strategies. Our detailed analysis of the performance of various strategies with respect to several different metrics has corroborated that, for a game whose structure is far from zero-sum, the traditional game-theoretic notions of rationality and optimality may turn out to be rather unsatisfactory. Our investigations raise several interesting questions, among which we are particularly keen to further investigate the following:

(i) To what extent simple models of reinforcement learning, such as Q-learning, can be really expected to help performance?

(ii) To what extent complex models of the other agent really help an agent increase its payoff in the repeated play?

(iii) Why are performances of various TFT-based strategies so broadly different from each other? This opens up interesting questions from meta-learning [19, 20] and meta-reasoning standpoints: how can one design TFT-based strategies that are likely to do well across tournaments (that is, choices of opponents) and across performance metrics.

(iv) What effects on strategies and their performance would an adjustment in the bonus/malus have? For prior research on how human behavior changes with a change in bonus/malus, see [7] and [12].

In our future work, in addition to more detailed analysis of the existing strategies and study of some new ones, we plan to pursue a systematic comparative analysis of how groups of closely related strategies perform against each other when viewed as *teams*. We also plan to further investigate other notions of game equilibria, and try to determine which such notions adequately capture what our intuition would tell us constitutes good ways of playing the iterated TD and other 'far-from-zero-sum' two-player games.

References

- [1] J. S. Rosenschein and G. Zlotkin, *Rules of encounter: designing conventions for automated negotiation among computers*. MIT Press, 1994.
- [2] S. Parsons and M. Wooldridge, "Game theory and decision theory in Multi-Agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 5, pp. 243–254, 2002.
- [3] M. Wooldridge, *An Introduction to MultiAgent Systems*. John Wiley and Sons, 2009.
- [4] R. Axelrod, "Effective choice in the prisoner's dilemma," *Journal of Conflict Resolution*, vol. 24, no. 1, pp. 3–25, Mar. 1980.
- [5] —, "The evolution of cooperation," *Science*, vol. 211, no. 4489, pp. 1390–1396, 1981.
- [6] T. Becker, M. Carter, and J. Naeve, "Experts playing the traveler's dilemma," Department of Economics, University of Hohenheim, Germany, Tech. Rep., Jan. 2005.
- [7] C. M. Capra, J. K. Goeree, R. Gómez, and C. A. Holt, "Anomalous behavior in a traveler's dilemma?" *The American Economic Review*, vol. 89, no. 3, pp. 678–690, Jun. 1999.
- [8] M. Pace, "How a genetic algorithm learns to play traveler's dilemma by choosing dominated strategies to achieve greater payoffs," in *Proc. of the 5th international conference on Computational Intelligence and Games*, 2009, pp. 194–200.
- [9] K. Basu, "The traveler's dilemma: Paradoxes of rationality in game theory," *The American Economic Review*, vol. 84, no. 2, pp. 391–395, May 1994.
- [10] J. V. Neumann and O. Morgenstern, *Theory of games and economic behavior*. Princeton Univ. Press, 1944.
- [11] K. Basu, "The traveler's dilemma," *Scientific American Magazine*, Jun. 2007.
- [12] J. K. Goeree and C. A. Holt, "Ten little treasures of game theory and ten intuitive contradictions," *The American Economic Review*, vol. 91, no. 5, pp. 1402–1422, Dec. 2001.
- [13] R. Axelrod, *The evolution of cooperation*. Basic Books, 2006.
- [14] P. Dasler and P. Tosić, "The iterated traveler's dilemma: Finding good strategies in games with 'bad' structure: Preliminary results and analysis," in *8th Euro. Workshop on Multi-Agent Systems, EUMAS'10*, Dec 2010.
- [15] A. Rapoport and A. M. Chammah, *Prisoner's Dilemma*. Univ. of Michigan Press, Dec. 1965.
- [16] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
- [17] M. L. Littman, "Friend-or-Foe q-learning in General-Sum games," in *Proc. of the 18th Int'l Conf. on Machine Learning*. Morgan Kaufmann Publishers Inc., 2001, pp. 322–328.
- [18] F. F. Zeuthen, *Problems of monopoly and economic warfare / by F. Zeuthen ; with a preface by Joseph A. Schumpeter*. London: Routledge and K. Paul, 1967, first published 1930 by George Routledge & Sons Ltd.
- [19] R. Sun, "Meta-Learning processes in Multi-Agent systems," *Proceedings of Intelligent Agent Technology*, pp. 210–219, 2001.
- [20] P. T. Tosić and R. Vilalta, "A unified framework for reinforcement learning, co-learning and meta-learning how to coordinate in collaborative multi-agent systems," *Procedia Computer Science*, vol. 1, no. 1, pp. 2211–2220, May 2010.

