RISC

RESEARCH INSTITUTE FOR SYMBOLIC COMPUTATION

JOHANNES KEPLER UNIVERSITY LINZ

9th International Symposium on Symbolic Computation in Software Science (SCSS 2021), short and work-in-progress papers

Temur Kutsia (editor)

August 2021

RISC Report Series No. 21-16

ISSN: 2791-4267 (online)

Available at https://doi.org/10.35011/risc.21-16

JOHANNES KEPLER UNIVERSITY LINZ
Altenberger Str. 69
4040 Linz, Austria
www.jku.at
DVR 0093696

Ninth International Symposium on

SYMBOLIC COMPUTATION IN SOFTWARE SCIENCE

SCSS 2021

SHORT AND WORK-IN-PROGRESS PAPERS

Temur Kutsia (editor)

September 8–10, 2021
Research Institute for Symbolic Computation
Johannes Kepler University Linz, Austria

Preface

This collection contains short and work-in-progress papers presented at the 9th International Symposium on Symbolic Computation in Software Science, SCSS 2021.

Symbolic Computation is the science of computing with symbolic objects (terms, formulae, programs, representations of algebraic objects etc.). Powerful algorithms have been developed during the past decades for the major subareas of symbolic computation: computer algebra and computational logic. These algorithms and methods are successfully applied in various fields, including software science, which covers a broad range of topics about software construction and analysis.

The purpose of the International Symposium on Symbolic Computation in Software Science – SCSS is to promote research on theoretical and practical aspects of symbolic computation in software science, combined with modern artificial intelligence techniques.

SCSS 2021 was organized at the Research Institute for Symbolic Computation (RISC) of the Johannes Kepler University Linz. Due to the COVID-19 pandemic, the symposium was held completely online.

The SCSS program featured a keynote talk by Bruno Buchberger (Johannes Kepler University Linz) and three invited talks given by Tateaki Sasaki (University of Tsukuba), Martina Seidl (Johannes Kepler University Linz), and Stephen M. Watt (University of Waterloo). The symposium received 25 submissions with contributing authors from 17 countries. These submissions have been divided into two tracks: 16 in the category of regular papers and tool/dataset descriptions, and nine in the category of short and work-in-progress papers. Twenty PC members and 15 external reviewers took part in the refereeing process, after which 10 regular / dataset papers and 9 short contributions have been accepted for the presentation at the symposium. The accepted regular and tool/dataset papers have been included in a volume of Electronic Proceedings in Theoretical Computer Science (EPTCS). This technical report contains short and work-in-progress papers presented at the symposium. In addition to the main program, a special session on Computer Algebra and Computational Logic has been held.

I thank the keynote and invited speakers, Program Committee, and all the participants for contributing to the success of the symposium. The EasyChair conference management system has been a very useful tool for PC work, and the technical support team at RISC greatly contributed to running the conference smoothly.

Temur Kutsia
Program Chair of SCSS 2021

Conference Information

General Chairs

Adel Bouhoula (Arabian Gulf University, Bahrain)
Tetsuo Ida (University of Tsukuba, Japan)

Program Chair

Temur Kutsia (RISC, Johannes Kepler University Linz, Austria)

Program Committee

David Cerna (Czech Academy of Sciences, Czechia, and Johannes Kepler University Linz, Austria)
Changbo Chen (Chinese Academy of Sciences, China)
Rachid Echahed (CNRS, Grenoble, France)
Seyed Hossein Haeri (UC Louvain, Belgium)
Mohamed-Bécha Kaâniche (Sup'Com, Carthage University, Tunisia)
Cezary Kaliszyk (University of Innsbruck, Austria)
Yukiyoshi Kameyama (University of Tsukuba, Japan)
Michael Kohlhase (University of Erlangen-Nuremberg, Germany)
Laura Kovács (Vienna University of Technology, Austria)
Zied Lachiri (ENIT, University of Tunis El Manar, Tunisia)
Christopher Lynch (Clarkson University, USA)
Mircea Marin (West University of Timisoara, Romania)
Yasuhiko Minamide (Tokyo Institute of Technology, Japan)
Yoshihiro Mizoguchi (Kyushu University, Japan)
Julien Narboux (Strasbourg University, France)
Michaël Rusinowitch (INRIA, France)
Wolfgang Schreiner (Johannes Kepler University Linz, Austria)
Sofiane Tahar (Concordia University, Canada)
Dongming Wang (CNRS, Paris, France)

Organization

Temur Kutsia
Cleopatra Pau
Werner Danielczyk-Landerl
Ralf Wahner

Table of Contents

Incremental One-Class Framework for Human Face Detection

Mohamed Khalil Ben Salah
Digital Security Research Lab
Higher School of Communication of Tunis
University of Carthage
Tunis, Tunisia
MohamedKhalil.BenSalah@insat.u-carthage.tn

Takoua Kefi-Fatteh
Digital Security Research Lab
Higher School of Communication of Tunis
University of Carthage
Tunis, Tunisia
takoua.kefi@supcom.tn

Adel Bouhoula
Department of Next-Generation Computing
23 College of Graduate Studies
Arabian Gulf University
Kingdom of Bahrain
a.bouhoula@agu.edu.bh

Abstract. Systems that rely on Face Detection have gained great importance ever, since large-scale databases of thousands of face images are collected from several sources. Thus, the use of an outperforming face detector becomes a challenging problem. Different classification frameworks have been studied and applied for face detection. However, such models involve large scale datasets, which requires huge memory and enormous amount of training time. Therefore, in this paper we investigate the performance of different feature extraction methods, then incrementally projecting data in low variance directions. Extensive tests on human faces, and comparative experiments need to be carried out to find the best combination for accurate human face detection framework.

1 Introduction

For the past few decades, Human Face Detection (HFD) has been a hot topic in machine learning and computer vision. It has been considered as a fundamental step for face understanding and analysis tasks, particularly for cognitive vision applications.

In fact, face detection is a key stone in all authentications and security systems. It has brought many benefits to users, such as, improving security and automated identification. For example, thanks to HFD, transactions has become more secure than using credit. HFD has an impact in fighting crimes, by allowing the detection and identification of criminals within a crowd, and track them. Besides, HFD is an important phase to be used in airports to check travelers' identities by using face-scanning. Thus we can easily and reliably control access to sensitive areas.

The main contribution of our research project is to build a novel framework for HFD based on incremental Covariance-guided One-class Support Vector Machine (iCOSVM) for data classification. The used classifier takes advantage of the incremental strategy while emphasizing the low variance directions in order to minimize target dispersion and improve classification performance. This phase is preceded by a feature extraction process. Besides, it is not trivial to find the most convenient model for feature extraction. Thus, we intend to study the effectiveness of Histogram of Oriented Gradients (HOG) and Local Binary Pattern (LBP) for human features extraction.

T. Kutsia (Ed.): Symbolic Computation
in Software Science (SCSS'21)
Short and work-in-progress papers, pp. 1–5

The remaining paper is structured as follows: Section 2 presents a brief literature review of the different proposed methods for face detection on single images. Section 3 discusses the phases of our framework. An experimental evaluation is conducted on MIT-CBCL database and presented in Section 4. Finally, conclusion and future directions are outlined in Section 5.

2 Related Works

HFD is one the research focuses in computer vision. It has been applied to solve several critical problems related to face analysis such as face analysis, such as, face verification and recognition, target tracking, etc... Works on automated HFD can be traced back to more than 50 years ago [9]. Yet, researchers have successfully worked for the improvement of the proposed algorithms and techniques.

In [8], it has been shown that applying HOG with SVM classification performance achieves the highest results in terms of accuracy, precision and sensitivity. These results were strengthened in [6] where different types of images were used to compare HOG against other state-of-the-art feature extractors. HOG was proven to be fast and accurate in detecting human faces in images.

Another face detector scheme was introduced[1]. The proposed model is based on LBP feature extraction method. The fundamental contribution of the method is the relative position of of the threshold pixel to the coding pixel of the original LBP are spaced on a circle or an ellipse that is centered at a threshold pixel. However, there are no similar constraints on the threshold pixel and coding pixel of the newly proposed method.

A fusion of HOG and LBP was successfully performed in [5] and [7]. In spite of the challenging conditions from one used dataset to another, the method showed a high efficiency.

Once useful features are extracted from face images, a classification algorithm have to be applied. For instance, we can use Convolutional Neural Network (CNN), Artificial Neural Network (ANN), Random Forests, Support vector Machines (SVM) and many other algorithms. Yet, SVM has become widely and successfully used for human face detection[3]. Thanks to the structural risk minimization, the expected generalization error is limited.

3 Face Detection Framework

Our model is composed of two main parts : (1) feature extraction and (2) data classification. Each phase will be described in the following subsections.

3.1 Feature Extraction

Due to enlargement of image size, taking out the most relevant information has become crucial phase in computer vision. Feature Extraction means extracting various features of the image with the aim of ignoring noises that may lead to miss-classification. Many Feature Extraction techniques were used in different field, among these methods we can cite HOG and LBP.

HOG feature descriptor is a fast and effective feature extraction method . As presented in [2], HOG method starts with dividing image into small cells, 4×4 for example. These cells are used to compute the edge directions where each pixels of the considered region is assumed to a variable for edge orientation and associated with gradient element. To get the histogram, we need to calculate

horizontal and vertical gradients through filtering image using kernel filters. Then, the magnitude and the direction of the gradients are calculated. The next step in HOG is to create histogram from these gradients, using the orientation and direction of each cell. Subsequently, after grouping the cells together into connected blocks and normalizing them, we have to concatenate all histogram values to obtain HOG features.

From another side, LBP is a powerful feature extractor, used as local structures descriptor. It dictates that every block of 3×3 window is processed separately, and every pixel of an image is labeled using a threshold of center of matrix block compared to 8 neighbors. In matrix of 3×3 window, every neighbor of threshold will take a binary value, it takes 1 if his values equals or exceeds than the central value. Otherwise, it takes 0. Due to problems related to scale structures, LBP operator was extended to take different size of neighborhoods, assigning for every block a circle with a different radius. Based on new generated images, we extract the histogram of each region then we create a new and bigger histogram through concatenating each histogram of each grid.

3.2 Classification

In our proposed framework we used for the classification phase the incremental Covariance guided One-Class Support Vector Machine (iCOSVM). The main idea of iCOSVM is to take into consideration the covariance matrix Δ and add it to the dual optimization problem. A key value used $\eta \in [0,1]$ to get the balance between the contributions of the kernel matrix K and the covariance matrix to the objective function, so the optimization problem will be:

$$\min_{\alpha} W(\alpha, b) = \frac{1}{2}\alpha^T(\eta K + (1-\eta)\Delta)\alpha - b(1 - \sum_{i=1}^{N} \alpha_i) \tag{1}$$

$$s.t. \quad 0 \leq \alpha_i \leq \frac{1}{vN} = C, \quad \sum_{i=1}^{N} \alpha_i = 1 \tag{2}$$

where, $\Delta = K(I - 1_N)K^T$, $v \in (0,1]$ represent a key that controls the fraction of outliers, C penalty weight, $K(x_i, x_j) =< \phi(x_i), \phi(x_j) >, \forall i, j \in \{1, 2, ..., N\}$ is a kernel matrix and α are the Lagrange multipliers.

Both the kernel matrix **K** and the covariance matrix Δ are positive definite. Therefore, the proposed method still results in a convex optimization problem. Thus, the solution to this optimization problem will have one global optimum solution and can be solved efficiently using a mathematical method. Karush-Kuhn-Tucker (KKT) conditions are among the most important theoretical optimization methods. The key of our method is to construct a solution recursively, by adding one point at a time, and retain the KKT conditions on all previously acquired data.

4 Experimental Results

To evaluate the effectiveness of our framework, we used the MIT-CBCL dataset. It consists of 2901 images of faces and 28121 images of non-faces of size 19×19 pixels. The dataset is divided into a training set containing 2429 faces and 4548 non-faces and a test set containing 472 faces and 23573 non-faces. The performance of the iCOSVM was compared to SVM-based methods, namely, the the incremental OSVM and the incremental Support Vector Data Description. For the kernelization,

Feature extraction	Classifier	F-score	Recall	FNR (%)
HOG	iSVDD	0.658	0.490	51.00
	iOSVM	1.00	1.00	0.00
	iCOSVM	1.00	1.00	0.00
LBP	iSVDD	0.875	0.777	22.3
	iOSVM	0.992	1.00	0.00
	iCOSVM	0.998	1.00	0.00
HOG-LBP	iSVDD	0.7937	0.6580	34.2
	iOSVM	0.9901	1.00	0.00
	iCOSVM	1.00	1.00	0.00

Table 1: Comparison of Feature extraction methods using different classifiers

the RBF kernel is used. To make sure that our results are not coincidental or overoptimistic, we used a 10-fold cross-validation process. Besides, as performance metrics, we computed the F-score the Hit Rate, and the False Negative Rate (FNR).

Table 1 contains the average F-Score and Hit-Rate values obtained for each classifier with different face extraction methods, i.e. HOG, LBP and HOG&LBP, on the human faces datasets. As we can see, iCOSVM provides signicantly better results by averaging over 10 different models. This strengthens our claim that by emphasizing the low variance directions with the incorporation of the covariance matrix, iCOSVM can indeed lead to improved performance. The iSVDD gives almost the worst metrics values, as SVDD and its derivatives are constructed to give the better separation.

As far as the feature extraction methods are considered, we can notice that LBP is the most convenient feature extraction method to be used with iSVDD, since it improves the F-score and the Recall values, and reduces the FNR. However, the HOG-LBP fusion does not enhance the detection accuracy of iOSVM and iCOSVM, and the best performance measures are obtained with HOG feature extraction method, which needs to be further investigated.

In fact, each descriptor processes in a different manner and produces a different output, even though they are both based on the gradient information. HOG proves very effective when it comes to capturing the outlines and angles. LBP on the other hand uses 8 directions for each pixel and generates a 256 bin histogram representing the pixel distribution of the input image. The concatenation of the LBP and HOG generated vectors is a solution to form a single feature pool, but a curse of dimensionality appears and needs to be resolved.

5 Conclusion

In this paper, we have shown that the incremental Covariance-guided One-class Support Vector Machine (iCOSVM) is a good choice for face detection. Besides, we tested different feature extraction methods to analyze their effects on the classifier performance. It is assumed that HOG method is the most convenient for iOSVM and iCOSVM. Whereas, iSVDD performs better when combined with LBP.

The proposed framework can be used as the first component of a reliable surveillance system,used to identify faces and track targets and improve national security. Besides, to ensure that our system performs as expected, we intend to use reliable machine learning techniques [4].

References

[1] Jie chun Chen, Jun Wang, Li ping Zhao & Jin He (2020): *Branch-structured detector for fast face detection using asymmetric LBP features*. Signal, Image and Video Processing 14, pp. 1699–1706, doi:10.1007/s11760-020-01710-7.

[2] Navneet Dalal & Bill Triggs (2005): *Histograms of Oriented Gradients for Human Detection*. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*, IEEE Computer Society, pp. 886–893, doi:10.1109/CVPR.2005.177. Available at `https://doi.org/10.1109/CVPR.2005.177`.

[3] Mina Farmanbar & Önsen Toygar (2017): *Spoof detection on face and palmprint biometrics*. Signal, Image and Video Processing 11, pp. 1253–1260, doi:10.1007/s11760-017-1082-y.

[4] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri & Martin Vechev (2018): *AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation*, pp. 3–18. doi:10.1109/SP.2018.00058.

[5] Bayezid Islam, Firoz Mahmud & Arfat Hossain (2018): *High Performance Facial Expression Recognition System Using Facial Region Segmentation, Fusion of HOG & LBP Features and Multiclass SVM*. *2018 10th International Conference on Electrical and Computer Engineering (ICECE)*, pp. 42–45, doi:10.1109/ICECE.2018.8636780.

[6] Mohammed G. Mohammed & Amera I. Melhum (2020): *Implementation of HOG Feature Extraction with Tuned Parameters for Human Face Detection*. International Journal of Machine Learning and Computing 10, pp. 654–661, doi:10.18178/ijmlc.2020.10.5.987.

[7] M Santosh & A Sharma (2020): *Fusion of multi representation and multi descriptors for facial expression recognition*. 1057, pp. 1–24, doi:doi:10.1088/1757-899X/1057/1/012093.

[8] K PNithish Sriman, P Raj Kumar, A Naveen & R Saravana Kumar (2021): *Comparison of Paul Viola – Michael Jones algorithm and HOG algorithm for Face Detection*. IOP Conference Series: Materials Science and Engineering 1084(1), p. 012014, doi:10.1088/1757-899x/1084/1/012014.

[9] Stefanos Zafeiriou, Cha Zhang & Zhengyou Zhang (2015): *A survey on face detection in the wild: Past, present and future*. Comput. Vis. Image Underst. 138, pp. 1–24, doi:10.1016/j.cviu.2015.03.015.

Q-learning and MCTS techniques for improving an algorithm to compute discrete vector fields on finite topological spaces*

Julián Cuevas-Rozo
National University of Colombia
University of La Rioja, Spain
jucuevas@unirioja.es

Jose Divasón
University of La Rioja, Spain
jose.divason@unirioja.es

Laureano Lambán
University of La Rioja, Spain
lalamban@unirioja.es

Ana Romero
University of La Rioja, Spain
ana.romero@unirioja.es

Abstract. In this work we present an ongoing project on the improving of a previous symbolic computation algorithm computing discrete vector fields on finite topological spaces. To this aim, we consider different strategies to choose each one of the possible vectors at each step of the algorithm and we apply some reinforcement learning techniques.

1 Introduction

A *finite topological space* (or simply a *finite space*) is a topological space with finitely many points. Finite spaces and finite preordered sets are basically the same objects considered from different perspectives [1]. The study of homotopical invariants can be restricted to the class of finite T_0-spaces, since any finite topological space is known to be homotopy equivalent to a finite T_0-space [17]. Finite T_0-spaces correspond to finite partially ordered sets or posets. Given a finite T_0-space X, the incidence matrix corresponding to the order relation of its associated poset and the adjacency matrix of the Hasse diagram $\mathscr{H}(X)$ are called *topogenous matrix* and *Stong matrix* of the space, respectively.

In [6], some algorithms and programs to compute topological invariants of finite spaces have been presented, which are based on new developed constructive versions of theoretical results [4, 11]. In particular, a new module for the Kenzo symbolic computation system [7] was developed allowing the computation of homology groups of h-regular spaces without constructing the order complex associated with the poset, by defining a chain complex directly from the finite space. Moreover, our new Kenzo module includes an algorithm to determine homologically admissible Morse matchings on finite spaces, in order to use the Minian's version of discrete Morse theory. A *Morse matching* M on a finite space X is a set of edges of its Hasse diagram where no edges share a common vertex and such that the directed graph $\mathscr{H}(X)$, modified by reversing the orientation of the edges in M, is acyclic. If a Morse matching satisfies the property of being *homologically admissible* (see [11] for details), the homology of X can be determined by means of a chain complex smaller than the canonical one associated to the finite space generated by the *critical* points of the matching, which are the elements of X that are not incident to any edge in M. A Morse matching on a finite space can also be seen as a *discrete vector field* [14] on the associated chain complex. Then, the longer the Morse matching is, the smaller the number of generators we need to describe a chain complex to compute homology by means of the critical complex is.

*Partially supported by the Spanish Ministry of Science, Innovation and Universities, projects MTM2017-88804-P and PID2020-116641GB-I00.

T. Kutsia (Ed.): Symbolic Computation
in Software Science (SCSS'21)
Short and work-in-progress papers, pp. 6–10

In this work, we try to improve the algorithms presented in [6] by studying different strategies and methods to maximize the size of a discrete vector field. To this aim, we first consider 25 different strategies to choose which vector should be added in each step of the Morse matching algorithm. We also try some reinforcement learning techniques such as Q-learning and MCTS. The code and the experiments of our work can be found at:

`https://github.com/jodivaso/Q_Learning_MCTS_DVF`

2 Strategies for computing discrete vector fields on finite spaces

In our algorithm for computing Morse matchings on finite topological spaces presented in [6], we search for homologically admissible edges such that the set of such edges does not contain cycles in the modified Hasse diagram of the poset associated to the finite space. In this searching, we go through the columns and the rows of the Stong matrix in ascending order (an edge in the Morse matching, or a vector in the associated chain complex, corresponds to an element equal to 1 in the Stong matrix). The algorithm finishes when it is not possible to add a new edge to the matching satisfying the desired property, so that the corresponding discrete vector field is maximal. However, a different vector field could exist with a bigger size (the problem of finding a vector field of maximal length is difficult for big spaces).

In order to improve our algorithm and try to find Morse matchings on a finite space as big as possible, we have decided to consider different *strategies* to sort rows and columns of the Stong matrix, and compare them analyzing if there exist remarkable differences between the size of the discrete vector fields obtained in each case. We have considered five different strategies to sort a list of elements in a finite T_0-space: *:standard* (in ascending order, the one used in our initial implementation of the algorithm), *:indegree* (sorted by the number of "head ends" adjacent to each vertex), *:reverse-indegree* (reverse order to the previous one), *:outdegree* (sorted by the number of "tail ends" adjacent to a vertex), and *:reverse-outdegree* (reverse order to the previous one).

The strategies considered above have been tested in random finite h-regular spaces of different sizes between 10 and 100 elements. On each one of these spaces, we have computed 25 discrete vector fields in Kenzo by using the five strategies in order to sort the column and row indexes of the Stong matrices. In these experiments we have observed that the strategy *:outdegree - :reverse-outdegree* (that is, considering outdegree order for column indexes and reverse-outdegree for row indexes) predominates over the others, obtaining the maximum of all strategies in 81.5% of all spaces. The second best strategy is *:outdegree - :indegree* (maximum in 68% of all cases) and the third one is *:outdegree - :standard* (65%). We can also observe that there is not a unique strategy which performs well in all cases and, moreover, in most of the finite spaces we have considered, the different strategies have produced discrete vector fields with low difference between their lengths.

3 Application of Q-learning algorithm

After trying the different strategies to compute discrete vector fields on finite topological spaces introduced in the previous section and seeing that there is not a unique strategy which performs the best in all spaces, we have tried to improve our symbolic computation algorithm by means of *machine learning* methods. Due to the type of problem we are studying, we have focused on the so-called *reinforcement learning*. Machine learning methods have already been applied in other computer algebra problems [3, 13, 16].

Reinforcement learning is a type of machine learning technique where the key point of is to establish a suitable reward system. That is, when the process has reached a concrete state and it has to select a possible action in order to continue, a reward is applied, which depends on the profit obtained after the corresponding change of state. In particular, we use the *Q-learning* algorithm [18], which is a particular case of reinforcement learning. This algorithm requires the definition of *states* (the possible situations which can be encountered), *actions* (transitions from one state to another state) and (positive or negative) *rewards* received after each transition. More concretely, the Q-learning algorithm is based on the construction of a *Q-table*, which is a matrix where we have a row for every state and a column for every action. It is first initialized to 0, and then values are updated after training, taking into account the rewards obtained. The process of *training* is done by iterating a sequence of actions starting from the initial state, combining *exploration* (choosing a random action) and *exploitation* (choosing actions based on already learned Q-values). For each state (each row in the table), the best action is supposed to be the one with the highest value in the corresponding column of the Q-table. See [18] for more details.

In our problem, we have chosen to use the Python library for reinforcement learning *Gym* [12]. We construct a new class called `DVFMatrixEnv` which implements the interface `gym.Env`, which represents a learning `environment`. This interface requires the definition of a set of `states` and a set of `actions`, and the implementation of four methods: `init`, `reset`, `render` and `step`. The most important method is `step`, which, for an input pair of state and action, returns a new state, a reward and a boolean variable called `done` which determines if the problem ends or not after applying the action.

The input data to construct an environment of type `DVFMatrixEnv` is the Stong matrix of the finite T_0-space where we want to compute a discrete vector field. The elements of the set of actions are the possible vectors we can select in the space. Then, a state represents a set of possible vectors (some of the sets are not valid, but this will be specified in the algorithm by means of a negative reward). The number of possible states is $2^{\text{numberOfActions}}$. Both actions and states are represented as numbers. The initial state is 0. Once we have defined our environment, we perform the training of the Q-learning algorithm with a given number of repetitions. After a high number of iterations, the Q-table allows one to define a good process to obtain a Morse matching as big as possible.

We have tested the method on some of the random examples studied in the previous section. In all the cases where the algorithm works, the Q-learning technique has provided us satisfactory results. That is, for each space, the vector field we obtained by using this approach has the same number of vectors as the best one obtained by applying the strategies given in Section 2. Nevertheless, the size of the Q-table results to be an evident problem to be treated. In fact, we have only tested satisfactorily the Q-learning technique in 13 spaces; for other spaces, the Q-table cannot be built. It is a common drawback of this type of machine learning and there exist some well-studied alternatives to avoid it, such as the one studied in the next section.

4 Monte Carlo tree search

Monte Carlo tree search (MCTS) is a heuristic search algorithm for some kinds of decision processes. It is commonly used in the field of game theory as a method to solve a game tree by analysing the most promising movements. In fact, MCTS has proved to be very competitive in deterministic games with large branching factors for many years. One of the most famous applications is the AlphaGo program, in which a neural network combined with MCTS beat a professional human Go player in 2015.

In our problem, we can see the different possibilities for selecting the vectors as a search tree: each node is a 1 of the Stong matrix that can be chosen in a concrete step; branches correspond to the sequence

of vectors that have been selected. The final goal is to get a branch as large as possible.

Essentially, MCTS is a smart search, since in each step it accumulates value estimates to guide towards highly rewarding trajectories in the search tree. In other words, MCTS focuses on nodes that are more promising, avoiding applying brute force algorithms (like minimax), which are unfeasible over big state spaces. More concretely, each MCTS iteration consists of four steps: selection (choose a promising node p that can be expanded), expansion (create one or several child nodes of p, one for each possible action at that point), simulation (choose one of those child nodes and apply random decisions until finish) and backpropagation (use the result of the simulation to update the information of the tree).

We have implemented a MCTS version based on *Gym*. To balance exploration *vs.* explotation, in the selection phase we employed the UCB formula (Upper Confidence Bound, see [2, 9]): $v_i + c\sqrt{\log(N)/n_i}$, where v_i is the value estimate of the node, N the number of visits of the parent node, n_i the number of visits of the node and c is the exploration hyperparameter, which theoretically is equal to $\sqrt{2}$ but can be empirically chosen depending on the problem to strengthen exploration or explotation. That is, the UCB-value of a node decreases in the number of visits (exploration) and increases in node's value (exploit). Thus, UCB permits to *sort* the nodes: nodes with a high UCB-value are more promising or need to be explored. In order to avoid prioritizing nodes that have never been visited (otherwise, a non-visited node has ∞ as UCB-value and will always have higher priority over an already visited one), we decide to give parent's UCB-value to a non-visited node. The simulation step consists of randomly choosing vectors until reaching a state in which no more admissible vectors can be added. The reward is the number of vectors that we reached (the depth of the branch). That information is backpropagated to the tree and the UCB-values of each node are updated. MCTS permits to handle big spaces, since there is no need to create a Q-table. It can give a result immediately and indeed, MCTS improves its result with more time. It is a decision process which applies random actions over the most promising nodes, and the information obtained from the simulation is used to update the list of the most promising nodes (based on the UCB-value). At the end, the best decision is the one with higher reward, in our case, the largest branch.

The algorithm essentially requires as input the Stong matrix of the finite space and the number of MCTS iterations (also named as playouts) that one wants to perform. Our first experiments show that MCTS obtains the same or more vectors than applying any of the 25 strategies and the Q-learning algorithm in 80% of all spaces, with no need to explore the whole tree and performing better than a pure iterated random strategy. Concretely, by using MCTS we obtain the same number of vectors in 42% of cases and more vectors in 37.5% of spaces.

5 Conclusions and further work

This work presents some improvements that we carried out over a symbolic computation algorithm for computing Morse matchings on finite topological spaces, concretely on how to select vectors to maximize the size of a discrete vector field. We first studied 25 different strategies to choose which vector should be added. We also tried techniques that are usually employed in reinforcement learning contexts, such as Q-learning and MCTS. The initial experiments show promising results. We plan to extend this work by employing deep Q-learning, which uses neural networks to approximate the Q-table, and also combine it with MCTS. In addition, further work is necessary to optimize the implementation of the algorithms (for instance, to avoid recalculations and to exploit parallelism) and tune appropriately the parameters, hyperparameters and rewards to get a fast and good result.

References

[1] P. Alexandroff (1937): *Diskrete Rume*. *Mat. Sb. (N.S.)* 2, pp. 501–518.

[2] Peter Auer, Nicolo Cesa-Bianchi & Paul Fischer (2002): *Finite-time analysis of the multiarmed bandit problem*. *Machine learning* 47(2), pp. 235–256, doi:10.1023/A:1013689704352.

[3] Y. Bengio, A. Lodi & A. Prouvost (2020): *Machine Learning for Combinatorial Optimization: a Methodological Tour dHorizon*. `https://arxiv.org/pdf/1811.06128.pdf`.

[4] N. Cianci & M. Ottina (2017): *A new spectral sequence for homology of posets*. *Topology and its Applications* 217, pp. 1–19, doi:10.1016/j.topol.2016.12.001.

[5] J. Cuevas-Rozo (2020): *Finite topological spaces in Kenzo*. `https://github.com/jcuevas-rozo/finite-topological-spaces`.

[6] J. Cuevas-Rozo, L. Lambn, A. Romero & H. Sarria (2020): *Effective homological computations on finite topological spaces*. *Applicable Algebra in Engineering, Communication and Computing*, doi:10.1007/s00200-020-00462-8. In press.

[7] X. Dousson, J. Rubio, F. Sergeraert & Y. Siret (1999): *The Kenzo program*. Institut Fourier, Grenoble. `http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/`.

[8] A. I. Khan & S. Al-Habsi (2020): *Machine Learning in Computer Vision*. *Procedia Computer Science* 167, pp. 1444–1451, doi:10.1016/j.procs.2020.03.355.

[9] Levente Kocsis & Csaba Szepesvári (2006): *Bandit Based Monte-Carlo Planning*. In Johannes Fürnkranz, Tobias Scheffer & Myra Spiliopoulou, editors: *Machine Learning: ECML 2006*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 282–293, doi:10.1007/11871842_29.

[10] A. Maier, C. Syben, T. Lasser & C. Riess (2019): *A gentle introduction to deep learning in medical image processing*. *Zeitschrift fr Medizinische Physik* 29(2), pp. 86–101, doi:10.1016/j.zemedi.2018.12.003.

[11] G. Minian (2012): *Some remarks on Morse theory for posets, homological Morse theory and finite manifolds*. *Topology and its Applications* 159(12), pp. 2860–2869, doi:10.1016/j.topol.2012.05.027.

[12] OpenAI (2016): *Gym*. `https://gym.openai.com/`.

[13] D. Peifer, M. Stillman & D. Halpern-Leistner (2020): *Learning Selection Strategies in Buchberger's Algorithm*. `https://arxiv.org/pdf/2005.01917.pdf`.

[14] A. Romero & F. Sergeraert (2010): *Discrete Vector Fields and Fundamental Algebraic Topology*. `https://arxiv.org/pdf/1005.5685.pdf`.

[15] D. Silver, A. Huang & C. Maddison (2016): *Mastering the game of Go with deep neural networks and tree search*. *Nature* 529, pp. 484–489, doi:10.1038/nature16961.

[16] L. R. Silverstein (2019): *Probability and Machine Learning in Combinatorial Commutative Algebra*. Ph.D. thesis, University of California Davis.

[17] R. E. Stong (1966): *Finite topological spaces*. *Trans. Amer. Math. Soc.* 123(2), pp. 325–340, doi:10.1090/S0002-9947-1966-0195042-2.

[18] C. J. C. H. Watkins (1989): *Learning from Delayed Rewards*. Ph.D. thesis, Cambridge University.

PρLog: a system for rule-based programming

Besik Dundua
Kutaisi International University
Kutaisi, Georgia
Ilia Vekua Institute of Applied Mathematics, Ivane Javakhishvili Tbilisi State University
Tbilisi, Georgia
bdundua@gmail.com

1 Brief overview

PρLog [11] is a rule-based system that supports programming with individual, hedge, function and context variables. It extends Prolog with rule-based programming capabilities to manipulate sequences of terms, also known as *hedges*. The four kinds of variables help to traverse tree forms of expressions both in horizontal and vertical directions, in one or more steps. It facilitates to have expressive pattern matching that often helps to write short and intuitive code.

Another important feature of PρLog is the use of strategies. They provide a mechanism to control complex rule-based computations in a highly declarative way. With the help of strategies, the user can combine simpler transformation rules into more complex ones. In this way, PρLog conveniently combines the whole Prolog power with rule-based strategic programming features.

PρLog is based on ρLog calculus [17], where the inference system basically is the SLDNF-resolution with normal logic program semantics [14]. It has been successfully used in the extraction of frequent patterns from data mining workflows [20], XML transformation and web reasoning [7], modeling of rewriting strategies [9] and access control policies [18], etc.

The ρLog calculus has been influenced by the ρ-calculus [4, 5], which, in itself, is a foundation for the rule-based programming system ELAN [2]. There are some other languages for programming by rules, such as, e.g., ASF-SDF [3], CHR [12], Maude [6], Stratego [21], Tom [1]. The ρLog calculus and, consequently, PρLog differs from them, first of all, by its pattern matching capabilities. Besides, it adopts logic programming semantics (clauses are first class concepts, rules/strategies are expressed as clauses) and makes a heavy use of strategies to control transformations. Earlier works about ρLog and its implementation in Mathematica include [16, 19, 15].

PρLog is available at `https://www.risc.jku.at/people/tkutsia/software/prholog`.

2 The PρLog language

We write PρLog constructs in `typewriter` font. They are *terms* (including a special kind of terms, called *contexts*) and hedges. These objects are constructed from function symbols without a fixed arity (so called unranked or variadic function symbols), a special constant `hole` (called the *hole*), and individual, functional, context and hedge variables. These variables are denoted by the identifiers whose names start respectively with `i_`, `f_`, `c_`, and `h_` (e.g., `i_X`, `f_X`, `c_X`, `h_X`). Terms `t` and hedges `h` are constructed in a standard way:

$$\mathtt{t} ::= \mathtt{hole} \mid \mathtt{i\_X} \mid \mathtt{f}(\mathtt{h}) \mid \mathtt{f\_X}(\mathtt{h}) \mid \mathtt{c\_X}(\mathtt{t}) \quad \text{(terms)}$$
$$\mathtt{h} ::= \mathtt{eps} \mid \mathtt{t} \mid \mathtt{h\_X} \mid (\mathtt{h},\mathtt{h}) \quad \text{(hedges)}$$

T. Kutsia (Ed.): Symbolic Computation
in Software Science (SCSS'21)
Short and work-in-progress papers, pp. 11–16

where f is a function symbol and eps stands for the empty hedge and is omitted whenever it appears as a subhedge of another hedge. A *context* is a term with a single occurrence of the hole constant. Application of a context C to a term t is a term derived by replacing the hole in C with t. For instance, applying f(i_X,g(i_Y,hole),a) to g(b,hole) gives another context f(i_X,g(i_Y,g(b,hole)),a), while applying it to g(b,c) gives a non-context term f(i_X,g(i_Y,g(b,c)),a).

A *substitution* is a mapping from individual variables to hole-free terms, from hedge variables to hole-free hedges, from function variables to function variables and symbols, and from context variables to contexts, such that all but finitely many individual, sequence, and function variables are mapped to themselves, and all but finitely many context variables are mapped to themselves applied to the hole. This mapping can be extended to terms and hedges in the standard way. For instance, for a substitution σ ={c_Ctx↦f(hole), i_Term↦g(h_X), f_Funct↦g, h_Hedge1↦eps, h_Hedge2↦(b,c)} and a hedge h=(c_Ctx(i_Term),f_Funct(h_Hedge1,a,h_Hedge2)), by applying σ to h we get the hedge σ(h) = (f(g(h_X)),g(a,b,c)).

Matching problems are pairs of hedges, one of which is ground (i.e., does not contain variables). Such matching problems may have zero, one, or more (finitely many) solutions, called matching substitutions or *matchers*. For instance, the hedge (h_1,f(i_X),h_2) matches (f(a),f(b),c) in two different ways: one by the matcher {h_1↦(),i_X↦a,h_2↦(f(b),c)} and other one by the matcher {h_1↦f(a),i_X↦b, h_2↦c}. Similarly, the term c_X(f_Y(a)) matches the term f(a,g(a)) with the matchers {c_X↦f(hole,g(a)),f_Y↦f} and {c_X↦f(a,g(hole)),f_Y↦g}. Matching is the main computational mechanism in PρLog.

Instantiations of sequence and context variables can be restricted by regular hedge and regular context languages, respectively. We do not go into the details of this feature of PρLog matching here.

A ρLog *atom* (ρ-atom) is a quadruple consisting of a hole-free term st (a *strategy*), two hole-free hedges h1 and h2, and a set of regular constraints R where each variable is constrained only once, written as st :: h1 ==> h2 where R. Intuitively, it means that the strategy st transforms h1 to h2 when the variables satisfy the constraint R. We call h1 the left hand side and h2 the right hand side of this atom. When R is empty, we omit it and write st :: h1 ==> h2. The negated atom is written as st :: h1 =\=> h2 where R. A ρLog *literal* (ρ-literal) is a ρ-atom or its negation. A PρLog *clause* is either a Prolog clause, or a clause of the form st :: h1 ==> h2 where R :- body (in the sequel called a ρ-clause) where body is a (possibly empty) conjunction of ρ- and Prolog literals.

A PρLog *program* is a sequence of PρLog clauses and a *query* is a conjunction of ρ- and Prolog literals. There is a restriction on variable occurrence imposed on clauses: ρ-clauses and queries can contain only ρLog variables, and Prolog clauses and queries can contain only Prolog variables. If a Prolog literal occurs in a ρ-clause or query, it may contain only ρLog individual variables that internally get translated into Prolog variables.

3 Inference and strategies

PρLog execution principle is based on depth-first inference with leftmost literal selection in the goal. If the selected literal is a Prolog literal, then it is evaluated in the standard way. If it is a PρLog atom of the form st :: h1 ==> h2, due to the syntactic restriction called well-modedness (formally defined in [9]), st and h1 do not contain variables. Then a (renamed copy of a) program clause st' :: h1' ==> h2' :- body is selected, such that the strategy st' matches st and the hedge h1' matches h1 with a substitution σ. Next, the selected literal in the query is replaced with the conjunction σ(body),id :: σ(h2') ==> h2, where id is the built-in strategy for identity: it succeeds iff its

right-hand side matches the left-hand side. Evaluation continues further with this new query. Success and failure are defined in the standard way. Backtracking explores other alternatives that may come from matching the selected query literal to the head of the same program clause in a different way (since context/sequence matching is finitary, see, e.g., [8, 13]), or to the head of another program clause. Negative literals are processed by negation-as-failure.

When instead of the exact equality one uses proximity as in [10], then in place of `id`, PρLog introduces another built-in strategy in the new query: $\texttt{prox}(\lambda)$, which succeeds if iff its right-hand side matches the left-hand side approximately, at least with the degree λ.[1] Proximity relations indicate by which degree two expressions are close to each other, where the degree is a real number in $[0,1]$. Proximity with degree 1 means that the terms are equal, while degree 0 means that they are distinct. Hence, `id` can be seen as an abbreviation of $\texttt{prox}(1)$. Proximity is a fuzzy reflexive, symmetric, non-transitive relation, suitable for modeling imprecise, incomplete information.

Some of the other predefined strategiesof PρLog and their intuitive meanings are the following:

- $\texttt{compose}(\texttt{st}_1,\texttt{st}_2,\ldots,\texttt{st}_\texttt{n})$, $n \geq 2$, first transforms the input hedge by $\texttt{st}_1$ and then transforms the result by $\texttt{compose}(\texttt{st}_2,\ldots,\texttt{st}_\texttt{n})$ (or by $\texttt{st}_2$, if $n = 2$). Via backtracking, all possible results can be obtained. The strategy fails if either $\texttt{st}_1$ or $\texttt{compose}(\texttt{st}_2,\ldots,\texttt{st}_\texttt{n})$ fails.
- $\texttt{choice}(\texttt{st}_1,\ldots,\texttt{st}_\texttt{n})$, $n \geq 1$, returns a result of a successful application of some strategy $\texttt{st}_\texttt{i}$ to the input hedge. It fails if all $\texttt{st}_\texttt{i}$'s fail. By backtracking it can return all outputs of the applications of each of the strategies $\texttt{st}_1,\ldots,\texttt{st}_\texttt{n}$.
- $\texttt{first\_one}(\texttt{st}_1,\ldots,\texttt{st}_\texttt{n})$, $n \geq 1$, selects the first $\texttt{st}_\texttt{i}$ that does not fail on the input hedge and returns only one result of its application. `first_one` fails if all $\texttt{st}_\texttt{i}$'s fail. Its variation, `first_all`, returns via backtracking all the results of the application to the input hedge of the first strategy $\texttt{st}_\texttt{i}$ that does not fail.
- $\texttt{map}(\texttt{st})$ maps the strategy `st` to each term in the input hedge and returns the result hedge. Backtracking generates all possible output hedges. `st` should operate on a single term and not on an arbitrary hedge. $\texttt{map}(\texttt{st})$ fails if `st` fails for at least one term from the input hedge.

4 Examples

In this section we bring some examples to illustrate features and the expressive power of PρLog.

Example 1 (Sorting)**.** The following program illustrates how bubble sort can be implemented in PρLog.

```
swap(f_Ordering) :: (h_X, i_I, i_J, h_Y) ==> (h_X, i_J, i_I, h_Y) :-
    not(f_Ordering(i_I, i_J)).
bubble_sort(f_Ordering) := first_one(nf(swap(f_Ordering))).
```

In the first clause, the user-defined strategy `swap` swaps two neighboring elements `i_I, i_J` in the given hedge if they violate the given ordering `f_Ordering`. The use of sequence variables `h_I, h_J` helps to identify the violating place in the given hedge by pattern matching, without the need to explicitly define the corresponding recursive procedure. The sorting strategy `bubble_sort` is then defined as an exhaustive application of `swap` (via the built-in strategy `nf`), which will lead to a sorted hedge. The strategy `first_one` guarantees that only the first answer computed by `nf(swap(f_Ordering))` is returned: it does not make sense to sort a hedge in different ways to get the same answer over and over again via backtracking.

The way how the `bubble_sort` strategy is defined above is just an abbreviation of the clause

[1] This is an experimental feature, not yet included in the official distribution.

```
bubble_sort(f_Ordering) :: h_X ==> h_Y :-
    first_one(nf(swap(f_Ordering))) :: h_X ==> h_Y.
```

PρLog allows the user to write such abbreviations. To use this strategy for sorting a hedge, we could call, e.g.,

```
?(bubble_sort(=<) :: (1,3,4,3,2) ==> h_X, Result).
```

and PρLog would return a single result in the form of a substitution:

```
Result = [h_X ---> (1,2,3,3,4)].
```

Example 2 (Rewriting)**.** One step of rewriting a term by some rule/strategy can be straightforwardly defined in PρLog:

```
rewrite_step(i_Str) :: c_Ctx(i_X) ==> c_Ctx(i_Y) :- i_Str :: i_X ==> i_Y.
```

It finds a subterm `i_X` in `c_Ct(i_X)` to which the strategy `i_Str` applies and rewrites it. Due to the built-in matching algorithm that finds the relevant instantiation of the context variable `c_Ctx`, this step corresponds to a leftmost-outermost rewriting step. In [9] we have illustrated how easily other rewriting strategies can be modeled in PρLog.

Example 3 (Using proximity)**.** We assume that a proximity relation between function symbols is given. (Based on it, we can compute proximity between terms as well.) The task is to remove from a given hedge approximate duplicates, i.e., if the hedge contains two elements that are proximal to each other (by a predefined degree), we should get a hedge where only one of the proximal elements is retained. The strategy `merge_dubles(i_D)` below does it. It checks, whether the hedge contains somewhere two elements `i_X` and `i_Y` that are close to each other at least by the degree `i_D` and removes `i_Y`. `merge_all_dubles(i_D)` removes all duplicates and returns one answer:

```
merge_dubles(i_D) :: (h_X, i_X, h_Y, i_Y, h_Z) ==> (h_X, i_X, h_Y, h_Z) :-
    prox(i_D) :: i_X ==> i_Y.
merge_all_dubles(i_D) := first_one(nf(merge_dubles(i_D))).
```

Assume our proximity relation is such that `a` and `b` are proximal with the degree 0.6 and `b` is close to `c` with the degree 0.8. Then we have:

```
?(merge_dubles(0.5) :: (a,b,d,b,c) ==> h_Ans, Result).
Result = [h_Ans --> (d,c)] ;
false.

?(merge_dubles(0.7) :: (a,b,d,b,c) ==> h_Ans, Result).
Result = [h_Ans --> (a,d,c)] ;
false.
```

Due to nontransitivity of proximity relations and the fact that matching finds the first proximal pair (from the left), the order of hedge elements affects the answer. For instance, if we put `a` at the end, we get

```
?(merge_dubles(0.5) :: (b,d,b,c,a) ==> h_Ans, Result).
Result = [h_Ans --> (d,c,a)] ;
false.

?(merge_dubles(0.7) :: (b,d,b,c,a) ==> h_Ans, Result).
Result = [h_Ans --> (d,c,a)] ;
false.
```

It happens because `a` and `c` are not close to each (although `a` and `b` as well as `b` and `c` are).

5 Summary

The main advantages of PρLog are: compact and declarative code; capabilities of expression traversal without explicitly programming it; the ability to use clauses in a flexible order with the help of strategies. Besides, PρLog has access to the whole infrastructure of its underline Prolog system. These features make PρLog suitable for nondeterministic computations, manipulating XML documents, implementing rule-based algorithms and their control, etc.

Acknowledgments. This work has been supported by the Shota Rustaveli National Science Foundation of Georgia under the grant YS-18-1480.

References

[1] Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau & Antoine Reilles (2007): *Tom: Piggybacking Rewriting on Java*. In Franz Baader, editor: *Term Rewriting and Applications, 18th International Conference, RTA 2007, Paris, France, June 26-28, 2007, Proceedings*, *Lecture Notes in Computer Science* 4533, Springer, pp. 36–47, doi:10.1007/978-3-540-73449-9_5.

[2] Peter Borovanský, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau & Marian Vittek (1996): *ELAN: A logical framework based on computational systems*. In José Meseguer, editor: *First International Workshop on Rewriting Logic and its Applications, RWLW 1996, Asilomar Conference Center, Pacific Grove, CA, USA, September 3-6, 1996*, *Electronic Notes in Theoretical Computer Science* 4, Elsevier, pp. 35–50, doi:10.1016/S1571-0661(04)00032-5.

[3] Mark van den Brand, Arie van Deursen, Jan Heering, Hayco de Jong, Merijn de Jonge, Tobias Kuipers, Paul Klint, Leon Moonen, Pieter A. Olivier, Jeroen Scheerder, Jurgen J. Vinju, Eelco Visser & Joost Visser (2001): *The Asf+Sdf Meta-Environment: a Component-Based Language Development Environment*. *Electron. Notes Theor. Comput. Sci.* 44(2), pp. 3–8, doi:10.1016/S1571-0661(04)80917-4.

[4] Horatiu Cirstea & Claude Kirchner (2001): *The rewriting calculus - Part I*. *Log. J. IGPL* 9(3), pp. 339–375, doi:10.1093/jigpal/9.3.339.

[5] Horatiu Cirstea & Claude Kirchner (2001): *The rewriting calculus - Part II*. *Log. J. IGPL* 9(3), pp. 377–410, doi:10.1093/jigpal/9.3.377.

[6] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer & Jose F. Quesada (2002): *Maude: specification and programming in rewriting logic*. *Theor. Comput. Sci.* 285(2), pp. 187–243, doi:10.1016/S0304-3975(01)00359-0.

[7] Jorge Coelho, Besik Dundua, Mário Florido & Temur Kutsia (2010): *A Rule-Based Approach to XML Processing and Web Reasoning*. In Pascal Hitzler & Thomas Lukasiewicz, editors: *Web Reasoning and Rule Systems - Fourth International Conference, RR 2010, Bressanone/Brixen, Italy, September 22-24, 2010. Proceedings*, *Lecture Notes in Computer Science* 6333, Springer, pp. 164–172, doi:10.1007/978-3-642-15918-3_13.

[8] Hubert Comon (1998): *Completion of Rewrite Systems with Membership Constraints. Part II: Constraint Solving*. *J. Symb. Comput.* 25(4), pp. 421–453, doi:10.1006/jsco.1997.0186.

[9] Besik Dundua, Temur Kutsia & Mircea Marin (2009): *Strategies in PRholog*. In Maribel Fernández, editor: *Proceedings Ninth International Workshop on Reduction Strategies in Rewriting and Programming, WRS 2009, Brasilia, Brazil, 28th June 2009*, *EPTCS* 15, pp. 32–43, doi:10.4204/EPTCS.15.3.

[10] Besik Dundua, Temur Kutsia, Mircea Marin & Cleo Pau (2019): *Extending the ρLogCalculus with Proximity Relations*. In George Jaiani & David Natroshvili, editors: *Applications of Mathematics and Informatics in Natural Sciences and Engineering*, Springer, Cham, pp. 83–100, doi:10.1007/978-3-030-56356-1_6.

[11] Besik Dundua, Temur Kutsia & Klaus Reisenberger-Hagmayer (2017): *An Overview of PρLog*. In Yuliya Lierler & Walid Taha, editors: *Practical Aspects of Declarative Languages - 19th International Symposium,*

PADL 2017, Paris, France, January 16-17, 2017, Proceedings, *Lecture Notes in Computer Science* 10137, Springer, pp. 34–49, doi:10.1007/978-3-319-51676-9_3.

[12] Thom W. Frühwirth (1998): *Theory and Practice of Constraint Handling Rules*. *J. Log. Program.* 37(1-3), pp. 95–138, doi:10.1016/S0743-1066(98)10005-5.

[13] Temur Kutsia & Mircea Marin (2005): *Matching with Regular Constraints*. In Geoff Sutcliffe & Andrei Voronkov, editors: *Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference, LPAR 2005, Montego Bay, Jamaica, December 2-6, 2005, Proceedings*, *Lecture Notes in Computer Science* 3835, Springer, pp. 215–229, doi:10.1007/11591191_16.

[14] John W. Lloyd (1987): *Foundations of Logic Programming, 2nd Edition*. Springer, doi:10.1007/978-3-642-83189-8.

[15] Mircea Marin & Tetsuo Ida (2005): *Rule-Based Programming with ρLog*. In Daniela Zaharie, Dana Petcu, Viorel Negru, Tudor Jebelean, Gabriel Ciobanu, Alexandru Cicortas, Ajith Abraham & Marcin Paprzycki, editors: *Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2005), 25-29 September 2005, Timisoara, Romania*, IEEE Computer Society, pp. 31–38, doi:10.1109/SYNASC.2005.61.

[16] Mircea Marin & Temur Kutsia (2003): *On the Implementation of a Rule-Based Programming System and some of its Applications*. In Boris Konev & Renate Schmidt, editors: *Proc. 4th International Workshop on the Implementation of Logics, WIL'04*, pp. 55–69.

[17] Mircea Marin & Temur Kutsia (2006): *Foundations of the rule-based system ρLog*. *J. Appl. Non Class. Logics* 16(1-2), pp. 151–168, doi:10.3166/jancl.16.151-168.

[18] Mircea Marin, Temur Kutsia & Besik Dundua (2019): *A Rule-based Approach to the Decidability of Safety of ABACα*. In Florian Kerschbaum, Atefeh Mashatan, Jianwei Niu & Adam J. Lee, editors: *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies, SACMAT 2019, Toronto, ON, Canada, June 03-06, 2019*, ACM, pp. 173–178, doi:10.1145/3322431.3325416.

[19] Mircea Marin & Florina Piroi (2004): *Deduction and Presentation in ρLog*. *Electron. Notes Theor. Comput. Sci.* 93, pp. 161–182, doi:10.1016/j.entcs.2003.12.033.

[20] Phong Nguyen (2015): *Meta-mining: a meta-learning framework to support the recommendation, planning and optimization of data mining workflows*. Ph.D. thesis, University of Geneva, doi:10.13097/archive-ouverte/unige:86131.

[21] Eelco Visser (2001): *Stratego: A Language for Program Transformation Based on Rewriting Strategies*. In Aart Middeldorp, editor: *Rewriting Techniques and Applications, 12th International Conference, RTA 2001, Utrecht, The Netherlands, May 22-24, 2001, Proceedings*, *Lecture Notes in Computer Science* 2051, Springer, pp. 357–362, doi:10.1007/3-540-45127-7_27.

An introduction to Mathdialog

Carlos E. Freites
Alcala de Henares 16 7C
Guadalajara, Spain
carlos_freites@yahoo.com

Abstract. We present Mathdialog, the computer implementation of the CZFC set theory. Although we cannot discuss CZFC here, Mathdialog may be of interest as a pedagogical tool for students. It is not as powerful as the leading Interactive Theorem Provers such as Coq. Isabelle or LEAN. Because Mathdialog is not a programming language and it is based on ZFC (in the form it is introduced by [5]), the standard mathematical fundament, users may learn it easily. Proving a theorem in Mathdialog may even be perceived as an interesting puzzle because of its simplicity. It is not only for checking proofs; it is a primitive form of knowledge transmission. After a proof is checked, Mathdialog can show it in an interactive environment to students where they will comfortably find answers to their usual doubts; to use it, students only need a basic mathematical background and no knowledge of CFZC nor Mathdialog. Mathdialog is suitable for undergraduate textbooks content, not for high-level mathematics, but may give a convincing illusion of proof understanding. Mathdialog can be tested in [2] by clicking Practice Area.

1 The simplicity of the CZFC syntax and the Mathdialog GUI

Context is widely used in mathematical natural language at all levels. For example, we can find the expression F(x) without an explicit indication that F is a function or see a mention of a group G without an explicit indication about its binary operator. However, it is hard to find its inclusion in formal presentations (for example, in [7]).

Labels are another widely used resource in mathematical natural language as they are used in textbooks. An interesting example is when a theorem of existence and uniqueness is labeled in a textbook, for example as "Inverse uniqueness theorem." It is easy to use that label in mathematical natural language to define the object whose existence is assured by the theorems referred by it.

A label and a context are two elements that have in common the syntax of the main CZFC sentences (theorems and definitions). They make the formulas those sentences contain make sense. Formulas are defined in the usual recursive way. The informal CZFC syntax of a theorem is:

```
THEOR[label;local context;formula]
```

Where `label` is a string of characters without blanks, and `local context` is a list of atomic formulas. This list is not commutative because each element may be the local context of the elements that follow it. CZFC has five sentences that allow us to define, inside the language, new predicates, and sets. For example, the following informal syntax will enable us to define new predicates and new sets by comprehension:

```
DEF PRED[label;local context;label(args) <==> formula]
DEF OBC[label;local context;label(args) = {BG(v,T):formula}]
```

T. Kutsia (Ed.): Symbolic Computation
in Software Science (SCSS'21)
Short and work-in-progress papers, pp. 17–21

Where `T` is a term and `BG(v,T)` represents $(v \in T)$. The role of local context is to introduce variables by associating hypotheses to them. It is the most important kind of context, but there are two more. The list of variables `args` must have all variables introduced in `local context`.

2 An overview of the proving process

In Mathdialog, proving is interacting with a dynamic and hierarchical list of hypotheses and goals called proof list. Logic commands sentences place hypotheses and/or a goal to the proof list. A proof starts putting the theorems local context as hypotheses and its formula as the main goal in the proof list. A proof is a sequence of logic commands whose last member place in the proof list, a hypothesis that matches the main goal. There are 25 logic commands [1], one for each kind of proof step in a textbook.

Each logic command has a particular requirement to add its arguments to the proof list. Because in CZFC a formula cannot live without a context, Mathadialog takes as context for the formulas in logic commands arguments, some of the available hypotheses present in the proof list.

In Mathdialog, each proof has three forms: the human-to-machine proof, the CZFC proof (its internal representation made of the sequence of logic commands mentioned above), and the machine-to-human proof. The human-to-machine proof helps humans find the CZFC proof, which Mathdialog uses to build the machine-to-human proof automatically. It is interactive, in mathematical English, and can be understood without knowing CZFC nor Mathdialog. Three minutes videos of a human-to-machine proof, and the resulting machine-to-human proof, can be seen in [3] and [4], respectively.

The Mathdialog GUI has two windows; the Command Window is where users write sentences; the Blackboard is where Mathdialog places its responses. TE, FA, BG, and EQ are used in human-to-machine proofs instead of the standard notations $\exists$, $\forall$, $\in$, and $=$ respectively. In Mathdialog, users can access the ”Theorems by name” and ”Definitions by name” on the ”Consults” blue menu. The theorems and definitions, displayed in floating windows, are in mathematical English. Users do not need to know CZFC nor Mathdialog to understand them.

Users write a definition, a theorem or a logic command in the Command Window and sent it for processing by clicking on the ”SEND COMMAND” button. In the case of a definition, if the syntax is correct and passes other verifications, Mathdialog stores it. In the case of a theorem, the human-to-machine proof process starts by displaying in the Blackboard the theorem local context as the proof first hypotheses and its formula as the main goal. At this point, the Command Window only accepts logic commands. If the proof is successful, the proof list will be the human-to-machine proof.

For example, after proving the theorem `THEOR[EMPTY_P01; SET(X); SUBSET(EMPTY,X)]`, the Blackboard content, and so the human-to-machine proof, is bellow. The lines starting with `GL` are goals, and the ones with `H` are hypotheses. Mathdialog generates one of them or both as a response to theorems or logic commands. Mathdialog replicates the logic commands the user sends with a `U` in the proof list. The text after N: in each line are comments not generated by Mathdialog.

```
GL1 - SUBSET(EMPTY,X)                N:The theorem thesis (formula).
H1 - SET(X)                          N:The theorem hypothesis (local context).
U - IP[FA(BG(x,EMPTY):BG(x,X))]      N:The first logic command sent by the user.
GL1.1 - FA(BG(x,EMPTY):BG(x,X))      N:Response to the first logic command.
U - UQ_RED[]                         N:The second logic command sent by the user.
H1.1 - SET(x)                        N:These two lines are the Mathdialog response
GL1.2 - BG(x,EMPTY) ==> BG(x,X)      N:to the second logic command.
```

```
U - BY_THEOR[NOTBG(x,EMPTY)] EXISTENCE_AXIOM
H1.2 - NOTBG(x,EMPTY)
U - DEF_OF[NOTBG(x,EMPTY)]
H1.2 - NOT(BG(x,EMPTY))
U - PROP_CONS[BG(x,EMPTY) ==> BG(x,X)]
GL1 - SUBSET(EMPTY,X)
HD1.1 - BG(x,EMPTY) ==> BG(x,X)
HD1 - FA(BG(x,EMPTY):BG(x,X))
U - ATOMIC_OF[FA(BG(x,EMPTY):BG(x,X)),SUBSET(EMPTY,X)]
HD - SUBSET(EMPTY,X)
```

The CZFC proof is the list of logic commands sent by the user. It is what Mathdialog stores as the theorem's proof. So, the CZFC proof of `EMPTY_P01` is:

```
IP[FA(BG(x,EMPTY):BG(x,X))],
UQ_RED[],
BY_THEOR[NOTBG(x,EMPTY)],
DEF_OF[NOTBG(x,EMPTY)],
PROP_CONS[BG(x,EMPTY) ==> BG(x,X)],
ATOMIC_OF[FA(BG(x,EMPTY):BG(x,X)),SUBSET(EMPTY,X)]
```

From this six elements list, Mathdialog builds the interactive machine-to-human proof shown in figure 1. Now we will highlight some Mathdialog features. First, in each floating window with definitions or theorems, the user can click the link "Show the formal-regular version" to flip the English version to the formal version and vice versa. Users can use it to learn the CZFC syntax or to solve potential ambiguities in the English version. Second, selecting any word, for example by double-clicking it, hovering the Consult menu item, and clicking "The selected name," opens a floating window with the definition of that word or the theorem it names. Third, in the machine-to-human proof, the user has simple access to the answer to usual questions such as: "What is the justification of that hypothesis?", "What says the theorem justifying that hypothesis?", "What is the justification of this step?" and "How is this defined?". If the machine-to-human proof can answer those kinds of questions, can we conclude that, in a weak Turing test sense, Mathdialog has "understood" the proof?

The human-to-machine proofs process cannot enforce the pedagogical quality of the machine-to-human proofs Mathdialog will generate. But the more pedagogically complete it is, the more intuitive, convincing, and useful for users will be the "understanding" illusion exhibited by the machine-to-human proof. For example, the proof that there is no universe ($\forall U \exists x : \neg(x \in U)$), may start as follow:

`THEOR[THERE_IS_NO_UNIVERSE;SET(U);TE(SET(x): NOT( BG(x,U) ))]`
`LET_TERM[Z,{BG(x,U):NOT(BG(x,x))}]` (The set of all $x \in U$ such that $\neg(x \in x)$.)
`EQ_RED[Z]` (Mathdialog responds with the new goal NOT(BG(Z,U)))
`BY_DEF_OBC[BG(Z,{BG(x,U):NOT(BG(x,x))})]`
(This logic command adds the definition of its argument as a hypothesis to the proof list, according to the axiom scheme of comprehension: $Z \in Z \iff Z \in U \wedge \neg(Z \in Z)$)

At this point, the user has two alternatives: Use `PROP_CONS[NOT(BG(Z,U))]` to finish the proof, or use `CONTRD_PROOF` and four more logic commands to complete the proof by contradiction. The first option may sound attractive because it finishes the whole proof with just four logic commands. However,

the resulting machine-to-human proof would be too difficult for users because it could not justify why the argument of `PROP_CONS` is a propositional consequence of the available hypothesis. `PROP_CONS` worked in the human-to-machine proof because it uses the theorem prover OTTER [6]. It is powerful, but it should be used when the propositional consequence is pedagogically obvious. The second option is better because it helps Mathdialog build a more clear machine-to-human proof for human beings.

3 Conclusion

We expect mathematicians will perceive Mathdialog as a toy with the potential to be an effective tool for students. Although Mathdialog is not made to deal with high level theorems, it may attract the attention of working mathematicians as a fun and interesting puzzle. At this time, logic commands in human-to-machine proofs cannot accept drawings, examples, or intuitive hints to show them in machine-to-human proofs at user requests. The human-to-machine proof process makes it reliable to locate the exact point to put them in the pedagogical right place. Implementing it is among our development priorities.

Enhance the GUI with spoken English understanding and spoken output would make Mathdialog even more attractive. The simple CZFC syntax makes it a perfect target for neural network training.

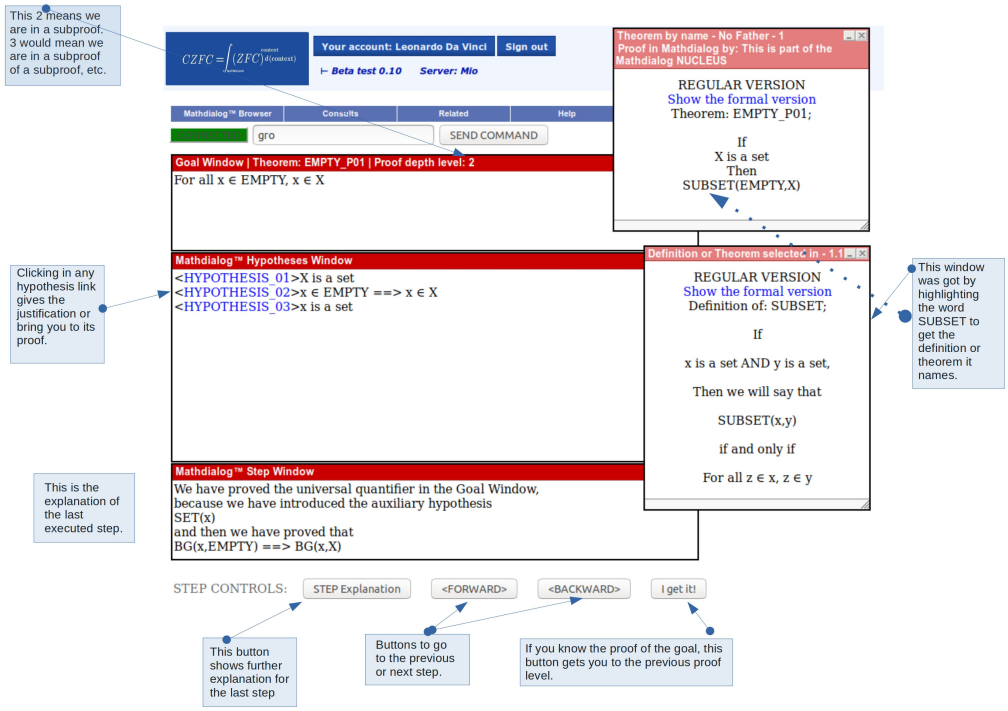

Figure 1: A machine-to-human proof

References

[1] Carlos E. Freites (2019): *Logic Commands Reference Guide*. Available at https://mathdialog.com/LogicCommandsReferenceGuide.pdf.

[2] Carlos E. Freites (2021): *Mathdialog Web Site*. Available at https://mathdialog.com.

[3] Carlos E. Freites (2021): *Human-to-Machine proof video, 3:14 minutes*. Available at https://www.youtube.com/watch?v=dMBF3yKSN4o.

[4] Carlos E. Freites (2021): *Machine-to-Human proof video, 4:14 minutes*. Available at https://www.youtube.com/watch?v=27dUUh1zkH4.

[5] Karel Hrbacek & Thomas Jech (1984): *Introduction to Set Theory*, second edition. 85, Marcel Dekker, Inc.

[6] William McCune (2003): *OTTER3.3 Reference Manual*. Available at https://www.cs.unm.edu/~mccune/otter/Otter33.pdf.

[7] Arnold Neumaierand & Flaviu Adrian Marginean (2020 Version 1.9): *Logic in Context*. Available at https://www.mat.univie.ac.at/~neum/FMathL/context.pdf.

A Systematic Search for Vertex Transitive Directed Strongly Regular Graphs

Štefan Gyürki
Department of Mathematics and Descriptive Geometry
Faculty of Civil Engineering
Slovak University of Technology
Bratislava, Slovakia
stefan.gyurki@stuba.sk

Abstract. We report about a computer assisted search for vertex transitive directed strongly regular graphs. The enumeration of such graphs was previously known for the number of vertices up to 20. Using computer algebra system GAP and the catalogue of all association schemes we extend this enumeration for graphs up to order 31.

1 Introduction

Since the main objects of interest in algebraic graph theory are highly symmetric graphs, strongly regular graphs are playing a central role in this area. A possibility how to quantify symmetry of graphs is by considering the properties of its group of automorphisms. Special attention is paid on the graphs, which are vertex transitive, that is, their group of automorphisms has only one orbit on the set of vertices.

One kind of generalization of the concept of strong regularity for directed graphs was given by Duval in 1988, see [2]. A catalogue of all directed strongly regular graphs up to order 12 was published by the current author in [5]. Small vertex transitive directed strongly regular graphs were considered systematically first in [3]. The authors described all such graphs up to order 20. Here we extend the enumeration of such graphs up to order 31. As the number of graphs needed to be considered grows quickly with the order, it was clear that this job can not be made by hand. Moreover, to achieve the results we needed to combine the theoretical knowledge about vertex transitive graphs, together with the availability of combinatorial catalogue of association schemes [8] and to use the SetOrbit package [15] for reducing the search space in an algebraic way. As a result of our efforts we announce that the number of equivalence classes of vertex transitive directed strongly regular graphs of order up to 31 is precisely 140. Among them 107 classes are such that they can be realized as Cayley graphs. At the time of writing this paper there are five parameter sets of order at most 31, on which the existence of a graph is still open. These are (27,7,4,1,2), (27,11,6,5,4), (28,6,3,2,1), (30,11,9,2,5) and (30,12,11,4,5). As a by-product of our search we can confirm that there are no vertex transitive graphs with these parameters, hence if there exists a graph realizing them, then it has necessarily intransitive group.

2 Preliminaries

Let us start with the used terminology from the area of graph theory.

The number of vertices of a graph Γ is called the *order* of Γ and it is usually denoted by n. A graph Γ of order n with adjacency matrix $A = A(\Gamma)$ is called *regular*, if there exists a positive integer k such that $AJ = JA = kJ$, where J is the all-one matrix of dimension $n \times n$. The number k is called *valency* of Γ.

T. Kutsia (Ed.): Symbolic Computation
in Software Science (SCSS'21)
Short and work-in-progress papers, pp. 22–27

A simple regular graph with valency k is said to be *strongly regular* (SRG, for short) if there exist integers λ and μ such that for each edge $\{u,v\}$ the number of common neighbours of u and v is exactly λ; while for each non-edge $\{u,v\}$ the number of common neighbours of u and v is equal to μ. Previous condition can be rewritten equivalently into the equation

$$A^2 = kI + \lambda A + \mu(J - I - A) \tag{1}$$

using the adjacency matrix of Γ. The quadruple (n,k,λ,μ) is called the *parameter set* of an SRG Γ.

A natural generalization of SRGs for directed graphs was given by Duval [2]. A *directed strongly regular graph* (DSRG) with parameters (n,k,t,λ,μ) is a regular directed graph on n vertices with valency k, such that every vertex is incident with t undirected edges, and the number of paths of length 2 directed from a vertex x to another vertex y is λ, if there is an arc from x to y, and μ otherwise. In particular, a DSRG with $t = k$ is an SRG, and a DSRG with $t = 0$ is a *doubly regular tournament*, in the sense of [17]. Here we consider only DSRGs satisfying $0 < t < k$, which are called *genuine* DSRGs. The adjacency matrix $A = A(\Gamma)$ of a DSRG with parameters (n,k,t,λ,μ) satisfies $AJ = JA = kJ$ and

$$A^2 = tI + \lambda A + \mu(J - I - A), \tag{2}$$

where I and J are the identitiy matrix and the all-one matrix of dimension $n \times n$, respectively.

The parameters n,k,t,λ,μ of DSRGs are not independent. The basic combinatorial and algebraic conditions they have to satisfy are called *feasibility conditions*, and can be found in [2]. A 5-tuple (n,k,t,λ,μ) is called a *feasible parameter set*, if its parameters satisfy the feasibility conditions. A feasible parameter set for which at least one DSRG Γ exists is called *realizable*, otherwise *non-realizable*. The lists of feasible parameter sets up to order $n \leq 110$ are displayed on the webpage of A.E. Brouwer and S. Hobart [1]. It is known that if Γ is a DSRG, then its reverse digraph Γ^T and also its complementary digraph $\bar{\Gamma}$ are DSRGs, see [2, 11]. We say that the graphs $\Gamma, \bar{\Gamma}, \Gamma^T$ and $\bar{\Gamma}^T$ form an *equivalence class* of DSRGs.

The second algebraic ingredient of our research is the theory of association schemes.

Under a *color graph* Γ we mean an ordered pair $(V,\mathscr{R})$, where V is a set of vertices and $\mathscr{R}$ a partition of $V \times V$ into binary relations. The elements of $\mathscr{R}$ are called *colors*, and the number of colors is the *rank* of Γ. In other words, a color graph is an edge-colored complete directed graph with loops, whose arcs are colored by the same color if and only if they belong to the same binary relation.

For color graphs $(V,\mathscr{R})$ we can define a few morphisms. An *isomorphism* of color graphs $(V,\mathscr{R})$ and $(V',\mathscr{R}')$ is a bijection φ from V to V' that induces a bijection of colors (relations) in $\mathscr{R}$ onto colors in $\mathscr{R}'$. A *weak* (or *color*) *automorphism* of $\Gamma = (V,\mathscr{R})$ is an isomorphism of Γ with itself. If the induced permutation of colors is the identity permutation, then we speak of a *(strong) automorphism.*

An *association scheme* is a color graph $\mathscr{M} = (\Omega,\mathscr{R})$, $\mathscr{R} = \{R_i \mid i \in I\}$, such that the following axioms are satisfied:

(i) The diagonal relation $\Delta_\Omega = \{(x,x) \mid x \in \Omega\}$ does belong to $\mathscr{R}$.

(ii) For each $i \in I$ there exists $i' \in I$ such that $R_i^T = R_{i'}$, where $R_i^T = \{(y,x) \mid (x,y) \in R_i\}$ is the relation transposed to R_i.

(iii) For any $i,j,k \in I$, the number $p_{i,j}^k$ of elements $z \in \Omega$ such that $(x,z) \in R_i$ and $(z,y) \in R_j$ is a constant depending only on i,j,k, and independent on the choice of $(x,y) \in R_k$.

The numbers $p_{i,j}^k$ are called *intersection numbers*, or sometimes *structure constants* of $\mathscr{M}$. An association scheme $\mathscr{M}$ is called *commutative*, if for all $i,j,k \in I$ we have $p_{i,j}^k = p_{j,i}^k$, otherwise *non-commutative*.

An association scheme $\mathscr{M} = (\Omega, \mathscr{R})$ is called *thin*, if for each $x \in \Omega$ and $R_i \in \mathscr{R}$ there is precisely one $y \in \Omega$ such that $(x, y) \in R_i$. Otherwise, it is called *non-thin*.

By an *algebraic automorphism* of an association scheme $\mathscr{M}$ we mean a permutation acting on basic relations such that it is preserving the tensor of structure constants of $\mathscr{M}$.

The notion of an association scheme may be reformulated in terms of matrices. A *coherent algebra* $\mathscr{W}$ is a set of square matrices of order n over the field $\mathbb{C}$ that forms a matrix algebra, is closed with respect to the operations of SchurHadamard multiplication and transposition, and contains both the identity matrix I and the all-one matrix J. The set of basic matrices $\{A_i \mid i \in I\}$ of an association scheme $\mathscr{M}$ serves as a standard basis of the corresponding coherent algebra $\mathscr{W}$, and in this case we write $\mathscr{W} = \langle A_1, A_2, \ldots, A_r \rangle$.

The intersection of coherent algebras is again a coherent algebra. This implies the existence of the *coherent closure* of B, denoted by $\langle\langle B \rangle\rangle$, this is the smallest coherent algebra containing a prescribed set B of matrices of order n. An efficient polynomial-time algorithm for the computation of coherent closure, often referred to as *Weisfeiler-Leman stabilization* (or *WL-stabilization*, for short), is described in [19].

For more definitions of the terms used but not defined here, we refer to [10, 12].

3 The process of enumeration

This research naturally stems from a few previous investigations of DSRGs.

On one hand, in [6] we were looking for DSRGs with parameters for which the existence of a graph realizing it was not known previously; while in [7] we provided a construction of infinite families of DSRGs from a smaller one, supposed on the existence of a specific equitable partition in it. Both results were achieved by essential support of computer algebra packages and theoretical interpretation of the obtained data.

On the other hand, all the vertex transitive DSRGs up to order 20 were classified in [3]. Motivated by the previous successes, following a similar strategy used in [6, 7], here we enumerate the vertex transitive DSRGs up to order 31.

Since graphs and digraphs can be regarded as binary relations, while association schemes are collections of binary relations in the sense of our definition, it was natural to join these concepts in searching for digraphs with specific properties. Hence our search for vertex transitive DSRGs heavily relied on the classification of association schemes and used the catalogue of association schemes by Hanaki and Miyamoto [8]. The most important theoretical result making this search possible is the following theorem by [11]:

Theorem. *Let Γ be a genuine directed strongly regular graph. Then the coherent closure of Γ is non-commutative and its rank is at least* 6.

In other words, we have to consider non-commutative association schemes of rank at least 6, when we are searching for directed strongly regular graphs as unions of relations in a prescribed association scheme. Hence the difficulty of our problems mainly depends on the number of non-commutative association schemes of specified order. For example, the number of such schemes is 242 in the case of order 24, while 0 for the order 25. Thus, we can say immediately that there are no vertex transitive DSRGs of order 25, while the complete enumeration of vertex transitive DSRGs of order 24 needs a lot of effort. On the other hand, the number of possible mergings of relations in a given association scheme grows exponentially with its rank, making the problem hard for higher ranks. Though the search space

for mergings can be reduced by considering algebraic automorphisms of a given association scheme, the complexity of the problem remains exponential in rank.

The realization of our theoretical ideas on a computer was done as follows. We run all computations in the software GAP [4] with its share packages:

- GRAPE [18] for computation with graphs, including nauty [14] for testing graph isomorphism;
- COCO-2P [13] for computation with association schemes and coherent configurations;
- SetOrbit [15] for finding representatives of orbits of group actions on sets of various size, documented in [16].

In fact, for a given order n, we were searching first for mergings in the set of vertex transitive non-thin association schemes of order n that are non-commutative of rank at least 6. The output was a set $\mathscr{G}$ of suitable graphs that are vertex transitive DSRGs of order n. This set $\mathscr{G}$ contains clearly all the vertex transitive DSRGs of order n which are either non-Cayley graphs, or Cayley DSRGs having more than n automorphisms.

Thus, it remained to search for those Cayley DSRGs of order n, which have precisely n automorphisms. This search was done generally by considering all possible Cayley generating sets that can produce Cayley DSRGs of order n, however with the aid of the SetOrbit package the space of generating sets under consideration was immediately reduced up to Cayley isomorphism. Thus, from the set of those generating sets that produce isomorphic graphs predictable just by the algebraic properties of the used group, only one representative was kept. The output from this step was the set $\mathscr{H}$ of all Cayley DSRGs of order n.

Clearly, the union $\mathscr{G} \cup \mathscr{H}$ is the set of all vertex transitive DSRGs of order n; while the set $\mathscr{G} \setminus \mathscr{H}$ is the set of vertex transitive DSRGs that are non-Cayley, and, finally, the set $\mathscr{H} \setminus \mathscr{G}$ is the set of Cayley DSRGs of order n having precisely n automorphisms.

4 The results

Complete enumeration of vertex transitive DSRGs up to order 20 was given in [3]. The authors found 35 equivalence classes of such graphs up to order 20. After performing our research, we confirm and extend this result for graphs up to order 31 and state that the number of equivalence classes up to this order is 140.

More detailed results are displayed in the Table 1. Notice that in Table 1 the prime orders are not displayed simply from the reason that there are no DSRGs of prime order, see [2]. The used abbreviations are: VT-EC – *number of equivalence classes of vertex transitive DSRGs*; Cay-EC – *number of equivalence classes of Cayley DSRGs*; VT-DSRG – *the number of pairwise non-isomorphic vertex transitive DSRGs*; Cay-DSRG – *the number of pairwise non-isomorphic Cayley DSRGs*. Also one can notice that the numbers of vertex transitive DSRGs and Cayley DSRGs of a given order are always even; this is because the parameter sets can be paired based on complementarity of graphs.

As we already mentioned in the *Introduction*, at the time of writing this paper there were five parameter sets of DSRGs of order at most 31, on which the existence of a graph was still open. These were (27,7,4,1,2), (27,11,6,5,4), (28,6,3,2,1), (30,11,9,2,5) and (30,12,11,4,5). As a by-product of our search we can confirm that there are no vertex transitive graphs with these parameters, hence if there exists a graph realizing them, then it is necessarily with intransitive group.

Table 1: Summary of the results

Order	VT-EC	Cay-EC	VT-DSRG	CayDSRG
≤ 20 [3]	35	30	102	84
21	4	4	14	14
22	4	4	14	14
24	33	30	116	104
25	0	0	0	0
26	11	7	40	24
27	6	6	20	20
28	8	6	24	18
30	39	20	142	70
≤ 31	140	107	472	348

5 Work in progress

Above, we reported about the results of our exhaustive search for vertex transitive DSRGs up to order 31. In fact, we are close to complete similar data for the order 32. This order is quite special, because here a kind of combinatorial explosion appears: the number of non-commutative association schemes is a few of thousands. When this order will be finished, the continuation of this kind of search will be probably much faster up to order 47. Here we notice, that very recently Holt and Royle [9] published the census of all vertex transitive undirected and directed graphs up to order 47. Of course, our goal is more modest as we are considering digraphs with a strong regularity property, but still the huge number of combinatorial objects being considered makes the problem very difficult.

During the above-mentioned search, as a by-product, also a few DSRGs were found, which are not vertex transitive, but their coherent closure is an association scheme. This can happen only when the association scheme is non-Schurian, that is, when it does not appear as a set of 2-orbits of a permutation group. This phenomenon creates for us a very interesting challenge.

Another interesting challenge is obtained by investigating the question:

"How far can be a vertex transitive DSRG from being a Cayley graph?"

That is, let Γ be a vertex transitive DSRG of order n and G its group of automorphisms. Clearly, n divides $|G|$. Let H be the smallest subgroup of G that acts vertex transitively on Γ. Let us call the ratio $\kappa = |H|/n$ as the *Cayley deficiency of* Γ. Trivially, if Γ is a Cayley graph, then $\kappa = 1$, otherwise $\kappa > 1$. Most of the discovered vertex transitive non-Cayley DSRGs have $\kappa = 2$, however, we also discovered DSRGs of order 28 with Cayley deficiency 6. Hence, we pose the following problem:

"Construct a family of vertex transitive DSRGs with arbitrarily large Cayley deficiency."

Acknowledgements

This research was supported by the APVV Research Grants 17-0428 and 19-0308, and the VEGA Research Grants 1/0238/19 and 1/0206/20. The author is very grateful to Misha Klin, Christian Pech, Sven Reichard and Matan Ziv-Av for helpful communications on various topics from computer algebra systems and algebraic graph theory.

References

[1] A.E. Brouwer & S. Hobart (2021): Tables of directed strongly regular graphs, http://homepages.cwi.nl/~aeb/.

[2] A.M. Duval (1988): *A directed graph version of strongly regular graphs*, J. Combin. Th. A **47**, pp. 71–100, https://doi.org/10.1016/0097-3165(88)90043-X.

[3] F. Fiedler, M.H. Klin & M. Muzychuk (2002): *Small vertex-transitive directed strongly regular graphs*, Discrete Math. **255**, pp. 87–115, https://doi.org/10.1016/S0012-365X(01)00391-0.

[4] The GAP Group (2021): GAP – Groups, Algorithms, and Programming – a System for Computational Discrete Algebra, Version 4.11.1, 2021. http://www.gap-system.org.

[5] Š. Gyürki (2020): *Small directed strongly regular graphs*, Algebra Colloquium, **27(1)**, pp. 1130, https://doi.org/10.1142/S1005386720000036.

[6] Š. Gyürki & M. Klin (2014): *Sporadic examples of directed strongly regular graphs obtained by computer algebra experimentation*, in Gerdt, Vladimir P. (ed.) et al., *Computer algebra in scientific computing*, Lecture Notes in Computer Science **8660**, pp. 155–170, https://doi.org/10.1007/978-3-319-10515-4_12.

[7] Š. Gyürki (2016): *Infinite families of directed strongly regular graphs using equitable partitions*, Discrete Math. **339** pp. 2970–2986, https://doi.org/10.1016/j.disc.2016.06.005.

[8] A. Hanaki & I. Miyamoto (2021): *Catalogue of association schemes*, http://math.shinshu-u.ac.jp/~hanaki/as/.

[9] D. Holt & G. Royle (2020): *A census of small transitive groups and vertex-transitive graphs*, J. Symbolic Comput. **101**, pp. 5160, https://doi.org/10.1016/j.jsc.2019.06.006.

[10] M. Klin & Š. Gyürki (2015): *Selected Topics from Algebraic Graph Theory (Lecture notes),* Belianum, Banská Bystrica, 209 pages, ISBN 978-80-557-1063-1.

[11] M. Klin, A. Munemasa, M. Muzychuk & P.H. Zieschang (2004): *Directed strongly regular graphs obtained from coherent algebras,* Lin. Alg. Appl., **377**, pp. 83–109, https://doi.org/10.1016/j.laa.2003.06.020.

[12] M. Klin, Ch. Pech, S. Reichard, A. Woldar & M. Ziv-Av (2010): *Examples of computer experimentation in algebraic combinatorics*, Ars Math. Contemp. **3** no. 2, pp. 237–258, https://doi.org/10.26493/1855-3974.119.60b.

[13] M. Klin, Ch. Pech & S. Reichard (2021): COCO2P *package for* GAP, https://usermanual.wiki/Document/manual.1325036396/help.

[14] B. D. McKay (2009): nauty *Users Guide (Version 2.4)*, Department of Computer Science, Australian National University, Canberra, http://users.cecs.anu.edu.au/~bdm/nauty/nug.pdf.

[15] Ch. Pech & S. Reichard (2009): *The* SetOrbit *package for* GAP, http://www.math.tu-dresden.de/~pech.

[16] Ch. Pech & S. Reichard (2009): *Enumerating set orbits*. in: M. Klin et al., Algorithmic Algebraic Combinatorics and Gröbner Bases, Springer, Berlin, Heidelberg, pp. 137–150, https://doi.org/10.1007/978-3-642-01960-9_4.

[17] K.B. Reid & E. Brown (1972): *Doubly regular tournaments are equivalent to Hadamard matrices*, J. Combin. Th. A, **12**, pp. 332-338, https://doi.org/10.1016/0097-3165(72)90098-2.

[18] L.H. Soicher (1993): GRAPE*: A system for computing with graphs and groups*, Groups and computation (New Brunswick, 1991), DIMACS Ser. Discrete Math. Theoret. Comput. Sci., 11, pp. 287–291, Amer. Math. Soc., Providence, RI, https://doi.org/10.1090/dimacs/011/20.

[19] B. Weisfeiler, editor (1976): *On construction and identification of graphs*, Lecture Notes in Math., 558, Springer-Verlag, Berlin.

On the integrality of algebraic Witt vectors over imaginary quadratic fields

Yasuhiro Ishitsuka
Institute of Mathematics for Industry
Kyushu University, Japan
yasu-ishi@imi.kyushu-u.ac.jp

Takeo Uramoto
Institute of Mathematics for Industry
Kyushu University, Japan
uramoto@imi.kyushu-u.ac.jp

The aim of this short note is to report some progress on numerical computations of algebraic Witt vectors over imaginary quadratic fields K based on a previous work of the second author. In particular, when K is of class number one, we describe an effective procedure to decide whether a given algebraic Witt vector $\xi \in E_K = K \otimes W^a_{O_K}(O_{\bar{K}})$ is an integral Witt vector $\xi \in W^a_{O_K}(O_{\bar{K}})$.

1 Introduction

The aim of this short note is to report some progress on numerical computations of algebraic Witt vectors over imaginary quadratic fields K, which is a natural continuation of the previous work [10] of the second author. This work is also deeply related to the *algebraic automata theory* (e.g. [4]); for this background, the interested reader is referred to [7, 8]; see also §5.

In [10] the second author proved the *modularity theorem* (Theorem 4.3.1, [10]), which states that the *algebraic Witt vectors* over an imaginary quadratic field K (cf. §2) are precisely those vectors obtained by special values of certain deformation families of modular functions, called *modular vectors* (cf. §4.2 [10]); and moreover, in the stronger form of the modularity theorem (Theorem 4.4.1, [10]), he also proved that they are actually generated by the specific Witt vectors $\widehat{f_a}$ ($a \in \mathbb{Q}^2/\mathbb{Z}^2$) arising from a certain family of modular functions f_a called *Fricke functions* (§2). In particular, this gives an analytic method to construct algebraic Witt vectors over imaginary quadratic fields; from a more computational viewpoint, this result enables us to compute all of them explicitly by a numerical manner, and provides us the useful symbolic representation of (a priori mysterious) algebraic Witt vectors ξ as K-polynomials $\xi = F(\widehat{f_a})$, which are computationally enumerable and manageable.

The major result in this note concerns this symbolic computational aspect of [10], or in particular, the *decidability* of the integrality of algebraic Witt vectors over imaginary quadratic fields, which is related to a problem unsolved in [10]: In the case of the rational number field $\mathbb{Q}$, Corollary 3.3.1 in [10] gave an explicit characterization of which algebraic Witt vectors over $\mathbb{Q}$ are integral: that is, while algebraic Witt vectors over $\mathbb{Q}$ are given as $\mathbb{Q}$-linear combinations of the vectors of the form $\zeta^{(\gamma)} = (e^{2\pi i n \gamma})$ for $\gamma \in \mathbb{Q}/\mathbb{Z}$, they are integral if and only if they are $\mathbb{Z}$-*linear* combinations of $\zeta^{(\gamma)}$'s. For imaginary quadratic fields, on the contrary, a characterization of integral Witt vectors among algebraic ones remained unsolved in [10]. For the time being, we do not yet have a satisfactory solution to this problem; in fact, even in the case of the rational number field $\mathbb{Q}$, the characterization of the integral Witt vectors among algebraic ones (Corollary 3.3.1, [10]) heavily relies on (or is essentially due to) the non-trivial result of Borger and de Smit, Theorem 3.4 [2]; the case of imaginary quadratic fields is much worse in that we do not yet have even a non-trivial example of integral Witt vectors other than the one given by the j-function (cf. §4.1 [10]). In view of this, it seems reasonable to consider the *decidability* of the integral Witt vectors among algebraic ones as an auxiliary step toward a completely explicit characterization of them. In fact,

T. Kutsia (Ed.): Symbolic Computation
in Software Science (SCSS'21)
Short and work-in-progress papers, pp. 28–33

this decidability result enables us to enumerate all of integral Witt vectors in an algorithmic way, and hence, produces (in principle, all) examples of integral Witt vectors over imaginary quadratic fields; also this will be useful to formulate some conjecture on a characterization of integral Witt vectors.

This note reports an effective procedure to decide the integrality of algebraic Witt vectors, focusing on the case of imaginary quadratic fields of class number one. To be more specific, let $K = \mathbb{Q}(\sqrt{-d})$ be an imaginary quadratic field (of class number one) and O_K the ring of integers in K. The (strong version of) modularity theorem (Theorem 4.4.1 [10]) shows that the K-algebra E_K of algebraic Witt vectors over K is equal to the K-algebra $K[\widehat{f_a} \mid a \in \mathbb{Q}^2/\mathbb{Z}^2]$ generated over K by the specific (modular) vectors $\widehat{f_a}$ arising from the Fricke functions f_a ($a \in \mathbb{Q}^2/\mathbb{Z}^2$); therefore, each algebraic Witt vector $\xi \in E_K$ is specified by a K-polynomial $F(z_a) \in K[z_a \mid a \in \mathbb{Q}^2/\mathbb{Z}^2]$ with variables z_a indexed by $a \in \mathbb{Q}^2/\mathbb{Z}^2$ so that we have an identity $\xi = F(\widehat{f_a})$ in E_K. Our target problem here is to decide whether a given $\xi = F(\widehat{f_a})$ is integral or not looking at the polynomial $F(z_a)$.

There are two major technical challenges for solving this problem. First, from the analytic specification $\xi = F(\widehat{f_a})$ of each $\xi \in E_K$, a priori we can know only numerical approximations of the values of its coefficients $\xi_{\mathfrak{a}}$; in particular, in order to judge the integrality of ξ (as Witt vectors), we first need to judge whether the coefficients $\xi_{\mathfrak{a}}$ are algebraic integers only from their approximate numerical values. We will firstly overcome this issue by employing an idea from the theory of elliptic curves, namely, the idea of *division polynomials* (§3), which tells us upper bounds on denominators of $\xi_{\mathfrak{a}}$'s necessary to judge their integrality. The second challenge arises from the fact that, for $\xi \in E_K$ to be an integral Witt vector, it is *not* sufficient that the coefficients $\xi_{\mathfrak{a}}$'s are all algebraic integers; in fact, the distribution of $\xi_{\mathfrak{a}}$'s must be *smooth with respect to arithmetic derivations*, i.e. infinitely many differentiable by arithmetic derivations (cf. §2). This arithmetic smoothness is a priori an infinitary condition on ξ, thus, we need to reduce it to some finitary condition. In §4 we will overcome this issue by means of elementary module-theoretic argument: Informally speaking, we see that ξ is arithmetically smooth (i.e. differentiable inifinitely many times) if and only if it is differentiable at least some finite M times, where an upper bound of M is calculable for ξ; this upper bound comes from the finiteness of ξ over O_K, or rather, the Noetherianity of the integral closure of O_K within a certain finite K-algebra X_ξ (cf. §3.1 [10]).

This note describes the central ideas for this procedure; the detailed proofs will be given elsewhere. We summarize in §2 some necessary notations and terminology in [10, 9], and in §3 the idea of division polynomials in relation to the first technical challenge mentioned above. In §4 we then discuss the second technical challenge concerning the arithmetic smoothness of Witt vectors. The last section §5 is devoted to concluding remarks, including a few comments on the above-mentioned connection to the algebraic automata theory.

We are grateful to Prof. Mizoguchi for introducing us to SCSS 2021, and also to the anonymous reviewer, who provided several corrections on this note. The first author is supported by JSPS KAKENHI Grant-number 21K13773. This work is supported by 2021 Joint Use Research Program Short-term Joint Research "New interplays between algebraic language theory and classical class field theory".

2 Preliminaries

Throughout this note let $K = \mathbb{Q}(\sqrt{-d})$ be an imaginary quadratic field of class number one; we guess that this assumption on class number will be unnecessary but we assume it here because it is useful for describing our procedure (as discussed in §4). Due to the page limitation, we freely use the notations and terminology in [10, 9], to which the reader is referred for more detail. To be specific, we denote by E_K the K-algebra of *algebraic Witt vectors* and by $A_K = W^a_{O_K}(O_{\bar{K}})$ the ring of *integral Witt vectors*, whence

$E_K = K \otimes A_K$ (cf. §3.2 [9]). The set of maximal ideals of the integer ring O_K of K is denoted by P_K, while I_K denotes the monoid of the non-zero integral ideals of O_K. The Frobenius lifts $\psi_{\mathfrak{p}} : A_K \to A_K$ for each $\mathfrak{p} \in P_K$ extend to $\psi_{\mathfrak{p}} : E_K \to E_K$ and to $\psi_{\mathfrak{a}} : E_K \to E_K$ for $\mathfrak{a} \in I_K$ (cf. Remark 2, §2.2 [9]). Each algebraic Witt vector $\xi \in E_K$ is a vector of the form $\xi = (\xi_{\mathfrak{a}})_{\mathfrak{a} \in I_K} \in (K^{ab})^{I_K}$, where each $\xi_{\mathfrak{a}} \in K^{ab}$ is called the *component* or *coefficient* of ξ at $\mathfrak{a} \in I_K$; the operator $\psi_{\mathfrak{b}} : E_K \to E_K$ for each $\mathfrak{b} \in I_K$ acts on $\xi \in E_K$ *by the* $\mathfrak{b}$*-shift* in the sense that $\psi_{\mathfrak{b}}\xi = (\xi_{\mathfrak{b}\mathfrak{a}})_{\mathfrak{a} \in I_K}$.

Take $\pi_{\mathfrak{p}} \in O_K$ a uniformizer of each $\mathfrak{p} \in I_K$ so that $\mathfrak{p} = \pi_{\mathfrak{p}} O_K$; here let us define the *(arithmetic)* $\mathfrak{p}$*-derivation* $\delta_{\mathfrak{p}}\xi \in \bar{K}^{I_K}$ of $\xi \in O_{\bar{K}}^{I_K}$ by $\delta_{\mathfrak{p}}\xi := \pi_{\mathfrak{p}}^{-1}(\psi_{\mathfrak{p}}\xi - \xi^{N\mathfrak{p}})$.[1] In general, the components of $\delta_{\mathfrak{p}}\xi$ are not necessarily integers; but if every component of $\delta_{\mathfrak{p}}\xi$ is an integer, we shall say that ξ is *(arithmetically) differentiable* by $\mathfrak{p}$; also we simply say that ξ is *differentiable* if it is differentiable by any $\mathfrak{p} \in P_K$. With this terminology, we then say that $\xi \in O_{\bar{K}}^{I_K}$ is *(arithmetically) smooth* if it is differentiable inifitely many times; we also call such arithmetically smooth vectors $\xi \in O_{\bar{K}}^{I_K}$ as *Witt vectors* over O_K with coefficients in $O_{\bar{K}}$ (see Definition 1, §2.1 [9] for the general definition). The ring of these Witt vectors is denoted $W_{O_K}(O_{\bar{K}})$; the above-mentioned ring $A_K = W^a_{O_K}(O_{\bar{K}})$ of *integral* Witt vectors is the integral closure of O_K within $W_{O_K}(O_{\bar{K}})$, hence the name (cf. §3.2, [9]).

The arithmetic smoothness, together with the integrality over O_K, imposes strong constraints on the distribution of the values of $\xi_{\mathfrak{a}}$'s as discussed in [9]. This makes the structure of the ring A_K of integral Witt vectors as well as the K-algebra $E_K = K \otimes A_K$ rather mysterious, so that they are deeply relevant to the class field theory of K [3, 9, 10]. But in the case of imaginary quadratic fields K, how the coefficients of $\xi_{\mathfrak{a}}$'s for $\xi \in E_K$ can distribute could be explicitly described in [10] through the analytic construction of (modular) vectors $\widehat{f_a}$ ($a \in \mathbb{Q}^2/\mathbb{Z}^2$) using *Fricke functions* f_a (cf. §6 [5]). In particular, by construction of the modular vectors $\widehat{f_a}$ (cf. Example 4.2.4 and Lemma 4.2.8 [10]) and by virtue of Shimura's reciprocity law (cf. Theorem 6.31 [5]), we obtain the following basic facts:

Theorem 2.1. *Concerning the construction of the modular vectors* $\widehat{f_a}$*:*

1. *each coefficient* $\widehat{f_a}(\mathfrak{a}) \in K^{ab}$ *is calculable; and*
2. *the action of* $[s] \in G_K^{ab}$ *on* $\widehat{f_a}(\mathfrak{a}) \in K^{ab}$ *is calculable;*

where $[-] : \mathbb{A}_K^* \ni s \mapsto [s] \in G_K^{ab}$ *denotes the Artin map. Thus, by the strong modularity theorem (Theorem 4.4.1 [10]), the same is true for all* $\xi = F(\widehat{f_a}) \in E_K$.

Now we shall proceed to the problem of judging the integrality of algebraic Witt vectors. As outlined in §1, the major technical challenge is two-fold. Firstly, it is necessary for us to check whether each coefficient $\xi_{\mathfrak{a}}$ of an algebraic Witt vector $\xi = F(\widehat{f_a}) \in E_K$ is an algebraic integer, which is the subject of §3; for this task, we shall employ the idea of *division polynomials* [6] from the theory of elliptic curves. As remarked in §1 too, nevertheless, in order that $\xi \in E_K$ is an integral Witt vector (i.e. $\xi \in A_K$), it is not sufficient that every coefficient $\xi_{\mathfrak{a}}$ of ξ is an algebraic integer: the distribution of the integers $\xi_{\mathfrak{a}}$ must be smooth with respect to arithmetic derivations, namely, infinitely many differentiable by any $\mathfrak{p} \in P_K$. The subject of §4 is thus to develop an effective procedure of checking this arithmetic smoothness, based on an elementary module-theoretic idea.

[1] This makes sense only when K is of class number one (as we assume now) and depends on the choice of $\pi_{\mathfrak{p}}$'s. The notion of arithmetic smoothness defined here using $\delta_{\mathfrak{p}}$ is, however, equivalent to the general one in [9, 10]. The operator $\delta_{\mathfrak{p}}$, which is globally unavailable in general K, plays a central role in our procedure to decide the integrality of algebraic Witt vectors; cf. §4. In the case where K is not necessarily of class number one, we will need some localization argument in order to perform a similar algorithm; cf. §5.

3 Integrality of coefficients

The Fricke functions f_a, used in the construction of the above modular vectors $\widehat{f_a} \in E_K$, are defined in terms of certain Eisenstein series (cf. [5]); hence, we can compute the coefficients $\widehat{f_a}(\mathfrak{a})$ of the modular vectors $\widehat{f_a} \in E_K$ (= special values of f_a), and therefore those of all $\xi \in E_K$ as well, approximately with an arbitrary accuracy. The problem here is to judge whether the coefficients $\xi_{\mathfrak{a}}$ of $\xi = F(\widehat{f_a})$ are integers using only this approximate numerical values.

Concerning this issue, note first that, for any $\xi = F(\widehat{f_a}) \in E_K$, the indices $a \in \mathbb{Q}^2/\mathbb{Z}^2$ appearing in the polynomial $F(z_a) \in K[z_a \mid a \in \mathbb{Q}^2/\mathbb{Z}^2]$ are in $N^{-1}\mathbb{Z}^2/\mathbb{Z}^2$ for some $N \geq 1$, whence the coefficients $\xi_{\mathfrak{a}}$ of $\xi \in E_K$ all belong in the ray class field K_{NO_K} with the conductor $NO_K \in I_K$. Therefore, the integrality of $\xi_{\mathfrak{a}}$'s can be judged by checking the integrality of the symmetric functions $s(\xi_{\mathfrak{a}}^{\sigma} \mid \sigma \in Gal(K_{NO_K}/K)) \in K$ of the conjugates of $\xi_{\mathfrak{a}} \in K_{NO_K}$ over K, at least whose values are computable by Theorem 2.1. Of course, as mentioned above too, we can know only approximate values of $s = s(\xi_{\mathfrak{a}}^{\sigma} \mid \sigma \in Gal(K_{NO_K}/K)) \in K$; but if we could estimate upper bounds of their denominators, we can judge the integrality of such values s since we know that the (e.g. decimal) expansions of s_1, s_2 where $s = s_1 + s_2\omega_K$ with $O_K = \mathbb{Z} + \mathbb{Z}\omega_K$ are periodic (because $s_1, s_2 \in \mathbb{Q}$), and that the periods can be estimated by the size of the denominator of s; also we can compute their decimal expansions in any accuracy.

Moreover, since all $\xi \in E_K$ are expresented as K-polynomials of $\widehat{f_a}$'s, we can estimate the upper bound of the denominator of the above s for all $\xi \in E_K$ if we could do this just for the special values of f_a's at imaginary quadratics. We then solve this problem by employing the idea of *division polynomials* from the theory of elliptic curves (cf. [6]); in fact, division polynomials are explicitly computable and tell us minimal polynomials of the special values of f_a's, hence are useful to give upper bounds of their denominators. Therefore, using this fact, we can judge the integrality of the components of the modular vectors $\widehat{f_a}$ ($a \in \mathbb{Q}^2/\mathbb{Z}^2$), hence the integrality of the components $\xi_{\mathfrak{a}}$ of any $\xi = F(\widehat{f_a}) \in E_K$ as well.

4 Arithmetic smoothness

The results discussed above provide a basis for our target procedure to judge the integrality of algebraic Witt vectors, which we shall sketch below. The problem here is that, given $\xi = F(\widehat{f_a}) \in E_K$, we need to judge whether ξ belongs to A_K, or in elementary words, we need to judge whether the coefficients of the arithmetic derivatives $\delta_{\mathfrak{p}_1}\delta_{\mathfrak{p}_2}\cdots\delta_{\mathfrak{p}_N}\xi$ of ξ are integers for *all* $N \geq 1$ and *any* $\mathfrak{p}_1, \cdots, \mathfrak{p}_N \in P_K$; apparently, this itself is a priori impossible to check within finite steps. Therefore, our challenge here is to reduce this infinitary problem to some finitary one.

This reduction can be divided into two steps. First, for each $\xi \in E_K$, it suffices to consider arithmetic derivatives $\delta_{\mathfrak{p}}$ only for *finitely many* $\mathfrak{p} \in P_K$, depending on ξ: As discussed in §4.4 [10], each ξ belongs to a certain Λ-ring X_{NO_K}, which is, as a finite etale K-algebra, of the form $X_{NO_K} = \prod_{\mathfrak{d}|NO_K} K_{NO_K/\mathfrak{d}}$, where $K_{\mathfrak{f}}$ denotes the ray class field of K with conductor $\mathfrak{f} \in I_K$. Therefore, for those $\mathfrak{p} \in P_K$ coprime to NO_K, the action of $\psi_{\mathfrak{p}}$ on X_{NO_K} is equal to that of the Frobenius automorphism $\sigma_{\mathfrak{p}}$ on each component $K_{NO_K/\mathfrak{d}}$ of X_{NO_K}, hence on ξ as well. This implies that our problem for $\xi \in X_{NO_K}$ is reduced to considering the arithmetic derivatives $\delta_{\mathfrak{p}}$ only for those $\mathfrak{p}$ that divide NO_K; also $N \in \mathbb{N}$ can be chosen so that the indices $a \in \mathbb{Q}^2/\mathbb{Z}^2$ appearing in $\xi = F(\widehat{f_a})$ are all in $N^{-1}\mathbb{Z}^2/\mathbb{Z}^2$. Moreover, since $X_{NO_K} = \mathrm{Hom}_{G_K}(DR_{NO_K}, \bar{K})$, all the coefficients $\xi_{\mathfrak{a}}$ of $\xi \in X_{NO_K}$ are determined by those at $\mathfrak{a}_1, \cdots, \mathfrak{a}_k$ which represent the elements of the finite *ray class monoid* $DR_{NO_K} = \bigsqcup_{\mathfrak{d}|NO_K} C_{NO_K/\mathfrak{d}}$, where $C_{\mathfrak{f}}$ denotes the ray class group of K with conductor $\mathfrak{f} \in I_K$ (cf. §2.1 [10])

Nevertheless this first step only is insufficient for our goal; in fact, even with finitely many $\mathfrak{p} \mid NO_K$,

we still need to judge whether $\xi \in X_{NO_K}$ is differentiable *inifinitely many times* with respect to those finite $\mathfrak{p} \mid NO_K$. The second step to reduce this is to show that $\xi \in X_{NO_K}$ is differentiable infinitely many times with respect to all $\mathfrak{p} \mid NO_K$ if and only if it is differentiable at least some *finite M times* by those $\mathfrak{p} \mid NO_K$, an upper bound of which is calculable for $\xi \in X_{NO_K}$. We can show this fact based on the finiteness of the integral closure B_ξ of O_K in X_ξ (cf. §3.1 [10]); to be more specific, we shall estimate the above $M \in \mathbb{N}$ for ξ based on two facts on certain O_K-algebras D_n: Let us denote by $D_n \subseteq X_\xi$ for $n \in \mathbb{N}$ the O_K-subalgebra of X_ξ generated by r-times derivatives $\delta_{\mathfrak{p}_{i_1}} \cdots \delta_{\mathfrak{p}_{i_r}} \xi$ of ξ with respect to $\mathfrak{p}_{i_l} \mid NO_K$ up to $r \leq n$; this gives rise to the ascending sequence $D_0 \subseteq D_1 \subseteq \cdots \subseteq D_n \subseteq \cdots \subseteq X_\xi$ of O_K-subalgebras. Concerning this, note first that D_n becomes *full rank* in X_K (i.e. $K \otimes D_n = X_\xi$) after some n_1 which is explicitly calculable from our current N; this follows just from the facts that (i) for $X_\xi = K[\psi_\mathfrak{a}\xi \mid \mathfrak{a} \in I_K]$, we can choose finitely many $\mathfrak{a}$'s from representatives of DR_{NO_K} so that such finitely many $\psi_\mathfrak{a}\xi$'s already generate X_ξ; and (ii) $\psi_\mathfrak{p}\xi = \xi^{N\mathfrak{p}} + \pi_\mathfrak{p}\delta_\mathfrak{p}\xi$ and conversely $\delta_\mathfrak{p}\xi = \pi_\mathfrak{p}^{-1}(\psi_\mathfrak{p}\xi - \xi^{N\mathfrak{p}})$ by definition, hence $\psi_\mathfrak{p}\xi$ and $\delta_\mathfrak{p}\xi$ generate the same algebra over K together with ξ. Therefore, for all $n \geq n_1$, the O_K-algebras D_n are of full rank in X_ξ. Note also that we have either (I) some n-times derivative of ξ is judged as non-integral by means of §3 at some $n \geq n_1$, whence we conclude that ξ is not integral; or (II) the ascending sequence of D_n's terminates at some $n_2 \geq n_1$ because B_ξ is finite over O_K, thus by construction, the quotient O_K-module B_ξ/D_{n_1} is of finite cardinarity. The point in the latter case (II) is that (i) such n_2 can be estimated from ξ, and also (ii) if ξ is at least n_2-times differentiable for all $\mathfrak{p} \mid NO_K$, it follows that ξ is in fact differentiable infinitely many times (thanks to $D_{n_2} = D_{n_2+1} = \cdots$), whence we can conclude that ξ is an integral Witt vector. (Thus we can take $M = n_2$.) On the whole, this procedure is finitary, which is what we needed.

5 Concluding remarks

This note reported the decidability of the integrality of algebraic Witt vectors over imaginary quadratic fields of class number one. As demonstrated in this note, the strong modularity theorem (Theorem 4.4.1, §4 [10]) provides us a useful symbolic representation of algebraic Witt vectors over imaginary quadratic fields. Since this symbolic representation is general enough (i.e. available for any imaginary quadratic fields), it will be useful also for other computational problems on algebraic Witt vectors over imaginary quadratic fields, not limited to the decidability of the integrality; the computability result mentioned in §2 will be of general use. We expect that the assumption on the class number that we made in this note can be actually removed so that a similar algorithm works for any imaginary quadratic fields, in view of the good behavior of the Witt vector functor [1] with respect to the localization operations. Although this note postponed working on this problem, this natural generalization of our algorithm should be further investigated.

Finally, as briefly mentioned in §1 too, this work is deeply related to the algebraic automata theory (e.g. [4]); see [7, 8, 9]. Indeed, as the *arithmetic Christol theorem* [9] shows, the fact that the integrality of Witt vectors over O_K is equivalent to their *automaticity* (i.e. being generated by finite automata, cf. §3 [9]; equivalently, the orbit-finiteness with respect to the action of the Frobenius lifts $\psi_\mathfrak{p}$) is in some sense a key for our algorithm to terminate. This algorithmic aspect of [9, 10] relevant to their connection [7, 8] to the algebraic automata theory [4] would deserve being particularly highlighted here.

References

[1] James Borger (2011): *The Basic Geometry of Witt Vectors, I: The affine case*. Algebra and Number Theory 5(2), pp. 231–285, doi:10.2140/ant.2011.5.231.

[2] James Borger & Bart de Smit (2008): *Galois theory and integral models of* Λ*-rings*. Bulletin of London Mathematical Society 40(3), pp. 439–446, doi:10.1112/blms/bdn024.

[3] James Borger & Bart de Smit (2011): *Lambda actions of rings of integers*. ArXiv:1105.4662.

[4] Jean-Eric Pin: *Mathematical Foundation of Automata Theory*. Available from: `http://www.liafa.jussieu.fr/~jep/PDF/MPRI/MPRI.pdf`.

[5] Goro Shimura (1971): *Introduction to the arithmetic theory of automorphic functions*. Princeton University Press.

[6] Joseph Silverman (1986): *The arithmetic of elliptic curves*. Springer-Verlag.

[7] Takeo Uramoto (2016): *Semi-galois Categories I: The Classical Eilenberg Variety Theory*. In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'16)*, pp. 545–554, doi:10.1145/2933575.2934528.

[8] Takeo Uramoto (2017): *Semi-galois Categories I: The Classical Eilenberg Variety Theory*. ArXiv:1512.04389v4 (Extended version of LICS'16 paper).

[9] Takeo Uramoto (2018): *Semi-galois Categories II: An arithmetic analogue of Christol's theorem*. J. Algebra 508(2018), pp. 539–568, doi:10.1016/j.jalgebra.2018.04.033.

[10] Takeo Uramoto (2020): *Semi-galois Categories III: Witt vectors by deformations of modular functions*. ArXiv:2007.13367v5.

Towards algebraic specification and verification of airport baggage handling systems algorithms

Iakovos Ouranos
Hellenic Civil Aviation Authority, Heraklion, Greece
School of Applied Mathematical and Physical Sciences
National Technical University of Athens, Greece
iouranos@central.ntua.gr

Petros Stefaneas
School of Applied Mathematical and Physical Sciences
National Technical University of Athens, Greece
petros@math.ntua.gr

Kazuhiro Ogata
School of Information Science
Japan Advanced Institute of Science and Technology
ogata@jaist.ac.jp

Abstract. We deal with the formal modeling and verification of a complex industrial real time system, the airport baggage handling system. Such systems are critical for the smooth operation of each airport and usually their automation is based on PLCs (Programmable Logic Computers) and exhaustive testing. In our approach, we claim that before the implementation of such safety critical systems, at least the most important parts of them, should be formally analyzed. To this end, we have formally specified an abstract model of a baggage handling system and the window reservation algorithm which is used to avoid baggage collisions, with CafeOBJ algebraic specification language as a Timed Observational Transition System (TOTS). Then we express the critical safety properties of the system in terms of the same language, in order to prove them, using the proof score semi-formal method and the CafeOBJ term-rewriting system. During this ongoing work, we have proved the no collision property of baggages for two out of four logical windows of the collector conveyor, and we are now working the proof of the property for the third one. In this paper some interesting results and lessons learned are also presented.

1 Introduction

Safety critical industrial systems are systems that support difficult and complex tasks. In most cases, a malfunction of them can be very dangerous for people lives; can have environmental consequences and/or loss or severe damage to equipment/property. One important parameter of them is the software that supports their automation. Examples of safety critical systems can be found at medicine, infrastructure, transport, nuclear engineering, space flight, etc. In this paper we deal with the airport baggage handling system or BHS, which is one of the most complex airport operational systems. It is responsible for moving, controlling, screening, sorting and storing the passenger baggage from the check-in area to the departure gates. Because the system is mainly composed of a series of conveyors that are connected as a whole system, a bottleneck in any part of the system could possibly affect the whole system [4]. With respect to potential bottlenecks in the BHS, check-in systems have often been identified as problem areas, particularly because several input conveyors merge into a collector conveyor [5]. The algorithm which is used for controlling the merge configuration is called window reservation, and is a First-in-First-out (FIFO) based control logic. This effectively control the merge configuration by allocating a window, which is a logical space reserved for incoming baggage on a collector conveyor [4]. The current study models a part of an airport baggage handling system, which consists of four check-in counter desks

T. Kutsia (Ed.): Symbolic Computation
in Software Science (SCSS'21)
Short and work-in-progress papers, pp. 34–39

with their corresponding baggage conveyors and photocells together with a collector conveyor and the window reservation algorithm, as a Timed Observational Transition System [10]. We have also already verified the "no collision property" for two out of four logical windows, which say that There is always at most one baggage in each logical window of the collector conveyor. We mention that the version of the method we have applied, cannot be used for an arbitrary number of check-in counter desks. The rest of the paper is organized as follows. In Section 2, the Timed OTS/CafeOBJ method is presented in short. Section 3 describes initially informally the baggage handling system and then an abstract formal model is presented as a Timed OTS. Section 4 deals with verification of the no collision property, a safety property of the system and finally, section 5 concludes the paper.

2 The Timed OTS/CafeOBJ method

CafeOBJ [2] is an executable formal specification language based on constructor based order sorted algebras, etc. The OTS/CafeOBJ is a formal method in which a system is modeled as an observational transition system (OTS), its specification is described in CafeOBJ, and it is verified formally by using specification execution function in CafeOBJ, called the proof score method [8] [9]. The logical foundations of it are presented at [3]. The Timed OTS/CafeOBJ method [10] [6] is a version of the OTS/CafeOBJ method for modeling real time systems. The main advantage of these methods is that systems specification and verification is written in terms of equations, which are the most fundamental logical formulas, easy to learn and understand. For more details on the method, the reader can consult the above cited publications.

3 Description of the baggage handling system

3.1 Informal description

Figure 1 shows a part of a baggage handling system, consisting of four counter desks (*ci*) with their corresponding weight label (*wli*) and dispatch (*dsi*) conveyors. Each dispatch conveyor feeds the collector conveyor (*cc*) which then feeds the rest of the baggage handling system. The collector conveyor is divided into four logical windows (*wi*) with a fixed window size, where baggages are forwarded from the dispatch conveyors (*dsi*). The algorithm which assigns each baggage to a window is called the window reservation algorithm and works as follows: Each check-in counter (*ci*), is equipped with a foot-switch (*fsi*) and two conveyors, the weight/label conveyor (*wli*) and dispatch conveyor (*dsi*), Each conveyor has its corresponding photo sensor, which returns whether the conveyor is empty or not.

We assume that the collector conveyor is moving at a constant rate, one position (i.e. one logical window) at a time unit (t.u.). The *wli* conveyor is activated and feeds the *dsi*, whenever a) the *fsi* is pressed by the staff member of counter *ci*, b) the *wli* photo sensor returns that is not empty and c) *dsi* photo sensor returns that *dsi* conveyor is empty. After the baggage is transferred to the dispatch conveyor, the photo sensor of the corresponding *dsi* returns that the conveyor is not empty and the window reservation procedure begins. The system writes the request through a counter which increases whenever a baggage is transferred to a dispatch conveyor (each baggage has a unique number) and the checks are being performed. We assume that these checks are being performed instantly. The conditions under which the dispatch conveyor transfers a baggage to the assigned window after one (1) time unit, being synchronized with the collector conveyor, are as following: a) the baggage on the *dsi* has requested a window earlier than a downstream baggage, i.e. a baggage on a *dsj*, $j > i$, i.e. its counter is a smaller number, b) the

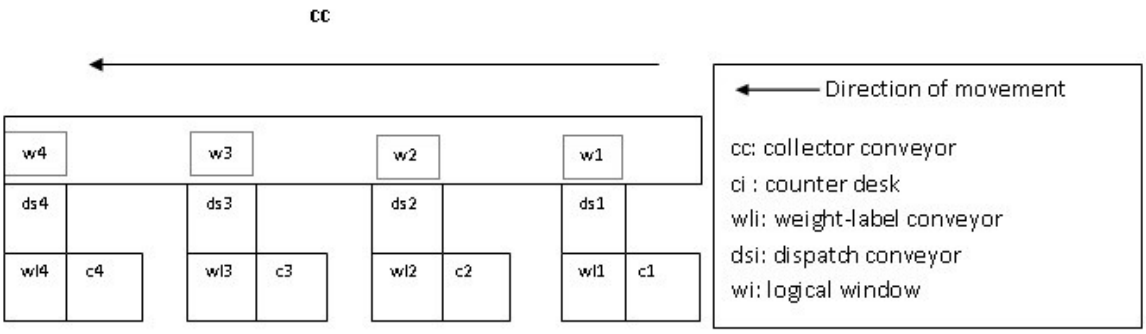

Figure 1: Baggage handling system

window w_{i-1}, called upstream, is empty and has not been assigned to another baggage. The above system has been formally specified as a Timed Observational Transition System and we have already proved that our specification meets the two out of four no-collision properties (one for each window) with the proof score approach and CafeOBJ term rewriting system. These safety properties say that There is always at most one baggage in each logical window of the collector conveyor.

3.2 Formal specifications

The specification of the system consists of CafeOBJ modules which models the datatypes used and the systems behavior as an observational transition system. More specifically, the modules:

- `BID` models the baggage ID
- `COUNTER` models a counter of natural numbers that increments by one when a bag arrives at a dispatch conveyor, requesting a window
- `LOC` models the location of a bag, i.e. the conveyor at which it resides
- `STATE` models the state of a window, i.e. whether it is assigned to a bag or not
- `TIMEVAL` models non-negative real numbers as stated above

The above modules import other CafeOBJ built-in modules such as `BOOL`, `NAT` that model Boolean and Natural numbers respectively.

- The module `BHS` models the systems behavior as an OTS. It imports the datatype modules prepared beforehand. The state space is denoted by the sort `Sys`. There are observations that return observable values of the system at a given state together with clock observers related to real time issues, the set of initial states, that return the observable values of the system at the initial state, and the set of actions or conditional transitions that change the state of the system under certain conditions, together with the time advancing transition tick.

The whole specification consists of 1680 lines of code and the most complex part of it, is the conditions under which the transition moving the dispatch conveyor can be applied. For example, in the case of the dispatch conveyor *ds2* the effective condition under which the action for moving the conveyor can be applied, consists of eight subcases written in a predicate, `c-move-d2(S, B1, B2, B3, B4)` where each subcase is connected with an `or` with the other. We have distinguished the subcases based on the state of each conveyor which represents a state of the system. For example the first and most

complex subcase, represents the state where w1 is empty, and the counter of baggage B2 is smaller than the counters of B3 and B4, and each dispatch conveyor has a baggage on it, B1 to B4, and the B2 has been assigned a window and the lower time bound is smaller than the current time.

```
eq c-move-d2(S, B1, B2, B3, B4) = (empty?(S, w1) and
(read c(S, B2) < read c(S, B3)) and
(read c(S, B2) < read c(S, B4)) and (loc(S, B1) = d1)
and (loc(S, B2) = d2) and (loc(S, B3) = d3) and
(loc(S, B4) = d4) and not empty?(S, d1) and not
empty?(S, d2) and not empty?(S, d3) and not empty?(S, d4)
and not state(S, B2) = assigned and (l-move-d2(S) <= now(S))) .
```

Then the lower and upper bounds of the dispatch transition are set to d1 time units later, as is the time needed for the movement of the collector conveyor by one position. The dispatch transition is applied then, and the baggage is forwarded to the collectors conveyor window. The corresponding effective condition is:

```
eq c-desp-ds2(S, B1) = (state(S, B1) =  assigned) and (loc(S, B1) = d2)
and (l-desp-ds2(S) <= now(S)) .
```

The non timing part of the effective condition for the movement of the collector conveyor is always `true`. The conveyor moves every `d1` time units and changes the location of a bag on the conveyor together with observer `empty?` of each window.

4 Verification with proof scores

We have already proved that our specification satisfies the non collision properties for windows *w1* and *w2* while now we are working the proof for the window *w3*. The non collision property is a safety property and says that There is always at most one baggage in each logical window of the collector conveyor. Since our system consists of four check in counters and corresponding windows, we have to prove four different invariants expressed as follows, in a module `INV`.

```
eq inv1(S, B, B') = ((loc(S,B) = w1) and (loc(S,B') = w1)) implies (B=B') .
eq inv2(S, B, B') = ((loc(S,B) = w2) and (loc(S,B') = w2)) implies (B=B') .
eq inv3(S, B, B') = ((loc(S,B) = w3) and (loc(S,B') = w3)) implies (B=B') .
eq inv4(S, B, B') = ((loc(S,B) = w4) and (loc(S,B') = w4)) implies (B=B') .
```

The method uses induction and we write a module `ISTEP1` in which basic formulas to prove in each inductive case are expressed as CafeOBJ terms. For the second invariant the predicate declared is `istep2`:

```
op istep2 : -> Bool .
eq istep2 = inv2(s, b1, b2) implies inv2(s', b1, b2) .
```

Then we write proof scores for each subcase. In such complex systems always lemmas are used that then need to be proved. For the second invariant we needed totally 12 more lemmas that was then

proved. Due to the complexity of the system we had to split the effective condition `c-move-di` into a set of equations without logical or. Without this splitting, the CafeOBJ system got stuck and the proof could not finished. To show the length of the proof, we mention that the basic proof for the second window (`proof02.mod`) is about 2500 line of code and used two lemmas, `inv12` and `inv21`:

```
inv12(S, B, L) = (loc(S, B) = L) implies not empty?(S, L) .
inv21(S, B) = (l-desp-ds2(S) <= now(S)) and not empty?(S, d2) and
loc(S, B) = d2 and state(S, B) = assigned implies empty?(S, w2) .
```

5 Conclusions and lessons learned

The proof score approach that we have used, is a semi formal theorem proving method. That is because the proofs are not automated, but written by humans who may omit a critical subcase of the proof, which leads to errors. As we have mentioned in our previous work [11], and encountered also in this case study, an incorrect systems specification may lead to unsuccessful verifications, which implies specification revisions/updates and verification retries. Especially in such a complex system with timing constraints, many verification retries and specification revisions were performed. The reason was basically systems functions misunderstanding, that lead to wrong descriptions. This version is the third one written by hand, and we are still make modifications to the proof passages and/or the specification documents. If someone wants to formally describe every aspect of a complex system in detail, then manual theorem proving is very difficult and time consuming, especially for inexperienced users. Abstracting from implementation details can be a way to shrink the code and the corresponding sub cases. In addition, more automated techniques can be used to tackle the errors arise when humans write proofs. Model checking and automated theorem provers may be used and researchers from the CafeOBJ and Maude community have developed such tools[1] [12] [13] [7]. On the other hand the approach we have used has the advantage of better understanding of the system which may help later, during the development and implementation stages. We believe that at least the most important parts of safety critical industrial systems should be formally analyzed before the implementation. To this end, formal techniques such as these mentioned in this paper are useful and need to be applied. In the future we plan to finish the verification of our model with the proof score approach, and then apply more automated techniques to compare with. In addition we plan to investigate how to specify and verify a system with n check-in counters and not with a fixed number, as in this paper, applying a revised method [6].

References

[1] Peter Csaba Ölveczky & José Meseguer (2000): *Real-Time Maude: A Tool for Simulating and Analyzing Real-Time and Hybrid Systems1 1Supported by DARPA through Rome Laboratories Contract F30602-C-0312, by Office of Naval Research Contract N00014-99-C-0198, and by National Science Foundation Grant CCR-9900334.* Electronic Notes in Theoretical Computer Science 36, pp. 361–382, doi:10.1016/S1571-0661(05)80134-3.

[2] Razvan Diaconescu & Kokichi Futatsugi (1998): *Cafeobj Report*. World Scientific, doi:10.1142/3831.

[3] Daniel Gin, Kokichi Futatsugi & Kazuhiro Ogata (2012): *Constructor-based Logics*. Journal of Universal Computer Science 18(16), pp. 2204–2233, doi:10.3217/jucs-018-16-2204.

[4] Gukhwa Kim, Junbeom Kim & Junjae Chae (2017): *Balancing the baggage handling performance of a check-in area shared by multiple airlines*. Journal of Air Transport Management 58, pp. 31–49, doi:10.1016/j.jairtraman.2016.08.017.

[5] Vu Thanh Le, James Zhang, Michael Johnstone, Saeid Nahavandi & Doug Creighton (2012): *A generalised data analysis approach for baggage handling systems simulation*. In: *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 1681–1687, doi:10.1109/ICSMC.2012.6377979.

[6] Masaki Nakamura, Shuki Higashi, Kazutoshi Sakakibara & Kazuhiro Ogata (2020): *Formal verification of Fischers real-time mutual exclusion protocol by the OTS/CafeOBJ method*. In: *2020 59th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, pp. 1210–1215, doi:10.23919/SICE48898.2020.9240272.

[7] Masaki Nakamura, Kazutoshi Sakakibara & Kazuhiro Ogata (2020): *Specification description and verification of multitask hybrid systems in the OTS/CafeOBJ method*. CoRR abs/2010.15280. Available at `https://arxiv.org/abs/2010.15280`.

[8] Kazuhiro Ogata & Kokichi Futatsugi (2003): *Proof Scores in the OTS/CafeOBJ Method*. In Elie Najm, Uwe Nestmann & Perdita Stevens, editors: *Formal Methods for Open Object-Based Distributed Systems*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 170–184, doi:10.1007/978-3-540-39958-2_12.

[9] Kazuhiro Ogata & Kokichi Futatsugi (2006): *Some Tips on Writing Proof Scores in the OTS/CafeOBJ Method*. In Kokichi Futatsugi, Jean-Pierre Jouannaud & José Meseguer, editors: *Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, *Lecture Notes in Computer Science* 4060, Springer, pp. 596–615, doi:10.1007/11780274_31.

[10] Kazuhiro Ogata & Kokichi Futatsugi (2007): *Modeling and verification of real-time systems based on equations*. *Science of Computer Programming* 66(2), pp. 162–180, doi:10.1016/j.scico.2006.10.011.

[11] Iakovos Ouranos, Kazuhiro Ogata & Petros Stefaneas (2014): *TESLA Source Authentication Protocol Verification Experiment in the Timed OTS/CafeOBJ Method: Experiences and Lessons Learned*. *IEICE Transactions on Information and Systems* E97.D(5), pp. 1160–1170, doi:10.1587/transinf.E97.D.1160.

[12] Adrián Riesco & Kazuhiro Ogata (2018): *Prove It! Inferring Formal Proof Scripts from CafeOBJ Proof Scores*. *ACM Trans. Softw. Eng. Methodol.* 27(2), doi:10.1145/3208951.

[13] Duong Dinh Tran & Kazuhiro Ogata (2020): *Formal verification of an abstract version of Anderson protocol with CafeOBJ, CiMPA and CiMPG*. In Raúl García-Castro, editor: *The 32nd International Conference on Software Engineering and Knowledge Engineering, SEKE 2020, KSIR Virtual Conference Center, USA, July 9-19, 2020*, KSI Research Inc., pp. 287–292, doi:10.18293/SEKE2020.

A Term-Rewriting Semantics for Imperative Style Programming: Summary

David A. Plaisted
Department of Computer Science
UNC Chapel Hill
Chapel Hill, North Carolina, USA
plaisted@cs.unc.edu

Lee Barnett
Institute for Formal Models and Verification
Johannes Kepler Universität
Altenbergerstraße 69, 4040 Linz, Austria
lee.barnett@jku.at

Abstract. Term rewriting systems have a simple syntax and semantics and facilitate proofs of correctness. However, pure functional languages in general are not as popular in industry or academia as imperative languages. We define a term-rewriting based abstract programming language with an imperative style and a precise semantics allowing programs to be translated into efficient imperative languages, to obtain proofs of correctness together with efficient execution. This language is designed to facilitate translations into correct programs in imperative languages with assignment statements, iteration, recursion, arrays, pointers, and side effects. It can also be used in place of a pseudo-programming language to specify algorithms.

1 Introduction

Term rewriting systems[1] have a simple syntax and semantics. Here a first-order term-rewriting based programming style is introduced that facilitates translations into imperative languages. The purpose of this system is to provide a way to express abstract algorithms that has a simple syntax and semantics but is also close to imperative languages in style. Then abstract algorithms can be written and verified in such a system and translated into many imperative programming languages, so that the translated programs are correct and efficient. This facilitates verification and also avoids the need to rewrite the same algorithm over and over again in many languages. At this stage only pure algorithms are considered, and features such as interrupts and input-output are not considered. After the abstract algorithm has been translated into a target language and inserted into a larger program, such features can be added in a language-specific way. The idea of an abstract language that can be translated into others was presented earlier [16, 17], but the abstract languages considered before were imperative and not fully specified or not specified at all.

The K framework [19] provides a way to specify the formal semantics of imperative programming languages. Some impressive semantics specifications have been done in it including the specification of the formal semantics of the C programming language [8]. The purpose of the present system is to provide an abstract term-rewriting based notation for imperative-style algorithms with a precise, accessible syntax and semantics. Unlike the K framework, which gives a formal semantics for arbitrary programs in an imperative language, the purpose of the present system is to provide a formalism for abstract programs which permits these programs to be translated into programs in a target imperative language with a semantics similar to the semantics of the abstract programs.

In the present system, abstract programs are written in a formalism that is pure and close to logical notation, so proofs of correctness may be easier than for imperative programs. This is due to the simple syntax and semantics of first-order term-rewriting systems. In order for a proof of correctness in this system to imply the correctness of a translated program, it is necessary that the semantics of this rewriting

T. Kutsia (Ed.): Symbolic Computation
in Software Science (SCSS'21)
Short and work-in-progress papers, pp. 40–45

system be similar to the semantics of imperative languages; in particular, modifying a data structure will cause all references to it also to be modified, which cannot happen in pure functional languages. This rewriting language is based on graph rewriting, which permits multiple pointers to the same structure.

This system is untyped. It is much simpler than other systems that have been proposed to give abstract descriptions of algorithms, such as Coq[6, 9] or other logical frameworks[10, 15], which may use the Curry-Howard isomorphism[13]. Thus this system may be easier for programmers to understand, though the ideas presented here can also be extended to more sophisticated formalisms. It is *not* the purpose of this language to have the most expressive features such as higher-order functions, nonlinear rules, overlapping rules or AC unification; the purpose is to keep the language as simple as possible while meeting its objectives.

A language based on these ideas is described in great detail in a previous unpublished paper [18] by the authors. Here we just review and slightly reformulate its basic features to make the ideas accessible to a wider community.

2 Syntax

The usual definitions of terms, rewrite rules, and so on will be assumed [1]. We concentrate on constructor systems that are complete in that all terms of the form $f(t_1,\ldots,t_n)$ for defined symbol f and constructor terms t_i are reducible. Also, we will use left-linear and non-overlapping (orthogonal) systems because they seem to correspond naturally to imperative programs. Left linearity is a reasonable restriction for a programming language in the style of an imperative language, because it corresponds to the lack of repeated formal parameters in imperative programming languages. Disjointness is also reasonable because it means that each term can be rewritten in at most one way.

A *procedure* for a defined function symbol f is a set of rewrite rules with all left-hand sides of the form $f(t_1,\ldots,t_n)$ for some terms t_i. If such a procedure consists of all rules in R of the specified form then it is called the R-procedure for f. A system R is *complete* if for all non-constructors f appearing in R, for all terms $f(t_1,\ldots,t_n)$ where the t_i are ground constructor terms, $f(t_1,\ldots,t_n)$ is reducible by the R-procedure for f.

Completeness corresponds to the fact that all inputs can be processed by typical imperative languages. It implies that all non-constructor ground terms are reducible; as a consequence, all normal forms of ground terms are constructor terms.

Definition 2.1. *A* OCC system *(called a COC system in the extended paper) is a constructor term rewriting system that is complete and orthogonal.*

Such systems seem to correspond naturally to imperative programs. In such a system R, if R is terminating, then for each ground term r there is a unique term s such that $r \Rightarrow !_R s$, and s must be a constructor term. Also, term rewriting systems and constructor terms automatically give records and references.

A possible extension to the current formalism would be to allow nondeterminism. Also, if R is not terminating, then one can naturally extend the semantics to allow infinite constructor terms as normal forms, but this will not be considered here.

2.1 Procedures and programs

In this system, conditional statements can be defined in a straightforward manner. Also, procedures and programs can be defined in this language in a straightforward manner. Some procedures are *compiled*;

an example is addition where 4 + 5 rewrites to 9. Such procedures can be implemented by procedures in the underlying language and need not be explicitly defined by term rewriting. They can be thought of as an infinite sets of ground rewrite rules.

Of interest are the following:

Replace:

$replace(i, y, f(x_1, \ldots, x_n)) \rightarrow f(x_1, \ldots, y, \ldots, x_n)$

where the i-th argument of f has been replaced by y.

One rule is needed for each non-constant constructor f and each i.

D-Replace:

There are also similar rules for $d\_replace(i, y, f(x_1, \ldots, x_n))$ that return the same value as $replace$ but are destructive, in that the term $f(x_1, \ldots, x_n)$ is not copied but the ith argument is replaced. This will be explained further in the following sections.

In order to simulate arrays with a number of elements declared at run time, it is necessary to extend the language in some manner with functions of variable arity.

3 Graph rewriting

The semantics of the language is essentially given by graph rewriting. Each node of the graph has an associated function symbol. If a node N has an associated function symbol f of arity k then it has k children ordered left to right. Suppose they are $N_1, \cdots, N_k$ and the term t_i is associated with N_i. Then the term associated with N is $f(t_1, \cdots, t_k)$. Common subterms can be represented by pointers to the same node, although two syntactically identical terms need not share structure. If a structure is destructively modified by a destructive assignment then all references to it will also be modified. This can cause side effects, which violate the semantics of functional programming and the usual semantics of term rewriting systems, but this conforms to the semantics of imperative programming languages. Such destructive assignments can also create circular structures that correspond to infinite terms.

The semantics of the present language can be seen as a sequence of graphs where each graph is obtained from the previous one by the execution of a rewrite rule. This covers both the terminating and nonterminating cases, and also handles programs that create circular structures. Nondeterminism can be handled by sets of sequences of graphs, or by branching sequences of graphs. In this way one obtains an operational semantics that is much simpler than a more axiomatic or denotational semantics. Perhaps such abstract programs should be called graph rewriting automata to make this formalization of semantics more acceptable to the research community.

A problem with pure functional languages is that it is difficult to efficiently implement array operations. In a pure functional language, replacing an element of an array entails creating a new copy of the array, which is inefficient in time and space. This can be helped some by representing arrays as binary trees, but there is still a logarithmic cost for array operations, which can be done in constant time in imperative languages. Also, it can be convenient to use side effects for some applications. Furthermore, in order that verification of abstract programs can be applied to verification of programs translated into imperative languages, the semantics of the abstract language has to model the semantics of imperative languages, which have side effects and which modify arrays in a destructive manner.

In the system previously described [18], the graph is represented as an equivalence relation on terms, and rewriting is replaced by *decorated rewriting*. For details see the referenced paper.

Another option is to use monads, but these are difficult to understand and even with monads, to get destructive operations one needs to have a state variable and use a single threaded style of programming.

It's not clear that anything is gained by this.

Up to this point the *base language* has been defined. In the referenced paper [18], extended features are defined in terms of the base language. Among these are assignment statements and iterative statements. However, every program in the extended language can be compiled down to a program in the base language.

4 Defining algorithms

Pseudocode is used in many algorithm texts to define algorithms. However, this code generally does not have a precisely specified syntax and semantics. Here is what a well-known algorithms text [7] says on page 16:

> In this book, we shall typically describe algorithms as programs written in a *pseudocode* that is similar in many respects to C, C++, Java, Python, or Pascal.

Another problem is that there is not a precise definition for what an algorithm is [20, 2, 12, 3, 14, 5, 11, 4]. The first algorithm in the text by Cormen et al [7] is insertion sort. However, it is not precisely defined how one tells if an algorithm is insertion sort or not.

While not claiming to solve this latter problem, the rewriting language can be used for pseudocode to define algorithms. It does have a precise syntax and operational semantics, so that in principle programs can be verified in it. The operational semantics is defined in terms of sequences of graphs. The language also has the appearance of imperative languages, making it suitable for defining algorithms in an imperative style. Furthermore, translations of this language into high-level programming languages can be written in a way that preserves the structure of the algorithm, in that assignment statements translate to assignment statements, iterative statements to iterative statements, procedure calls to procedure calls, and conditional statements to conditional statements. It is intended that programs resulting from such a translation will have a semantics similar to the semantics of the abstract program. Preserving the structure of the abstract program also makes the translated program easier to understand for the programmer and easier to maintain directly without going back to the abstract language for every change.

5 Conclusion

A language based on first-order term rewriting has been developed to have something of the style of imperative programming languages and to permit efficient translations into such languages. This language also permits destructive operations with side effects. The pure rewriting language is confluent, but the version with destructive operations is not. The approach given in this paper could be extended to more sophisticated languages such as Coq with highly developed type systems.

We wish to emphasize that this language has been made as simple as possible to meet the stated objective of providing an efficient and direct translation into imperative programming languages. Therefore extensions such as higher-order unification and AC-unification have not been added. This simplicity makes it easier for the typical programmer to understand the language and also makes the language easier to translate, interpret, and possibly compile. It also facilitates proofs of correctness in the language.

We also have tried to make the language as close as possible to typical imperative programming languages. This is even necessary if proofs of correctness of the abstract language are to imply the correctness of programs translated into typical imperative languages. It is our intention that assignment statements in the present language will translate to assignment statements in an imperative language,

conditional statements will translate to conditional statements, iterative statements will translate to iterative statements, procedure calls will translate to procedure calls, and recursion will translate to recursion. This means that the translated program will have essentially the same structure as the original program and hopefully a similar semantics. This gives the programmer a high degree of control over the form of the translated program and also makes it easier to translate the proof of correctness and to maintain and perform complexity analysis on the translated programs. This also makes the language attractive as a replacement for the pseudo-code often used in typical algorithms texts to specify algorithms, but without a rigorous abstract semantics.

In the current programming environment, the same algorithms are programmed again and again in different programming languages, but the structure of the code is essentially the same. It would be better to represent the algorithm once in an abstract language such as that presented here and then write translations into various high-level imperative languages. This also makes it more economical to verify the algorithm once in the abstract setting rather than verifying each implementation of it in a specific language; the algorithm will typically only be part of a program, and the entire program will still be verified in the target language.

We also want to emphasize that our concern is with algorithms such as shortest path algorithms, maximum flow algorithms, and others that may be used again and again. An algorithm can be seen as a function from abstract mathematical objects such as graphs to other abstract mathematical objects such as integers, as in a shortest path algorithm on graphs. We are not interested in programs that use many interrupts, for example, or even in entire programs, but only in portions of them that can be regarded as algorithms. There are many interesting algorithms such as garbage collection algorithms and storage allocation algorithms that are not directly covered by the present system. The present approach can be used to provide implementations of algorithms in larger programs in which some of the code is provided directly in the target language.

In order to avoid conflicts with procedures in the target language program, there should be a facility for renaming the defined symbols and constructors in the abstract or translated programs. Also, it would be good to have encapsulation to help with this. However, a detailed translation example is beyond the scope of this paper.

Another problem with the current approach to programming is that programs generally become obsolete after about 15 years and have to be updated or completely rewritten. It should be possible to write an algorithm once and never have to write it again. The system presented in this paper is designed to make this possible and is maximally accessible to the typical programmer.

References

[1] Franz Baader & Tobias Nipkow (1998): *Term Rewriting and All That*. Cambridge University Press, Cambridge, England, doi:10.1017/CBO9781139172752.

[2] Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser & György Turán, editors (2012): *SOFSEM 2012: Theory and Practice of Computer Science - 38th Conference on Current Trends in Theory and Practice of Computer Science, Špindlerův Mlýn, Czech Republic, January 21-27, 2012. Proceedings*. *Lecture Notes in Computer Science* 7147, Springer, doi:10.1007/978-3-642-27660-6.

[3] Andreas Blass, Nachum Dershowitz & Yuri Gurevich (2009): *When are two algorithms the same?* *Bulletin of Symbolic Logic* 15(2), pp. 145–168, doi:10.2178/bsl/1243948484.

[4] Andreas Blass & Yuri Gurevich (2003): *Abstract state machines capture parallel algorithms*. *ACM Transactions on Computational Logic* 4, pp. 578–651, doi:10.1145/1352582.1352587.

[5] Andreas Blass & Yuri Gurevich (2003): *Algorithms: A Quest for Absolute Definitions*. Bulletin of the EATCS 81, pp. 195–225.

[6] Thierry Coquand & Gerard Huet (1988): *The Calculus of Constructions*. Inf. Comput. 76(2-3), pp. 95–120, doi:10.1016/0890-5401(88)90005-3.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein (2009): *Introduction to Algorithms, Third Edition*, 3rd edition. The MIT Press, doi:10.2307/2583667.

[8] Chucky Ellison & Grigore Roşu (2012): *An Executable Formal Semantics of C with Applications*. In: *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'12)*, ACM, pp. 533–544, doi:10.1145/2103656.2103719.

[9] Yannick Forster & Gert Smolka (2018): *Call-by-Value Lambda Calculus as a Model of Computation in Coq*. Journal of Automated Reasoning 63, pp. 393–413, doi:10.1007/s10817-018-9484-2.

[10] M.J. Gordon & T.F. Melham, editors (1993): *Introduction to HOL: A Theorem-Proving Environment for Higher-Order Logic*. Cambridge University Press.

[11] Yuri Gurevich (2000): *Sequential Abstract-state Machines Capture Sequential Algorithms*. ACM Trans. Comput. Logic 1(1), pp. 77–111, doi:10.1145/343369.343384.

[12] Robin Hill (2015): *What an Algorithm Is*. Philosophy & Technology 56, doi:10.1007/s13347-014-0184-5.

[13] William A. Howard (1980): *The formulas-as-types notion of construction*. In J. P. Seldin & J. R. Hindley, editors: *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, Academic Press, pp. 479–490. Reprint of 1969 article.

[14] Yiannis N. Moschovakis (2001): *What Is an Algorithm?*, pp. 919–936. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-642-56478-9_46.

[15] L.C. Paulson (1994): *Isabelle: A Generic Theorem Prover*. LNCS Volume 828. Springer Verlag, New York, doi:10.1007/BFb0030541.

[16] David A. Plaisted (2003): *An Abstract Programming System*. CoRR cs.SE/0306028. Available at `http://arxiv.org/abs/cs.SE/0306028`.

[17] David A. Plaisted (2013): *Source-to-source translation and software engineering*. Journal of Software Engineering and Applications Vol.6 No.4A, pp. 30–40, doi:10.4236/jsea.2013.64A005.

[18] David A. Plaisted & Lee Barnett (2020): *A Term-Rewriting Semantics for Imperative Style Programming*. CoRR abs/2007.03075. Available at `https://arxiv.org/abs/2007.03075`.

[19] Grigore Roşu & Traian Florin Şerbănuţă (2010): *An overview of the K semantic framework*. The Journal of Logic and Algebraic Programming 79(6), pp. 397–434, doi:10.1016/j.jlap.2010.03.012.

[20] Moshe Y. Vardi (2012): *What is an Algorithm?* Communications of the ACM 55(3), pp. 5–5. doi:10.1145/2093548.2093549.

An Intelligent Solution to Detect Security Policy Violations in SDN Data Plane

Amina Sahbi[1], Faouzi Jaidi[1, 2] and Adel Bouhoula[3]

[1]University of Carthage, Higher School of Communication of Tunis (Sup'Com),
LR18TIC01 Digital Security Research Lab, Tunis, Tunisia.

[2]University of Carthage, National School of Engineers of Carthage, Tunis, Tunisia.

[3]Department of Next-Generation Computing, College of Graduate Studies
Arabian Gulf University, Kingdom of Bahrain

amina.essahbi@supcom.tn; faouzi.jaidi@supcom.tn; a.bouhoula@agu.edu.bh

Abstract. Nowadays, emerging Software Defined Networks (SDN) present an innovative and a promotional research axis thanks to their advantages compared to conventional networks. Nevertheless, security aspects within SDN networks are still challenging tasks because of the rapid expansion and the dynamic updates of these networks. Security and cyber-security incidents such as nodes bad behaviors, configurations anomalies, vulnerabilities, attacks and intrusions, forwarding problems and miss-configurations are crucial issues for SDN. To deal with cyber-security within SDN, industrials and researchers need to improve new methods and approaches for detecting infected nodes and cyber-attacks. In this paper, we present an approach for detecting and resolving security policy violations in SDN Data Plane based on the benefit of Machine Learning Algorithms and Artificial Intelligence. The main contribution of this work is the proposition of a comprehensive online, dynamic and intelligent solution to efficiently detect and resolve different security policy violations.

Keywords: Software Defined Networks; SDN Data Plane; Security; Machine Learning; Security Policy.

1 Introduction

Due to the complexity of configuration, the expensive costs and the rigidity of conventional network architecture, the appearance of the new paradigm Software Defined Networks (SDN) was a revolutionary potential to resolve these problems by centralizing the control over the network. SDN break the vertical integration of the conventional network architecture. In this way, SDN solve the problem of interdependence between the control plane and the data plane. In addition, SDN provide more programming capabilities for the network. They enable network operators to manage their infrastructures in a simple and dynamic way.

However, security issues remain a significant concern and impede SDN from being widely adopted. SDN and security have a reciprocal relationship. These networks are very vulnerable to external as well as internal attacks that can easily overload the SDN resulting in critical degradation of the network performance. One of the main security research challenges is ensuring the security of the infrastructure itself. SDN Data plane has to be continuously monitored for detecting policy violations or suspicious traffic such as Intrusions or DOS attacks. The need of a system that can detect any harmful activities and generate results to the management authority is a necessity.

In order to detect and resolve security policy violations, we had recourse to Machine learning, artificial intelligence, and other modern statistical methods. They are providing new opportunities to operationalize previously untapped and rapidly growing sources of data benefit. We aim to optimize our

T. Kutsia (Ed.): Symbolic Computation
in Software Science (SCSS'21)
Short and work-in-progress papers, pp. 46–51

approach performance by using different machine learning algorithms such us Support Vector Machines (SVMs), k-Nearest Neighbor (k-NN), Deep Learning, neural networks, etc. Our main goal is to propose a mature solution that allows solving the current challenges of the security of the SDN data plane, in a precise and online way; we optimize our approach via the artificial intelligence benefits.

The rest of the manuscript is organized as follows: in Section 2, we present the state of the art and discuss the security challenges within SDN. We review and discuss related works in Section 3. In Section 4, we introduce and technically detail our proposed solution and discuss different perspectives. Finally, in Section 5, we present our conclusion and ongoing works.

2 State of the Art

The dynamic and programmable nature of SDN faces many security issues, which demand innovative and smart security solutions. To do so, it is necessary to improve SDN security by understanding their weaknesses.

2.1 SDN Security

Providing and managing security in computer networks is a complex task. The network administrator must prevent the network from internal and external intrusions. Network security aims to protect and defend network assets to preserve their security dimensions. Security protection includes detection, prevention and mitigation mechanisms against network threats, before the occurrence of network attacks. Security defense includes detection, remediation, and mitigation against the actions of network attackers. Security dimensions properties are classified into categories like illustrated by figure 1.

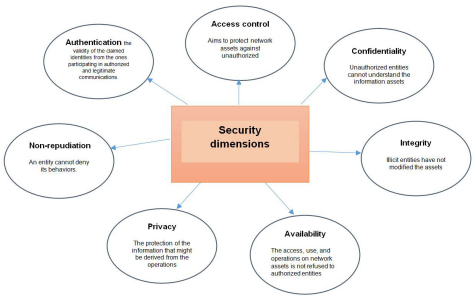

Figure 1: The basic properties of a secure communications network

The relation between SDN and security is bipolar. On the one hand, a SDN enables the improvement of network security thanks to its concepts. In fact, SDN centralization provides means for the construction of a holistic knowledge of the network. However, on the other hand, the SDN concept brings out new attack vectors that were not present in the conventional architecture. It intensifies the severity of some attacks. For example, an attacker who takes the commands of a controller will be able to access other SDN components and corrupt them [2]. An adversary can even use a SDN asset as an attack vector or a zombie to assault another one. Moreover, the breakdown of the controller will result in the unavailability of services running in the application layer. An attacker can even take advantage of this situation to replace the legitimate controller with its corrupted controller. This usurpation will give him the commands of the controller. Due to the sturdy dependability and reachability between SDN layers, an attack on a layer can affect the other layers [4]. Depending on its sophistication and complexity, it can open new

breaches towards other layers. In addition, it can reduce their performance [4]. An attacker can exploit APIs vulnerabilities to access SDN resources in different layers. He can modify network traffic and even SDN functionalities. Also, he can insert itself in the middle of two layers to snoop the communications between SDN entities [4]. Therefore, interfaces need to integrate into their heart security mechanisms such as access control, encryption and integrity checks to protect the network. Network elements rely entirely on the control layer. The latter specifies their forwarding behavior. In the case of the connectivity between both layers failed, network elements will not be able to handle network traffic because they do not have any intelligence. In this case, the forwarding and routing capabilities of network elements become inoperative, and they induce the network elements to drop the traffic. As a result, the network becomes unreliable. An attacker can exploit this issue to take control of network elements. For example, it initiates a first threat that disconnects a network element from its controller. Then, it connects an impersonated controller to this network element to command it. Consequently, the attacker reprograms all the connected network elements [8]. We present in table 1 a summary of security issues and attacks with regards to affected or targeted layers of SDN.

Table 1: Security issues within SDN layers.

Security Issue/Attack		Unauthorized Access		Data Leakage		Data Modification	Malicious Applications		Denial of Service		Configuration Issues	
		Unauthorized Controller Access	Unauthenticated Application	Flow Rule Discovery (Side Channel Attack on Input Buffer)	Forwarding Policy Discovery (Packet Processing Timing Analysis)	Flow Rule Modification to Modify Packets	Fraudulent Rule Insertion	Controller Hijacking	Controller-Switch Communication Flood	Switch Flow Table Flooding	Lack of TLS (or other Authentication Technique) Adoption	Configuration Issues
SDN Layer Affected or Targeted	Application Layer		✓				✓					✓
	Control Layer	✓	✓			✓	✓	✓	✓		✓	✓
	Data Layer	✓		✓	✓	✓		✓	✓	✓	✓	

2.2 Data Plane Security

The data plane layer consists of thousands of switches that are responsible for forwarding the packets. It is important to identify possible threats and corresponding countermeasures in this layer, as switches are the direct entry point to network access for end users. Attackers could target the network elements from within the network itself. An attacker could theoretically gain unauthorized physical or virtual access to the network or compromise a host that is already connected to the SDN and then try to perform attacks to destabilize the network elements. This could be a type of Denial of Service (DoS) attack or it could be a type of fuzzing attack to try to attack the network elements [3]. Possible security threats in this layer are Man-In-The-Middle attacks that occur between the switch and the controller, denial of service (DoS) attacks or attempted attack on network components DATA LAYER . Added to the external possible attacks, switches misconfigurations have a direct impact on the security and the efficiency of the network. There are a list of network confusions that may occur during different network update scenarios, such as *forwarding black hole*, *forwarding loop*, *link congestion* and *network policy violation*.

2.3 Security Policy Violations

In SDN-based networks, it is easier to define security policies dynamically using system properties and network statistics than in traditional networks where most security mechanisms are managed using static security policies. Besides, network policies can also produce new vulnerabilities such as opening access to unauthorized intruders or blocking an authorized source.

Access attacks enable an unauthorized entity to penetrate into the network. These attacks abuse the access control and authentication security dimensions. They exploit vulnerabilities in network policies and network misconfigurations to gain access. Violation of SDN data plane integrity might lead to abnormal behaviors of the overall network. During the network policy update, packets may violate the policies. Such confusion is known as network policy violation. This challenge can influence the path parsed by some packets. More, flow rules may overlap each other in the same switch or between switches which cause indirect network breaches. Depending on the complexity of an overlap found in violated space, we distinguish between two types of access violations: (i) *Entire Violation*: if the fields domain of the decision path covers the whole space (denied or accepted) of the security policy; and (ii) *Partial Violation*: if the fields domain of the decision path partially covers the space of the security policy. Figure 2 highlights a list of network confusions that may occur during different network update scenarios.

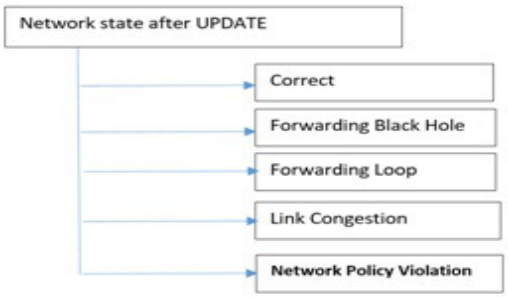

Figure 2: Problems related to network updates.

3 Related works

There are several research works about the security of SDNs data plane due to the importance of this layer. In [5], the authors focus on identifying challenges faced in securing the data plane of SDN. They have presented a survey of available solutions and their limitations as well as future research recommendations. They presented a covert channel defender (CCD) that prevents covert channel attacks by verifying and resolving rule conflicts. Specifically, CCD tracks all rule insertion and modification messages from applications running on the controller. It analyzes the correlation among rules based on multiple packet header fields and resolves any identified rule conflict in real time before rule installation. In [7], the authors proposed a comprehensive discovering and fixing of data plane security invariants. The solution considers Flow entries Decision Diagram (FeDD) as data structure and relies on formal techniques for analyzing the policy defects and resolving misconfigurations. It allows ensuring that the operators policies are correctly applied in an optimal way. It is an offline approach to fix violations at data plane side and a fine-grained control of SDN switches flow tables. A detection method in [1], has been proposed to find compromised switches in data plane by analyzing the behavior of these switches. This method depends on how a switch handles incoming packets which contravene the controller rules then the switch is considered compromised. The design is composed of two algorithms for detection: Forwarding Detection and Weighting Detection which has been implemented in Ryu controller.

Despite research efforts and valuable proposals addressing the security concept of SDN data plane, it is still not mature and represents an open challenge. Indeed, most of existing solutions are offline based approaches for detecting or correcting defects. More, the scalability of a lot of proposals is not

verified as well the overhead associated to several solutions constitutes a handicap. Taking benefits from emerging technologies may help in resolving the issue. Particularly, by using an AI-based approach we aim to define a smart solution with a better accuracy to deal with this crucial aspect taking into account the management of false positive and negative scenarios.

4 Proposed Solution

Our proposed approach is to detect and correct different types of access violations of the security policy in the SDN data plane. Our goal is to provide a complete, online and offline, reliable and performed solution using the advantages of Artificial Intelligence (AI). As input, our SDN network topology (flow tables, network traffic) is considered. SDN allows a logically centralized control, global view of the network, software-based traffic analysis, and dynamic updating of forwarding rules. The detection and resolution processes are linked to the security policy, composed of: (i) firewalls rules (FWR); (ii) access control lists (ACL); (iii) controllers policies (CTRLP). In order to efficiently organize, manage, maintain

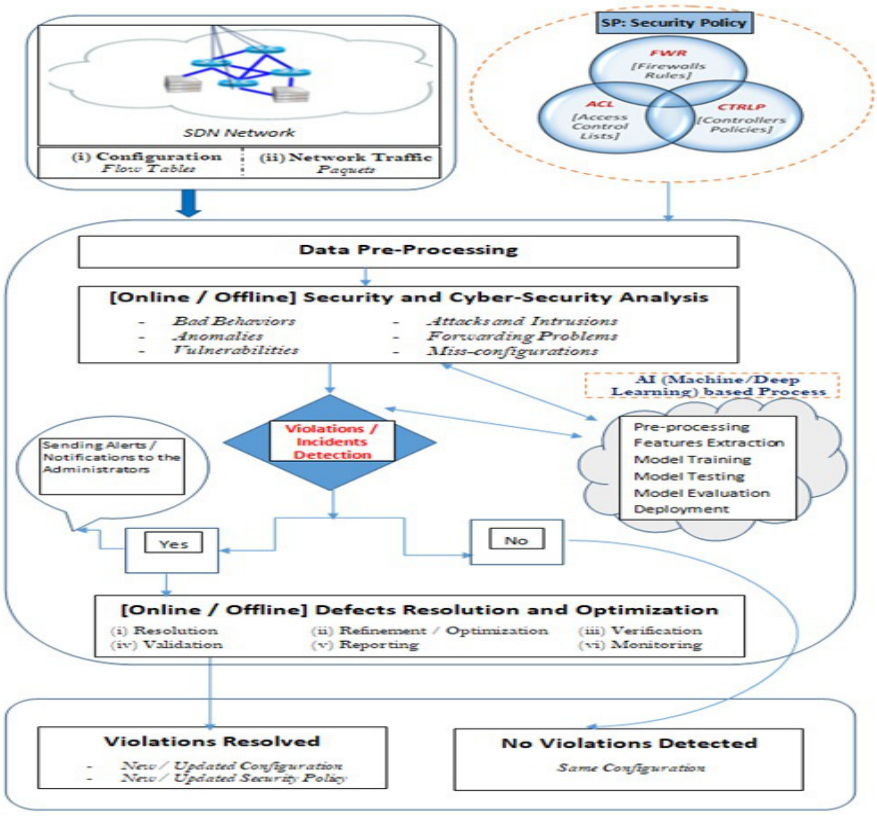

Figure 3: Smart security approach for SDN

and optimize networking systems, more intelligence needs to be deployed. We propose to apply machine learning techniques. The first step is Data Pre-Processing [6], by collecting and preparing data. Then we will extract features from rules (Source IP address, Destination IP address, Port destination and Action). Our online/offline approach aims to analyze security and cyber-Security properties and invariants to detect any bad behaviors, anomalies, vulnerabilities, attacks and intrusions, forwarding problems or

miss-configurations. Second, we train the model to detect and mitigate some attacks and intrusions or bad behaviors, misconfigurations of the policies. We test and evaluate the model, and finally we deploy the machine learning algorithm. If no incidents are detected we keep the same configuration. If any violation is detected, an alert will be send to the administrator and the process of Resolution is triggered. As output, violations will be resolved and a new updated configuration and an updated Security Policy will be set.

Figure 3 details our proposed architecture, as input we have the SDN network, and a security policy. Our approach will detect access violations by using machine learning techniques, which will improve the accuracy and reduce the time of processing. Once the detection step is validated, the next step will be to resolve the various violations as well as to optimize the network.

5 Conclusion

The emergence of Big Data has led to the need for dynamic networks that offer the possibility of quickly adapting to changes in requirements. Software Defined Networks (SDN) emerged as an innovative and a promotional solution with several advantages compared to conventional networks while security mechanisms such as access control, encryption and integrity checks to protect the network, must be preserved in a real time context. Our proposal consists of an online solution that can assure the detection and the resolution of access and security policies violations.

References

[1] Po-Wen Chi, Chien-Ting Kuo, Jing-Wei Guo & Chin-Laung Lei (2015): *How to detect a compromised SDN switch*. In: *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, IEEE, pp. 1–6, doi:10.1109/NETSOFT.2015.7116184.

[2] Paulo César Fonseca & Edjard Souza Mota (2017): *A survey on fault management in software-defined networks*. *IEEE Communications Surveys & Tutorials* 19(4), pp. 2284–2321, doi:10.1109/COMST.2017.2719862.

[3] Ghandi Hessam, Ghassan Saba & M Iyad Alkhayat (2021): *A new approach for detecting violation of data plane integrity in Software Defined Networks*. *Journal of Computer Security* (Preprint), pp. 1–18, doi:10.3233/JCS-200094.

[4] Deepak Kumar & Manu Sood (2014): *Software Defined Networking: A Concept and Related Issues*. *International Journal of Advanced Networking and Applications* 6(2), p. 2233.

[5] Qi Li, Yanyu Chen, Patrick PC Lee, Mingwei Xu & Kui Ren (2018): *Security policy violations in SDN data plane*. *IEEE/ACM Transactions on Networking* 26(4), pp. 1715–1727, doi:10.1109/TNET.2018.2853593.

[6] Raihan Ur Rasool, Khandakar Ahmed, Zahid Anwar, Hua Wang, Usman Ashraf & Wajid Rafique (2021): *CyberPulse++: A machine learning-based security framework for detecting link flooding attacks in software defined networks*. *International Journal of Intelligent Systems*, doi:https://doi.org/10.1002/int.22442.

[7] Wejdene Saied, Faouzi Jaidi & Adel Bouhoula (2020): *A Comprehensive Solution for the Analysis, Validation and Optimization of SDN Data-Plane Configurations*. In: *2020 16th International Conference on Network and Service Management (CNSM)*, IEEE, pp. 1–7, doi:10.23919/CNSM50824.2020.9269124.

[8] Arash Shaghaghi, Mohamed Ali Kaafar, Rajkumar Buyya & Sanjay Jha (2020): *Software-defined network (SDN) data plane security: issues, solutions, and future directions*. *Handbook of Computer Networks and Cyber Security*, pp. 341–387, doi:10.1007/978-3-030-22277-2_14.

Author Index