

Arnold Beckmann  
Costas Dimitracopoulos  
Benedikt Löwe (Editors)



## Logic and Theory of Algorithms

---

Fourth Conference on Computability in Europe, CiE 2008  
Local Proceedings

June 15–20, 2008

University of Athens



## Preface

**CiE 2008: Logic and Theory of Algorithms**  
Athens, Greece, June 15–20, 2008



*Computability in Europe* (CiE) is an informal network of European scientists working on computability theory, including its foundations, technical development, and applications. Among the aims of the network is to advance our theoretical understanding of what can and cannot be computed, by *any* means of computation. Its scientific vision is broad: computations may be performed with discrete or continuous data by all kinds of algorithms, programs, and machines. Computations may be made by experimenting with any sort of physical system obeying the laws of a physical theory such as Newtonian mechanics, quantum theory, or relativity. Computations may be very general, depending on the foundations of set theory; or very specific, using the combinatorics of finite structures. CiE also works on subjects intimately related to computation, especially theories of data and information, and methods for formal reasoning about computations. The sources of new ideas and methods include practical developments in areas such as neural networks, quantum computation, natural computation, molecular computation, computational learning. Applications are everywhere, especially, in algebra, analysis and geometry, or data types and programming. Within CiE there is general recognition of the underlying relevance of computability to physics and a broad range of other sciences, providing as it does a basic analysis of the causal structure of dynamical systems.

The conference was based on invited tutorials and lectures, a set of special sessions on a range of subjects as well as contributed papers and informal presentations. The formal proceedings volume appeared in the Springer LNCS series, Vol. 5028 and contains 25 of the invited lectures and 36 of the submitted contributed papers. The present volume represents the informal abstract booklet which contains another 86 of the contributed talks and two invited plenary talks as well as 23 abstracts of informal presentations. The informal abstract booklet is not a scholarly publication, and papers appearing in it should not be considered published or as selected in a peer review process. There will be a number of post-proceedings publications, including special issues of *Theory of Computing Systems*, *Archive for Mathematical Logic*, and *Journal of Algorithms*.

The first three meetings of CiE were at the University of Amsterdam in 2005, at the University of Wales Swansea in 2006, and at the University of Siena in 2007. The large number of mathematicians and computer scientists attending these conference had their view of computability theory enlarged and transformed: they discovered that its foundations were deeper and more mysterious, its technical development more vigorous, its applications wider and more challenging than they had known. The Athens meeting promised to extend and enrich that process.

The annual CiE conference, based on the Computability in Europe network, has become a major event, and is the largest international meeting focused on computability theoretic issues. The series is coordinated by the CiE Conference Series Steering Committee:

Arnold Beckmann (Swansea)  
 Paola Bonizzoni (Milan)  
 S. Barry Cooper (Leeds)  
 Benedikt Löwe (Amsterdam, Chair)

Elvira Mayordomo (Zaragoza)  
 Dag Normann (Oslo)  
 Peter van Emde Boas (Amsterdam).

We will reconvene 2009 in Heidelberg and 2010 in Ponta Delgada (Açores).

## Organization and Acknowledgements

The conference CiE 2008 was organized by: Dionysis Anapolitanos (Athens), Arnold Beckmann (Swansea), Costas Dimitracopoulos (Athens, Chair), Michael Mytilinaios (Athens) †, Thanases Pheidas (Heraklion), Stathis Zachos (Athens and New York NY).

The Programme Committee was chaired by Arnold Beckmann and Costas Dimitracopoulos:

Luigia Aiello (Rome)  
 Thorsten Altenkirch (Nottingham)  
 Klaus Ambos-Spies (Heidelberg)  
 Giorgio Ausiello (Rome)  
 Arnold Beckmann (Swansea)  
 Lev Beklemishev (Moscow)  
 Paola Bonizzoni (Milan)  
 Stephen A. Cook (Toronto ON)  
 Barry Cooper (Leeds)  
 Costas Dimitracopoulos (Athens)  
 Rod Downey (Wellington)  
 Elias Koutsoupias (Athens)  
 Orna Kupferman (Jerusalem)  
 Sophie Laplante (Orsay)  
 Hannes Leitgeb (Bristol)  
 Benedikt Löwe (Amsterdam)  
 Elvira Mayordomo (Zaragoza)

Franco Montagna (Siena)  
 Michael Mytilinaios (Athens) †  
 Mogens Nielsen (Aarhus)  
 Isabel Oitavem (Lisbon)  
 Catuscia Palamidessi (Palaiseau)  
 Thanases Pheidas (Heraklion)  
 Ramanujam (Chennai)  
 Andrea Schalk (Manchester)  
 Uwe Schöning (Ulm)  
 Helmut Schwichtenberg (Munich)  
 Alan Selman (Buffalo NY)  
 Andrea Sorbi (Siena)  
 Ivan Soskov (Sofia)  
 Christopher Timpson (Oxford)  
 Stathis Zachos (Athens and New York NY)

We are delighted to acknowledge and thank the following for their essential financial support: City of Athens, Bank of Greece, Graduate Program in Logic, Algorithms and Computation (MPLA), Hellenic Ministry of Education, John S. Latsis Foundation, Kleos S.A., National and Kapodistrian University of Athens, Public Power Corporation S.A., Rizareio Foundation, The Elsevier Foundation.

The high scientific quality of the conference was possible through the conscientious work of the Programme Committee, and the special session organizers. While papers in this abstract booklet are not to be considered publications, some of the papers in this volume greatly benefitted from comments of the referees during our reviewing process, and we would like to thank the referees for their work. The list of referees is given on pp. IX - X of the mentioned LNCS volume for this conference.

We thank Andrej Voronkov for his EasyChair system which facilitated the work of the Programme Committee and the editors considerably. We also thank Nikolaos S. Papaspyrou for his great support in producing this abstract booklet.

Swansea, Athens and Amsterdam, June 2008

Arnold Beckmann  
Costas Dimitracopoulos  
Benedikt Löwe

# Table of Contents

## Invited Talks.

The Computing Species .....	1
<i>Keith Devlin</i>	

Logic, Automata, Games, and Algorithms .....	2
<i>Moshe Y. Vardi</i>	

## Contributed Talks.

On the Performance of Automata Minimization Algorithms .....	3
<i>Marco Almeida, Nelma Moreira, Rogério Reis</i>	

Decidability Results for Mobile Membranes derived from Mobile Ambients	15
<i>Bogdan Aman, Gabriel Ciobanu</i>	

Sophisticated Infinite Sequences .....	25
<i>Luis Antunes, André Souto</i>	

On Detecting Deadlock in the Pi-Calculus .....	35
<i>Tiago Azevedo, Mario Benevides, Fábio Protti, Marcelo Sihman</i>	

Algorithms for Analytic Functions and Applications to Toeplitz Operators	45
<i>Edwin Beggs, Annelies Gerber</i>	

Triviality and Minimality in the Degrees of Monotone Complexity .....	55
<i>William Calhoun</i>	

Extraction of Efficient Programs from Correct Proofs: The Case of Structural Induction over Natural Numbers .....	64
<i>Luca Chiarabini</i>	

Phase Shifts of LFSM as Pseudorandom Number Generators for BIST for VLSI .....	77
<i>Sung-Jin Cho, Un-Sook Choi, Han-Doo Kim, Yoon-Hee Hwang, Jin-Gyoung Kim</i>	

Introducing Service Schemes and Systems Organization in the Theory of Interactive Computation .....	87
<i>Antônio Carlos Costa, Graçaliz Dimuro</i>	

Online-division with Periodic Rational Numbers .....	97
<i>Gregorio de Miguel Casado, Juan Manuel García Chamizo, Higinio Mora Mora</i>	

Abstract Geometrical Computation with Accumulations: Beyond the Blum, Shub and Smale model .....	107
<i>Jérôme Durand-Lose</i>	
Notions of Bisimulation for Heyting-Valued Modal Languages .....	117
<i>Pantelis Eleftheriou, Costas Koutras, Christos Nomikos</i>	
Algorithmic Properties of Structures for Languages with Two Unary Functional Symbols .....	127
<i>Ekaterina Fokina</i>	
Verification of Newman's and Yokouchi's Lemmas in PVS .....	137
<i>André Luiz Galdino, Mauricio Ayala-Rincón</i>	
Computation over Groups .....	147
<i>Christine Gaßner</i>	
Singularities of Holomorphic Functions in Subsystems of Second Order Arithmetic .....	157
<i>Yoshihiro Horihata, Keita Yokoyama</i>	
Modelling Linear Cellular Automata with the Minimum Stage Corresponding to CCSG based on LFSR .....	165
<i>Yoon-Hee Hwang, Sung-Jin Cho, Un-Sook Choi, Han-Doo Kim</i>	
Multitape Ordinal Machines and Primitive Recursion .....	175
<i>Bernhard Irrgang, Benjamin Seyfferth</i>	
Prescribed Learning of Indexed Families .....	185
<i>Sanjay Jain, Frank Stephan, Nan Ye</i>	
Lower Bounds for Syntactically Multilinear Algebraic Branching Programs	195
<i>Maurice Jansen</i>	
The Use of Information Affinity in Possibilistic Decision Tree Learning and Evaluation .....	205
<i>Ilyes Jenhani, Salem Benferhat, Zied Elouedi</i>	
Ordering Finite Labeled Trees .....	215
<i>Herman Ruge Jervell</i>	
Towards Reverse Proofs-as-Programs .....	224
<i>Reinhard Kahle</i>	
Plotkin Definability Theorem for Atomic-Coherent Information Systems .	234
<i>Basil Karádais</i>	
Safety Properties Verification for Pfaffian Dynamics .....	246
<i>Margarita Korovina, Nicolai Vorobjov</i>	

On Extending Wand’s Type Reconstruction Algorithm to Handle Polymorphic Let .....	254
<i>Sunil Kothari, James Caldwell</i>	
Simulations Between Tilings .....	264
<i>Grégoire Lafitte, Michael Weiss</i>	
Discrete Non Determinism and Nash Equilibria for Strategy-Based Games	274
<i>Stéphane Le Roux</i>	
Combining Concept Maps and Petri Nets to Generate Intelligent Tutoring Systems .....	284
<i>Maikel León, Isis Bonet, Zenaida García</i>	
Query-Optimal Oracle Turing Machines for Type-2 Computations .....	294
<i>Chung-Chih Li</i>	
From Hilbert’s Program to a Logic Toolbox .....	304
<i>Johann Makowsky</i>	
From Program Verification to Certified Binaries .....	324
<i>Angelos Manousaridis, Michalis Papakyriakou, Nikolaos Papaspyrou</i>	
Limiting Recursion, FM–Representability, and Hypercomputations .....	332
<i>Marcin Mostowski</i>	
Using Tables to Construct Non-Redundant Proofs .....	344
<i>Vivek Nigam</i>	
Classifying the Phase Transition Threshold for Unordered Regressive Ramsey Numbers .....	354
<i>Florian Pelupessy, Andreas Weiermann</i>	
Almost Partial m-Reducibility .....	361
<i>Katya Petrova, Boris Solon</i>	
Two-Dimensional Cellular Automata Transforms for a Novel Edge Detection .....	367
<i>Yongri Piao, Seok-Tae Kim, Sung-Jin Cho</i>	
Computable Counter-examples to the Brouwer Fixed-point Theorem .....	377
<i>Petrus Potgieter</i>	
Polynomial Iterations over Finite Fields .....	387
<i>Mihai Prunescu</i>	
Simulations of Quantum Turing Machines by Quantum Multi-Counter Machines .....	397
<i>Daowen Qiu</i>	

Optimal Proof Systems and Complete Languages . . . . .	407
<i>Zenon Sadowski</i>	
On the Complexity of Computing Winning Strategies for Finite Poset Games . . . . .	415
<i>Michael Soltys, Craig Wilson</i>	
A Statistical Mechanical Interpretation of Algorithmic Information Theory	425
<i>Kohtaro Tadaki</i>	
Solving Tripartite Matching by Interval-valued Computation in Polynomial Time . . . . .	435
<i>Ákos Taji, Benedek Nagy</i>	
Probabilistic Machines vs. Relativized Computation . . . . .	445
<i>Hayato Takahashi, Kazuyuki Aihara</i>	
Quantum Query Algorithms for AND and OR Boolean Functions . . . . .	453
<i>Alina Vasilieva</i>	
Space Complexity in Ordinal Turing Machines . . . . .	463
<i>Joost Winter</i>	
Reverse Mathematics for Fourier Expansion . . . . .	473
<i>Keita Yokoyama</i>	
Induced Matchings in Graphs of Maximum Degree Three . . . . .	483
<i>Grażyna Zwolińska</i>	

### **Abstracts of Informal Presentations.**

Subsystems of Iterated Inductive Definitions . . . . .	493
<i>Bahareh Afshari, Michael Rathjen</i>	
Query Algorithms for Detecting Hamming and Reed-Solomon Codes . . . . .	494
<i>Ruben Agadžanjan</i>	
Expressive Power Of Graph Logic . . . . .	495
<i>Timos Antonopoulos, Anuj Dawar</i>	
The Lost Melody Theorem for Infinite Time Register Machines . . . . .	496
<i>Merlin Carl, Peter Koepke</i>	
Comparing Notions of Fractal Dimension . . . . .	497
<i>Chris Conidis</i>	
Clockable Ordinals for Infinite Time Register Machines . . . . .	498
<i>Tim Fischbach, Peter Koepke, Miriam Nasfi, Gregor Weckbecker</i>	
Embedding the Enumeration Degrees in the $\omega$ -Enumeration Degrees . . . . .	499
<i>Hristo Ganchev</i>	

Computable Models Spectras of Ehrenfeucht Theories . . . . .	500
<i>Alexander Gavryushkin</i>	
Deterministic Subsequential Transducers with Additional FIFO-memory . . . . .	501
<i>Stefan Gerdjikov</i>	
Proof Fragments and Cut-Elimination . . . . .	502
<i>Stefan Hetzl</i>	
$q$ -Overlaps in the Random Exact Cover Problem . . . . .	503
<i>Gabriel Istrate, Romeo Negrea</i>	
Inductive Definitions over Domain Representable Spaces . . . . .	504
<i>Petter Kristian Køber</i>	
The Computable Dimension of Free Projective Planes . . . . .	505
<i>Nurlan Kogabaev</i>	
A Classification of Theories of Truth . . . . .	506
<i>Graham Leigh, Michael Rathjen</i>	
Monotonicity Conditions over Characterisations of PSPACE . . . . .	507
<i>Bruno Loff, Isabel Oitavem</i>	
Some Results on Local LR-degree Structures . . . . .	508
<i>Anthony Morphett</i>	
The Axiomatic Derivation of Absolute Lower Bounds . . . . .	509
<i>Yiannis Moschovakis</i>	
Exploring the Computational Contribution of a Non-constructive Combinatorial Principle . . . . .	510
<i>Diana Ratiu, Trifon Trifonov</i>	
Autostability of Automatic Linear Orders . . . . .	511
<i>Alexandra Revenko</i>	
Quantifiers on Automatic Structures . . . . .	512
<i>Sasha Rubin</i>	
The Almost Zero $\omega$ -Enumeration Degrees . . . . .	513
<i>Ivan N. Soskov</i>	
A Sequent Calculus for Intersection and Union Logic . . . . .	514
<i>Anastasia Veneti, Yiorgos Stavrinos</i>	
Anhomomorphic Logic: The Logic of Quantum Realism . . . . .	515
<i>Petros Wallden</i>	

# The Computing Species

Keith Devlin

Center for the Study of Language and Information (CSLI)  
Stanford University  
Cordura Hall, 210 Panama Street  
Stanford, CA 94305-4115 USA  
[devlin@csli.stanford.edu](mailto:devlin@csli.stanford.edu)

We are characteristically the “symbolic species,” Terrence Deacon claimed in his celebrated 1997 book of that name. Deacon’s focus was on language, a capacity unique to humans, the development of which changed in fundamental ways the way humans think and live. Computation is a particular form of symbolic processing. At four distinct stages in the development of modern society, acquisition of the ability to carry out certain new kinds of computation likewise changed - also in fundamental ways - how we humans understand the world and live our lives.

The fourth such change is taking place during our lifetime, brought about by the invention of machines that can be instructed to compute for us. The others occurred in 8,000 B.C., the 13<sup>th</sup> century, and the 17<sup>th</sup> century. I’ll look at how human life and cognition changed at each of those three stages.

# Logic, Automata, Games, and Algorithms

Moshe Y. Vardi

Department of Computer Science  
Rice University  
6100 S. Main Street  
Houston, TX 77005-1892, USA  
[vardi@cs.rice.edu](mailto:vardi@cs.rice.edu)

The automata-theoretic approach to decision procedures, introduced by Buechi, Elgot, and Trakhtenbrot in the late 1950s, is one of the most fundamental approaches to decision procedures. Recently, this approach has found industrial applications in formal verification of hardware and software systems. The path from logic to practical algorithms goes not only through automata, but also through games, whose algorithmic aspects were studies by Chandra, Kozen, and Stockmeyer in the late 1970s. In this tutorial we describe the path from logic to algorithms via games and automata.

# On the Performance of Automata Minimization Algorithms

Marco Almeida, Nelma Moreira, and Rogério Reis  
`{mfa,nam,rvr}@ncc.up.pt`

DCC-FC & LIACC, Universidade do Porto  
R. do Campo Alegre 1021/1055, 4169-007 Porto, Portugal

**Abstract.** Apart from the theoretical worst-case running time analysis not much is known about the average-case analysis or practical performance of finite automata minimization algorithms. On this paper we compare the running time of four minimization algorithms based on experimental results, and apply them to both deterministic and non-deterministic random automata. Although there is no clear winner, some conclusions can be taken for specific cases. Hopcroft's algorithm performs better for DFAs with small alphabets, and, regardless of the alphabet size, Brzozowski's algorithm is the clearly the fastest dealing with NFAs.

**Key words:** deterministic finite automata, non-deterministic finite automata, minimal automata, minimization algorithms, random generation

## 1 Introduction

The problem of writing efficient algorithms to find the minimal DFA equivalent to a given automaton can be traced back to the 1950's with the works of Huffman [Huf55] and Moore [Moo58]. Over the years several alternative algorithms were proposed. Authors typically present the running time worst-case analysis of their algorithms, but the practical experience is sometimes different. The comparison of algorithms performance is always a difficult problem, and little is known about the practical running time performance of automata minimization algorithms. In particular, there are no studies of average-case analysis of these algorithms, an exception being the work of Nicaud [Nic00], where it is proved that the average-case complexity of Brzozowski's algorithm is exponential for group automata. Lhoták [Lho00] presents a general data structure for DFA minimization algorithms to run in  $O(kn \log n)$ , where  $n$  is the number of states of the DFA and  $k$  is the size of the alphabet. Bruce Watson [Wat95] presents some experimental results but his data sets were small and biased. Tabakov and Vardi [TV05] compared Hopcroft's and Brzozowski's algorithms. Baclet *et al.* [BP06] analysed different implementations of the Hopcroft's algorithm. More recently, Bassino *et al.* [BDN07] compared Moore's and Hopcroft's algorithms.

Using the Python programming language, we implemented the algorithms due to Hopcroft (**H**) [Hop71], Brzozowski (**B**) [Brz63], Watson (**W**) [Wat95], and also using full memoization (**WD**) [WD03]. The choice of the algorithms

was due to the disparate worst-case complexities and doubts about the practical behaviour of each algorithm. The input data was obtained with random automata generators. For the (initially-connected) deterministic finite automata we used a uniform random generator and thus our results are statistically accurate. Lacking an equivalent uniform random generator for NFAs, we implemented a non-uniform one. Although not statistically significant, the results in this case are still fairly interesting.

The text is organised as follows. In Section 2 we present some definitions and the notation used throughout the paper. In Section 3 we describe each of the algorithms, briefly explain them, and present the respective pseudo-code. In Section 4 we describe the generation methods of the random automata. In Section 5 we present the experimental results and in Section 6 we analyze these results. Finally, on Section 7 we summarize and compare our results with previous work.

## 2 Preliminaries

A *deterministic finite automaton* (DFA)  $\mathcal{D}$  is a tuple  $(Q, \Sigma, \delta, q_0, F)$  where  $Q$  is a finite set of *states*,  $\Sigma$  is the *input alphabet* (any non-empty set of symbols),  $\delta : Q \times \Sigma \rightarrow Q$  is the *transition function*,  $q_0$  is the *initial state* and  $F \subseteq Q$  is the set of *final states*. When the transition function is total, the automaton  $\mathcal{D}$  is said to be *complete*. Any finite sequence of alphabet symbols  $a \in \Sigma$  is a *word*. Let  $\Sigma^*$  denote the set of all words over the alphabet  $\Sigma$  and  $\epsilon$  denote the *empty word*. We define the *extended transition function*  $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$  in the following way:  $\hat{\delta}(q, \epsilon) = q$ ;  $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$ . A state  $q \in Q$  of a DFA  $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$  is called *accessible* if  $\hat{\delta}(q_0, w) = q$  for some  $w \in \Sigma^*$ . If all states in  $Q$  are accessible, a complete DFA  $\mathcal{D}$  is called (complete) *initially-connected* (ICDFA). The *language* accepted by  $\mathcal{D}$ ,  $L(\mathcal{D})$ , is the set of all words  $w \in \Sigma^*$  such that  $\hat{\delta}(q_0, w) \in F$ . Two DFAs  $\mathcal{D}$  and  $\mathcal{D}'$  are *equivalent* if and only if  $L(\mathcal{D}) = L(\mathcal{D}')$ . A DFA is called *minimal* if there is no other equivalent DFA with fewer states. Given a DFA  $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$ , two states  $q_1, q_2 \in Q$  are said to be *equivalent*, denoted  $q_1 \approx q_2$ , if for every  $w \in \Sigma^*$ ,  $\hat{\delta}(q_1, w) \in F \Leftrightarrow \hat{\delta}(q_2, w) \in F$ . Two states that are not equivalent are called *distinguishable*. The equivalent minimal automaton  $\mathcal{D}/\approx$  is called the *quotient automaton*, and its states correspond to the equivalence classes of  $\approx$ . It is proved to be unique up to isomorphism.

A *non-deterministic finite automaton* (NFA) is also a tuple  $(Q, \Sigma, \Delta, I, F)$ , where  $I$  is a *set of initial states* and the *transition function* is defined as  $\Delta : Q \times \Sigma \rightarrow 2^Q$ . Just like with DFAs, we can define the *extended transition function*  $\hat{\Delta} : 2^Q \times \Sigma^* \rightarrow 2^Q$  in the following way:  $\hat{\Delta}(S, \epsilon) = S$ ;  $\hat{\Delta}(S, xa) = \bigcup_{q \in \hat{\Delta}(S, x)} \delta(q, a)$ . The *language* accepted by  $\mathcal{N}$  is the set of all words  $w \in \Sigma^*$  such that  $\hat{\Delta}(I, w) \cap F \neq \emptyset$ . Every language accepted by some NFA can also be described by a DFA. The *subset construction* method takes a NFA  $\mathcal{A}$  as input and computes a DFA  $\mathcal{D}$  such that  $L(\mathcal{A}) = L(\mathcal{D})$ . This process is also referred to as *determinization* and has a worst-case running time complexity of  $O(2^{|Q|})$ .

Following Leslie [Les95], we define the *transition density* of an automaton  $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$  as the ratio  $\frac{t}{|Q|^2|\Sigma|}$ , where  $t$  is the number of transitions in

$\mathcal{A}$ . This density function is normalised, giving always a value between 0 and 1. We also define *deterministic density* as the ratio of the number of transitions  $t$  to the number of transitions of a complete DFA with the same number of states and symbols, i.e.,  $\frac{t}{|Q||\Sigma|}$ .

The *reversal* of a word  $w = a_0a_1 \cdots a_n$ , written  $w^R$ , is  $a_n \cdots a_1a_0$ . The reversal of a language  $L \subseteq \Sigma^*$  is  $L^R = \{w^R \mid w \in L\}$ . Further details on regular languages can be found in the usual references (Hopcroft [HMU00] or Kozen [Koz97], for example).

### 3 Minimization Algorithms

Given an automaton, to obtain the associated minimal DFA we must compute the equivalence relation  $\approx$  as defined in Section 2. The computation of this relation is the key difference of the several minimization algorithms. Moore's algorithm and its variants, for example, aim to find pairs of distinguishable states. **H**, on the other hand, computes the minimal automaton by refining a partition of the states' set.

Of the three minimization algorithms we compared, **H** has the best worst-case running time analysis. **B** is simple and elegant, and, despite its exponential worst-case complexity, it is supposed to frequently outperform other algorithms (including **H**). **B** also has the particularity of being able to take both DFAs and NFAs as input. **W** can be halted at any time yielding a partially minimized automaton. The improved version, **WD**, includes the use of full memoization. Because one of our motivations was to check the minimality of a given automaton, not to obtain the equivalent minimal one, this algorithm was of particular interest.

#### 3.1 Hopcroft's Algorithm (**H**)

**H** [Hop71], published in 1971, achieves the best known running time worst-case complexity for minimization algorithms. It runs on  $O(kn \log n)$  time for a DFA with  $n$  states and an alphabet of size  $k$ . Let  $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$  be a DFA. **H**, proceeds by refining the coarsest partition until no more refinements are possible. The initial partition is  $P = \{F, Q - F\}$  and, at each step of the algorithm, a block  $B \in P$  and a symbol  $a \in \Sigma$  are selected to *refine* the partition. This refinement process *splits* each block  $B'$  of the partition according to whether the states of  $B'$ , when consuming the symbol  $a$ , go to a state which is in  $B$  or not. Formally, we call this procedure **split** and define it by

$$\text{split}(B', B, a) = (B' \cap \delta^{-1}(B, a), B' \cap \overline{\delta^{-1}(B, a)})$$

where  $\check{\delta}(S, a) = \bigcup_{q \in S} \delta(q, a)$ .

The algorithm terminates when there are no more blocks to refine. In the end, each block of the partition is a set of equivalent states. Because, for any two blocks  $B, B' \in P$ , every state  $q \in B$  is distinguishable from any state

$q' \in B'$ , the elements of  $P$  represent the states of a new minimal DFA. The complete pseudo-code is presented in Algorithm 1.1.

```

def hopcroft () :
    L = {}
    if |F| < |Q-F|:
        P = {Q-F, F}; L = {F}
    else:
        P = {F, Q-F}; L = {Q-F}
    while L ≠ ∅:
        S = extract(L)
        for a in Σ:
            for B in P:
                (B1, B2) = split(B, S, a)
                P = P - {B}; P = P ∪ {B1}; P = P ∪ {B2}
                if |B1| < |B2|:
                    L = L ∪ {B1}
                else:
                    L = L ∪ {B2}
    return P

```

**Algorithm 1.1.** Hopcroft's algorithm (**H**).

The set  $L$  contains blocks of  $P$  not yet treated. The `extract` procedure removes one element of  $L$  to be used in the splitting process. The choice of the element does not influence the correctness of the algorithm.

### 3.2 Brzozowski's Algorithm (**B**)

**B** [Brz63] is based on two successive reverse and determinization operations and the full pseudo-code is presented (in one single line!) on Algorithm 1.2.

```

def brzozowski (fa):
    return det (rev (det (rev (fa))))

```

**Algorithm 1.2.** Brzozowski's algorithm (**B**).

Having to perform two determinizations, the worst-case running time complexity of **B** is exponential. Watson's thesis, however, presents some surprising results about **B** practical performance, usually outperforming **H**. As for the peculiar way that this algorithm computes a minimal DFA, Watson assumed it to be unique and, in his taxonomy, placed it apart all other algorithms. Later, Champarnaud et al. [CKP02] analysed the way the sequential determinizations perform the minimization and showed that it does compute state equivalences.

### 3.3 An Incremental Algorithm (**W**)

Watson presented an incremental DFA minimization algorithm (**W**) [Wat95]. This algorithm can be halted at any time yielding a partially minimized DFA that recognises the same language as the input DFA. Later, Watson and Daciuk presented an improved version of the same algorithm (**WD**) [WD03] which makes

use of full memoization. While the first algorithm has a worst-case exponential running time, the memoized version yields a  $O(n^2\alpha((n^2)))$  algorithm, where  $\alpha$  is the inverse of the Ackermann function. Since  $\alpha(x) \leq 5$  for any  $x \leq 2^{2^{16}}$ ,  $\alpha(x)$  can be considered a constant for all “practical” values of  $x$ . It was not clear, however, that this algorithm would outperform **H** as the experimental results in [WD03] seemed to point to. Since the use of memoization introduces some considerable overhead in the algorithm, we wanted to discover at what point this extra work begins to payoff.

**W** uses an auxiliary function, `equiv`, to test the equivalence of two states. The third argument, an integer  $k$ , is used to control the recursion depth only for matters of efficiency. Also for matters of efficiency, a variable  $S$  that contains a set of presumably equivalent pairs of states is made global. The pseudo-code for a non-memoized, specialized for ICDFAs, implementation of `equiv` is presented in Algorithm 1.3. The memoized algorithm (**WD**) is quite extensive and can be found in Watson and Daciuk’s paper [WD03].

```
def equiv(p, q, k):
    if k == 0:
        return (p in F and q in F) or (not p in F and not q in F)
    elif (p, q) in S:
        return True
    else:
        eq = (p in F and q in F) or (not p in F and not q in F)
        S = S ∪ {(p, q)}
        for a in Σ:
            if not eq:
                return False
            eq = eq and equiv(δ(p, a), δ(q, a), k - 1)
        S = S - {(p, q)}
    return eq
```

**Algorithm 1.3.** Pairwise state equivalence algorithm.

Having a method to verify pairwise state equivalence, it is possible to implement a test that calls `equiv` for every pair of states and returns *False* if some pair is found to be equivalent.

## 4 Random Automata Generation

Even if we consider only (non-isomorphic) ICDFAs, the number of automata with  $n$  states over an alphabet of  $k$  symbols grows so fast [RMA05] that trying to minimize every one is not feasible, even for small values of  $n$  and  $k$ . The same applies to NFAs. In order to compare the practical performance of the minimization algorithms, we must have an arbitrary quantity of “good” randomly generated automata available, i.e. the samples can not be unbiased. This can be achieved with a uniform random generator. We used the DFA string representation and random generation method proposed by Almeida et al. [AMR07]. This approach, unlike the one by Bassino et al. [BN07], does not require a rejection

step. The generator produces a string that is a canonical representation of an ICDFA with  $n$  states and  $k$  symbols without final states information, as described by Reis et al. [RMA05,AMR07]. Given an order over  $\Sigma$ , it is possible to define a canonical order over the set of states by traversing the automaton in a breadth-first way choosing at each node the outgoing edges using the order of  $\Sigma$ . In the string representation, each of the  $i$  blocks, for  $1 \leq i \leq n$ , corresponds to the transitions from the state  $i - 1$ . The random generator produces the random strings from left to right, taking into account the number of ICDFAs that, at a given point, would still be possible to produce with a given prefix. The set of final states is computed by generating an equiprobable bitstream of size  $n$  and considering final all the states that correspond to a non-zero position.

Lacking a uniform random generator for NFAs, we implemented one which combines the van Zijl bit-stream method, as presented by Champarnaud et al. [CHPZ04], with one of Leslie’s approaches [Les95], which allows us both to generate initially connected NFAs (with one initial state) and to control the transition density. Leslie presents a “generate-and-test” method which may never stop, so we added some minor changes that correct this situation. A brief explanation of the random NFA generator follows. Suppose we want to generate a random NFA with  $n$  states over an alphabet of  $k$  symbols and a transition density  $d$ . Let the states (respectively the symbols) be named by the integers  $0, \dots, n - 1$  (respectively  $0, \dots, k - 1$ ). A sequence of  $n^2k$  bits describes the transition function in the following way: the occurrence of a non-zero bit at the position  $ink + jk + a$  denotes the existence of a transition from state  $i$  to state  $j$  labelled by the symbol  $a$ .

Starting with a sequence of zero bits, the first step of the algorithm is to create a connected structure and thus ensure that all the states of the final NFA will be accessible. In order to do so, we define the first state as 0, mark it as visited, generate a transition from 0 to any not-visited state  $i$ , and mark  $i$  as visited. Next, until all states are marked as visited, randomly choose an already visited state  $q_1$ , randomly choose a not-visited state  $q_2$ , add a transition from  $q_1$  to  $q_2$  (with a random), and mark  $q_2$  as visited. At this point we have an initially connected NFA and proceed by adding random transitions. Until the desired density is achieved, we simply select one of the bitstream’s zero bits and set it to one. By maintaining a list of visited states on the first step and keeping record of the zero bits on the second step, we avoid generating either a disconnected NFA or a repeated transition and guarantee that the algorithm always halts. The set of final states can be easily obtained by generating an equiprobable bitstream of size  $n$  and considering final all the states that correspond to a non-zero position in the bitstream.

## 5 Experimental Results

To compare algorithms is always a difficult problem. The choice of the programming language, implementation details, and the hardware used may harm the rigour of any benchmark. In order to produce realistic results, the input data

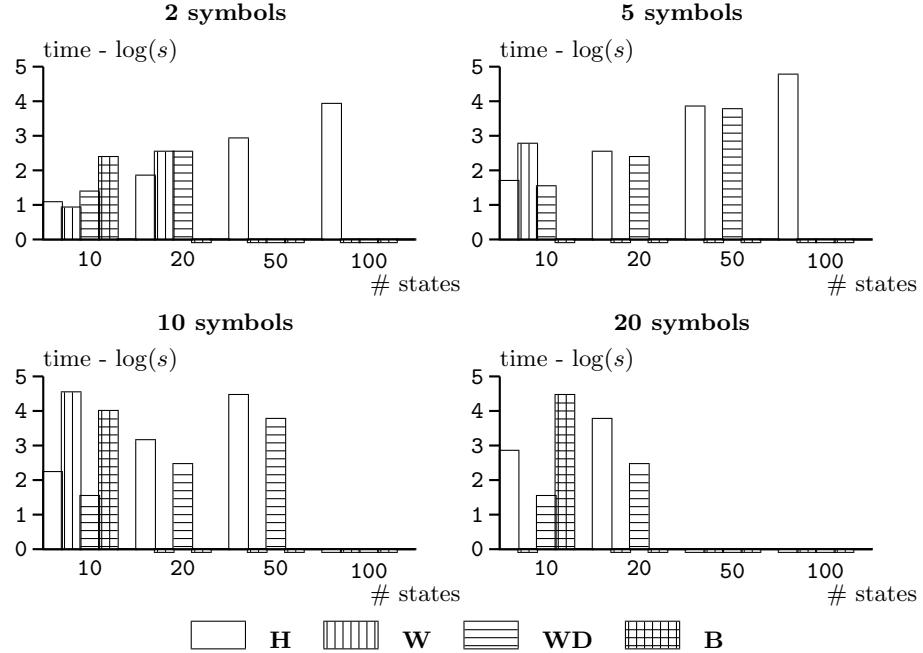
should be random so that it represents a typical usage of the algorithm and the test environment should be identical for all benchmarks. We implemented all the algorithms in the Python 2.4 programming language, using similar data structures whenever possible. All the tests were executed in the same computer, an Intel® Xeon® 5140 at 2.33GHz with 2GB of RAM. We used samples of 10.000 automata, with  $5 \leq n \leq 100$  states and alphabets with  $k \in \{2, 5, 10, 20\}$  symbols. For the data sets obtained with the uniform random generator, the size of each sample is sufficient to ensure results with a 95% confidence level within a 1% error margin. It is calculated with the formula  $n = (\frac{z}{2\epsilon})^2$ , where  $z$  is obtained from the normal distribution table such that  $P(-z < Z < z) = \gamma$ ,  $\epsilon$  is the error margin, and  $\gamma$  is the desired confidence level.

### 5.1 Random IC DFA Minimization

On his thesis, Watson used a fairly biased sample. It consisted of 4833 DFAs of which only 7 had 23 states. As Watson himself states, being constructed from regular expressions, the automata “... are usually not very large, they have relatively sparse transition graphs, and the alphabet frequently consists of the entire ASCII character set.”. On their paper on the incremental minimization algorithm [WD03], Watson and Daciuk also present some performance comparisons of automata minimization algorithms. They used four different data sets, one from experiments on finite-state approximation of context-free grammars and three that were automatically generated. These are not, however, uniform random samples, and thus, do not represent a typical usage of the algorithms.

The graphics on Figure 1 show the running times of the tested algorithms while minimizing samples of 10.000 random ICDFAs. This running time is the total time required for minimizing an entire sample and was limited to 24 hours. The batches which did not finish under this limit appear with the value  $-1$ , in order to be distinguished from those that actually took almost 0 seconds.

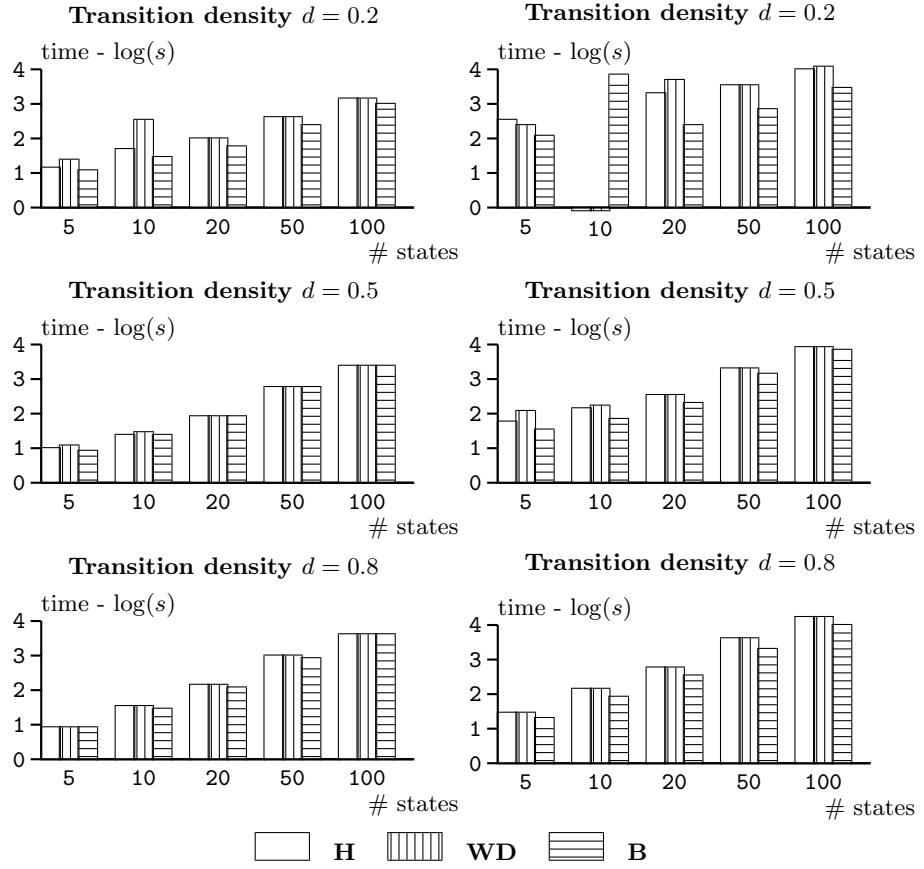
For small alphabets ( $k = 2$ ), **H** is always the fastest. When the alphabet size grows ( $k \geq 5$ ), however, **H** is clearly outperformed by the **WD**, which was over twice as fast as **H** when minimizing ICDFAs with an alphabet of size  $k \geq 10$ . **W** showed itself quite slow in all tests. It is important to point out that for  $k \geq 5$  all the automata were already minimal and so the speed of the incremental algorithm can not be justified by the possibility of halting whenever two equivalent states are found. The fact that almost all ICDFAs are minimal was observed by several authors, namely Almeida *et al.* [AMR07] and Bassino *et al.* [BDN07]. As Watson himself stated, the incremental algorithm may show exponential performance for some DFAs. This was the case in one of our tests. For the sample of 5 symbols and 15 states **WD** took an unusual amount of time. **B** is never the fastest algorithm. In fact, even for small alphabets it was not possible to use it on ICDFAs with more than 15 states.



**Fig. 1.** Running time results for the minimization of 10.000 ICDFAs with  $k \in \{2, 5, 10, 20\}$ .

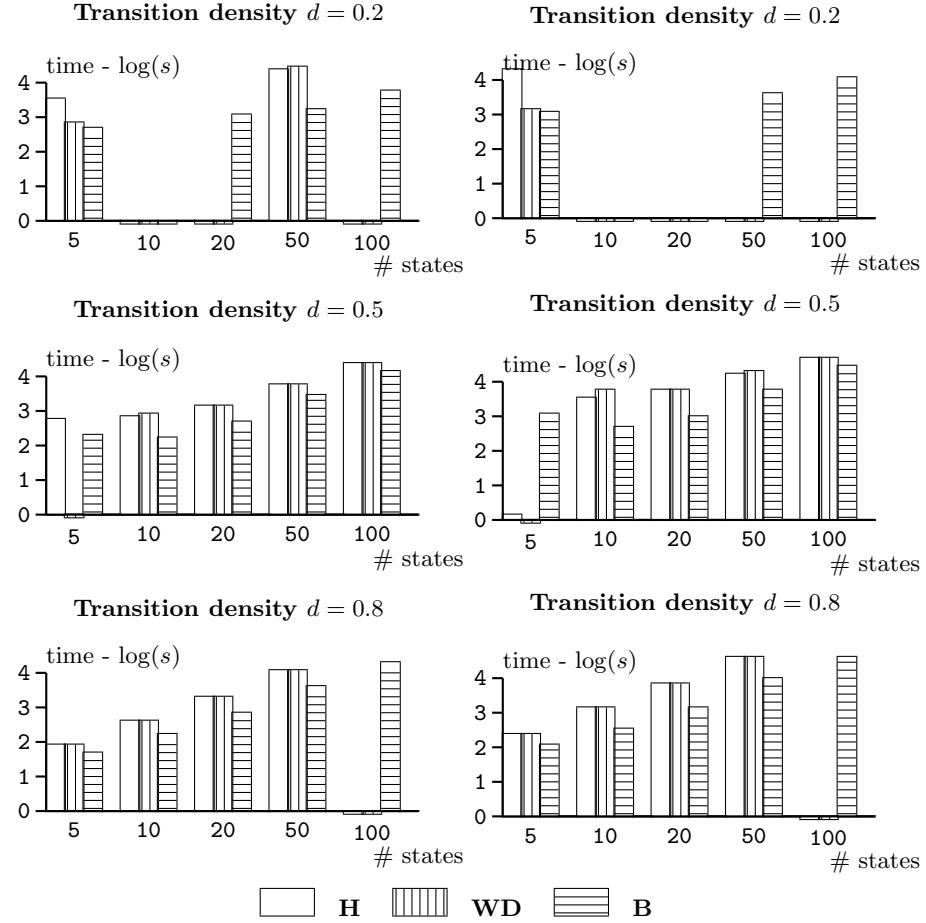
## 5.2 Random NFAs Minimization

The next set of graphics shows the execution times of the three algorithms when applied to a set of 10.000 random NFAs. The running time limit for all algorithms was set to 15 hours and, again, the batches that did not finish under this time limit are shown with the value  $-1$ . It is important to note that the NFA generator we used is not a uniform one, and so we can not prove that each sample is actually a good representative of the universe. Because we are dealing with NFAs, the transition density is an important factor and so each sample was generated with three different transition densities ( $d$ ): 0.2, 0.5, and 0.8. For both the **WD** and **H**, which are only able to minimize DFAs, we also accounted for the time spent in the subset construction method. For alphabets with two symbols there are no significant differences in any of the algorithms' general performance, although **B** is usually the fastest. **H** outperforms **B** for less than 4% only when  $d = 0.5$  and  $n \in \{50, 100\}$ . For alphabets with  $k = 5$ , **B** is always the fastest and, except for occasional cases, **H** is slightly faster than the incremental algorithm. When the alphabet size increases, **B**'s performance becomes quite remarkable. For an alphabet with size  $k \in \{10, 20\}$ , **B** is definitively the fastest, being the only algorithm to actually finish the minimization process of almost all random samples within the 15 hour limit. As for **H** and **WD**, except for two cases, there are no significant performance differences. For  $d = 0.2$  and



**Fig. 2.** Running time results for the minimization of 10,000 NFAs with  $k = 2$  (left) and  $k = 5$  (right).

$n = 10$  all the algorithms showed a particularly bad performance. For  $k = 5$  only **B** finished the minimization process (taking an unusual high amount of time) and for  $k \in \{10, 20\}$  none of the algorithms completed the minimization process within the 15 hour time limit. This result corroborates Leslie's conjecture [Les95] about the number of states obtained with the subset construction method for a given deterministic density  $d_d$ . Leslie's conjecture states that randomly generated automata exhibit the maximum execution time and the maximum number of states at an approximate deterministic density of 2.0. While generating the random NFAs, we considered the transition density  $d = \frac{t}{n^2k}$ , which is related to the deterministic density  $d_d = \frac{t}{nk}$  by  $d_d = nd$ . It is easy to see that in our case  $d_d = nd = 10 \times 0.2 = 2.0$ , which will make the subset construction computationally expensive. In order to achieve the same exponential behaviour in the subset method for  $d \in \{0.5, 0.8\}$  the number of states would have to be  $n \in \{4, 2.5\}$ , but for such a small number of states the exponential blowup is not meaningful. This



**Fig. 3.** Running time results for the minimization of 10.000 NFAs with  $k = 10$  (left) and  $k = 20$  (right).

explains why there are no similar results for the test batches with  $d \in \{0.5, 0.8\}$ . Considering we used a variation of one of Leslie's random NFA generators, this result does not come with any surprise.

## 6 Analysis of the Results

In this work, we experimentally compare four automata minimization algorithms (**H**, **W**, **WD** and **B**). As data sets we use two different types of randomly generated automata (ICDFAs and NFAs) with a range of different number of states and symbols. The ICDFAs' data set was obtained using a uniform random generator and is large enough to ensure a 95% confidence level within a 1% error margin. For ICDFAs with only two symbols, **H** is the fastest algorithm. As the

alphabet size grows, **WD** begins to perform better than **H**. With alphabets larger than 10, **WD** becomes clearly faster. As for **W** and **B** algorithms, we can safely conclude that neither performs well, regardless of the number of states and symbols. As for the NFAs, it is important to note that the random generator used was not a uniform one, and thus does not have the same statistical accuracy as the first one. **B** is definitively the fastest algorithm. Both **H** and **WD** consistently show equally slower results.

Since all these algorithms make use of the subset construction at least once, the reason for **B**'s good performance is even less evident. It would be interesting to make an average-case running time complexity analysis for the DFA reversal, and thus possibly explain **B**'s behaviour with ICDFAs minimization.

## 7 Comparison with Related Work

In this final section we summarise and compare our experiments with some of the results of the works cited before.

Bruce Watson implemented five minimization algorithms: two versions of Moore's algorithm as well as **H**, **W** and **B**. As we have mentioned before, the data set then used was fairly biased and his results lead him to conclude that the two Moore based algorithms perform very poorly and that **B** normally outperforms the other two. Baclet *et al.* [BP06] implemented **H** with two different queue strategies: LIFO and FIFO. The implementations were tested with several random (non-uniform) automata having thousands of states but very small alphabets ( $k \leq 4$ ), concluding that the LIFO strategy is better, at least for alphabets of one or two symbols. Bassino *et al.* [BDN07] used an ICDFAs' uniform random generator based on a Boltzmann sampler to create a data set and compared the performance of the **H** (with the two strategies refereed above) and Moore's algorithms. The automata generated have up to some thousands of states for alphabets of size 2. Their results are statistically accurate, and indicate that Moore's algorithm is, for the average case, more efficient than **H**. Moreover, no clear difference were found between the two queue strategies for **H**. These results are interesting because they neither corroborate the results of the works mentioned above nor the general idea that **H** outperforms Moore's algorithm in practice. It remains unstudied a comparison between Moore's algorithm with **WD** using a uniformly generated data set. Finally, Tabakov and Vardi [TV05] studied **H** and **B** performance with a data set of random NFAs. The random model they used is similar to the one we describe in Section 4 but considering a deterministic density,  $d_d$ . They choose  $0 \leq d_d \leq 2.5$  and the samples were relatively small: 200 automata with  $n < 50$  and  $k = 2$ . The conclusion was that **H** is faster for low-density automata ( $d_d < 1.5$ ), while **B** is better for high-density automata. For the example studied of  $n = 30$  the deterministic density  $d_d < 1.5$ , corresponds to a normalised transition density that we used,  $d < 0.05$ . This phase transition may be due to the fact that for such a low normalised transition density the probability of a connected NFA being deterministic is very high.

## References

- [AMR07] M. Almeida, N. Moreira, and R. Reis. Enumeration and generation with a string automata representation. *Theoretical Computer Science*, 387(2):93–102, 2007.
- [BDN07] F. Bassino, J. David, and C. Nicaud. A library to randomly and exhaustively generate automata. In *Implementation and Application of Automata*, volume 4783 of *LNCS*, pages 303–305. Springer-Verlag, 2007.
- [BN07] F. Bassino and C. Nicaud. Enumeration and random generation of accessible automata. *Theoretical Computer Science*, 381(1-3):86–104, 2007.
- [BP06] M. Baclet and C. Pagetti. Around hopcroft’s algorithm. pages 114–125, Taipei, Taiwan, 2006. Springer.
- [Brz63] J. A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. In J. Fox, editor, *Proc. of the Sym. on Math. Theory of Automata*, volume 12 of *MRI Symposia Series*, pages 529–561, NY, 1963.
- [CHPZ04] J.-M. Champarnaud, G. Hansel, T. Paranthoën, and D. Ziadi. Random generation models for nfas. *J. of Automata, Lang. and Comb.*, 9(2), 2004.
- [CKP02] J.-M. Champarnaud, A. Khorsi, and T. Paranthoën. Split and join for minimizing: Brzozowski’s algorithm. In M. Balík and M. Simánek, editors, *Proc. of PSC’02*, Report DC-2002-03, pages 96–104. Czech Technical University of Prague, 2002.
- [HMU00] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 2000.
- [Hop71] J. Hopcroft. An  $n \log n$  algorithm for minimizing states in a finite automaton. In *Proc. Inter. Symp. on the Theory of Machines and Computations*, pages 189–196, Haifa, Israel, 1971. AP.
- [Huf55] D. A. Huffman. The synthesis of sequential switching circuits. *The Journal of Symbolic Logic*, 20(1):69–70, 1955.
- [Koz97] D. C. Kozen. *Automata and Computability*. Undergrad. Texts in Computer Science. Springer-Verlag, 1997.
- [Les95] T. Leslie. Efficient approaches to subset construction. Master’s thesis, University of Waterloo, Ontario, Canada, 1995.
- [Lho00] O. Lhoták. A general data structure for efficient minimization of deterministic finite automata. Technical report, University of Waterloo, 2000.
- [Moo58] E. F. Moore. Gedanken-experiments on sequential machines. *The Journal of Symbolic Logic*, 23(1):60, 1958.
- [Nic00] C. Nicaud. *Étude du comportement en moyenne des automates finis et des langages rationnels*. PhD thesis, Université de Paris 7, 2000.
- [RMA05] R. Reis, N. Moreira, and M. Almeida. On the representation of finite automata. In C. Mereghetti, B. Palano, G. Pighizzini, and D. Wotschke, editors, *Proc. of DCFS’05*, pages 269–276, Como, Italy, 2005.
- [TV05] D. Tabakov and M. Vardi. Evaluating classical automata-theoretic algorithms. In *LPAR’05*, 2005.
- [Wat95] B. W. Watson. *Taxonomies and toolkit of regular languages algortihms*. PhD thesis, Eindhoven Univ. of Tec., 1995.
- [WD03] B. W. Watson and J. Daciuk. An efficient DFA minimization algorithm. *Natural Language Engineering*, pages 49–64, 2003.

# Decidability Results for Mobile Membranes Derived from Mobile Ambients

Bogdan Aman<sup>1</sup> and Gabriel Ciobanu<sup>1,2</sup>

<sup>1</sup> Romanian, Academy, Institute of Computer Science  
Bldv. Carol I no.8, 700505 Iași, Romania

<sup>2</sup> “A.I.Cuza” University, Faculty of Computer Science  
Bldv. Carol I no.11, 700506 Iași, Romania  
[baman@iit.tuiasi.ro](mailto:baman@iit.tuiasi.ro), [gabriel@info.uaic.ro](mailto:gabriel@info.uaic.ro)

**Abstract.** Mobile ambients and membrane systems are two new computation models. We investigate some decidability properties in mobile membranes, using also some known decidability results of the mobile ambients.

## 1 Introduction

Ambients calculus [6] and membrane systems [15] represent two new computation models which are quickly evolving. During the last decade, both of them were emergent research fields in computer science. Both mobile ambients and mobile membranes (a variant of membrane systems introduced in [11]) have similar structures and common concepts. Both formalisms work with boundaries representing locations, have a hierarchical structure representing the nesting relationship among locations, and describe mobility in complex systems. We consider these new computing models, studying their computability aspects and their relationship.

Mobile ambients are designed to model both mobile computing (provided by mobile devices as laptops or PDA's), and mobile computation (mobile code like applets or agents moving between computers and devices). An ambient is a location (bounded place) where computation happens. Ambients can be nested, having a hierarchical structure. An ambient has a name, and can move with all the subambients and processes it contains. Ambient calculus is designed to describe distributed and mobile computation. In contrast with other formalisms for mobility such as the  $\pi$ -calculus [13] whose computational model is based on the notion of *communication*, the ambient calculus is based on the notion of *movement*. An ambient is the unit of movement. Ambients mobility is controlled by the capabilities *in*, *out*, and *open*. Moreover, in an ambient we have processes which may exchange messages.

Membrane systems are introduced in [14] as a class of parallel computing devices inspired by the biological systems which are complex hierarchical structures, with a flow of materials and information which underlies their functioning. Essentially, the membrane systems (also called P systems) are composed of various compartments with different tasks, all of them working simultaneously to

accomplish a more general task. There are several variants of membrane systems. The mobile membranes were introduced in [2] where the mobility is expressed through the operations of gemmation and fusion of mobile membranes. In this paper we use mobile membrane systems (also called P systems with mobile membranes) which express mobility using two biologically inspired operations: exocytosis and endocytosis.

The structure of the paper is as follows. In Section 2 we present the ambient calculus. In the last sections of the paper we present the mobile membrane systems and investigate some decidability problems (reachability, boundedness, periodicity) in mobile membranes. Conclusions and references end the paper.

## 2 Mobile Ambients

In this section we provide a short description of the mobile ambients; more details can be found in [6]. The following table describes the syntax of mobile ambients.

<b>Table 1: Mobile Ambients Syntax</b>			
$c$	channel name	$P, Q ::=$	processes
$n, m$	ambient names	<b>0</b>	inactivity
$x$	variable	$M.P$	movement
$M ::=$	capabilities	$n[P]$	ambient
$in\ n$	can enter $n$	$P \mid Q$	composition
$out\ n$	can exit $n$	$(\nu n)P$	restriction
$open\ n$	can open $n$	$c!(m).P$	output action
		$c?(x).P$	input action
		$*P$	replication

Process **0** is an inactive process (it does nothing). A movement  $M.P$  is provided by a capability  $M$ , followed by the execution of process  $P$ . An ambient  $n[P]$  represents a bounded place labelled by  $n$  in which a process  $P$  is executed.  $P \mid Q$  is a parallel composition of processes  $P$  and  $Q$ .  $(\nu n)P$  creates a new unique name  $n$  within the scope of process  $P$ . An output action  $c!(m).P$  releases a name  $m$  on channel  $c$ , and then behaves as process  $P$ . An input action  $c?(x).P$  captures a name from channel  $c$ , and binds it to a variable  $x$  within the scope of process  $P$ .  $*P$  denotes an unbounded replication of a process  $P$ , producing as many parallel copies of process  $P$  as needed.

Semantics of the ambient calculus is provided by two relations: structural congruence and reduction. The *structural congruence*  $P \equiv_{amb} Q$  relates different syntactic representations of the same process; it is also used to define the reduction relation. The *reduction relation*  $P \Rightarrow_{amb} Q$  describes the evolution of ambients and processes. We denote by  $\Rightarrow_{amb}^*$  the reflexive and transitive closure of  $\Rightarrow_{amb}$ , and by  $\Rightarrow_{amb}^+$  its transitive closure.

The structural congruence is defined as the least relation over processes satisfying the axioms from the table below:

**Table 2:** Structural Congruence

$P   Q \equiv_{amb} Q   P$	$P \equiv_{amb} Q$ implies $Q \equiv_{amb} P$
$(P   Q)   R \equiv_{amb} P   (Q   R)$	$P \equiv_{amb} Q, Q \equiv_{amb} R$ implies $P \equiv_{amb} R$
$*P \equiv_{amb} P   *P$	$P \equiv_{amb} Q$ implies $(\nu n)P \equiv_{amb} (\nu n)Q$
$(\nu n)(\nu m)P \equiv_{amb} (\nu m)(\nu n)P$ if $n \neq m$	$P \equiv_{amb} Q$ implies $P   R \equiv_{amb} Q   R$
$(\nu n)(P   Q) \equiv_{amb} P   (\nu n)Q$ if $n \notin fn(P)$	$P \equiv_{amb} Q$ implies $*P \equiv_{amb} *Q$
$(\nu n)m[P] \equiv_{amb} m[(\nu n)P]$ if $n \neq m$	$P \equiv_{amb} Q$ implies $n[P] \equiv_{amb} n[Q]$
$P   \mathbf{0} \equiv_{amb} P$	$P \equiv_{amb} Q$ implies $M.P \equiv_{amb} M.Q$
$(\nu n)\mathbf{0} \equiv_{amb} \mathbf{0}$	$P \equiv_{amb} Q$ implies $c?(x).P \equiv_{amb} c?(x).Q$
$P \equiv_{amb} P$	$P \equiv_{amb} Q$ implies $c!(m).P \equiv_{amb} c!(m).Q$

The rules from the left side of the table describe the commutativity and associativity of the parallel components, unfolding recursion, stretching of a restriction scope, renaming of bounded names. The rules from the right side describe how structural congruence is propagated across processes. The set  $fn(P)$  of free names of a process  $P$  is defined as:

$$fn(P) = \begin{cases} \emptyset & \text{if } P = \mathbf{0} \\ fn(R) \cup \{n\} & \text{if } P = in\ n.R \text{ or } P = out\ n.R \text{ or } P = open\ n.R \\ & \text{or } P = n[R] \text{ or } P = c!(n).R \\ fn(R) \cup fn(Q) & \text{if } P = R | Q \\ fn(R) - \{n\} & \text{if } P = (\nu n)R \\ fn(R) & \text{if } P = c?(x).R \text{ or } P = *R \end{cases}$$

The reduction relation is defined as the least relation over processes satisfying the following set of axioms:

**Table 3:** Reduction Rules

<b>(In)</b>	$n[in\ m.\ P   Q]   m[R] \Rightarrow_{amb} m[n[P   Q]   R]$
<b>(Out)</b>	$m[n[out\ m.\ P   Q]   R] \Rightarrow_{amb} n[P   Q]   m[R]$
<b>(Open)</b>	$open\ n.\ P   n[Q] \Rightarrow_{amb} P   Q$
<b>(Com)</b>	$c!(m).\ P   c?(x).\ P' \Rightarrow_{amb} P   P'\{m/x\}$
<b>(Res)</b>	$P \Rightarrow_{amb} Q$ implies $(\nu n)P \Rightarrow_{amb} (\nu n)Q$
<b>(Amb)</b>	$P \Rightarrow_{amb} Q$ implies $n[P] \Rightarrow_{amb} n[Q]$
<b>(Par)</b>	$P \Rightarrow_{amb} Q$ implies $P   R \Rightarrow_{amb} Q   R$
<b>(Struct)</b>	$P' \equiv P, P \Rightarrow_{amb} Q, Q \equiv Q'$ implies $P' \Rightarrow_{amb} Q'$

The first four rules are the one-step reductions for *in*, *out*, *open* and *communication*. In the rule **(Com)** we write  $P'\{m/x\}$  for the substitution of name  $m$  for each free occurrence of variable  $x$  in process  $P'$ . The next three rules propagate reductions across scopes, ambient nesting and parallel composition. The final rule allows the use of structural congruence during the evolution given by the reduction relation.

### 3 Decidability Results for Mobile Membranes

A *membrane system* consists of a hierarchy of nested membranes, placed inside a distinguishable membrane called *skin*. The space outside the skin membrane is

called *environment*. A membrane contains multisets of *objects*, *evolution rules*, and possibly other *membranes*. The multisets of objects from a membrane correspond to the chemicals swimming in the solution in the cell compartment, while the rules correspond to the chemical reactions possible in the same compartment. The rules contain target indications specifying the membranes where the new obtained objects are sent. The new objects either remain in the same membrane whenever they have attached a *here* target, or they pass through membranes in two directions: they can be sent *out* of the membrane, or can be sent *in* one of the nested membranes which is precisely identified by its label. In one step, the objects can pass only through one membrane. A membrane without any other membranes inside is called *elementary*, while a membrane containing other membranes is a *composite* one.

The membrane systems are synchronous: at each time unit of a global clock, a transformation of the system takes place by applying the rules in a nondeterministic and maximally parallel manner. This means that the objects and the rules involved in such a transformation are chosen in a nondeterministic way, and the application of rules is performed in a maximally parallel way. After a choice was made, no rule can be applied anymore in the same evolution step: there are not enough objects and membranes available for any rule to be applied. Several membranes can evolve in parallel. Many variants of this basic model are discussed in the literature [7, 15]. In this paper we use mobile membrane systems. This is a variant of membrane systems with active membranes, but having none of the features like polarizations, label change and division of non-elementary membranes.

Reachability is the problem of deciding whether a system may reach a given configuration during its execution. This is one of the most critical properties in the verification of systems; most of the safety properties of computing systems can be reduced to the problem of checking whether a system may reach a “unintended state”.

Boundedness is a property of systems whose production and consumption of resources may be bounded. From a biological point of view, boundedness can be interpreted as a property of sustainable development, in the sense that a cell can accumulate and work with only a finite amount of material.

When working with Petri nets, reachability and boundedness are two general properties of interest. Given a net with initial marking  $\mu_0$ , we say that the marking  $\mu$  is reachable if there exists a sequence of firings  $\mu_0 \rightarrow \mu_1 \rightarrow \dots \mu_n = \mu$  of the net. We say that a net is bounded if the set of reachable markings is finite. A  $k$ -bounded net implies that there exists some integer  $k$  bounding the number of tokens which may be present at each place.

The boundedness and reachability problems are decidable in Petri nets, even if they tend to have a very large complexity in practice. A good survey of the known decidability issues for Petri nets is given in [9].

We use various results to investigate reachability and boundedness for mobile membranes.

### 3.1 Mobile Membranes with Replication

We prove in [1] that reachability in the class of mobile membranes defined by Definition 1 can be decided by reducing it to the reachability problem of a version of pure and public ambient calculus from which the `open` capability has been removed. It is proven in [4] that the reachability for this fragment of ambient calculus is decidable by reducing it to marking reachability for Petri nets, which is proven to be decidable in [12]. Problems like reachability and boundedness are investigated in [8] for other classes of P systems, namely for extensions of PB systems with volatile membranes. In what follows we present a sketch of the work done in [1], and we extend it with new results.

First we define a new class of mobile membranes with replication rules.

**Definition 1.** A mobile membrane system *with replication rules* is a structure  $\Pi = (V \cup \bar{V}, H \cup \bar{H}, \mu, w_1, \dots, w_n, R)$ , where:

1.  $n \geq 1$  represents the initial degree of the system;
2.  $V \cup \bar{V}$  is an alphabet (its elements are called objects), where  $V \cap \bar{V} = \emptyset$ ;
3.  $H \cup \bar{H}$  is a finite set of labels for membranes, where  $H \cap \bar{H} = \emptyset$ ;
4.  $\mu$  is a membrane structure, consisting of  $n$  membranes, labelled (not necessarily in a one-to-one manner) with elements of  $H$ ;
5.  $w_1, w_2, \dots, w_n$  are multisets of objects from  $V \cup \bar{V}$  placed in the  $n$  membranes of the system;
6.  $R$  is a finite set of developmental rules, of the following forms:
  - (a)  $[\bar{a}\downarrow \rightarrow \bar{a}\downarrow a\downarrow]_h$ , for  $h \in H$ ,  $a\downarrow \in V$ ,  $\bar{a}\downarrow \in \bar{V}$ ; replication rule  
The objects  $\bar{a}\downarrow$  are used to create new objects  $a\downarrow$  without being consumed.
  - (b)  $[\bar{a}\downarrow a\downarrow \rightarrow \bar{a}\downarrow]_h$ , for  $h \in H$ ,  $a\downarrow \in V$ ,  $\bar{a}\downarrow \in \bar{V}$ ; consumption rule  
The objects  $a\downarrow$  are consumed.
  - (c)  $[\bar{a}\uparrow \rightarrow \bar{a}\uparrow a\uparrow]_h$ , for  $h \in H$ ,  $a\uparrow \in V$ ,  $\bar{a}\uparrow \in \bar{V}$ ; replication rule  
The objects  $\bar{a}\uparrow$  are used to create new objects  $a\uparrow$  without being consumed.
  - (d)  $[\bar{a}\uparrow a\uparrow \rightarrow \bar{a}\uparrow]_h$ , for  $h \in H$ ,  $a\uparrow \in V$ ,  $\bar{a}\uparrow \in \bar{V}$ ; consumption rule  
The objects  $a\uparrow$  are consumed.
  - (e)  $[a\downarrow]_h [ ]_a \rightarrow [ ]_h [ ]_a$ , for  $a, h \in H$ ,  $a\downarrow \in V$ ; endocytosis  
An elementary membrane labelled  $h$  enters the adjacent membrane labelled  $a$ , under the control of object  $a\downarrow$ . The labels  $h$  and  $a$  remain unchanged during this process; however the object  $a\downarrow$  is consumed during the operation. Membrane  $a$  is not necessarily elementary.
  - (f)  $[ ]_h [ a\uparrow ]_a \rightarrow [ ]_h [ ]_a$ , for  $a, h \in H$ ,  $a\uparrow \in V$ ; exocytosis  
An elementary membrane labelled  $h$  is sent out of a membrane labelled  $a$ , under the control of object  $a\uparrow$ . The labels of the two membranes remain unchanged; the object  $a\uparrow$  of membrane  $h$  is consumed during the operation. Membrane  $a$  is not necessarily elementary.
  - (g)  $[ ]_{\bar{h}} \rightarrow [ ]_{\bar{h}} [ ]_h$  for  $h \in H$ ,  $\bar{h} \in \bar{H}$  division rules  
An elementary membrane labelled  $\bar{h}$  is divided into two membranes labelled by  $\bar{h}$  and  $h$  having the same objects.

In 2.,  $V \cap \bar{V} = \emptyset$  states that the objects from  $\bar{V}$  can participate only in rules of type (a) and (b). Similarly in 3.,  $H \cap \bar{H} = \emptyset$  states that the membranes having labels from the set  $\bar{H}$  can participate only in rules of type (g).

The rules are applied using the following principles:

1. In biological systems molecules are divided into classes of different types. We make the same decision here and split the objects into four classes:  $a \downarrow$  - objects which control the *endocytosis*,  $a \uparrow$  - objects which control the *exocytosis*, and  $\bar{a} \downarrow$ ,  $\bar{a} \uparrow$  - objects which produce new objects from the first two classes without being consumed.
2. All the rules are applied in parallel, non-deterministically choosing the rules, the membranes, and the objects in such a way that the parallelism is maximal; this means that in each step we apply a set of rules such that no further rule, no further membranes and objects can evolve at the same time.
3. A membrane  $a$  from each rule of type (e) and (f) is said to be **passive**, while membrane  $h$  is said to be **active**. In any step of a computation, any object and any active membrane can be involved in at most one rule, but the passive membranes are not considered involved in the use of rules (hence they can be used by several rules at the same time).
4. When a membrane is moved across another membrane, by endocytosis or exocytosis, its whole content (its objects) is moved.
5. If a membrane is divided, then its content is replicated in the two new copies.
6. The skin membrane can never be divided.

According to these rules, we get transitions among the configurations of the system. For two mobile membrane systems  $M$  and  $N$ , we say that  $M$  reduces to  $N$  if there is a sequence of rules applicable in the membrane system  $M$  in order to obtain the membrane system  $N$ .

**Theorem 1 ([1]).** *For two arbitrary mobile membrane systems  $M_1$  and  $M_2$ , it is decidable whether  $M_1$  reduces to  $M_2$ .*

The main steps of the proof are as follows:

1. we reduce mobile membranes systems to pure and public mobile ambients without the capability *open*;
2. we show that the reachability problem for two arbitrary mobile membranes can be expressed as the reachability problem for the corresponding mobile ambients;
3. we use the result that the reachability problem is decidable for a fragment of pure and public mobile ambients without the capability *open*, using the following decidability theorem from Petri nets:

**Theorem 2 ([12]).** *For all Petri nets  $P$ , for all markings  $m, m'$  for  $P$ , one can decide whether  $m'$  is reachable from  $m$ .*

We define the following translation steps:

1. any object  $a\downarrow$  is translated into a capability  $in\ a$ ;
2. any object  $a\uparrow$  is translated into a capability  $out\ a$ ;
3. any object  $\bar{a}\downarrow$  is translated into a replication  $!in\ a$ ;
4. any object  $\bar{a}\uparrow$  is translated into a replication  $!out\ a$ ;
5. a membrane  $h$  is translated into an ambient  $h$ ;
6. an elementary membrane  $\bar{h}$  is translated into a replication  $!h[\ ]$  where all the objects inside membrane  $h$  are translated into capabilities of ambient  $h$  by using the above steps.

A correspondence exists between the rules from mobile membranes and the rules from mobile ambients:

rule (e) corresponds to rule **(In)**;

rule (f) corresponds to rule **(Out)**;

rules (a), (b), (c) and (d) correspond to instances of the structural congruence rule  $*P \equiv_{amb} P \mid *P$ , namely (a) and (c) for creation of new instances, while (b) and (d) for consuming existing instances.

If we consider a mobile membrane system  $M$ , we denote by  $\mathcal{T}(M)$  the mobile ambient obtained using the above translation steps. For example, starting from the membrane system  $M = [m\downarrow m\uparrow]_n[\ ]_m$ , we obtain  $\mathcal{T}(M) = n[in\ m \mid out\ m] \mid m[\ ]$ .

**Proposition 1 ([1]).** *Given two mobile membrane systems  $M$  and  $N$ ,  $M$  reduces to  $N$  by applying one rule if and only if  $\mathcal{T}(M)$  reduces to  $\mathcal{T}(N)$  by applying only one rule.*

**Theorem 3 ([1]).** *For two arbitrary ambients  $A$  and  $B$  of the restricted fragment, it is decidable whether  $A$  reduces to  $B$ .*

By restricted fragment we refer to the class of mobile ambients obtained through translation from the class of mobile membrane systems from Definition 1. As a corollary, we prove that periodicity is a decidable property. *Periodicity* is a central notion in biological processes which have a periodic or quasi-periodic evolution, such as the ATP cycle, for example.

**Theorem 4 (Periodicity).** *Given a mobile membrane system and an initial configuration  $\mu_0$ , if the system evolves it is decidable whether the system is periodic.*

*Proof.* From Theorem 1, considering  $\mu_0$  both as an initial and as a target configuration.

We say that a mobile membrane system, with an initial configuration  $\mu_0$ , is *bounded* if there are only finitely many configurations which are reachable starting from  $\mu_0$ . Since we have obtained the translation of the class of mobile membrane systems with replication into Petri nets, the following result can be viewed as a corollary.

**Theorem 5 (Boundedness).** *Given a mobile membrane system and an initial configuration  $\mu_0$ , it is decidable whether the system is bounded.*

The decidability of the deadlock problem is also a consequence of the encoding into the Petri nets.

**Theorem 6 (Non-termination).** *Given a mobile membrane system and an initial configuration  $\mu_0$ , it is decidable whether all computations starting with  $\mu_0$  are infinite.*

For the decidability results from Petri nets we refer to [9].

### 3.2 Mobile Membrane Systems with Dissolution

A mobile membrane system with dissolution provides, in addition to the rules from Definition 1, a new rule of the form:

$$a_\delta [ ]_a \rightarrow \varepsilon$$

where  $\varepsilon$  is the empty multiset. The intuitive meaning of this new rule is that after applying it, the membrane  $a$  is dissolved and its content, including the inner membranes (if any), are moved to the parent of membrane  $a$ .

In order to simulate this rule in mobile ambients we use the following translation step: the object  $a_\delta$  is translated into a capability *open*  $a$ . The class of pure mobile ambients containing the capability *open* is shown to be undecidable in [5]. Using this result and the fact that the mobile membrane systems from Definition 1 together with the dissolution rule can be encoded in pure mobile ambients with the capability *open*, we obtain the following result:

**Theorem 7.** *Reachability is undecidable in mobile membrane systems with dissolution rules.*

## 4 Termination in Mobile Membrane Systems

A computation terminates if all its reduction sequences are of finite length. Termination is an interesting property in concurrency. For some configurations and sets of rules, we may want to know that an answer is eventually produced. Termination alone does not guarantee that an answer is eventually produced since other properties, e.g. deadlock-free, are also involved. A first attempt to formalize some halting conditions has been done quite recently in [10], by defining certain sets of configurations which satisfy some conditions.

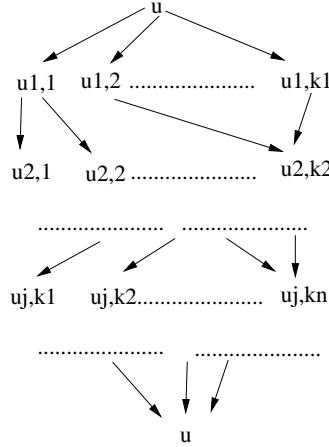
In order to assure that a computation terminates, the set of rules applied to a membrane system must not contain rules which produce periodicity in the system. In the rest of this section we study some of rules that can produce periodicity.

If  $\prod$  is a membrane system, then  $\prod_i$  denotes the membrane systems obtained after  $i$  steps. Inspired from [3] we define in what follows the periodicity of a membrane system:

**Definition 2.** *A membrane systems  $\prod$  is periodic if  $\prod_i = \prod$  for some  $i > 0$ .*

**Definition 3.** A membrane systems  $\Pi$  is eventually periodic if  $\Pi_{i+k} = \Pi_k$  for some  $i, k > 0$ . In this case  $k$  is called the transient and  $i$  the period.

Generally this can be described as:



In the figure above all the objects from line 1 are obtained using only objects from the initial multiset  $u$ , then the multisets from line 2 are obtained using only objects obtained in the first line, and so on. A special case is when at each step we apply a single rule, namely we have

$$u_1 \rightarrow u_2, u_3 \rightarrow u_4 \dots u_{2n-1} \rightarrow u_{2n}, \text{ with } u_1, \dots, u_{2n} \in V^*, \\ u_{2i+1} \preceq u_{2i}, \text{ for } i \in \{1, \dots, n-1\} \text{ and } u_1 = u_{2n}$$

The simplest case has the following form

$$u \rightarrow u, \text{ where } u \in V^*$$

for which if the precondition is satisfied, then it can be applied forever.

## 5 Conclusion

Ambient calculus is a process algebra using interleaving semantics, compositionality and bisimulation. Membrane computing is a branch of natural computing defining computation in the Turing sense, i.e., making use of automata, formal languages and complexity tools. This paper presents some existing computability results of the mobile ambient calculus and mobile membrane systems, and emphasizes on a formal relationship between mobile ambients and mobile membranes.

We investigate the problem of reaching a certain configuration of a system of mobile membranes starting from another configuration. In order to do this we use a result of [4] where the reachability problem for the pure and public ambient calculus without open capability is proven to be decidable. The same problem is tackled in [5], where the authors do not take into account the replication of ambients which in our case is used to simulate the division rules in mobile

membranes. We proved that the reachability can be decided by reducing this problem to the reachability problem for a fragment of ambient calculus. Starting from this approach, some other decidability results with respect to boundedness, periodicity and termination are proven.

## Acknowledgments

We thank Oana Agrigoroaiei for her useful remarks and help in improving early drafts of the paper.

## References

1. B. Aman, G. Ciobanu. On the Reachability Problem in P Systems with Mobile Membranes. *Proceedings of the Eight Workshop on Membrane Computing*, 111-121, 2007.
2. D. Besozzi, C. Zandron, G. Mauri, N. Sabadini. P Systems with Gemmation of Mobile Membranes. *Italian Conference on Theoretical Computer Science*, Lecture Notes in Computer Science vol.2202, Springer, 136-153, 2001.
3. L. Bianco, F. Fontana, G. Franco, V. Manca. P Systems for Biological Dynamics. *Applications of Membrane Computing*, Springer, 83-128, 2005.
4. I. Boneva, J.-M. Talbot. When Ambients Cannot be Opened. *Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science vol.2620, Springer, 169-184, 2003.
5. N. Busi, G. Zavattaro. Deciding Reachability in Mobile Ambients. *Lecture Notes in Computer Science*, Springer vol.3444, 248-262, 2005.
6. L. Cardelli, A. Gordon. Mobile Ambients. *Theoretical Aspects of Computer Software*, Lecture Notes in Computer Science vol.1378, Springer, 140-155, 1998.
7. G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez. *Application of Membrane Computing*, Springer, 2006.
8. G. Delzanno, L. Van Begin. On the Dynamics of PB Systems with Volatile Membranes. *Proceedings of the Eight Workshop on Membrane Computing*, 279-300, 2007.
9. J. Esparza, M. Nielson. Decidability issues for Petri nets - a survey. *Journal of Informatik Processing and Cybernetics*, vol.30, 143-160, 1994.
10. R. Freund, S. Verlan. A Formal Framework for P Systems. *Proceedings of the Eight Workshop on Membrane Computing*, 317-330, 2007.
11. S.N. Krishna, Gh. Păun. P Systems with Mobile Membranes. *Natural Computing*, vol.4(3), Springer, 255-274, 2005.
12. E.W. Mayr. An Algorithm for the General Petri Net Reachability Problem. *SIAM Journal of Computing*, vol.13(3), 441-460, 1984.
13. R. Milner. *Communicating and mobile systems: the  $\pi$ -calculus*, Cambridge University Press, 1999.
14. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, vol.61(1), 108-143, 2000.
15. Gh. Păun. *Membrane Computing. An Introduction*, Springer, 2002.

# Sophisticated Infinite Sequences

Luis Antunes and André Souto\*

University of Porto  
lfa@ncc.up.pt, andresouto@ncc.up.pt

**Abstract.** In this paper we revisit the notion of sophistication for infinite sequences. Koppel defined sophistication of an object as the length of the shortest (finite) total program ( $p$ ) that with some (finite or infinite) data ( $d$ ) produce it and  $|p| + |d|$  is smaller than the shortest description of the object plus a constant. However the notion of “description of infinite sequences” is not appropriately defined. In this work, we propose a new definition of sophistication for infinite sequences as the limit of the ratio of sophistication of the initial segments and its length. As the main result we prove that highly sophisticated sequences are dense when the sophistication is defined with  $\limsup$  and are sparse when we consider the  $\liminf$ . We also prove that, similarly to what happens for finite strings, sophistication and depth, for infinite sequences are distinct complexity measures.

**Key words:** Kolmogorov complexity, Sophistication, Constructive dimensions

## 1 Introduction

Solomonoff [Sol64], Kolmogorov [Kol65] and Chaitin [Cha66] independently defined the complexity of an object  $x$  as the length of the shortest program that produces  $x$ . The Kolmogorov structure function divides the smallest program producing the object into two parts: the part accounting for the *useful* regularity present in the object and the part accounting for the remaining *accidental* information. Kolmogorov suggested that the useful information is represented by a finite set in which the object is a typical member, so that the two-part description of the finite set together with the index of the object is as shortest as the one-part description. This approach has been generalized to computable probability mass functions. The combined theory has been developed in detail in [GTV01] and called “Algorithmic Statistics”. The most general way to proceed is perhaps to express the useful information as a recursive function. The resulting measure has been called the *sophistication* of the object in [Kop87,Kop88,AK91]. Later Antunes and Fortnow [AF01] revisited the notion of sophistication for finite strings and formalize a connection between sophistication and a variation of computational depth (intuitively the useful or nonrandom information in a string), proving the existence of strings with maximum sophistication and showing that they are the deepest of all strings.

In 1982, at a seminar in the Moscow State University (see [V'y99]), Kolmogorov raised the question if “absolutely non-random” (or non-stochastic) objects exist. Gács *et al.* [GTV01] and Antunes and Fortnow [AF01], independently, proved that these (finite) objects do exist.

In this work we take a fresh look at the sophistication of infinite sequences. We start by redefining the notion of sophistication for infinite sequences, introducing the

---

\* Departamento de Ciéncia de Computadores, Rua Campo Alegre, 1021/1055, 4169 - 007 Porto - Portugal; The authors are partially supported by KCrypt (POSC/EIA/60819/2004) and funds granted to LIACC through the Programa de Financiamento Plurianual, Fundação para a Ciéncia e Tecnologia and Programa POSI

lower and upper sophistication as the  $\liminf$  and  $\limsup$ , respectively, of the ratio between the sophistication of the initial segment and the length of that segment. Then we prove that the set of sequences with upper sophistication different from 0 is dense and the set of sequences with lower sophistication different from 0 is sparse. So, the answer for Kolmogorov's question for infinite sequences is affirmative if we use the upper sophistication and probably negative if we use the lower sophistication.

Koppel claimed the equivalence between sophistication and logical depth. However, he used a different definition of logical depth imposing totality in the functions and defining the length of a time bound by the smallest program describing it. Later Antunes and Fortnow [AF01] gave an example, for the finite case where the equivalence is not valid considering the classical definition of Kolmogorov complexity and computational depth. In this work we go further and give an example of an infinite sequence, the diagonal of the Halting Problem, for which the sophistication is small and the dimensional depth is high.

The rest of this paper is organized as follows: in the next section, we give some background on the basic concepts necessary to the rest of the paper. In section 3 we formally present the new definitions of sophistications and establish a relationship with Hausdorff dimension and packing dimension. In section 4 we prove that highly sophisticated sequences are dense if sophistication is defined with  $\limsup$  and are rare if in the definition we consider  $\liminf$ . In section 5 we relate sophistication with dimensional depth by proving that these two complexity measures are of different type.

## 2 Preliminaries

To avoid confusions any element of  $\Sigma^\infty$  will be called a sequence and will be denoted by Greek letters and any element of  $\Sigma^*$  will be called a string. We denote the initial segment of length  $n$  of a sequence  $\alpha$  by  $\alpha_{[1..n]}$  and the  $i^{th}$  bit by  $\alpha_i$ . The length of a binary string  $x$  is denoted by  $|x|$ . The function  $\log$  will always mean the logarithmic function of base 2. If  $x$  and  $y$  are strings  $x \leq y$  means that  $x$  is prefix of  $y$ .

### 2.1 Kolmogorov Complexity

We refer the reader to the book of Li and Vitányi [LV97] for a complete study on Kolmogorov complexity. Here we give essential definitions and basic results in Kolmogorov complexity necessary to the rest of this paper.

**Definition 1.** Let  $U$  be a fixed universal Turing machine. The plain Kolmogorov complexity of  $x \in \Sigma^*$  is,  $C(x) = \min_p \{|p| : U(p) = x\}$ . If  $t$  is a constructible function, the  $t$ -time-bounded Kolmogorov complexity of  $x \in \Sigma^*$  is,  $C^t(x) = \min_p \{|p| : U(p) = x \text{ in at most } t(|x|) \text{ steps}\}$ .

Notice that the choice of Universal Turing machine affects the running time of a program at most by a logarithmic factor and the program length at most a constant number of extra bits. So the Kolmogorov complexity theory is machine independent.

**Definition 2.** A string  $x$  is  $c$ -incompressible if  $C(x) \geq |x| - c$ .

**Proposition 1.** There are at least  $2^n \times (1 - 2 - c) + 1$  binary strings  $x \in \Sigma^n$  that are  $c$ -incompressible.

We need prefix-free Kolmogorov complexity defined using prefix free Turing machines: Turing machines with a one-way input tape (the input head can only read from left to right and crashes if it reads past the end of the input), a one-way output tape and a two-way work tape.

**Definition 3.** Let  $U$  be a fixed prefix free universal Turing machine. Then for any string  $x \in \Sigma^*$ , the prefix free Kolmogorov complexity of  $x$  is,  $K(x) = \min_p \{|p| : U(p) = x\}$ .

For any time constructible  $t$ , the  $t$ -time-bounded prefix-free Kolmogorov complexity of  $x \in \Sigma^*$  is,  $K^t(x) = \min_p \{|p| : U(p) = x \text{ in at most } t(|x|) \text{ steps}\}$ .

We extend the definition of Kolmogorov complexity to finite sets in the following way: the Kolmogorov complexity of a set  $S$  (denoted  $C(S)$ ) is the Kolmogorov complexity of its characteristic sequence. As noted by Cover [Cov85], Kolmogorov proposed in 1973 at the Information Theory Symposium, Tallin, Estonia the following function:

**Definition 4.** The Kolmogorov structure function  $H_k(x|n)$  for  $x \in \Sigma^n$  is defined by

$$H_k(x|n) = \min\{\log |S| : x \in S \text{ and } C(S|n) \leq k\}.$$

Of special interest is the value

$$C^*(x|n) = \min\{k : H_k(x|n) + k = C(x|n)\}.$$

A program for  $x$  can be written in two stages:

1. Use  $p$  to print the indicator function for  $S$ .
2. The desired string is the  $i^{th}$  string in a lexicographic ordering of the elements of this set.

This program has length  $|p| + \log |S| + O(1)$ , and  $C^*(x|n)$  is the length of the shortest program  $p$  for which this two-stage description is as concise as the shortest one stage description. Note that  $x$  must be maximally random (a typical element) with respect to  $S$  otherwise the two stage description could be improved, contradicting the minimality of  $C(x|n)$ . Gács *et al.* [GTV01] generalize the model class from finite sets to probability distributions, where the models are computable probability density functions.

In 1982, at a seminar in the Moscow State University (see [V'y99]), Kolmogorov raised the question if “absolutely non-random” (or non-stochastic) objects exist.

**Definition 5.** Let  $\alpha$  and  $\beta$  be natural numbers. A string  $x \in \Sigma^n$  is called  $(\alpha, \beta)$ -stochastic if there exists a finite set  $S$  such that  $x \in S$ ,  $C(S) \leq \alpha$  and  $C(x|S) \geq \log |S| - \beta$ .

Gács *et al.* [GTV01] and Antunes and Fortnow [AF01], independently, proved that  $(\alpha, \beta)$ -stochastic objects do exist.

## 2.2 Sophistication

Koppel [Kop87] used total functions to represent the useful information, and the resulting measure has been called *sophistication*. The definition of sophistication is based on process (monotonic) complexity defined by Schnorr. A function  $f : \Sigma^* \rightarrow \Sigma^*$  is monotonic if  $x \leq y$  ( $x$  is a prefix of  $y$ ) implies that  $f(x) \leq f(y)$  for all  $x$  and  $y$ .  $S_\Sigma$  is a sample space consisting of all finite and infinite sequences over  $\Sigma$ .

**Definition 6.** Let  $U$  be the reference monotone machine. The monotone complexity of  $x$  with respect to  $y$ , is defined as,

$$Km(x|y) = \min_p \{|p| : U(p, y) = x\omega, \omega \in S_\Sigma\}.$$

Koppel [Kop87] defined a *description* of a finite or infinite binary string  $\alpha$  as a pair  $(p, d)$  such that  $p$  is a total, i.e.,  $U(p, d)$  is defined for all  $d$ ,  $p$  is a self-delimiting program and  $U(p, d) \leq \alpha$ , i.e.,  $U(p, d)$  is an initial segment of  $\alpha$ . He also defined the *complexity* of  $\alpha$  by

$$H(\alpha) = \min\{|p| + |d| : (p, d) \text{ is a description of } \alpha\}.$$

**Definition 7 ([Kop87]).** *The  $c$ -sophistication of  $\alpha \in \Sigma^\infty$ , is*

$$\text{soph}_c(\alpha) = \min_p \{|p| : \text{exists } d \text{ s.t. } (p, d) \text{ is a description of } \alpha \text{ and } |p| + |d| \leq H(\alpha) + c\}$$

Note that Koppel's notion of *description of infinite sequences* is not appropriately defined. Also, as Koppel remark in [Kop88], it is not defined for every sequence  $\alpha$ . In order to avoid this problem Koppel defined a weak version of sophistication based on “weak” compression programs for  $\alpha$ . Antunes and Fortnow [AF01] later, revisited the notion of sophistication and, using the plain Kolmogorov complexity, adapted Koppel's definition for finite sequences as:

**Definition 8 ([AF01]).** *Let  $c$  be a constant,  $x \in \Sigma^n$  and  $U$  the universal reference Turing machine. The  $c$ -sophistication of  $x$  is*

$$\text{soph}_c(x) = \min_p \left\{ |p| : p \text{ is total and exists } d \text{ s.t. } U(p, d) = x \text{ and } |p| + |d| \leq C(x) + c \right\}.$$

*Remark 1.* An important observation about sophistication is the fact that it is not known rather if it is or not a robust measure. Indeed it is not known if slightly variations on the parameter  $c$  influences largely the length of sophistication program.

The definitions of sophistication for infinite sequences introduced here use this notion of sophistication for finite strings. We will need the following result, on the existence of highly sophisticated finite strings.

**Theorem 1 ([AF01]).** *Let  $c$  be a constant. There exists  $x \in \Sigma^n$  such that*

$$\text{soph}_c(x) > n - 2 \log n - 2c.$$

### 2.3 Dimension

Hausdorff [Haus79] augmented Lebesgue measure theory with a theory of dimension. It assigns to every subset  $X$  of a given metric space a real number  $\dim(X)$ , called the *Hausdorff dimension* of  $X$ . Lutz [Lut00] proved a gale characterization of Hausdorff dimension. This characterization gives an exact relationship between the Hausdorff dimension of a set  $X$  consisting of all infinite binary sequences, and the growth rates achievable by martingales betting on the sequences in  $X$ . This gale characterization of Hausdorff dimension was a breakthrough since it enabled the definition of *effective versions* of Hausdorff dimension by imposing various computability and complexity constraints on the gales.

Later Mayordomo [May02] showed that constructive Hausdorff dimension can be fully characterized in terms of Kolmogorov complexity.

**Theorem 2 ([May02]).** *For every sequence  $\alpha$ ,*

$$\dim(\alpha) = \liminf_{n \rightarrow \infty} \frac{K(\alpha_{[1..n]})}{n}$$

where  $\dim(\alpha)$  is the constructive Hausdorff dimension.

Packing dimension was introduced independently by Tricot [Tic82] and Sullivan [Sul84]. Later, Athreya *et al.* [AHLM07] showed how to characterize packing dimension in terms of gales, a dual of the gale characterization of the Hausdorff dimension. By imposing computational and complexity constraints on the gales Athreya *et al.* obtained a variety of *effective strong dimensions* that are exactly duals of the effective Hausdorff dimensions. In particular, it was proved the following characterization in terms of algorithmic information theory that is the dual of the previous one.

**Theorem 3 ([AHLM07]).** *For every sequence  $\alpha$ ,*

$$\text{Dim}(\alpha) = \limsup_{n \rightarrow \infty} \frac{K(\alpha_{[1..n]})}{n}$$

where  $\text{Dim}(\alpha)$  is the constructive packing dimension.

### 3 The Sophistication for Infinite Sequence Redefined

Based on the strong relationship between constructive Hausdorff dimension (respectively packing dimension) and the  $\limsup$  (respectively  $\liminf$ ) of the ratio between the Kolmogorov complexity of the initial segment and its length, in this section we introduce a new and clean approach to the sophistication of infinite sequences.

**Definition 9.** *We define lower sophistication of a sequence  $\alpha \in \Sigma^\infty$  by*

$$\underline{\text{soph}}_c(\alpha) = \liminf_n \frac{\text{soph}_c(\alpha_{[1..n]})}{n}$$

and the upper sophistication by

$$\overline{\text{soph}}_c(\alpha) = \limsup_n \frac{\text{soph}_c(\alpha_{[1..n]})}{n}$$

The first observation is that the lower and the upper sophistication of a sequence are well defined. Notice that the  $\liminf$  and  $\limsup$  of well defined real numbers are themselves well defined. Now we give some properties of the new measure and establish a connection to some known concepts, namely constructive Hausdorff dimension and constructive Packing dimension.

**Proposition 2.** *For all sequence  $\alpha$  and constant  $c$ ,  $\underline{\text{soph}}_c(\alpha) \leq \text{dim}(\alpha)$ .*

*Proof.*

$$\begin{aligned} \underline{\text{soph}}_c(\alpha) &= \liminf_n \frac{\text{soph}_c(\alpha_{[1..n]})}{n} \leq \liminf_n \frac{C(\alpha_{[1..n]}) + c}{n} \\ &\leq \liminf_n \frac{C(\alpha_{[1..n]})}{n} + \liminf_n \frac{c}{n} \leq \liminf_n \frac{K(\alpha_{[1..n]})}{n} + \liminf_n \frac{c}{n} \\ &= \liminf_n \frac{K(\alpha_{[1..n]})}{n} = \text{dim}(\alpha) \end{aligned}$$

**Proposition 3.** *For all sequence  $\alpha$  and constant  $c$ ,  $\overline{\text{soph}}_c(\alpha) \leq \text{Dim}(\alpha)$ .*

*Proof.*

$$\begin{aligned} \overline{\text{soph}}_c(\alpha) &= \limsup_n \frac{\text{soph}_c(\alpha_{[1..n]})}{n} \leq \limsup_n \frac{C(\alpha_{[1..n]}) + c}{n} \\ &\leq \limsup_n \frac{C(\alpha_{[1..n]})}{n} + \limsup_n \frac{c}{n} \leq \limsup_n \frac{K(\alpha_{[1..n]})}{n} + \limsup_n \frac{c}{n} \\ &= \limsup_n \frac{K(\alpha_{[1..n]})}{n} = \text{Dim}(\alpha) \end{aligned}$$

In the next proposition we show that the lower and upper sophistication of infinite sequences is a non increasing function with  $c$ .

**Proposition 4.** *If  $c > c'$  then  $\underline{\text{soph}}_c(\alpha) \leq \underline{\text{soph}}_{c'}(\alpha)$  and  $\overline{\text{soph}}_c(\alpha) \leq \overline{\text{soph}}_{c'}(\alpha)$ .*

The proof of this result follows immediately from the fact that for any finite string  $x$ , if  $c > c'$  then  $\text{soph}_c(x) \leq \text{soph}_{c'}(x)$ .

We can, in fact, prove a sharper result for the upper sophistication. We show that there are sequences for which the upper sophistication is strictly smaller than the packing dimension.

**Proposition 5.** *There exist sequences  $\alpha$  such that  $\overline{\text{soph}}_c(\alpha) < \text{Dim}(\alpha)$  for some constant  $c$ .*

*Proof.* The idea is to use a sequence with high Kolmogorov complexity. Chaitin, in [Cha66] and Martin-Löf [ML71] observed that there exists  $\alpha$  such that from some  $n_0$  onwards  $K(\alpha_{[1..n]}) \geq n - \log n - \log \log n$ .<sup>1</sup> Thus

$$\text{Dim}(\alpha) = \limsup_n \frac{K(\alpha_{[1..n]})}{n} \geq \limsup_n \frac{n - \log n - \log \log n}{n} = 1$$

On the other hand, it is known that infinitely many initial segments of  $\alpha$  satisfy  $C(\alpha_{[1..n]}) = n - c'$  for some constant  $c'$ . Hence for some appropriate constant  $c$ , that only depends on  $c'$   $\text{soph}_c(\alpha_{[1..n]}) = O(1)$ , for infinitely many  $n$ , since the program  $p$  that prints  $\alpha_{[1..n]}$  when  $\alpha_{[1..n]}$  is given as data satisfies  $|p| + |\alpha_{[1..n]}| \leq C(\alpha_{[1..n]}) + c$ . So,

$$\overline{\text{soph}}_c(\alpha) = \limsup_n \frac{\text{soph}_c(\alpha_{[1..n]})}{n} \leq \limsup_n \frac{O(1)}{n} = 0.$$

#### 4 On the Existence of Highly Sophisticated Sequences

In this section we investigate the existence of highly sophisticated sequences. In particular, we show that the set of sequences with upper sophistication different from 0 is dense and the set of sequences with lower sophistication different from 0 is sparse. To formally present these results, we consider the standard metric in the Cantor space  $\Sigma^\infty$  and use a known result for complete metric spaces.

**Definition 10.** *In the Cantor set  $\Sigma^\infty$ , given  $\alpha$  and  $\beta$  in  $\Sigma^\infty$ , the standard metric is defined by:*

$$d(\alpha, \beta) = \max_i \{2^{-i} : \alpha_i \neq \beta_i\}$$

It is well known that  $(\Sigma^\infty, d)$  is a complete metric space, i.e., is a metric space in which every Cauchy sequence converges.

*Remark 2.* The less the distance between  $\alpha$  and  $\beta$ , the bigger the initial segment common to  $\alpha$  and  $\beta$ .

Consider the following set:

$$V_{i,e} = \{\alpha \in \Sigma^\infty : (\forall n \geq i) \text{soph}_c(\alpha_{[1:n]}) \leq n^e\}$$

where  $c$  and  $e$  are fixed constants and  $0 < e < 1$  is a rational number.  $V_{i,e}$  is the set of sequences that from its  $i^{th}$  bit their initial segments are not highly sophisticated.

Then:

---

<sup>1</sup> In fact, almost all sequences  $\alpha$  with respect to the binary measure have this property, since  $\sum_{n \in \mathbb{N}} 2^{-\log n - \log \log n}$  converges.

1.  $V_{i,e} \subset V_{i+1,e}$ .

If  $\alpha \in V_{i,e}$  then for all  $n \geq i$ ,  $\text{soph}_c(\alpha_{[1:n]}) \leq n^e$ . In particular, for all  $n \geq i+1$ , we have that  $\text{soph}_c(\alpha_{[1:n]}) \leq n^e$ . So  $\alpha \in V_{i+1,e}$ .

2.  $V_{i,e}$  is non empty.

For example the sequence such that all bits are equal to 0 has low sophistication since it has low Kolmogorov complexity.

3. For all  $e$  and sufficiently large  $i$ ,  $V_{i,e} \neq \Sigma^\infty$ .

Set  $n > i+1$  such that  $n - 2\log n - 2c > n^e$ . Notice that  $\lim_{n \rightarrow \infty} \frac{n - 2\log n - 2c}{n^e} = \infty$  and thus such  $n$  exists. Consider  $x \in \Sigma^n$  that satisfies  $\text{soph}_c(x) \geq n - 2\log n - 2c$ . This string exists by Theorem 1. Then the sequence  $\alpha = x000\dots$  satisfies  $\text{soph}_c(\alpha_{[1:n]}) \geq n - 2\log n - 2c > n^e$  and thus  $\alpha \notin V_{i,e}$ .

4. All sets  $V_{i,e}$  are closed subsets of  $\Sigma^\infty$ .

To prove this fact we show that  $\Sigma^\infty - V_{i,e}$  are open subsets of  $(\Sigma^\infty, d)$ . This can be done by showing that given  $\alpha \in \Sigma^\infty - V_{i,e}$  there exists  $\varepsilon > 0$  such that  $B(\alpha, \varepsilon) = \{\beta \in \Sigma^\infty : d(\alpha, \beta) < \varepsilon\} \subset \Sigma^\infty - V_{i,e}$ .

If  $\alpha \in \Sigma^\infty - V_{i,e}$  then there exists  $n$  such that  $\text{soph}_c(\alpha_{[1:n]}) \geq n^e$ . Set  $\varepsilon = 2^{n-1}$ . Then, if  $d(\alpha, \beta) < \varepsilon$  it implies that for all  $i \leq n$ ,  $\alpha_i = \beta_i$ . So  $\text{soph}_c(\beta_{[1:n]}) = \text{soph}_c(\alpha_{[1:n]}) \geq n^e$ , which proves that  $\beta \in \Sigma^\infty - V_{i,e}$ .

We now present a known result for complete metric spaces that gives a condition to prove that a certain sequence of subsets of a metric space have dense intersection.

**Theorem 4 (Baire's theorem).** Let  $(X, m)$  be a complete metric space and let  $(A_n)_{n \in \mathbb{N}}$  be a sequence of open dense subsets of  $X$ . Then  $\bigcap_{n \in \mathbb{N}} A_n$  is dense.

So if we prove that  $\Sigma^\infty - V_{i,e}$  are dense then we prove that the set of all highly sophisticated sequences is dense in  $\Sigma^\infty$  since  $\bigcap_{i \in \mathbb{N}} \Sigma^\infty - V_{i,1-1/i} = \Sigma^\infty - \bigcup_{i \in \mathbb{N}} V_{i,1-1/i}$ .

Notice that  $\alpha \in \bigcup_{i \in \mathbb{N}} V_{i,1-1/i}$  if and only if  $\alpha$  satisfies  $\overline{\text{soph}}_c(\alpha) = 0$ .

To prove that each  $\Sigma^\infty - V_{i,e}$  is dense it is sufficient to show that given  $\varepsilon > 0$  and  $\alpha \in V_{i,e}$  there exists a sequence  $\beta \in \Sigma^\infty - V_{i,e}$  such that  $d(\alpha, \beta) < \varepsilon$ .

Intuitively this fact is true since we can consider the first bits of  $\alpha$  (to insure that  $d(\alpha, \beta) < \varepsilon$ ) and construct a sophisticated string with that prefix of a reasonable size.

**Proposition 6.** Each set  $\Sigma^\infty - V_{i,e}$  is dense.

*Proof.* Let  $\alpha$  be an element in  $V_{i,e}$  and  $\varepsilon > 0$  a real number. We construct  $\beta$  as follows:

Let  $i_0$  be the index such that  $2^{-i_0} \leq \varepsilon/2$ . Set  $\beta_i = \alpha_i$  for all  $i \leq i_0$ . With this condition we guarantee that  $d(\alpha, \beta) < \varepsilon$ .

Let  $n$  be sufficiently large satisfying  $n - 2\log n - 6i_0 - 2kc > (n + i_0)^e$  and  $n > i$ . Consider  $x \in \Sigma^n$  such that  $\text{soph}_{3i_0+kc}(x) > n - 2\log n - 6i_0 - 2kc$  where  $k$  is also a positive constant. Notice that since  $i_0$  is a constant value depending only on  $\varepsilon$ , this string  $x$  always exists by theorem 1. We stress that  $y = \alpha_{[1:i_0]}x$  has high sophistication.

Let  $p$  be a program corresponding to  $\text{soph}_c(y)$ . Then there exists data  $f$  such that  $|p| + |f| \leq C(y) + c$ . Consider the following program  $q$ :

**Algorithm 5** Given some data  $\langle 1^i 0 i f' \rangle$ :

1. If the data does not have this structure, stop outputting  $\epsilon$  the empty string.  
Otherwise
2. Run  $U$  with  $p$  and  $f'$  to produce  $y'$ .

3. Print all the string except the first  $i$  bits.

Since  $p$  is total it follows that  $q$  is also total. Notice that if  $f' = f$  and  $i = i_0$ ,  $U(q, \langle 1^{i_0} 0 f' \rangle) = x$ . Notice also that:

$$\begin{aligned} |q| + |\langle 1^{i_0} 0 f' \rangle| &\leq |p| + |f| + 2i_0 + O(1) = \text{soph}_c(y) + 2i_0 + O(1) \\ &\leq C(y) + 2i_0 + O(1) \\ &\leq C(\alpha_{[1:i_0]}) + C(x) + 2i_0 + O(1) \leq C(x) + 3i_0 + kc \end{aligned}$$

So  $|q|$  is an upper bound of  $\text{soph}_{3i_0+kc}(x)$  and

$$\text{soph}_c(y) \geq \text{soph}_{3i_0+kc}(x) > n - 2 \log n - 6i_0 - 2kc > (n + i_0)^e = |y|^e$$

So let  $\beta$  be the sequence  $\beta = \alpha_{[1:i_0]}x0000\cdots$ . Since  $n > i$  the above discussion means that  $\beta \notin V_{i,e}$ .

Thus, using Theorem 4 and Proposition 6 we have:

**Theorem 6.** *The set of sequences  $\alpha$  such that for some  $c$   $\overline{\text{soph}}_c(\alpha) \neq 0$  is dense.*

In what follows, we prove that the set of sequences with lower sophistication different from 0 is sparse. Consider now the following set:

$$A_{i,e} = \{\alpha \in \Sigma^\infty : \exists n > i \text{ such that } \text{soph}_c(\alpha_{[1..n]}) < n^e\}$$

where  $c$  and  $e$  are constants such that  $0 < e < 1$  and is a rational number.

$A_{i,e}$  is the set of sequences for which there exists a value  $n > i$  such that the sophistication of  $\alpha_{[1..n]}$  is less than  $n^e$ .

**Proposition 7.**  *$A_{i,e}$  is open.*

*Proof.* The idea is to use the same argument that proves that  $\Sigma^\infty - V_{i,e}$  is open. Let  $\alpha$  be a sequence in  $A_{i,e}$ . Let  $n$  be the index such that  $\text{soph}_c(\alpha_{[1..n]}) < n^e$ . Then taking  $\varepsilon = 2^{-n}$  it follows that if  $d(\alpha, \beta) < \varepsilon$  then  $\beta_i = \alpha_i$  for all  $i \leq n$ . So  $\beta \in A_{i,e}$ .

**Proposition 8.**  *$A_{i,e}$  is dense.*

*Proof.* Let  $\alpha \in \Sigma^\infty - A_{i,e}$  and  $\varepsilon > 0$ . Consider  $\beta$  defined by:  $\beta_i = \alpha_i$  for all  $i \leq -\log \varepsilon$  and  $\beta_i = 0$  otherwise. Then  $d(\alpha, \beta) < \varepsilon$  and it is clear that  $\beta \in A_{i,e}$ .

So using again Baire's theorem we conclude the following:

**Theorem 7.** *The set of sequences  $\alpha$  such that for some constant  $c$ ,  $\underline{\text{soph}}_c(\alpha) \neq 0$  is sparse.*

*Proof.* By Baire's theorem  $\bigcap_{i \in \mathbb{N}} A_{i,1-1/i}$  is dense. So if  $\alpha \in \bigcap_{i \in \mathbb{N}} A_{i,1-1/i}$  then there exists a sequence of indexes  $(i_n)_{n \in \mathbb{N}}$  such that  $\text{soph}_c(\alpha_{[1..i_n]}) \leq (i_n)^e$ . So  $\underline{\text{soph}}_c(\alpha) = 0$ . Notice that the sequence  $(i_n)_{n \in \mathbb{N}}$  can be constructed inductively.

## 5 Sophistication vs Depth of Infinite Sequences

Bennett [Ben88] formally defined the *s-significant logical depth* of an object  $x$  as the time required by a standard universal Turing machine to generate  $x$  by a program that is no more than  $s$  bits longer than the shortest descriptions of  $x$ . A deep string  $x$  should take a lot of effort to construct from its short description. Incompressible strings are trivially constructible from their shortest description, and therefore computationally shallow.

Koppel [Kop87], claimed that sophistication and logical depth are equivalent, for all infinite sequences. However the proof uses a different definition of logical depth imposing totality in the functions defining it.

The claimed equivalence between sophistication and logical depth would be, in fact, an unexpected result, as sophistication measures program length that is upper bounded by the length of the string and logical depth measures running time that can grow unbounded.

In [AF01] the authors proved that computational depth and sophistication are distinct measures of complexity for finite strings contradicting Koppel's intuition. In this section, we reinforce the distinctness of these two measures for infinite sequence by proving the existence of sequences that are deep but not very sophisticated.

**Definition 11.** *The dimensional depth of a sequence  $\alpha$  is defined as*

$$\text{depth}_{\text{dim}}^t(\alpha) = \liminf_{n \rightarrow \infty} \frac{\delta(\alpha_{[1..n]}/2^{-K^t(\alpha_{[1..n]})})}{n}.$$

where  $\delta$  is the random deficiency defined by  $\delta(x | \mu) = \left\lfloor \log \frac{2^{-K(x)}}{\mu(x)} \right\rfloor$ .

The next example shows that the diagonal of the Halting Problem is deep but not very sophisticated.

*Example 1.* Let  $H$  be the diagonal of Halting problem i.e.,  $H = \{i : M_i(i) \text{ halts}\}$ . From Barzdini's Lemma (see [LV97]) we know that for all  $n$

$$C(\chi_{H[1..n]}) \leq C(n) + C(\chi_{H[1..n]}|n) \leq 2 \log n.$$

So

$$\underline{\text{soph}}_c(\chi_H) \leq \overline{\text{soph}}_c(\chi_H) = \limsup_n \frac{\text{soph}_c(\chi_{H[1..n]})}{n} \leq \limsup_n \frac{2 \log n}{n} = 0$$

On the other hand we know that:

$$\begin{aligned} \text{depth}_{\text{dim}}^t(\chi) &= \liminf_{n \rightarrow \infty} \frac{\delta(\chi_{H[1..n]}/2^{-K^t(\chi_{H[1..n]})})}{n} \\ &= \liminf_{n \rightarrow \infty} \frac{\log(2^{-K(\chi_{H[1..n]})}/2^{-K^t(\chi_{H[1..n]})})}{n} \\ &= \liminf_{n \rightarrow \infty} \frac{K^t(\chi_{H[1..n]}) - K(\chi_{H[1..n]})}{n} \\ &= \liminf_{n \rightarrow \infty} \frac{\text{depth}^t(\chi_{H[1..n]})}{n} = 1 \end{aligned}$$

The last inequality is due to the fact that  $\chi_{H[1..n]}$  is very deep, i.e.,  $\text{depth}^t(\chi_{H[1..n]}) \approx n$ .

## Conclusions

In this paper we proposed two definitions for sophistication for infinite sequences. The major improvement of this work to the existent one is the fact that sophistication is a concept well defined for all sequences. The most important result proved here concerns the existence of highly sophisticated sequences. In fact, we proved that if sophistication is defined with  $\limsup$  then the set of sequences that have sophistication different from 0 is dense and if it is defined with  $\liminf$  this set is rare. The last result stating that sophistication and depth are different complexity measures strengthens a similar result already proved in [AF01] for finite strings.

### Acknowledgments

We thank to the CiE 2008 anonymous reviewers for pointing out some problems in the proofs of some results.

### References

- [AF01] L. Antunes and L. Fortnow, “*Sophistication Revisited*”, Proceedings of the 30th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, 2719:267-277, Springer, 2003.
- [AFMV06] L. Antunes, L. Fortnow, D. van Melkebeek and N. V. Vinodchandran, “*Computational depth: concept and applications*”, Theor. Comput. Sci., 354(3): 391-404, 2006.
- [AMVS07] L. Antunes, A. Matos, A. Souto and P. Vitányi, “*Depth as Randomness Deficiency*”, submitted to Theory of Computing Systems.
- [AHLM07] K. Athreya, J. Hitchcock, J. Lutz and E. Mayordomo, “*Effective Strong Dimension in Algorithmic Information and Computational Complexity*”, SIAM Journal on Computing, 37(3):671-705, 2007
- [AK91] H. Atlan and M. Koppel, *An almost machine-independent theory of program-length complexity, sophistication, and induction*, in Inf. Sci. 56(1-3):23-33, Elsevier Science Inc., 1991
- [Ben88] C. H. Bennett. Logical depth and physical complexity. In R. Herken, editor, *The Universal Turing Machine: A Half-Century Survey*, pages 227–257. Oxford University Press, 1988.
- [Cha66] G. Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM*, 13:4(1966), 145-149.
- [Cov85] T. M. Cover, *Kolmogorov Complexity, Data Compression and Inference*, Chapter in The Impact of Processing Techniques on Communications. Series E: Applied Sciences (91), Martinus Nijhoff Publishers, 1985
- [Gacs88] P. Gacs, “*Lecture notes on descriptional complexity and randomness*”, Tech. report, Boston University, Computer Science Dept., Boston, MA 02215, 1988.
- [GTV01] P. Gács, John Tromp and P. Vitányi. Algorithmic Statistics. *In IEEE Trans. Inform. Theory*, 47:6, 2443-2463, 2001
- [Haus79] F. Hausdorff, “*Dimension und äußeres Maß*”, in Math. Ann., 79:157–179, 1919
- [Kol65] A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problemy Inform. Transmission*, 1(1): 1-7,1965
- [Kop87] M. Koppel, “*Complexity, Depth, and Sophistication*”, in Complex Systems 1, pages = 1087-1091, 1987
- [Kop88] M. Koppel, “*Structure*”, in “The universal Turing machine: a half-century survey”, pages = 403–419, Springer-Verlag New York, Inc., 1988
- [LV97] M. Li and P. Vitányi, “*An introduction to Kolmogorov complexity and its applications*”, Springer Verlag, 2nd edition, 1997.
- [Lut00] J. H. Lutz, “*Dimension in complexity classes*”, Proceedings of the 15th IEEE Conference on Computational Complexity, IEEE Computer Society Press, 2000.
- [Lut02] J. H. Lutz, “The dimensions of individual strings and sequences”, Technical Report cs. CC/0203017, ACM Computing Research Repository, 2002.
- [ML71] P. Martin-Löf, “*Complexity oscillations in infinite binary sequences*”, Zeitschrift Wahrscheinlichkeitstheorie und Verwandte Gabiete, 19, pages=225-230, 1971.
- [May02] E. Mayordomo, “*A Kolmogorov complexity characterization of constructive Hausdorff dimension*”. *Information Processing Letters*, 84:1-3, 2002.
- [Sol64] R. Solomonoff. “*A Formal Theory of Inductive Inference, Part I*”, *Information and Control*, 7(4):1-22, 1964
- [Sul84] D. Sullivan, “*Entropy, Hausdorff measures old and new, and limit sets of geometrically finite Kleinian groups*”, in Acta Math., 153:259–277, 1984
- [Tic82] C. Tricot, “*Two definitions of fractional dimension*” in Math. Proc. Cambridge Philos. Soc., 91:57–74, 1982.
- [V'y99] V.V. V'yugin. “*Algorithmic complexity and stochastic properties of finite binary sequences.*” *The Computer Journal*, 42(4):294-317, 1999.

# On Detecting Deadlock in the $\pi$ -Calculus

Tiago Azevedo<sup>1</sup>, Mario R. F. Benevides<sup>2</sup>, Fábio Protti<sup>3</sup>, and Marcelo Sihman<sup>1</sup>

<sup>1</sup> COPPE/PESC - Universidade Federal do Rio de Janeiro  
Caixa Postal 68511 - 21945-970 - Rio de Janeiro - RJ - Brazil

<sup>2</sup> IM and COPPE/PESC - Universidade Federal do Rio de Janeiro  
Caixa Postal 68511 - 21945-970 - Rio de Janeiro - RJ - Brazil  
<sup>3</sup> IM and NCE - Universidade Federal do Rio de Janeiro  
Caixa Postal 2324 - 20001-970 - Rio de Janeiro - RJ - Brazil

[tiagoazevedo@cos.ufrj.br](mailto:tiagoazevedo@cos.ufrj.br), [mario@cos.ufrj.br](mailto:mario@cos.ufrj.br)  
[fabiop@nce.ufrj.br](mailto:fabiop@nce.ufrj.br), [marcelosihman@ufrj.br](mailto:marcelosihman@ufrj.br)

**Abstract.** In this work we deal with the notion of deadlock in the asynchronous polyadic  $\pi$ -calculus. Based on the proposed definition, we show that detecting deadlock in the asynchronous polyadic  $\pi$ -calculus is usually an undecidable problem. However, when restricting the language by forbidding replicating input prefixed processes, the detection turns out to be decidable, and a polynomial deadlock detection algorithm is presented.

**Key words:** Concurrency; Program Correctness; Programming Calculi; Program Specification.

## 1 Introduction

When specifying distributed systems and concurrent programs, one crucial question is to ensure absence of deadlock and run-time errors. However, standard mechanisms for detecting automatically such situations can hardly be found in general, since those notions are usually difficult to be dealt with.

In a previous work by Vasconcelos and Ravara [9], the notion of run-time (communication) errors in the asynchronous polyadic  $\pi$ -calculus was shown to be undecidable. Their result supports the use of type systems, a static method of analysis in preventing run-time errors. In many cases, when a language enjoys the subject reduction property, it is possible to prove type-safety, and this implies the absence of run-time errors. Therefore, the method is correct, but not complete as not all well-behaved programs are well-typed. However, the undecidability of the run-time error notion does not allow us to do better.

Although this undecidability result is not surprising, its proof uses and extends non-trivial results from the encoding of the  $\lambda$ -calculus into the asynchronous  $\pi$ -calculus described in [6, 8] (see Section 2.1), and adapts a technique to transfer undecidability results: it reduces the problem to a known undecidable property in the source language, and prove that if it is decidable in the target language, it would also be in the source language, attaining a contradiction.

The motivation of this work is to extend the above result to the notion of deadlock. We believe that it is worth obtaining formal proofs of undecidability

for such situations in the  $\pi$ -calculus, since they have remained up to now as “folklore” results. The use of the asynchronous  $\pi$ -calculus as the setting of this work was shown to be an easier way to give continuity to the ideas inspiring the cited paper [9]. This choice is justified by the fact that if these results hold in the asynchronous  $\pi$ -calculus, then they must also hold in superlanguages (e.g. the synchronous  $\pi$ -calculus) – while the inverse implication may not be necessarily true. It turns out more useful and simpler to prove undecidability results using this smaller language, which is the basis of several implementations.

The definition of deadlock proposed here (Section 3) intends to capture the potentiality a process possesses of reaching this situation after performing an *empty experiment* (a sequence of silent actions, where external agents observe nothing). A similar definition of deadlock can be found in [10–14], where the deadlock state is defined through the concept of *annotations*.

The undecidability proof presented here (Section 4) follows the strategy employed in [9]: the problem of deciding whether a lambda term has a normal form [3] is reduced to the problem of deciding whether a process is potentially capable of reaching a deadlock state. This is done by defining a computable function  $f$  from  $\lambda$ -terms into processes of the  $\pi$ -calculus such that  $f(M)$  is a deadlocking process if and only if the  $\lambda$ -term  $M$  is convergent. The definition of  $f$  embodies the above-mentioned encoding of the lazy  $\lambda$ -calculus into the asynchronous  $\pi$ -calculus in [6, 8].

When restricting the asynchronous polyadic  $\pi$ -calculus by forbidding replicating input prefixed process, deadlock detection turns out to be decidable. In section 5, a polynomial-time deadlock detection algorithm is presented for this case.

## 2 Background

The concepts and definitions of the asynchronous polyadic  $\pi$ -calculus are used here as usual [5, 4]. Below, we briefly present some definitions. Small letters  $a, b, p, q, x, y, \dots$  are *names*,  $\tilde{v}$  is a sequence of names, and  $\tilde{x}$  is a sequence of pairwise distinct names.

The set  $\Pi$  of processes of the asynchronous polyadic  $\pi$ -calculus is given by the following grammar:  $P ::= \bar{a}[\tilde{v}] \mid a(\tilde{x}).P \mid P|Q \mid \nu x P \mid !a(\tilde{x}).P \mid \mathbf{0}$

Here, ‘|’ is the operator for parallel composition of processes; ‘ $\nu$ ’ restricts the scope of a name to a process; and ‘ $!a(\tilde{x}).P$ ’ is a replicating input prefixed process. Prefixed operations ( $\alpha$ . and  $\nu a$ ) bind more tightly than composition.

The set of *free names* in a process  $P$  is denoted by  $fn(P)$ . *Alpha-conversion* is denoted by  $\equiv_\alpha$ . If  $\tilde{v}$  and  $\tilde{x}$  are sequences with the same length, the process  $P[\tilde{v}/\tilde{x}]$  is obtained by substituting the names in  $\tilde{v}$  for every free occurrence of the corresponding name in  $\tilde{x}$ .

The *structural congruence*, written  $\equiv$ , is determined by the following equations:  $P \equiv Q$  if  $P \equiv_\alpha Q$ ,  $P|\mathbf{0} \equiv P$ ,  $P|Q \equiv Q|P$ ,  $(P|Q)|R \equiv P|(Q|R)$ ,  $\nu x \mathbf{0} \equiv \mathbf{0}$ ,  $\nu xy P \equiv \nu yx P$ ,  $\nu x P|Q \equiv \nu x(P|Q)$  if  $x \notin fn(Q)$ ,  $!P \equiv !P \mid P$ .

The set of *action labels* is given by the following grammar, where  $\{\tilde{x}\} \subseteq \{\tilde{v}\} \setminus \{a\}$ :

$$\alpha ::= \tau \mid a(\tilde{v}) \mid \nu \tilde{x} \bar{a}[\tilde{v}]$$

The symbol  $\tau$  denotes a *silent action*, an internal communication within a process. The remaining actions are *observable actions* (external communications between processes): the *input action*  $a(\tilde{v})$  denotes the reception on  $a$  of the sequence of names  $\tilde{v}$ , and the *output action*  $\nu \tilde{x} \bar{a}[\tilde{v}]$  denotes the emission to  $a$  of the sequence of names  $\tilde{v}$ , where some of them are bound. The sets  $\text{fn}(\alpha)$  and  $\text{bn}(\alpha)$  consist of the *free names* and the *bound names* in an action  $\alpha$ , respectively. Given an observable action  $\alpha = a(\tilde{v})$  or  $\alpha = \nu \tilde{x} \bar{a}[\tilde{v}]$ , the name  $a$  is called the *subject* of the action and is denoted by  $\text{subj}(\alpha)$ . The *asynchronous transition relation* contains exactly those transitions which can be inferred from the following rules:

$$\begin{array}{ll} (IN) & a(\tilde{x}).P \xrightarrow{a(\tilde{v})} P[\tilde{v}/\tilde{x}] \\ (OUT) & \bar{a}[\tilde{v}] \xrightarrow{\bar{a}[\tilde{v}]} \mathbf{0} \\ (COM) & a(\tilde{x}).P \mid \bar{a}[\tilde{v}] \xrightarrow{\tau} P[\tilde{v}/\tilde{x}] \\ (PAR) & \frac{P \xrightarrow{\alpha} Q}{P \mid R \xrightarrow{\alpha} Q \mid R} \quad (\text{bn}(\alpha) \cap \text{fn}(R) = \emptyset) \\ (RES) & \frac{P \xrightarrow{\alpha} Q}{\nu x P \xrightarrow{\alpha} \nu x Q} \quad (x \notin \text{bn}(\alpha) \cup \text{fn}(\alpha)) \\ (OPEN) & \frac{P \xrightarrow{\nu \tilde{x} \bar{a}[\tilde{v}]} Q}{\nu x P \xrightarrow{\nu \tilde{x} \bar{a}[\tilde{v}]} Q} \quad (a \notin \{x, \tilde{x}\}) \\ (STRUCT) & \frac{P \xrightarrow{\alpha} P'}{Q \xrightarrow{\alpha} Q'} \quad \text{if } P \equiv Q \text{ and } P' \equiv Q' \end{array}$$

Note that there is no specific rule for replication. Its required behavior is given by the last rule of structural congruence.

The process  $a(\tilde{x}).P$  and  $\bar{a}[\tilde{v}]$  in rule (COM) are called *component processes*. The symbol  $\Rightarrow$  denotes the reflexive and transitive closure of  $\xrightarrow{\tau}$ , and  $\xrightarrow{\alpha}$  denotes  $\Rightarrow \xrightarrow{\alpha} \Rightarrow$ . Let us simply write  $P \xrightarrow{\alpha} Q$  (resp.  $P \xrightarrow{\alpha} Q$ ) to mean that there exists  $Q$  such that  $P \xrightarrow{\alpha} Q$  (resp.  $P \xrightarrow{\alpha} Q$ ).

Let  $s$  be a sequence  $s = \alpha_1, \alpha_2, \dots, \alpha_n$  of action labels. An *experiment*  $\xrightarrow{s}$  is defined in the following way:

$$\xrightarrow{s} \stackrel{\text{def}}{=} \Rightarrow \xrightarrow{\alpha_1} \Rightarrow \xrightarrow{\alpha_2} \Rightarrow \dots \xrightarrow{\alpha_n} \Rightarrow .$$

Again, write  $P \xrightarrow{s}$  to mean that there exists  $Q$  such that  $P \xrightarrow{s} Q$ .

The *empty experiment* is defined for  $n = 0$ ; in this case,  $\xrightarrow{s}$  is the same as  $\Rightarrow$ , indicating that silent actions may occur even if we observe nothing.

When no silent actions can occur within a process  $P$ , we shall call  $P$  *stable* (cfr. [7], p.35). Stability implies that  $P$  needs an observable action to proceed any further. Processes that have already finished all their possible computations and are reduced to  $\mathbf{0}$  are also stable.

A relation  $\mathcal{R} \subseteq \Pi \times \Pi$  is called an *expansion*(we refer the reader to [8]) if  $P\mathcal{R}Q$  implies:

1. if  $P \xrightarrow{\alpha} P'$  then there exists  $Q'$  such that  $Q \xrightarrow{\alpha} Q'$  and  $P'\mathcal{R}Q'$ ;
2. if  $Q \xrightarrow{\alpha} Q'$  and  $\alpha$  is an observable action then there exists  $P'$  such that  $P \xrightarrow{\alpha} P'$  and  $P'\mathcal{R}Q'$ ;
3. if  $Q \xrightarrow{\tau} Q'$  then either  $P\mathcal{R}Q'$  or there exists  $P'$  such that  $P \xrightarrow{\tau} P'$  and  $P'\mathcal{R}Q'$ .

We write  $P \lesssim Q$  if  $P\mathcal{R}Q$  for some expansion  $\mathcal{R}$ . Relation  $\lesssim$  is a preorder and a congruence.

## 2.1 Encoding the lazy $\lambda$ -calculus into the asynchronous polyadic $\pi$ -calculus.

The transfer of results from the  $\lambda$ -calculus to the asynchronous polyadic  $\pi$ -calculus is achieved by using the encoding of the lazy  $\lambda$ -calculus described below.

**Definition 1.** [2, 3, 8] *The set  $\Lambda$  of  $\lambda$ -terms is defined by the following grammar, where  $x$  ranges over the set of  $\lambda$ -calculus variables:*

$$M ::= x \mid \lambda x.M \mid MN$$

Free variables, closed terms, substitution, alpha-conversion etc. are defined as usual. The reduction relation is  $\longrightarrow$ , and the reflexive and transitive closure of  $\longrightarrow$  is  $\Longrightarrow$ . We write  $M \downarrow$  if  $M$  is convergent, and  $M \uparrow$  otherwise. In the lazy  $\lambda$ -calculus [1], the redex is always at the left extreme of a term.

The proofs in Section 5 need the following tools:

**Theorem 1.** [3] *The predicate ' $M \downarrow$ ' is undecidable.*  $\square$

**Definition 2.** [6, 8] *The encoding of the lazy  $\lambda$ -calculus into the asynchronous polyadic  $\pi$ -calculus is given by the following rules:*

$$\begin{aligned} \llbracket \lambda x.M \rrbracket_p &\stackrel{\text{def}}{=} p(x, q) \cdot \llbracket M \rrbracket_q \\ \llbracket x \rrbracket_p &\stackrel{\text{def}}{=} \bar{x}[p] \\ \llbracket MN \rrbracket_p &\stackrel{\text{def}}{=} \nu u v (\llbracket M \rrbracket_u \mid \bar{u}[v, p] \mid !v(q) \cdot \llbracket N \rrbracket_q) \end{aligned}$$

**Lemma 1.** [8, 9] *The following properties of the above encoding are valid:*

- (a) *If  $M \Longrightarrow \lambda x.N$  then  $\llbracket M \rrbracket_p \xrightarrow{p(x,q)} \gtrsim \llbracket N \rrbracket_q$*
- (b) *If  $M \Longrightarrow x$  then  $\llbracket M \rrbracket_p \xrightarrow{\bar{x}[p]} \gtrsim \mathbf{0}$*
- (c)  *$M \uparrow$  if and only if for no observable  $\alpha$   $\llbracket M \rrbracket_p \xrightarrow{\alpha}$*

### 3 Notion of deadlock

In this section we define the set  $DEAD$  of processes which are understood to be potentially deadlocked.

Informally,  $P$  is capable of *deadlock* if it may reach a situation in which the computation cannot evolve internally. The first definition of this section says that such  $P$  is a deadlocking process if it reduces to a restricted composition of mutually non-communicating stable processes  $P_1, P_2, \dots, P_n$ .

We use the following notation: for a process  $P$ , let

$$obs(P) = \{ subj(\alpha) \mid \alpha \text{ is observable and } P \xrightarrow{\alpha} \}.$$

In addition, say that two observable actions  $\alpha, \mu$  are *complementary* if  $subj(\alpha) = subj(\mu)$  and either  $\alpha$  is an input action and  $\mu$  is an output action, or  $\alpha$  is an output action and  $\mu$  is an input action.

**Definition 3.** *The set  $DEAD$  of processes which are capable of deadlock is defined in the following way:*

$DEAD = \{P \mid P \Longrightarrow \nu\tilde{x}(P_1 \mid P_2 \mid \dots \mid P_n),$   
 where: (i)  $P_j$  is stable, for all  $j = 1, \dots, n$ ;  
 (ii) for any  $a$  and for all  $j = 1, \dots, n$ , if  $a \in obs(P_j)$  then  $a$  occurs in  $\tilde{x}$ ;  
 (iii) for all  $j, k$  ( $j \neq k$ ), there are no complementary actions  $\alpha, \mu$  such  
     that  $P_j \xrightarrow{\alpha}$  and  $P_k \xrightarrow{\mu}$ ;  
 (iv)  $obs(P_i) \neq \emptyset$  for some  $i \in \{1, \dots, n\}$ . }

Let us discuss the elements of the above definition.

- Condition (i) says that every  $P_j$  needs an observable action (with respect to itself) to proceed. (Note that if some silent actions can still occur within some  $P_j$ , we can not properly speak of a ‘deadlocking state’.)
- Condition (ii) means that the process  $\nu\tilde{x}(P_1 \mid P_2 \mid \dots \mid P_n)$  cannot establish communication with the external world. By excluding this condition, it would be possible to continue the computation via non-restricted names.
- Condition (iii) ensures that  $P_1, \dots, P_n$  are mutually non-communicating.
- Condition (iv) excludes from the definition of  $DEAD$  those processes that terminate for all possible computations, e.g.  $Q_1 \stackrel{\text{def}}{=} \mathbf{0}$ . The case  $n = 1$  covers examples such as  $Q_2 \stackrel{\text{def}}{=} \nu a(a().\mathbf{0})$ , which is considered a deadlocking process.

### 4 Undecidability proof

Now we are ready to present the undecidability result.

**Theorem 2.** *The predicate ‘ $P \in DEAD$ ’ is undecidable.*

**Proof.** Let us show that if the predicate ‘ $P \in DEAD$ ’ is decidable, then the predicate ‘ $M \downarrow$ ’ is decidable, leading to a contradiction.

First, let  $f : \Lambda \longrightarrow \Pi$  be such that

$$f(M) \stackrel{\text{def}}{=} \nu p x_1 x_2 \dots x_n (\llbracket M \rrbracket_p)$$

where  $p$  is a new name and  $x_1, x_2, \dots, x_n$  are the free variables occurring in  $M$ . The function  $f$  is clearly computable by the encoding of the lazy  $\lambda$ -calculus into the asynchronous polyadic  $\pi$ -calculus.

Now, given  $M \in \Lambda$ , let us show that  $f(M) \in DEAD$  if and only if  $M \downarrow$ .

Assume first that  $f(M) \in DEAD$ . Under these assumption, observe that process  $\llbracket M \rrbracket_p$ , after zero or more silent actions, must reach a point in which it depends on an observable action to proceed further, since condition (iv) in the definition of  $DEAD$  must be satisfied. This means that  $\llbracket M \rrbracket_p \xrightarrow{\alpha}$ , for some observable  $\alpha$ . Therefore, by Lemma 1(c),  $M \downarrow$ .

On the other hand, assume that  $f(M) \notin DEAD$ . Since  $f(M)$  trivially satisfies conditions (ii) and (iii) in the definition of  $DEAD$ , it is mandatory that process  $\llbracket M \rrbracket_p$  keeps continuously performing internal computations, in order to invalidate either condition (i) or condition (iv). This means that  $\llbracket M \rrbracket_p$  does not perform any observable action after any sequence of silent actions. Therefore, by Lemma 1(c) again,  $M \uparrow$ .

To conclude the proof, assume that ‘ $P \in DEAD$ ’ is decidable, that is, given  $P \in \Pi$ , there is a finite time algorithm that decides whether  $P$  belongs to  $DEAD$  or not. Then the following algorithm decides whether a  $\lambda$ -term  $M$  is convergent or not: compute  $f(M)$  and verify whether  $f(M) \in DEAD$ ; if  $f(M) \in DEAD$  then the answer is ‘yes’, otherwise the answer is ‘no’. This contradicts the result of Theorem 1.  $\square$

## 5 An algorithm for deadlock detection in the restricted $\pi$ -calculus

Now we define the new set  $\Pi$  of processes of the restricted asynchronous polyadic  $\pi$ -calculus, given by the following grammar:

$$\begin{aligned} P ::= & \quad \nu x Q \\ Q ::= & \quad \bar{a}[\tilde{v}] \mid a(\tilde{x}).Q \mid Q|Q' \mid \mathbf{0} \end{aligned}$$

In other words, there is a single scope restriction acting on a parallel composition of processes (also called *component processes*). This simplification is reasonable since it is easy to see that every process containing no replication can be rewritten to the above form, via alpha-conversions and structural congruences.

The algorithm for deadlock detection in the restricted  $\pi$ -calculus receives as input a process and then simulates it, step by step, testing all the execution paths. In case there is some possibility of deadlock, it is detected and the location

of the deadlock is determined. The algorithm uses the fact that the deadlock state of a process can be identified by a *knot* in the *waiting graph* (see these definitions below). In addition, the algorithm shows how to locate it.

A description of the algorithm is given below.

---

**Algorithm 1** Deadlock Detection

---

```

1: /* External call,  $P$  is the input process*/
2:  $V(G) \leftarrow \{P\}$  /* Vertex set of Global State Graph*/
3:  $E(G) \leftarrow \{\}$  /* Edge set of Global State Graph*/
4: NewGlobalState( $P$ )

5: procedure NEWGLOBALSTATE( $P$ )
6:   for all  $A$  occurring in  $P$  do
7:     Build  $G_{A,P}$ 
8:     /* $G_{A,P}$  is the waiting graph for  $A$  in  $P$ */
9:     if  $G_{A,P}$  does not contain knot then
10:      for all possible handshake  $h$  of  $A$  in  $P$  do
11:        Let  $P_h$  be the new global state
12:        if  $P_h \notin N$  then
13:           $N \leftarrow N \cup \{P_h\}$ 
14:           $E \leftarrow E \cup \{P, P_h\}$ 
15:          NewGlobalState( $P_h$ )
16:        end if
17:        if  $(P, P_h) \notin E$  then
18:           $E \leftarrow E \cup \{P, P_h\}$ 
19:        end if
20:      end for
21:    else
22:      if current action of  $A$  is restricted AND
23:        not passed as a parameter by another process then
24:          Deadlock!
25:        end if
26:      end if
27:    end for
28: end procedure

```

---

We use the term *handshake* to mean a communication over some action  $x$  between two component processes  $A$  and  $B$ , according to

rule (COM).

A *global state* is the state reached by the component processes after a sequence of handshakes starting at the input process  $P$  (which is the initial global state). Each global state represents a single situation, i.e., there cannot exist two identical global states.

The global state graph  $G = (N, E)$  is the (directed) graph where each node in  $N$  represents a global state, two nodes  $P, P' \in N$  being connected by a directed

edge  $(P, P') \in E$  labelled  $h$  whenever global state  $P'$  is reached from  $P$  after executing the handshake  $h$ . In this case, we denote  $P'$  by  $P_h$ .

The *waiting graph* of a component process  $A$  in  $P$  is the (directed) graph  $G_{A,P} = (N_P, E_{A,P})$ , where  $N_P$  is the set of component processes occurring in  $P$ , and  $E_{A,P}$  is a set of labelled edges. If an edge  $(A, B) \in E_{A,P}$  is labelled  $x$ , then it means that process  $A$  is waiting for a handshake over action  $x$  which can be provided by process  $B$ .

A *knot* in a directed graph  $G$  is a strong connected component  $C$  of  $G$  such that either (i)  $C$  consists of a single node and is equal to  $G$  or (ii)  $C$  contains at least two nodes and no directed edge leaves it.

A *scope extrusion* occurs when a process  $A$  contains a private action  $x$  shared with a process  $B$ , and intends to send this action (as a parameter) to another process  $C$ . When this action is exported to  $C$ , the scope of the restriction is extended to  $C$ , characterizing the scope extrusion.

Now we will describe the execution of the algorithm. First, for every component process  $A$  in global state  $P$ , we build  $G_{A,P}$ . Next, we check whether  $P$  is in deadlock or not. Each component process is tested separately, each one with its corresponding waiting graph.

To build  $G_{A,P}$  (line 7 of the algorithm), we look for the complement of the current action  $x$  of  $A$  in other component process (*unification algorithm*), and for each complementary action found in a component process  $B$  we add to  $E_{A,P}$  a directed edge  $(A, B)$  (if  $x$  is an input action) or  $(B, A)$  (if  $x$  is an output action), labelled with  $x$ . If the complement of  $x$  is the current action in  $B$ , then no other edge is added from/to this node, because this action will be the next to execute. Otherwise, we recursively repeat the search for complements of the current action in  $B$ , adding labelled edges to  $G_{A,P}$ . The construction of  $G_{A,P}$  finishes when no more edges can be added in any of the nodes, meaning that all the paths (sequences of edges that are incrementally added) have ended. Note that some paths may have turned out to be cycles.

After constructing  $G_{A,P}$ , our goal is to detect whether  $A$  will succeed to communicate its current action with another process in the sequel of the simulation, i.e., if  $A$  is in deadlock or not. This is equivalent to checking whether  $G_{A,P}$  contains a knot (line 8 of the algorithm). For this purpose we use a *knot detection algorithm*, which computes strongly connected components of  $G_{A,P}$ .

In case that no knot is found in  $G_{A,P}$ , the simulation continues from the current global state  $P$ . For each possible handshake  $h$  between  $A$  and another component process in  $P$ , we add a new global state  $P_h$  to  $N$  (if  $P_h$  does not exist yet) and an edge  $(P, P_h)$  to  $E$  (if it has not been added yet). See lines 9 – 19 of the algorithm. If  $P_h$  is a pre-existing global state, the simulation of this path in  $G$  finishes, because  $P_h$  has already generated its associated simulation branches; otherwise, the simulation continues until we get to the first case (coming back to a pre-existing global state) or until a deadlock is detected (lines 21 – 23). The information on which handshakes can be executed in a step, from the current global state to the next one, is obtained from the waiting graphs of the component processes.

In case that  $G_{A,P}$  contains a knot  $C$ , we have to check whether  $C$  cannot be repaired by a scope extrusion (lines 21 – 22). If so, a deadlock is detected. (line 23)

### 5.1 Complexity of the algorithm

In this section we calculate the complexity of the deadlock detection algorithm. The number of process is denoted by  $n$ , and the number of actions in the process by  $m$ .

The time needed to build the waiting graph for some process is in the worst case  $O(n)$  for creating a vertex to each process, and  $O(n^2)$  to create all edges, because each of these edges always leaves a vertex representing a process and reaches another vertex. Thus the overall complexity is  $O(n^2)$ .

To search for complementary ports, we use the unification algorithm. This algorithm has a linear time complexity on the size of the input process, thus the complexity of this task takes  $O(n + m)$  time.

The complexity to determine whether the waiting graph contains a knot is  $O(n^2)$  (linear on the size of the waiting graph), via a depth-first search.

Finally as we build a waiting graph for each process in each created global state, in the worst case we spend  $O(n)$  time (processes) times  $O(n^2)$  (waiting graphs), which gives an  $O(n^3)$  time. Moreover, we spend  $O(n^3)$  time to check whether these waiting graphs contain knots, and in the affirmative case, we search for complementary ports in  $O(n + m)$  time. As we have  $O(m)$  handshakes (because there are  $m$  actions) and therefore  $O(m)$  global states (because there is a recursive call to each global state), the complexity for the deadlock detection algorithm is  $O(m(n^3 + n^3 + m + n)) = O(m(2n^3 + n + m)) = O(n^3m + m^2)$ .

### 5.2 The CCS Case

In the CCS case, processes cannot exchange actions with other process. The algorithm for deadlock detection is thus simpler, because we can determine *a priori* which processes can execute a handshake, preventing the search for complementary ports; moreover we only build waiting graphs for restricted process. Thus the complexity for the CCS case is  $O(m(n^3 + n^3)) = O(2mn^3) = O(mn^3)$ .

## References

1. S. Abramsky. The lazy lambda calculus. *Research Topics in Functional Programming*, pp. 65–116. Addison-Wesley, 1989.
2. H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics. Studies in Logic*, vol. 103. North Holland, 1984. Revised edition.
3. J. R. Hindley and J. P. Seldin. *Introduction to Combinators and  $\lambda$ -Calculus*. Cambridge University Press, 1986, p. 63.
4. K. Honda and M. Tokoro. An object calculus for asynchronous communication. *5th European Conference on Object-Oriented Programming*, LNCS 512 (1991) 141–162. Springer-Verlag.

5. R. Milner. The polyadic  $\pi$ -calculus: a tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, eds. *Logic and Algebra of Specification*, Springer-Verlag, 1993.
6. R. Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science* 2(2) (1992) 119–141.
7. R. Milner. Communicating and Mobile Systems: the  $\pi$ -Calculus. Cambridge University Press, 1999.
8. D. Sangiorgi and D. Walker. The Pi-Calculus: A Theory of Mobile Processes. Cambridge University Press, 2001.
9. V. T. Vasconcelos and A. Ravara. Communication errors in the  $\pi$ -calculus are undecidable. *Information Processing Letters* 71(5-6) (1999), 229–233.
10. Naoki Kobayashi A New Type System for Deadlock-Free Processes *CONCUR* 233-247, Springer, Lecture Notes in Computer Science, 2006
11. Naoki Kobayashi Type-based information flow analysis for the pi-calculus *Acta Informatica*, Vol. 42, No 4-5 pp. 291-347, 2005
12. Naoki Kobayashi and Shin Saito and Eijiro Sumii An Implicitly-Typed Deadlock-Free Process Calculus Lecture Notes in Computer Science, 1877, pp. 489–??,2000
13. Naoki Kobayashi A Partially Deadlock-Free Typed ProcessCalculus ACM Transactions on Programming Languages and Systems Volume 20, pp. 436–482, 1998
14. Eijoro Kobayashi and Naoki Kobayashi A Generalized Deadlock-Free Process Calculus HLCL, 1998

# Algorithms for Analytic Functions and Applications to Toeplitz Operators

E. J. Beggs and A. Gerber

Mathematics Department, Swansea University, Singleton Park,  
Swansea SA2 8PP, Wales  
`E.J.Beggs@Swansea.ac.uk, A.Gerber@Swansea.ac.uk`

**Abstract.** In this paper we present two algorithms for analytic functions. They are based on a quadtree datatype and we use methods from object oriented programming to describe them. The first algorithm decides at each level  $n$  whether a given square of length  $2^{-n}$  lies further than a minimal distance away from the graph of a function  $f(z)$  and then computes the winding number for the centre of this square with respect to  $f(z)$ . The second algorithm computes zeros of an analytic function  $f(z)$ . Many algorithms for computing zeros of analytic functions exist already. Our algorithm explicitly deals with the problem of when zeros occur on the boundaries of squares and when the domain of the function is potentially complicated. We perturbing the relevant edges of such squares to guarantee the method working. Then, we briefly explain how we can apply these algorithms to particular linear operators on Hilbert spaces called Toeplitz operators. The first algorithm allows us to draw the spectrum of such a Toeplitz operator  $T(a)$ . The second algorithm is used to determine the spectrum of a perturbed Toeplitz operator.

**Key words:** computable analysis, real computation, zeros of analytic functions, Toeplitz operators

## 1 Introduction

Computing the number of zeros of an analytic function in a given region by sub-dividing that region into smaller and smaller squares or rectangles is a well-known procedure. Several algorithms for locating the zeros of complex analytic functions  $f : \mathbb{C} \rightarrow \mathbb{C}$  exist. In [9] a derivative-free method for computing zeros of analytic functions specifically taylored to certain problems from physics is given. Dellnitz et al. [8] present an argument based algorithm that sub-divides a given region into rectangles, where any rectangle, which does not contain a zero, is discarded. They point out that zeros could be on the boundary of a rectangle and assign the value  $\infty$  to these rectangles but do not explicitly deal with them.

In [11], [12] Kravanja and Van Barel present a derivative-free algorithm based on winding numbers for computing zeros of analytic functions. The package ZEAL [13] written in Fortran 90 computes the actual zeros. If zeros occur on the initial outer boundary this boundary is perturbed by ZEAL. If an inner edge is too close to a zero, this edge is shifted. The guiding principle is that

subdivisions of rectangles are chosen randomly and in an asymmetric way so that the probability of a zero lying on an inner edge is effectively zero. The total number of zeros in a (perturbed) square is obtained in the same way as in the package ZEBEC [14] (see also [16]).

Here, we propose an algorithm to detect the zeros of an analytic function within a given, possibly complicated domain. We use a quadtree data structure that allows us to keep track of all rectangles examined and we also move edges to within a maximal threshold distance should there be a possibility of a zero occurring on the that edge. The difference between our algorithm and the one presented in [15] is similar in nature to the difference between a computable subset of  $\mathbb{N}$  and a computably enumerable subset. For a computable subset of  $\mathbb{N}$  the algorithm computing that set can find all elements in an initial segment of  $\mathbb{N}$  in finite time. The algorithm we present can determine, up to a given error  $\varepsilon > 0$ , all zeros of the function more than a given distance  $r > 0$  from the boundary of a bounded domain, and terminate in finite time.

We consider the Hilbert space  $\mathcal{H} = l^2(\mathbb{N})$ , i.e. the space of square summable sequences  $\|x\| = \sqrt{\sum_{n=1}^{\infty} |x_n|^2} < \infty$  over  $\mathbb{C}$  (for further details consult for instance [7] or [18]). Banded Toeplitz operators on  $\mathcal{H}$  are particular bounded operators. Work by Böttcher et al. regarding Toeplitz operators such as in [2–4] proved particularly useful for our purposes.

## 2 Computability

Turing [20] introduced what is now called a Turing machine and computability theory is based on Turing machines. One approach to computability in analysis taken by Weihrauch and others is based on TTE (type-2 theory of effectivity) [21]. For real numbers all definitions of computability are equivalent. We now give the following version:

**Definition 1.** *A real number  $x$  is computable if there exists a computable sequence  $\{r_k\}$  of rationals which converges effectively to  $x$ . Such a sequence  $\{r_k\}$  converges effectively to a real number  $x$  if there exists a recursive function  $e : \mathbb{N} \rightarrow \mathbb{N}$  such that for all  $n \in \mathbb{N}$ :  $k \geq e(n)$  implies  $|r_k - x| \leq 2^{-n}$ .*

We will work with *effective* or *computably Cauchy* sequences  $\{x_i\}$ . This means that there is a computable function  $e : \mathbb{N} \rightarrow \mathbb{N}$  so that for all  $n \in \mathbb{N}$  we have

$$\forall i, j \geq e(n) \quad |x_i - x_j| < 2^{-n}.$$

In the applications we present later in the paper we will make use of a *computability structure* on a separable Hilbert space  $\mathcal{H}$  and in particular of computable linear operators on  $\mathcal{H}$ . The finite dimensional case was discussed in detail in two articles by Ziegler and Brattka [23, 24]. In a further article [6] the computability of the spectrum of self-adjoint operators is examined. Pour-El and Richards [17] present three important results concerned with computability on Banach spaces and introduce notions such as that of an *effectively separable Hilbert space*.

It may also be of interest to know when the inverse of a computable linear operator is computable. In [5] Brattka proves a number of results regarding the computability of inverses of linear bijective operators on Banach spaces. He uses the compact-open topology whereas our results are based on the operator topology.

### 3 Algorithm for Winding Numbers Using Quadtrees

In this section we are going to present a method that first checks whether a given square is on the graph of a given computable function  $f : S^1 \rightarrow \mathbb{C}$  (here  $S^1$  is the unit circle in the complex numbers). Then (if the square is not on the curve) the winding number of the curve given by  $f$  around the square is calculated. We use an algorithm for the winding number of a curve, and the reader can find other results on this in [8, 11, 12]. However, we will give some details of the algorithm we use now, as the algorithm for zeros will refer back to this.

We denote the *winding number* of a point  $\lambda \in \mathbb{C}$  with respect to the function  $f(z)$ , meromorphic inside  $\gamma$ , by  $\text{wind}(f, \lambda)$ . For  $\lambda = 0$  the winding number of  $f(z)$  along a closed curve  $\gamma$  is given by

$$\oint_{\gamma} \frac{f'(t)}{f(t)} dt = 2\pi i(N - P) = 2\pi i \cdot \text{wind}(f, 0), \quad (1)$$

where  $N$  is the number of zeros and  $P$  the number of poles inside  $\gamma$ .

We are going to use an object-oriented approach and introduce the two classes **Graph** and **Square**. We also assume that the curve is contained in the square  $[0, 1]^2$  for convenience - it is not hard to remove this condition by scaling.

**Graph** - Describes a continuous function  $f$  from the unit circle to the complex numbers.

**Methods:**

`findValue( $x, \epsilon$ )` - Returns a value of  $f(e^{2\pi ix})$  (for dyadic  $x \in [0, 1]$ ) within an error of  $\epsilon > 0$ .

`findDelta( $\epsilon$ )` - Given  $\epsilon > 0$  returns a value of  $\delta > 0$  for which, for all  $z, w \in S^1$  if  $|z - w| < \delta$  then  $|f(z) - f(w)| < \epsilon$ .

**Square** - Describes a square in the quadtree data structure on  $[0, 1]^2$ .

**Variables:**

`level` - gives the depth of the object in the quadtree data structure.

`center` - gives the coordinates of the center of the square.

`north, south, east, west` - gives the squares in those directions.

`onGraph` - value ‘no’ (in which case `distanceFromGraph` is set) or ‘don’t know’.

`distanceFromGraph` - a lower bound on the distance from the square to the graph.

`windingNumber` - integer - the winding number of the graph around the square.

**Methods:**

`findDistance( $graph, \epsilon$ )` - returns a lower bound on the distance from the square to the graph or ‘don’t know’. Is guaranteed to return a bound if the actual distance

is greater than  $\epsilon > 0$ .

`findWindingNumber(graph)` - uses `distanceFromGraph` (and is thus only called if `onGraph='no'`). Returns the windingNumber for the graph.

Now start at level zero (the square  $[0, 1]^2$ ), where the square at level  $n$  has side  $2^{-n}$ .

**Main routine:** enumerate over level  $n$  squares

```
call findDistance(graph, 2-n) and set onGraph †
If onGraph='no' set distanceFromGraph
If onGraph='no' call findWindingNumber(graph)
terminate (according to a given resolution) or move to next level
```

The important fact is that in the line † the method `findDistance(graph, 2-n)` can be written to return a value in finite time because of the limit on the resolution  $2^{-n}$  and the data on uniform continuity contained in the object graph. The price paid is that certain squares are recorded as ‘don’t know’ even if they are actually a distance  $> 0$  from the graph. But this will be remedied at a lower level with the subsquares of these squares. The procedure could be made faster if we let subsquares inherit variables from their parent, if the parent had `onGraph='no'`.

We will now give a description of the method `findWindingNumber`. For simplicity we assume that the center of the square is at the origin. The winding number is  $\frac{1}{2\pi}$  times the change in the argument (or angle) of the complex number  $f(e^{2\pi ix})$  as  $x$  moves from 0 to 1. The critical point here is that for complex numbers  $z, w$  with  $|z|, |w| \geq r$ , we have  $|\frac{z}{|z|} - \frac{w}{|w|}| \leq 2|z - w|/r$ . If a square of side  $2^{-n}$  is known not to intersect the graph, then we know that every point on the graph is distance  $\geq 2^{-n-1} = 2r$  from the center. Now set  $\epsilon = 2^{-n-6}$ , and we are guaranteed that  $|\text{findValue}(x, \epsilon)| \geq 2^{-n-2} = r$ . The previous inequality gives

$$\left| \frac{\text{findValue}(x, \epsilon)}{|\text{findValue}(x, \epsilon)|} - \frac{f(e^{2\pi ix})}{|f(e^{2\pi ix})|} \right| \leq \frac{2\epsilon}{r} = \frac{1}{8}.$$

From this data we can find an approximation to the argument of  $f(e^{2\pi ix})$ . However this angle is only known up to adding a multiple of  $2\pi$ , and in moving round the circle in a finite number of steps we need to be careful to avoid the angle changing by more than half this amount in one step. Using the inequality again, for all  $x, y \in [0, 1]$ ,

$$\left| \frac{\text{findValue}(x, \epsilon)}{|\text{findValue}(x, \epsilon)|} - \frac{\text{findValue}(y, \epsilon)}{|\text{findValue}(y, \epsilon)|} \right| \leq \frac{4\epsilon}{r} + \frac{2|f(e^{2\pi ix}) - f(e^{2\pi iy})|}{r}.$$

We call `findDelta`( $\epsilon$ ), set  $\delta$  to be the value returned, and if  $|x - y| < \delta$  we have

$$\left| \frac{\text{findValue}(x, \epsilon)}{|\text{findValue}(x, \epsilon)|} - \frac{\text{findValue}(y, \epsilon)}{|\text{findValue}(y, \epsilon)|} \right| \leq \frac{6\epsilon}{r} = \frac{3}{8}. \quad (2)$$

Now step from  $x = 0$  to  $x = 1$  in a finite number of steps of length less than  $\delta$ , and calculate the argument at each point. By (2) we can avoid skipping multiples

of  $2\pi$  and get the winding number up to an error. But the winding number must be an integer, so we take the closest integer. Figures 3.1 and 3.2 illustrate this procedure.

0	0	0	0	0	0			0	0
0	0	0	0	0	0			0	0
0	0	0	0	0	0			0	0
0	0	0	0	0	0			0	0
0	0	0	0	0	0			0	0
0	0	0	0	0	0			0	0
0	0								
0									
0									
0	0	0	0	0	0			0	0

Fig 3.1

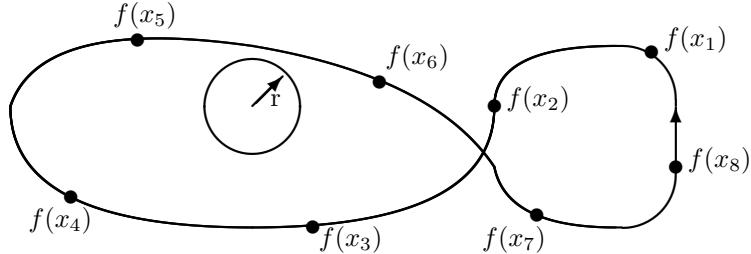


Fig 3.2

#### 4 Computing the Zeros of Holomorphic Functions

The total number of zeros (counted with multiplicity) of a complex analytic function  $f$  inside a given contour  $\gamma$  can be calculated by evaluating the winding number of  $f(z)$  as  $z$  moves counter-clockwise once round  $\gamma$ . We are going to make use of this and also of the important *principle of isolated zeros* for any analytic function [19]. This means that in any compact subset of the domain of the function, there can only be finitely many zeros. In the example we shall give later, the domain of the function is itself quite complicated, it is the set of points not on the graph where the winding number of the graph is zero. To describe such complicated domains, we will use a quadtree datatype. Then, given that a particular square is in the domain, we just calculate the winding number of  $f$  round its edges to find the number of zeros in the square. Unfortunately there is a complication - the edge may contain a zero of the function.

Consider a complex analytic function  $f$  defined on an open set (its domain) in the square  $[0, 1]^2$  in  $\mathbb{C}$ . We assume that  $f$  has no zeros on the edges of  $[0, 1]^2$ . The domain of  $f$  is defined by a quadtree data structure beginning with the level

$0$  square  $[0, 1]^2$ . A square at level  $n$  has a variable `inDomain`. If this takes the value ‘true’ then a slightly bigger square (for example, say the same center, but double the sides) lies in the domain of  $f$ . The other value of `inDomain` is ‘don’t know’. The union of all the squares for which `inDomain`=‘true’ is the domain of the function. Each level  $n$  square in the quadtree has nominal edges of side  $2^{-n}$ , but for our purposes these are not the actual edges that we choose. We choose edges of length  $5/2^{n+2}$  at a position shifted away from the center by an amount between  $0$  and  $2^{-n-3}$  from the nominal side. Fig. 4.1 shows a nominal square (with thick lines) and examples of the shifted edges. We only allow horizontal edges to move East and vertical edges to move South, which is sufficient.

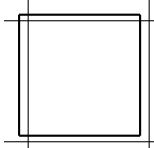


Fig. 4.1

Since the possible set occupied by any one shifted edge is a compact subset of the domain, by the principle of isolated zeros there can only be finitely many such shifted edges containing a zero of the function. We keep testing edges until we have one without a zero. Repeat for all four edges, then calculate the winding number. Of course, it is not quite that simple. First we can’t have adjacent squares disagreeing about what the common edge is. We take a class `Edge` and each edge is shifted once, so each object of class `Edge` contains a variable recording this shift. Squares query the objects of class `Edge`, referenced by the labels North, South, East and West.

Secondly, we can’t actually test if the function vanishes on a given shifted edge, as such a program may never terminate. What we can do is to run a test for an error  $\epsilon > 0$ , which is guaranteed to return a strictly positive lower bound for the function along the shifted edge if  $|f(z)| > \epsilon$  on the shifted edge, but may also return a ‘don’t know’ result. We test a list of trial shifted edges at successively reducing errors (say  $\epsilon = \frac{1}{m}$  on the  $m^{\text{th}}$  trial). The list includes infinitely many different edges, but also mentions each of these different edges infinitely many times. Then we must have a result in finite time giving a shifted edge and a strictly positive lower bound for  $|f(z)|$  on that edge.

The last complication is that the shifted edges are unlikely to form a simple grid pattern, but will have more complicated intersections at the corners. There are four possibilities, illustrated by the following diagrams in Fig. 4.2.

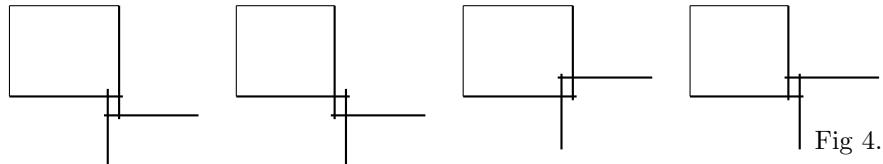


Fig 4.2

In each of the four possible cases we add any zeros occurring in the small rectangle to the perturbed, adjacent square to the North-West of it. It is impor-

tant to evaluate the integrals along the contours in each possible case detailed by Fig. 4.2 in the appropriate direction so as not to doubly count or omit zeros.

#### 4.1 Classes for Computing Zeros

**Function** - Describes an analytic function  $f$  defined on the domain.

**Methods:**

`findValue(z, ε)` - returns a value of  $f(z)$  for  $z$  in the domain within an error of  $ε > 0$ .

`findDelta(ε, edge)` - given  $ε > 0$  returns a value of  $δ > 0$  for which, for all  $z, w$  in the **Edge** edge if  $|z - w| < δ$  then  $|f(z) - f(w)| < ε$ .

**Square** - a quadtree datastructure square

**Variables:**

`level, center, northSquare, southSquare, eastSquare, westSquare, northEdge, southEdge, eastEdge, westEdge, inDomain, number_of_zeros`

**Methods:**

`find_in_domain` - determines if the enlarged square is in the domain

`find_number_of_zeros(function)` - determines the winding number by calling methods from the surrounding edges.

**HorizontalEdge** - a quadtree datastructure edge

**Variables:**

`edgePosition, level, northSquare, southSquare, inDomain, lowerBound`

**Methods:**

`find_edge_position(function)` - finds a shifted edgePosition where  $f(z)$  is guaranteed not to be zero, and a strictly positive lowerBound on  $|f(z)|$  there.

`change_in_argument(function, x, y)` - finds the change in argument of  $f(z)$  along the shifted edge from real part  $x$  to real part  $y$  to an accuracy of  $\frac{1}{16}$  of a revolution.

**VerticalEdge** - a quadtree datastructure edge - similar to HorizontalEdge

**Main routine:** enumerate over level  $n$  squares

call `find_in_domain` and set `inDomain` for square and surrounding edges

enumerate over level  $n$  edges with `inDomain=true`

call `find_edge_position` and set `edgePosition` and `lowerBound`

enumerate over level  $n$  squares

use `edgePosition` for surrounding edges to determine which of the corner cases applies

call `change_in_argument(function, x, y)` for surrounding edges and set `number_of_zeros`

We do not go into details of the calculation of the winding numbers, since this is similar to the previous discussion. The only difference is that we use

squares not circles, and do the calculation separately for each edge, only taking the closest integer on combining the data for the edges of the square.

There are a couple of points we should point out to the reader. The first is that the  $\epsilon$ - $\delta$  uniform continuity estimates for the function are carried out separately for the compact set of possible shifted edges in each `Edge` object. The reason is quite simple - the function is not analytic in the whole level 0 square, but on some open subset, the domain. Typically the function may become very badly behaved towards the boundary of the domain, so probably there is no global choice of  $\delta$  because the function is not globally uniformly continuous.

Secondly the output does not count the zeros in the original nominal squares, but in some slightly enlarged squares. However we have taken care not to double count any zeros, so a zero near an original nominal edge may end up counted in one or other of the neighbouring squares, but not both.

Thirdly we count zeros multiply, so  $f(z) = (z - a)^2$  counts as having two zeros at  $z = a$ . However the data given cannot distinguish between an actual double zero and two single zeros too close to separate at the given resolution.

## 5 Applications to Banded Toeplitz Operators

In a chosen basis  $(e_i)$  on a separable Hilbert space  $\mathcal{H}$ , following the notational conventions in [2], a Toeplitz operator  $T(a)$  is a linear operator given by

$$\begin{pmatrix} a_0 & a_{-1} & a_{-2} & \dots \\ a_1 & a_0 & a_{-1} & \dots \\ a_2 & a_1 & a_0 & \dots \\ \dots & \dots & \dots & \dots \end{pmatrix},$$

where  $a_i \in \mathbb{C}$  for  $i \in \mathbb{Z}$ . We can associate a function  $a(t) : S^1 \rightarrow \mathbb{C}$  with each Toeplitz operator consisting of the Laurent series with coefficients  $a_i$ . Here, we are interested in *banded* Toeplitz operators, which have a finite number of entries  $\{a_{-m}, a_{-m+1}, \dots, a_0, \dots, a_m\}$  only. In this case the associated function  $a(t)$  is simply a *Laurent polynomial*

$$a(t) = \sum_{k=-m}^m a_k t^k. \quad (3)$$

Criteria for the invertibility of Toeplitz operators are given in [2]. It is known [10] that the spectrum  $sp(T)$  of a Toeplitz operator  $T(a)$  is given by

$$sp(T) = a(S^1) \cup \{\lambda \in \mathbb{C} \setminus a(S^1) : \text{wind}(a, \lambda) \neq 0\}. \quad (4)$$

If  $T(a)$  is simply a shift operator then  $sp(T)$  consists of the closed unit disk. Böttcher uses MATLAB [22] to compute  $sp(T)$  for some examples in [4].

We now state how  $sp(T)$  can be computed by using the algorithm from Section 3. For that we assume that all entries of a Toeplitz operator are dyadic fractions, i.e. of the form  $\frac{a}{2^b}$ , where  $a$  and  $b$  are non-negative integers. In order to

compute  $sp(T)$  we simply need to use the methods for **Graph** for the Laurent polynomial (3) and then the main routine in Section 3. This will compute the complement of  $sp(T)$  up to a given accuracy depending on the level  $n$  of the squares in the quadtree.

From a private communication with Böttcher we learned that the spectrum of a Toeplitz operator  $T$  with a finite rank perturbation  $K$ , denoted by  $A = T + K$  and where  $K$  is contained in the first  $n$  rows and columns, is given by

$$sp(A) = sp(T) \cup \{\lambda \mid f(\lambda) := \det(\mathbb{I} + P_l(T - \lambda I)^{-1} K P_l) = 0\}. \quad (5)$$

Here  $P_l$  is the projection operator onto the first  $l$  basis vectors. Since  $f(\lambda)$  is analytic and not identically zero, this means that only countably many  $\lambda$  at most will be added to  $sp(T)$  to make up  $sp(A)$ .

Now, we briefly explain how  $sp(A)$  can be computed. The continuous part of  $sp(A)$  is simply  $sp(T)$  and can be computed as just stated. For the second, discrete part of  $sp(A)$  we retrieve all squares with centre  $\lambda$  for which  $windNumber = 0$ . Then, we check whether these  $\lambda$  are in the domain of  $f(\lambda)$  and run the main routine from Section 4 to determine the zeros of  $f(\lambda)$ . If a square with centre  $\lambda$  does not contain any zeros then it belongs to  $sp(A)^C$ . We subdivide all squares containing zeros and so on. This allows us to approximate  $sp(A)^C$  up to an accuracy depending on the level  $n$  of the quadtree structure.

Further results on computability of perturbed Toeplitz operators and a detailed account of the functional analysis behind the theory of Toeplitz operators are being written up in [1].

## Acknowledgements

The authors are indebted to Professor A Böttcher for numerous valuable discussions on Toeplitz operators and for providing very helpful advice on the literature. We also would like to thank Professor V Brattka for discussions relating to computable functional analysis. Both authors would like to thank EPSRC for supporting this work by the grant EP/C525361/1 .

## References

1. Beggs, E.J., Gerber, A.: Computability of banded Toeplitz Operators with Finite Rank Perturbations. In preparation
2. Böttcher, A., Grudsky, S.M.: Spectral properties of Toeplitz operators. SIAM (2005)
3. Böttcher, A., Embree, M., Sokolov, V.I.: Infinite Toeplitz and Laurent matrices with localised impurities. Lin. Alg. Appl. **343-344** (2002) 101-118
4. Böttcher, A., Embree, M., Sokolov, V.I.: On large Toeplitz band matrices with an uncertain block. Lin. Alg. Appl. **366** (2003) 87-97
5. Brattka, V.: The inversion problem for computable linear operators. STACS 2003. LNCS **2607** Springer, Berlin (2003), 391-402
6. Brattka, V., Dillhage, R.: Computability of the spectrum of self-adjoint operators. J. Univ. Comput. Sci. **11** (2005) 1884-1900

7. Debnath, L., Mikusinski, P.: Introduction to Hilbert spaces with applications. 2<sup>nd</sup> edition, Academic Press, (1999)
8. Dellnitz, M., Schütze, O., Zheng, Q.: Locating all the zeros of an analytic function in one complex variable. *J. Comput. Appl. Math.* **138** (2002) 325-333
9. Gillan, C.J., Schuchinsky, A., Spence, I.: Computing zeros of analytic functions in the complex plane without using derivatives. *Comp. Phys. Comm.* **175** (2006) 304-313
10. Gohberg, I.C.: On an application of the theory of normed rings to singular integral equations. *Uspekhi Matem. Nauk SSSR* **11****2** (1952) 149-156
11. Kravanja, P., Van Barel, M.: A derivative-free algorithm for computing zeros of analytic functions. *Computing* **63** (1999) 69-91
12. Kravanja, P., Van Barel, M.: Computing the zeros of analytic functions. *Lecture Notes in Mathematics* **1727** Springer, Berlin (2000)
13. Kravanja, P., Van Barel, M., Ragos, O., Vrahatis, M.N., Zafiroopoulos, F.A.: ZEAL: A mathematical software package for computing zeros of analytic functions. *Comp. Phys. Comm.* **124** (2000) 212-232
14. Kravanja, P., Ragos, O., Vrahatis, M.N., Zafiroopoulos, F.A.: ZEBEC: A mathematical software package for computing zeros of Bessel functions of real order and complex argument. *Comp. Phys. Comm.* **113** (1998) 220-238
15. Matheson, A., McNicholl, T.H.: Computable analysis and Blaschke products. *Proc. AMS* **136** (2008) 321-332
16. Piessens, R., de Doncker-Kapenga, E., Überhuber, C.W., Kahaner, D.K.: QUADPACK: A Subroutine package for automatic integration. Springer Series in Computational Mathematics Vol. **1** Springer, Berlin (1983)
17. Pour-El, M.B., Richards, J.I.: Computability in analysis and physics. Springer, (1989)
18. Rudin, W.: Functional analysis. 2<sup>nd</sup> edition, McGraw-Hill, (1991)
19. Rudin, W.: Principles of mathematical analysis. 3<sup>rd</sup> edition, McGraw-Hill, (1976)
20. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. *Proc. Lond. Math. Soc. Series 2* **42** (1936/37) 230-265
21. Weihrauch, K.: Computable analysis. Springer, Berlin (2000)
22. Wright, T.G.: MATLAB Pseudospectra GUI, 2000-2001. Available online at <http://www.comlab.ox.ac.uk/pseudospectra/psgui>
23. Ziegler, M., Brattka, V.: Computability in linear algebra. *Theor. Comput. Sci.* **326** (2004) 187-211
24. Ziegler, M., Brattka, V.: A computable spectral theorem. CCA 2000, Swansea, LNCS **2064** (2001) 378-388

# Triviality and Minimality in the Degrees of Monotone Complexity

William C. Calhoun

Department of Mathematics, Computer Science and Statistics  
Bloomsburg University, Bloomsburg, PA 17815, USA  
[wcalhoun@bloomu.edu](mailto:wcalhoun@bloomu.edu)

**Abstract.** This work extends the author's article *Degrees of Monotone Complexity* (2006). Monotone complexity,  $K_m$ , is a variant of Kolmogorov complexity that was introduced independently by Levin and Schnorr. Monotone complexity was used in the previous article to define the relative complexity (or randomness) of reals. Equivalence classes of reals under monotone complexity are the  $K_m$ -degrees, similar to the  $K$ -degrees defined via prefix-free complexity. In this paper, the  $K_m$ -trivial reals are defined and shown to be equivalent to the  $(K_m, K)$ -trivial reals. A nondecreasing, function  $f: \omega \rightarrow \omega$  is defined to be *computably infinitesimal* if it is dominated by every computable, nondecreasing, unbounded function. It is shown that if a real is  $K_m$ -trivial then its monotone complexity is computably infinitesimal. If a  $K_m$ -minimal real exists, it is  $K_m$ -trivial. The operation  $\otimes$ , defined previously, horizontally stretches the complexity graph of a real  $\alpha$  by a strictly increasing computable function  $f$ . Here,  $\alpha$  is defined to be *invariant under computable stretching* if  $\alpha \otimes f \equiv_{K_m} \alpha$  for any such  $f$ . It is shown that any  $K_m$ -minimal real must be invariant under computable stretching.

**Key words:** Kolmogorov complexity; monotone complexity; randomness; trivial reals

## 1 Introduction

Kolmogorov complexity was introduced by Solomonoff [17], Kolmogorov [7] and Chaitin [2] [3] in the 1960's and has been studied by many mathematicians and computer scientists since then. The present work is a continuation of the mathematical investigation of Kolmogorov complexity, in this case applied to the relative complexity (or relative randomness) of reals (binary sequences). We will use monotone complexity, a variant of Kolmogorov complexity defined by Levin [8]. Schnorr [13] independently defined a similar notion called process complexity that he later made equivalent to monotone complexity by slightly adjusting the definition [14]. In a previous article [1], the author studied the relative complexity of reals derived from monotone complexity for strings. Monotone complexity is arguably more appropriate for defining the complexity of a real than the more commonly used prefix-free complexity,  $K(\sigma)$ . Monotone programs can describe both finite strings and computable reals, and the bottom monotone degree is

the set of computable reals. In contrast, a prefix-free program can only describe a *finite* string and the bottom prefix-free degree is the set of  $K$ -trivial reals. See the book by Li and Vitányi [9] for additional background on monotone and prefix-free complexity.

Here we will focus on reals with very little complexity:  $K_m$ -trivial and  $K_m$ -minimal reals. We will define them formally in the following sections. Before delving into the mathematical details, it might be appropriate, in the spirit of the *Computability in Europe* conferences, to comment briefly on a potential application of Kolmogorov complexity. The close connection between Shannon's notion of entropy in information theory [15] and Kolmogorov complexity is well known. One example of a practical application of entropy is in computer data recovery and computer forensics [16]. Computing the entropy of recovered data can help determine the source of the data when directory information about the source file has been lost. High entropy indicates (seemingly) random data that is likely to come from a compressed or encrypted file. Low entropy indicates the data is redundant, possibly coming from a text or bitmap graphics file. Since the entropy is based on the frequencies of bytes in a file, it does not take into account the order of the bytes. Some researchers in computer forensics have used randomness measures based on Kolmogorov complexity to take the byte order into account. [19] In the setting of computer forensics, information may be sparsely scattered in data via steganography, a method of hiding messages in other data. One might think of the  $K_m$ -trivial reals informally as data in which information is sparsely scattered. The complexity of the initial segments grows so slowly that the real might be said to be "almost" computable. Nevertheless,  $K_m$ -trivial reals are noncomputable reals containing infinitely much information.

Of course our primary reason for studying  $K_m$ -trivial and  $K_m$ -minimal reals is to understand them as mathematical objects, rather than for any direct applications to computer forensics. The  $K_m$ -trivial reals are defined similarly to the intensively studied  $K$ -trivial reals. The connections and contrasts between  $K_m$ -trivial and  $K$ -trivial reals will be explored in the next section. The main question about  $K_m$ -minimal reals is whether they exist. The existence of minimal elements in a degree structure is one way the structure can fail to be dense. Density is considered an important property in the study of degree structures.

We close the introduction with some basic definitions that will be used in the analysis of trivial and minimal reals. A monotone machine  $M$  is a computably enumerable set of pairs  $\langle p, \sigma \rangle$  where  $p, \sigma \in 2^{<\omega}$  and for every  $\langle p, \sigma \rangle, \langle q, \tau \rangle \in M$ ,  $p \subseteq q$  implies  $\sigma \subseteq \tau$  or  $\tau \subseteq \sigma$ . (It is important to note that,  $p$  and  $q$  may be comparable or even equal.) We define the monotone complexity of a string  $\sigma$  with respect to  $M$  to be  $K_m^M(\sigma) = \min\{|p| : \langle p, \tau \rangle \in M \text{ for some } \tau \supseteq \sigma\}$ . One can show there is an optimal monotone machine  $U$ : for any other monotone machine  $M$ , there is a constant  $c$  such that  $K_m^U(\sigma) \leq K_m^M(\sigma) + c$  for all  $\sigma$ . For any string  $\sigma$  define  $K_m(\sigma) = K_m^U(\sigma)$ . For any  $\alpha \in 2^{<\omega}$  we define an *optimal monotone description* of  $\alpha$  to be a program  $p$  of minimal length such that  $\langle p, \beta \rangle \in U$  for some  $\beta \supseteq \alpha$ . In some constructions we approximate  $K_m$  by the computable functions  $K_m^s(\sigma) = \min\{|p| : \langle p, \tau \rangle \in U_s \text{ and } \sigma \subseteq \tau\}$ .

A useful method for constructing prefix-free machines is given by the Kraft-Chaitin Theorem.

**Theorem 1.** (*Kraft-Chaitin*) *From a computable sequence of pairs  $(\langle n_i, \sigma_i \rangle)_{i \in \omega}$  (known as axioms) such that  $\sum_{i \in \omega} 2^{-n_i} \leq 1$ , we can effectively obtain a prefix-free machine  $M$  such that for each  $i$  there is a  $\tau_i$  of length  $n_i$  with  $M(\tau_i) \downarrow = \sigma_i$ , and  $M(\mu) \uparrow$  unless  $\mu = \tau_i$ , for some  $i$ .*

When applying the Kraft-Chaitin theorem we shall refer to  $\sum_{\sigma_i = \mu} 2^{-n_i}$  as the *measure assigned to  $\mu$* . We will refer to  $\sum_{i \in \omega} 2^{-n_i}$  as the *total measure assigned*. Note that the total measure assigned is the sum over all strings  $\mu$  of the measure assigned to  $\mu$ . Using this terminology, the hypothesis of the Kraft-Chaitin theorem may be restated as follows: the total measure assigned is less than or equal to one.

As is done with prefix-free complexity, monotone complexity for binary strings may be used to define the complexity of reals via the complexity functions on initial segments. We will consider two complexity functions to be equivalent if their difference is bounded. We use the notation  $f \preceq g$  or  $g \succeq f$  to mean there is a constant  $c \in \omega$  such that  $f(n) \leq g(n) + c$  for all  $n \in \omega$ . We use the notation  $f \asymp g$  to mean that  $f \preceq g$  and  $g \preceq f$ . We write  $\alpha \leq_{K_m} \beta$  or  $\beta \geq_{K_m} \alpha$  if  $K_m(\alpha \upharpoonright n) \preceq K_m(\beta \upharpoonright n)$ . We write  $\alpha \equiv_{K_m} \beta$  if  $\alpha \leq_{K_m} \beta$  and  $\beta \leq_{K_m} \alpha$ . The equivalence classes of  $\equiv_{K_m}$  are the degrees of monotone complexity. See the book by Downey and Hirschfeldt [4] for additional background on relative randomness.

## 2 $K_m$ -Trivial Reals

Solovay [18] was the first to construct a noncomputable  $K$ -trivial real. (See also [5], [10] and [11].) The following definition was used in [1] to provide a notion of triviality for the monotone degrees.

**Definition 2.** *A real  $\alpha$  is  $(K_m, K)$ -trivial if  $K_m(\alpha \upharpoonright n) \preceq K(n)$ .*

Here we will more simply define triviality for the monotone degrees using only monotone complexity. First, we define the complexity of an integer  $n$  to be the complexity of a particular string of length  $n$ . For prefix-free complexity, we may define  $K(n) = K(1^n)$ . For monotone complexity, we must use a family of uniformly “self-delimiting” strings so that the length of the string can be computed from the pattern of bits in the string. A simple choice for this family is  $\lambda_n = 0^{n-1} \sim 1$  for  $n > 0$ . (Let  $\lambda_0$  be the empty string.)

**Definition 3.** *The monotone complexity of a positive integer  $n$  is defined by  $K_m(n) = K_m(\lambda_n)$*

We can now define a real to be  $K_m$ -trivial if the monotone complexity of its initial segments is dominated by the monotone complexity of the lengths of the initial segments.

**Definition 4.** *A real  $\alpha$  is  $K_m$ -trivial if  $K_m(\alpha \upharpoonright n) \preceq K_m(n)$ .*

The definitions  $(K_m, K)$ -trivial and  $K_m$ -trivial are actually equivalent as we will show. We begin with a theorem that shows that the  $K_m$  and  $K$  complexities are equivalent when applied to any computable set of incomparable strings.

**Theorem 5.** *If  $\{\alpha_n\}_{n \in \omega}$  is a computable set of pairwise incomparable binary strings, then  $K_m(\alpha_n) \asymp K(\alpha_n)$ .*

*Proof.* Since  $K$  dominates  $K_m$ , it is obvious that  $K_m(\alpha_n) \preceq K(\alpha_n)$ . For the other direction, we will use the Kraft-Chaitin Theorem to define a prefix-free machine  $M$  such that  $K^M(\alpha_n) \preceq K_m(\alpha_n)$ . Since  $K(\alpha_n) \preceq K^M(\alpha_n)$ , it follows that  $K(\alpha_n) \preceq K_m(\alpha_n)$ . Let  $S_n = \{s : K_m^s(\alpha_n) < K_m^{s-1}(\alpha_n)\}$ . (For this definition we set  $K_m^s(\alpha_n) = \infty$  when  $K_m^s(\alpha_n) \uparrow$ .) Note that for each  $n$  there is a stage  $t_n$  at which  $K_m^{t_n}(\alpha_n) = K_m(\alpha_n)$ . Therefore,  $S_n$  is finite for each  $n$ . To define  $M$ , we enumerate an axiom  $\langle K_m^s(\alpha_n) + 1, \alpha_n \rangle$  for each  $n$  and  $s$  where  $s \in S_n$ . Note that in the axioms  $(\langle n_i, \sigma_i \rangle)_{i \in \omega}$  assigned for each  $n$ , the  $n_i$  are distinct, and the least such  $n_i$  is  $K_m(\alpha_n) + 1$ . Then, for each  $n$ , the measure assigned to  $\alpha_n$  is  $\sum_{s \in S_n} 2^{-(K_m^s(\alpha_n)+1)} < 2(2^{-(K_m(\alpha_n)+1)}) = 2^{-K_m(\alpha_n)}$ . So the total measure assigned is less than  $\sum_{n \in \omega} 2^{-K_m(\alpha_n)}$ . Since the  $\alpha_n$  are pairwise incomparable, it follows that the total measure assigned is less than one. By the Kraft-Chaitin theorem, there is a prefix-free machine  $M$  such that  $K^M(\alpha_n) = K_m(\alpha_n) + 1$  for all  $n$ . Noting that  $K^M(\alpha_n) \preceq K_m(\alpha_n)$ , the proof is complete.  $\square$

As a corollary, we may deduce that  $K(n)$  and  $K_m(n)$  are equivalent functions.

**Corollary 6.**  $K(n) \asymp K_m(n)$ .

*Proof.* First, it is easy to see that  $K(1^n) \asymp K(\lambda_n)$ . Then we use the previous theorem to conclude that  $K(\lambda_n) \asymp K_m(\lambda_n)$ .  $\square$

We can conclude that  $(K_m, K)$ -triviality is equivalent to  $K_m$ -triviality.

**Corollary 7.** *A real is  $(K_m, K)$ -trivial iff it is  $K_m$ -trivial.*

As noted in [1] (for  $(K_m, K)$ -triviality), it is obvious that every  $K$ -trivial real is  $K_m$ -trivial, and hence that there are noncomputable  $K_m$ -trivial reals. D. Hirschfeldt and A. Nies have shown that every  $K$ -trivial real is Turing incomplete. [5] [10] On the other hand, F. Stephan has shown that the real  $\alpha$  defined by  $\alpha(n) = 1 \iff \forall m > n (K(m) > K(n))$  is  $K_m$ -trivial and Turing complete. [12] (The argument uses the fact that a universal machine with special properties can be constructed. [6]) Thus  $\alpha$  is a  $K_m$ -trivial real that is not  $K$ -trivial.

Since  $K_m(n)$  and  $K(n)$  are equivalent functions, we may deduce the asymptotic behavior of  $K_m(n)$  from known results for  $K(n)$ .

**Corollary 8.**

1.  $\lim_{n \rightarrow \infty} K_m(n) = \infty$ .
2. There is no upper bound on  $K_m(n_1) - K_m(n_2)$  for  $n_1 < n_2$ .

*Proof.* The corresponding statements are true of  $K(n)$ .  $\square$

For any real  $\alpha$  it is clear that  $K(|\alpha|) \preceq K(\alpha)$ . Hence, if  $\alpha$  is  $K$ -trivial, we have  $K(\alpha \upharpoonright n) \asymp K(n)$ . However, the corresponding fact does not hold for monotone complexity as indicated by the following theorem.

**Theorem 9.** *For any real  $\alpha$ ,  $K_m(\alpha \upharpoonright n) \not\asymp K_m(n)$ .*

*Proof.* Suppose for a contradiction that  $K_m(\alpha \upharpoonright n) \asymp K_m(n)$  for some real  $\alpha$ . Then for some constant  $b$ ,  $K_m(\alpha \upharpoonright n) - b \leq K_m(n) \leq K_m(\alpha \upharpoonright n) + b$ . But by Corollary 8 we may choose  $n_1 < n_2$  such that  $K_m(n_1) - K_m(n_2) > 2b$ . Combining the inequalities, we obtain  $K_m(\alpha \upharpoonright n_1) + b \geq K_m(n_1) > K_m(n_2) + 2b \geq K_m(\alpha \upharpoonright n_2) + b$ . But then  $K_m(\alpha \upharpoonright n_1) > K_m(\alpha \upharpoonright n_2)$ , which is impossible since  $K_m(\alpha \upharpoonright n)$  is nondecreasing.  $\square$

It follows immediately that  $K_m$ -trivial reals must have monotone complexity *strictly* below the complexity of the length.

**Corollary 10.** *If  $\alpha$  is a  $K_m$ -trivial real, then  $K_m(\alpha \upharpoonright n) \prec K_m(n)$ .*

However, it should be understood that the previous result only means that  $K_m(n) - K_m(\alpha \upharpoonright n)$  is bounded below and unbounded above. The difference can oscillate so that  $K_m(n) - K_m(\alpha \upharpoonright n)$  is small (less than a fixed bound) infinitely many times. This raises the question of whether a noncomputable real can have monotone complexity unboundedly less than the complexity of the length. We now state this question more precisely.

**Definition 11.** *We write  $f(n) \prec\prec g(n)$  if  $\lim_{n \rightarrow \infty} g(n) - f(n) = \infty$ .*

**Question 12.** Is there a noncomputable real  $\alpha$  such that  $K_m(\alpha \upharpoonright n) \prec\prec K_m(n)$ ?

We now show that  $K_m$ -trivial reals must have extremely slow-growing complexities. To state this precisely, we give a definition for nondecreasing functions that are below any computable, nondecreasing, unbounded function.

**Definition 13.** *A nondecreasing, function  $f : \omega \rightarrow \omega$  is computably infinitesimal if for every computable, nondecreasing, unbounded function  $g$ ,  $f(n) \preceq g(n)$ .*

Note that a computably infinitesimal function must grow more slowly than such slow-growing computable functions as  $\log_k$  for any  $k$  or the inverse Ackermann function. We now show the monotone complexity of any  $K_m$ -trivial real is computably infinitesimal, although it is unbounded.

**Theorem 14.** *If  $\alpha$  is a  $K_m$ -trivial real, then  $K_m(\alpha \upharpoonright n)$  is computably infinitesimal.*

*Proof.* Let  $g$  be a computable, nondecreasing, unbounded function. We wish to show that  $K_m(\alpha \upharpoonright n) \preceq g(n)$ . We may assume that  $g(0) = 0$  and  $g(n+1) - g(n) \leq 1$  for all  $n$ , since otherwise we can replace  $g$  with a function  $h(n) \preceq g(n)$  that satisfies those properties. We define a computable sequence  $\{n_i\}_{i \in \omega}$  by  $n_i$  = the greatest  $n$  such that  $g(n) = i$ . The definition provides a description of  $n_i$  of size

depending only on the size of  $i$ . Hence  $K_m(n_i) \preceq i = g(n_i)$ . Since  $\alpha$  is  $K_m$ -trivial,  $K_m(\alpha \upharpoonright n_i) \preceq K_m(n_i) \preceq g(n_i)$ . Since  $K_m(\alpha \upharpoonright n)$  and  $g(n)$  are monotone functions and  $g(n_{i+1}) - g(n_i) = 1$ , it follows that  $K_m(\alpha \upharpoonright n) \preceq g(n)$ .

□

As mentioned above, there is a  $K_m$ -trivial real that is not  $K$ -trivial. However, as a corollary to the previous result, we can show that every  $K_m$ -trivial real is “almost”  $K$ -trivial, in the sense that the difference between the prefix-free complexity of a  $K_m$ -trivial real and  $K(n)$  is computably infinitesimal.

**Corollary 15.** *If  $\alpha$  is  $K_m$ -trivial then  $K(\alpha \upharpoonright n) - K(n)$  is computably infinitesimal.*

*Proof.* Let  $\alpha$  be a  $K_m$ -trivial real. Then  $K(\alpha \upharpoonright n) \preceq K(n) + 2K_m(\alpha \upharpoonright n)$ . Hence  $K(\alpha \upharpoonright n) - K(n) \preceq 2K_m(\alpha \upharpoonright n)$ . Since  $K_m(\alpha \upharpoonright n)$  is computably infinitesimal, so is  $2K_m(\alpha \upharpoonright n)$ . □

### 3 $K_m$ -Minimal Reals

We now turn our attention to  $K_m$ -minimal reals.

**Definition 16.** *A real  $\alpha$  is  $K_m$ -minimal if  $\alpha >_{K_m} 0$  and for every real  $\beta \leq_{K_m} \alpha$  either  $\beta \equiv_{K_m} 0$  or  $\beta \equiv_{K_m} \alpha$ .*

It is not known whether a  $K_m$ -minimal real exists. We will prove several properties that such a real would have to satisfy. The following theorem connecting  $K_m$ -triviality and  $K_m$ -minimality is from [1].

**Theorem 17.** *If a real is  $K_m$ -minimal then it is  $K_m$ -trivial.*

As an immediate corollary, a  $K_m$ -minimal real must satisfy all the properties of  $K_m$ -trivial reals from the previous section.

**Corollary 18.** *If a real  $\alpha$  is  $K_m$ -minimal then*

1.  $K_m(\alpha \upharpoonright n) \prec K_m(n)$ ,
2.  $K_m(\alpha \upharpoonright n)$  is computably infinitesimal
3. and  $K(\alpha \upharpoonright n) - K(n)$  is computably infinitesimal.

We will close with a discussion of an additional property of  $K_m$ -minimal reals. An operation  $\otimes$  was introduced in [1] to “horizontally stretch” the graph of a real  $\alpha$  by a strictly increasing function  $f$ .

**Definition 19.** *Given any real  $\alpha$  and a strictly increasing function  $f: \omega \rightarrow \omega$ , let  $\alpha \otimes f$  be the real defined by*

$$(\alpha \otimes f)(n) = \begin{cases} \alpha(f^{-1}(n)) & \text{if } n \in \text{range}(f) \\ 0 & \text{otherwise} \end{cases}$$

**Theorem 20.** *If  $\alpha$  is any real and  $f: \omega \rightarrow \omega$  is a strictly increasing computable function, then  $K_m(\alpha \otimes f \upharpoonright n) \asymp K_m(\alpha \upharpoonright f^{-1}[n])$  where  $f^{-1}[n] = \max\{k : f(k) \leq n\}$ .*

It is conceivable that  $K_m(\alpha \upharpoonright n)$  could grow so slowly that  $\alpha \otimes f$  has monotone complexity equivalent to that of  $\alpha$ , for any strictly increasing computable function  $f$ .

**Definition 21.** *A real  $\alpha$  is invariant under computable stretching if for all strictly increasing computable functions  $f$ ,  $\alpha \otimes f \equiv_{K_m} \alpha$ .*

The monotone complexity of a real that is invariant under computable stretching must grow extremely slowly, as indicated by the next theorem.

**Theorem 22.** *If a real  $\alpha$  is invariant under computable stretching then for any strictly increasing computable function  $f$ ,  $K_m(\alpha \upharpoonright f(n)) - K_m(\alpha \upharpoonright n)$  is bounded.*

*Proof.* Let  $\alpha$  be invariant under computable stretching and  $f$  be a strictly increasing computable function. Since  $\alpha \oplus f \equiv_{K_m} \alpha$ , we have  $K_m(\alpha \oplus f \upharpoonright f(n)) \asymp K_m(\alpha \upharpoonright f(n))$ . By Theorem 20  $K_m(\alpha \upharpoonright f(n)) \asymp K_m(\alpha \upharpoonright f^{-1}[f(n)]) = K_m(\alpha \upharpoonright n)$ . Therefore  $K_m(\alpha \upharpoonright f(n)) \asymp K_m(\alpha \upharpoonright n)$  and the result follows.  $\square$

If a real  $\alpha$  is invariant under computable stretching, then by Theorem 20 its monotone complexity  $g(n) = K_m(\alpha \upharpoonright n)$  satisfies  $g(f^{-1}[n]) \asymp g(n)$  for any strictly increasing computable function  $f$ . It is not hard to construct a function  $g$  that satisfies that property.

**Theorem 23.** *There is a function  $g: \omega \rightarrow \omega$  such that  $g(f^{-1}[n]) \asymp g(n)$  for any strictly increasing computable function  $f$ .*

*Proof.* Let  $\{f_i\}_{i \in \omega}$  be a list of a strictly increasing computable functions. Define a sequence  $\{a_n\}_{n \in \omega}$  by  $a_0 = 0$  and, for  $n > 0$ ,  $a_n = \max\{f_i(a_{n-1}) : i < n\} + 1$ . Let  $g(x) = \max\{n : a_n \leq x\}$ . Now we verify that  $g$  satisfies the desired property. For any strictly increasing computable function  $f$ ,  $f = f_i$  for some  $i$ . For any integer  $x \geq i$  there is a unique integer  $n \geq i$  such that  $a_n \leq x < a_{n+1}$ . By definition  $a_n > f(a_{n-1})$ . So  $a_{n-1} \leq f^{-1}[a_n] \leq f^{-1}[x] \leq x$ . Since  $g$  is increasing,  $g(a_{n-1}) \leq g(f^{-1}[x]) \leq g(x)$ . By definition of  $g$ , we have  $g(a_{n-1}) = n - 1$  and  $g(x) = n$ . Therefore,  $g(x) - 1 \leq g(f^{-1}[x]) \leq g(x)$ .  $\square$

The function  $g$  constructed in the previous theorem is not necessarily the monotone complexity of a real, leaving open the question of whether a real exists that is invariant under computable stretching. However, if a  $K_m$ -minimal real exists, it must be invariant under computable stretching.

**Theorem 24.** *Every  $K_m$ -minimal real is invariant under computable stretching.*

*Proof.* Let  $\alpha$  be a  $K_m$ -minimal real and let  $f$  be a strictly increasing computable function. We have  $K_m(\alpha \otimes f \upharpoonright n) \asymp K_m(\alpha \upharpoonright f^{-1}[n])$  by Theorem 20. From  $\alpha >_{K_m} 0$  we may conclude that  $\lim_{n \rightarrow \infty} K_m(\alpha \upharpoonright f^{-1}[n]) = \infty$ . Therefore  $0 <_{K_m} \alpha \otimes f \leq_{K_m} \alpha$ . Since  $\alpha$  is  $K_m$ -minimal,  $\alpha \otimes f \equiv_{K_m} \alpha$ .  $\square$

Perhaps the results of this section will be useful in settling the main open question about  $K_m$ -minimal reals.

*Question 25.* Is there a  $K_m$ -minimal real?

**Acknowledgments:** The author would like to thank Frank Stephan for correspondence regarding the existence of  $K_m$ -trivial reals that are not  $K$ -trivial. The author would also like to thank the anonymous referees for detailed helpful comments.

## References

1. Calhoun, W.C.: Degrees of Monotone Complexity, *J. Symbolic Logic* 71, 1327–1341 (2006).
2. Chaitin, G.J.: On the Length of Programs for Computing Finite Binary Sequences, *J. ACM* 13, 547–569 (1966).
3. Chaitin, G.J.: On the Length of Programs for Computing Finite Binary Sequences: Statistical Considerations, *J. ACM* 16, 145–159 (1969).
4. Downey, R.G., Hirschfeldt, D.R.: Algorithmic Randomness and Complexity, Springer-Verlag (to appear).
5. Downey, R.G., Hirschfeldt, D.R., Nies, A., Stephan, F.: Trivial Reals. In: Downey, R.G., Hui, Q.Y., Ping, T.S., Decheng, D., Yasugi, Y. (eds.) *Proceedings of the 7th and 8th Asian Logic Conferences*, pp. 103–131. World Scientific (2003).
6. Figueira, S., Stephan, F., Wu, G.: Randomness and Universal Machines, *J. Complexity* 22, 738–751 (2006).
7. Kolmogorov, A.N.: Three Approaches to the Quantitative Definition of Information, *Problems Inform. Transmission* 1, 1–7 (1965).
8. Levin, L.A.: On the Notion of a Random Sequence, *Soviet Math. Dokl.* 14 1413–1416 (1973).
9. Li, M., Vitányi, P.: *An Introduction to Kolmogorov Complexity and Its Applications* (2nd ed.). Springer-Verlag, New York (1997).
10. Nies, A., Lowness Properties and Randomness, *Advances in Math.* 197, 274–305 (2005).
11. Nies, A., Stephan, F., Terwijn, S.: Randomness, Relativization and Turing Degrees, *J. Symbolic Logic* 70, 515–535 (2005).
12. Stephan, F., personal correspondence, (2006, 2008).
13. Schnorr, C.P.: Process Complexity and Effective Random Tests, *J. Comput. System Sci.* 7, 376–388 (1973).
14. Schnorr, C.P.: A Survey of the Theory of Random Sequences. In: Butts, R.E., Hintikka, J. (eds.) *Basic Problems in Methodology and Linguistics*, pp. 193–210. D. Reidel (1977).
15. Shannon, C.E.: The Mathematical Theory of Communication, *Bell System Tech. J.* 27, 379–423, 623–656 (1948).
16. Shannon, M.M.: Forensic Relative Strength Scoring: ASCII and Entropy Scoring, *Int. J. Digital Evidence* 2, 1–19 (2004).
17. Solomonoff, R.J.: A Formal Theory of Inductive Inference (Part 1 and Part 2), *Inform. Contr.* 7, 1–22, 224–254 (1964).

18. Solovay, R.M.: Draft of a Paper (or Series of Papers) on Chaitin's Work (Unpublished notes), IBM Thomas J. Watson Research Center, Yorktown Heights, NY (1975).
19. Veenman, C.J.: Statistical Disk Cluster Classification for File Carving. In: IEEE Third International Symposium on Information Assurance and Security, 393–398 (2007).

# Extraction of Efficient Programs from Proofs: The Case of Structural Induction over Natural Numbers

Luca Chiarabini

LMU Mathematisches Institut, Theresienstrasse 39, D-80333 München, Germany  
[chiarabi@mathematik.uni-muenchen.de](mailto:chiarabi@mathematik.uni-muenchen.de)

**Abstract.** Transforming a recursive procedure into a tail recursive one brings many computational benefits; in particular in each recursive call there is no context information to store. In this paper we consider the particularly simple induction schema over natural numbers, and we propose two methods to automatically turn it into another proof with tail recursive content: one *continuation* and one *accumulator* based.

**Key words:** Functional Programming, Program extraction from constructive proofs, Program development by proof transformation, CPS-Transformation, Defunctionalization.

## 1 Introduction

Let  $M$  be a proof by induction over  $n$  (natural number) of the property  $\forall n. \varphi(n)$ , and let, by the *Proofs-as-Program* paradigm,  $\llbracket M \rrbracket$  be the (recursive) content of  $M$ . In this paper we will try to answer the following question: *If  $\llbracket M \rrbracket$  is not tail recursive, is it possible (and if yes, how) to turn automatically  $M$  into another proof, say  $N$ , with tail recursive content?* During an informal chat in the first MATHLOGAPS meeting in Fischbachau, Germany, Andrej Bauer suggested to me an interesting idea[1] to mimic the behavior of a let-expression by a pure  $\lambda$ -calculus function. Here we present and develop his idea in a formal setting, which will provide a solution for the problem proposed above. Just to give an intuition of what we will see, let

```
define FACT = fun n -> (if n=0 then 1
                           else (n* (FACT (n-1))))
```

be the factorial function, written in an Ocaml-like syntax.  $\text{FACT}$  is not *tail* recursive because in each step of the computation the compiler has to store the context  $(n*[])$ , evaluate  $\text{FACT} (n-1) \mapsto v$ , and returns  $(n*v)$ . We turn  $\text{FACT}$  into a simpler function where it is not necessary to store any context information:

```
define FACT' = fun n -> (define FACT''= fun n,m,y ->
                           (if n=0 then y else FACT''(n-1)(m+1)((m+1)*y))) n 0 1
```

Now suppose FACT to be the computational content of the proof by induction  $M$ , with end formula  $\forall n. \varphi(n)$ , that states that for each natural  $n$  there exists its factorial. Given a natural  $n$ ,  $(\text{FACT}', n)$  is a function that takes the natural  $m$ , the witness  $y$  for  $\varphi(m)$  and returns a witness for  $\varphi(n+m)$ . One of the two proofs transformations proposed in the present paper (the *accumulator* based one) will regards how to turn  $M$  into a new proof  $M'$  with end formula  $(\forall n, m. \varphi(m) \rightarrow \varphi(n+m))$  and computational content equal to  $\text{Fact}'$ . It is clear that given  $n$ ,  $(\text{FACT}', n 0 1)$  is the witness for  $\varphi(n)$  as desired.

The paper is organized as follows: section two is a short introduction to minimal logic and program extraction. In section three we address two transformations over proofs in order extract *continuation* and *accumulator* based programs. The last section regards future works. All the proofs presented in the paper were developed with the MINLOG proof assistant[2].

## 2 Modified Realizability for First Order Minimal Logic

### 2.1 Gödel's T

Types are built from base types **N** (Naturals), **L**( $\rho$ ) (lists with elements of type  $\rho$ ) and **B** (booleans) by function ( $\rightarrow$ ) and pair ( $\times$ ) formation. The *Terms* of Gödel's T[3] are simply typed  $\lambda$ -calculus terms with pairs, projections ( $\pi_i$ ) and constants (constructors and recursive operators for the basic types)

$$\begin{aligned} \text{Types } \rho, \sigma ::= & \mathbf{N} \mid \mathbf{B} \mid \mathbf{L}(\rho) \mid \rho \rightarrow \sigma \mid \rho \times \sigma \\ \text{Const } c ::= & 0^{\mathbf{N}} \mid \text{Succ}^{\mathbf{N} \rightarrow \mathbf{N}} \mid \text{tt}^{\mathbf{B}} \mid \text{ff}^{\mathbf{B}} \mid []^{\mathbf{L}(\rho)} \mid ::^{\rho \rightarrow \mathbf{L}(\rho) \rightarrow \mathbf{L}(\rho)} \mid \mathcal{R}_{\mathbf{N}}^{\sigma} \mid \mathcal{R}_{\mathbf{L}(\rho)}^{\sigma} \mid \mathcal{R}_{\mathbf{B}}^{\sigma} \\ \text{Terms } r, s, t ::= & c \mid x^{\rho} \mid (\lambda x^{\rho} r^{\sigma})^{\rho \rightarrow \sigma} \mid (r^{\rho \rightarrow \sigma} s^{\rho})^{\sigma} \mid (\pi_0 t^{\rho \times \sigma})^{\rho} \mid (\pi_1 t^{\rho \times \sigma})^{\sigma} \mid (r^{\rho}, s^{\sigma})^{\rho \times \sigma} \end{aligned}$$

The types and conversion rules for the recursive operators, applications and projections are:

$$\begin{array}{ll} \mathcal{R}_{\mathbf{N}}^{\sigma} : \sigma \rightarrow (\mathbf{N} \rightarrow \sigma \rightarrow \sigma) \rightarrow \mathbf{N} \rightarrow \sigma & \mathcal{R}_{\mathbf{L}(\rho)}^{\sigma} : \sigma \rightarrow (\mathbf{L}(\rho) \rightarrow \sigma \rightarrow \sigma) \rightarrow \mathbf{L}(\rho) \rightarrow \sigma \\ (\mathcal{R}_{\mathbf{N}}^{\sigma} b f) 0 \mapsto b & (\mathcal{R}_{\mathbf{L}(\rho)}^{\sigma} b f) [] \mapsto b \\ (\mathcal{R}_{\mathbf{N}}^{\sigma} b f) (n + 1) \mapsto f n ((\mathcal{R}_{\mathbf{N}}^{\sigma} b f) n) & (\mathcal{R}_{\mathbf{L}(\rho)}^{\sigma} b f) (a :: l) \mapsto f l ((\mathcal{R}_{\mathbf{L}(\rho)}^{\sigma} b f) l) \\ \\ \mathcal{R}_{\mathbf{B}}^{\sigma} : \sigma \rightarrow \sigma \rightarrow \mathbf{B} \rightarrow \sigma & \pi_0(r, s) \xrightarrow{\beta} r \\ (\mathcal{R}_{\mathbf{B}}^{\sigma} r s) \text{tt} \mapsto r & \pi_1(r, s) \xrightarrow{\beta} s \\ (\mathcal{R}_{\mathbf{B}}^{\sigma} r s) \text{ff} \mapsto s & (\lambda x. r) s \xrightarrow{\beta} r[x := s] \end{array}$$

The term  $(\mathcal{R}_{\mathbf{B}}^{\sigma} r s)t$  is normally printed as  $(\text{if } r s \text{ then } t)$ . By  $\xrightarrow{\mathcal{R}_{\eta\beta}}$  we indicate the union of  $\xrightarrow{\beta}$ ,  $\mapsto$ , and  $\xrightarrow{\eta}$  the  $\eta$ -reduction defined as  $\lambda x. r s \xrightarrow{\eta} r$  if  $x \notin \text{FV}(r)$ . Finally we define the *extensional equality* relation  $=_{\mathcal{R}_{\eta\beta}}$ , as the least equivalence relation that contain  $\xrightarrow{\mathcal{R}_{\eta\beta}}$ . The *extensional equality* relation capture the idea that two functions should be considered equal if they yield equal results whenever applied to equal arguments.

## 2.2 Heyting Arithmetic

We define Heyting Arithmetic  $\text{HA}^\omega$  for our language based on Gödel's T, which is finitely typed. For falsity we can take the atomic formula  $F := \text{atom}(\text{ff})$  – called *arithmetical falsity* – built from the boolean constant  $\text{ff}$  and the predicate operator on booleans  $\text{atom}$ . Below we will also need the (logical) falsity  $\perp$ , which we can view as just a particular propositional symbol. We define *negation*  $\neg\varphi$  by  $\varphi \rightarrow F$  or  $\varphi \rightarrow \perp$  (depending on the context).

*Formulas:* Atomic formulas ( $Pt\rho$ ) ( $P$  a predicate symbol,  $t$ ,  $\rho$  lists of terms and types),  $\varphi \rightarrow \psi$ ,  $\forall x^\rho \varphi$ ,  $\exists x^\rho \varphi$ ,  $\varphi \wedge \psi$ .

*Derivations:* In spite of the Curry-Howard correspondence it is convenient to write derivations as terms: we define  $\lambda$ -terms  $M^\varphi$  for natural deduction proofs of formulas  $\varphi$  together with the set  $\text{OA}(M)$  of free assumptions in  $M$ :

$$\begin{aligned}
 (\text{ass}) \quad & u^\varphi, \text{OA}(u)=\{u\} \\
 (\wedge^+) \quad & (\langle M^\varphi, N^\psi \rangle^{\varphi \wedge \psi}), \text{OA}(\langle M, N \rangle)=\text{OA}(M) \cup \text{OA}(N) \\
 (\wedge_0^-) \quad & (M^{\varphi \wedge \psi} 0)^\varphi, \text{OA}(M 0)=\text{OA}(M) \\
 (\wedge_1^-) \quad & (N^{\varphi \wedge \psi} 1)^\psi, \text{OA}(N 1)=\text{OA}(N) \\
 (\rightarrow^+) \quad & (\lambda u^\varphi. M^\psi)^{\varphi \rightarrow \psi}, \text{OA}(\lambda u. M)=\text{OA}(M) \setminus \{u\} \\
 (\rightarrow^-) \quad & (M^{\varphi \rightarrow \psi} N^\varphi)^\psi, \text{OA}(M N)=\text{OA}(M) \cup \text{OA}(N) \\
 (\forall^+) \quad & (\lambda x^\rho. M^\varphi)^{\forall x^\rho \varphi}, \text{OA}(\lambda x. M)=\text{OA}(M) \\
 & \quad \text{provided } x^\rho \notin \text{FV}(\varphi), \text{ for any } u^\varphi \in \text{OA}(M) \\
 (\forall^-) \quad & (M^{\forall x^\rho \varphi} t^\rho)^\varphi, \text{OA}(M t)=\text{OA}(M)
 \end{aligned}$$

Usually we will omit type and formula indices in derivations if they are uniquely determined by the context or if they are not relevant and sometimes we will write  $M : \varphi$  instead of  $M^\varphi$ . In the above definition of derivations as terms we have left out the standard connectives  $\exists$  and  $\vee$  because for simplicity we want our derivation terms to be pure lambda terms formed just by lambda abstraction, application, pairing and projections. In spite of this omission we can use  $\exists$  and  $\vee$  in our logic, if we allow appropriate axioms as constant derivation terms, e.g. for  $\exists$ :

$$\begin{aligned}
 \exists_{x^\rho, \varphi}^+ & : \forall x^\rho (\varphi \rightarrow \exists x^\rho \varphi) \\
 \exists_{x^\rho, \varphi, \psi}^- & : \exists x^\rho \varphi \rightarrow (\forall x^\rho \varphi \rightarrow \psi) \rightarrow \psi
 \end{aligned}$$

with the usual proviso  $x \notin \text{FV}(\psi)$ . For  $\vee$  we could introduce similar axioms, but we do not do here, since we can define  $\vee$  from  $\exists$  via:

$$\varphi \vee \sigma \triangleq \exists p^{\mathbf{B}}. (p \rightarrow \varphi) \wedge ((p \rightarrow \perp) \rightarrow \psi)$$

Finally the induction axioms associated to the types  $\mathbf{N}, \mathbf{B}$  and  $\mathbf{L}(\rho)$  are:

$$\begin{aligned}
 \text{Ind}_{n, \varphi(n)} & : \varphi(0) \rightarrow (\forall n. \varphi(n) \rightarrow \varphi(n + 1)) \rightarrow \forall n^{\mathbf{N}}. \varphi(n) \\
 \text{Ind}_{t, \varphi(t)} & : \varphi(\mathbf{tt}) \rightarrow \varphi(\mathbf{ff}) \rightarrow \forall t^{\mathbf{B}}. \varphi(t) \\
 \text{Ind}_{l, \varphi(l)} & : \varphi([]) \rightarrow (\forall a, l. \varphi(l) \rightarrow \varphi(a :: l)) \rightarrow \forall l^{\mathbf{L}(\rho)}. \varphi(l)
 \end{aligned}$$

### 2.3 Modified Realizability

Clearly proper existence proofs have computational content. A well-known and natural way to define this concept is the notion of realizability, which can be seen as an incarnation of the Brouwer-Heyting-Kolmogorov interpretation of proofs.

**Type of a Formula** We define  $\tau(\varphi)$  as the type of the term (or “program”) to be extracted from a proof of  $\varphi$ . More precisely, we assign to every formula  $\varphi$  an object  $\tau(\varphi)$  (a type or the “nulltype” symbol  $\varepsilon$ ). In case  $\tau(\varphi) = \varepsilon$  proofs of  $\varphi$  have no computational content; such formulas  $\varphi$  are called *Harrop formulas*. The definition can be conveniently written if we extend the use of  $\rho \rightarrow \sigma$  and  $\rho \times \sigma$ , to the nulltype symbol:  $(\rho \rightarrow \varepsilon) := \varepsilon$ ,  $(\varepsilon \rightarrow \sigma) := \sigma$ ,  $(\varepsilon \rightarrow \varepsilon) := \varepsilon$ ,  $(\rho \times \varepsilon) := \rho$ ,  $(\varepsilon \times \sigma) := \sigma$ ,  $(\varepsilon \times \varepsilon) := \varepsilon$ :

$$\begin{aligned}\tau(Pt^\rho) &:= \varepsilon \\ \tau(\varphi \rightarrow \psi) &:= (\tau(\varphi) \rightarrow \tau(\psi)) \\ \tau(\varphi \wedge \psi) &:= (\tau(\varphi) \times \tau(\psi)) \\ \tau(\exists x^\rho. \varphi(x^\rho)) &:= \rho \times \tau(\varphi) \\ \tau(\forall x^\rho. \varphi) &:= (\rho \rightarrow \tau(\varphi))\end{aligned}$$

**Realize a formula** For a convenient definition we extend the use of term application and projection to the nullterm symbol:  $\varepsilon t := \varepsilon$ ,  $t\varepsilon := t$ ,  $\varepsilon\varepsilon := \varepsilon$ ,  $\pi_0\varepsilon := \varepsilon$ ,  $\pi_1\varepsilon := \varepsilon$ . We define the formula  $t \mathbf{mr} \varphi$ , to be read  $t$  realizes  $\varphi$ :

$$\begin{aligned}t \mathbf{mr} Pt &:= Pt \\ t \mathbf{mr} \exists x. \varphi(x) &:= \pi_0 t \mathbf{mr} \varphi(\pi_1 t) \\ t \mathbf{mr} (\varphi \rightarrow \psi) &:= \forall x. (x \mathbf{mr} \varphi \rightarrow tx \mathbf{mr} \psi) \\ t \mathbf{mr} (\varphi \wedge \psi) &:= (\pi_0 t \mathbf{mr} \varphi \wedge \pi_1 t \mathbf{mr} \psi) \\ t \mathbf{mr} (\forall x. \varphi) &:= \forall x. tx \mathbf{mr} \varphi\end{aligned}$$

Formulas which do not contain  $\exists$  play a special role in this context; we call them *negative*. Their crucial property is  $(\varepsilon \mathbf{mr} \varphi) = \varphi$ . Clearly every formula of the form  $(t \mathbf{mr} \varphi)$  is negative.

**Program Extraction** We now define the *extracted term*  $\llbracket M \rrbracket$  of a derivation  $M[4]$ . For derivations  $M^\varphi$  where  $\tau(\varphi) = \varepsilon$  (i.e.,  $\varphi$  is a Harrop formula) let  $\llbracket M \rrbracket := \varepsilon$  (the *nullterm* symbol). We extend the use of terms in the form  $(t_1, t_2)$  to nullterm symbol:  $(\varepsilon, t_2) := t_2$ ,  $(t_1, \varepsilon) := t_1$ ,  $(\varepsilon, \varepsilon) := \varepsilon$ . Moreover in case  $\tau(\varphi) = \varepsilon$ ,  $x_u^{\tau(\varphi)} := \varepsilon$  and  $(\lambda \varepsilon. \llbracket M \rrbracket)$  means just  $\llbracket M \rrbracket$ . Now assume that  $M$  derives a formula  $\varphi$  with  $\tau(\varphi) \neq \varepsilon$ . Then

$$\begin{aligned}
\llbracket u^\varphi \rrbracket &:= x_u^{\tau(\varphi)} \quad (x_u^{\tau(\varphi)} \text{ uniquely associated with } u^\varphi) \\
\llbracket (\lambda u^\varphi. N^\psi)^{\varphi \rightarrow \psi} \rrbracket &:= \lambda x_u^{\tau(\varphi)}. \llbracket N \rrbracket \\
\llbracket (M^{\varphi \rightarrow \psi} N^\varphi)^\psi \rrbracket &:= \llbracket M \rrbracket \llbracket N \rrbracket \\
\llbracket \langle M^\varphi, N^\psi \rangle^{\varphi \wedge \psi} \rrbracket &:= (\llbracket M \rrbracket, \llbracket N \rrbracket) \\
\llbracket (M^{\varphi \wedge \psi} i) \rrbracket &:= \pi_i \llbracket M \rrbracket \\
\llbracket (\lambda x^\rho. M^\varphi)^{\forall x \varphi} \rrbracket &:= \lambda x^\rho. \llbracket M \rrbracket \\
\llbracket (M^{\forall x^\rho \varphi} t^\rho)^{\varphi_x [t]} \rrbracket &:= \llbracket M \rrbracket t
\end{aligned}$$

We also need to define extracted terms for our axioms:

$$\begin{aligned}
\llbracket \exists_{x^\rho, \varphi, \psi}^- \rrbracket &:= \lambda p^{\rho \times \tau(\varphi)}, f^{\rho \rightarrow \tau(\varphi) \rightarrow \tau(\psi)}. (f(\pi_0 p)(\pi_1 p)), \text{ assuming } \tau(\varphi) \neq \varepsilon \\
\llbracket \exists_{x^\rho, \varphi}^+ \rrbracket &:= \lambda x^\rho, y^{\tau(\varphi)}. (x, y), \text{ assuming } \tau(\varphi) \neq \varepsilon \\
\llbracket \text{IF}_\varphi \rrbracket &:= \lambda b^{\mathbf{B}}, l^{\tau(\varphi)}, r^{\tau(\varphi)}. (\text{if } b \text{ } l \text{ } r), \text{ assuming } \tau(\varphi) \neq \varepsilon \\
\llbracket \text{Ind}_{n, \varphi(n)} \rrbracket &:= \mathcal{R}_{\mathbf{N}}^\sigma \\
\llbracket \text{Ind}_{l, \varphi(l)} \rrbracket &:= \mathcal{R}_{\mathbf{L}(\rho)}^\sigma \\
\llbracket \text{Ind}_{t, \varphi(t)} \rrbracket &:= \mathcal{R}_{\mathbf{B}}^\sigma
\end{aligned}$$

**Theorem 2.1 (Soundness)** *Let  $M$  be a derivation of a formula  $\varphi$  from assumptions  $u_i : \varphi_i$ . Then we can find a derivation of the formula  $(\llbracket M \rrbracket \text{ mr } \varphi)$  from assumptions  $\bar{u}_i : x_{u_i} \text{ mr } \varphi_i$ .*

*Proof.* Induction on  $M$ .

### 3 Proof Manipulation

**Definition 3.1 (Tail Expressions [5])** *The tail expressions of  $t \in \text{Terms}$ , are defined inductively as follows:*

1. If  $t \equiv (\lambda x. e)$  then  $e$  is a tail expression.
2. If  $t \equiv (\text{if } r \text{ } s)$  is a tail expression, then both  $r$  and  $s$  are tail expressions.
3. If  $t \equiv (\mathcal{R}_\iota \text{ } r \text{ } s)$  is a tail expression, then  $r$  and  $s$  are tail expressions.
4. Nothing else is a tail expression.

With  $\iota \in \{\mathbf{N}, \mathbf{L}(\rho)\}$ .

**Definition 3.2** *A tail call is a tail expression that is a procedure call.*

**Definition 3.3 (Tail Recursion [6])** *A recursive procedure is said tail recursive when it tail calls itself or calls itself indirectly through a series of tail calls.*

Now, Let consider  $F$  be the following induction proof over  $\mathbf{N}$ :

$$\frac{\begin{array}{c} |M| \\ \text{Ind}_{n, \varphi(n)} \quad \varphi(0) \quad \forall n. \varphi(n) \rightarrow \varphi(n+1) \end{array}}{\forall n. \varphi(n)}$$

The content of  $F$  is  $(\mathcal{R}_{\mathbf{N}}^\sigma b f)$  with  $b$  and  $f$  base and step case of the recursion operator, content of the proofs  $M$  and  $N$ .

### 3.1 Continuation based Tail Recursion

Given the procedure  $(\mathcal{R}_N^\sigma b f)$  defined in the previous section, let  $\Lambda$  be the term:

$$\mathcal{R}_N^{(\sigma \rightarrow \sigma') \rightarrow \sigma'} (\lambda k. kb)(\lambda n, p, k. p \lambda u. k(f n u))$$

In  $\Lambda$  we name *continuation* the input parameter of type  $(\sigma \rightarrow \sigma')$ .  $\Lambda$  is a function with just one tail recursive call and a functional accumulator parameter  $k$  with the follow property: for each  $n$ , at the  $i$ -th ( $0 < i \leq n$ ) step of the computation of  $(\Lambda n (\lambda x. x))$  the continuation has the form  $\lambda u. (f(n-1) (\dots (f(n-i) u) \dots))$ . At the  $n$ -th step the continuation  $\lambda u. (f(n-1) (\dots (f 0 u) \dots))$  is applied to the term  $b$  and returned. We see that such returned valued correspond to  $(\mathcal{R}_N^\sigma b f)_n$ . This fact is stated formally in the following,

**Theorem 3.1** *For each natural  $n$ :*

$$\Lambda n =_{\mathcal{R}_{\eta\beta}} \lambda k^{\sigma \rightarrow \sigma'}. k((\mathcal{R}_N^\sigma b f)n)$$

*Proof.* Appendix A

By Theorem 3.1,  $(\lambda n. \Lambda n (\lambda x. x))$  and  $(\mathcal{R}_N^\sigma b f)$  are extensionally equals and by definition 3.3  $\Lambda$  is tail recursive. The procedure call in  $(\lambda n. \Lambda n (\lambda x. x))$  is tail being the body of a lambda abstraction. The question that motivated the writing of this paper is: being

$$\llbracket F \rrbracket = (\mathcal{R}_N^\sigma b f)$$

then it is possible to turn  $F$  into  $F'$  such that

$$\llbracket F' \rrbracket = (\lambda n. \Lambda n (\lambda x. x))?$$

The answer is “yes”. The key point is understand the logical role of the continuation parameter in  $\Lambda$ : given a natural  $n$ , at each step  $i : n, \dots, 0$  in computing  $(\Lambda n (\lambda x. x))$ , the continuation is a function that takes the witness for  $\varphi(i)$  and returns the witness for  $\varphi(i+m)$ , for  $m$  such that  $i+m=n$ . So we expect  $\Lambda$  to be the computational content of a proof with end formula:

$$\forall n \forall^{\text{nc}} m. (\varphi(n) \rightarrow \varphi(n+m)) \rightarrow \varphi(n+m) \quad (1)$$

We observe that the counter  $m$  is introduced to count the “decrease” of  $n$  during the computation. It plays a “logical” role, but at programming level it is irrelevant. As we expect to extract programs from proofs, we explicitly underline the “hidden” role of  $m$  quantifying over it by the special *non-computational* quantifier  $\forall^{\text{nc}}$  [7, page 47]. Supposing to know the derivation terms  $M : \varphi(0)$  and  $N : \forall n. \varphi(n) \rightarrow \varphi(n+1)$ , let’s prove (1):

**Proposition 3.1**  $\varphi(0) \rightarrow (\forall n. (\varphi(n) \rightarrow \varphi(n+1))) \rightarrow \forall n \forall^{\text{nc}} m. (\varphi(n) \rightarrow \varphi(n+m)) \rightarrow \varphi(n+m)$

*Proof.* Assume  $b : \varphi(0)$  and  $f : \forall n. (\varphi(n) \rightarrow \varphi(n+1))$ . By induction on  $n$ .

$n = 0$  We have to prove

$$\forall^{\text{nc}} m. (\varphi(0) \rightarrow \varphi(m)) \rightarrow \varphi(m)$$

So assume  $m$  and  $k : (\varphi(0) \rightarrow \varphi(m))$ . Apply  $k$  to  $b : \varphi(0)$ .  
 $n + 1$  Assume  $n$ , the recursive call  $p : \forall^{\text{nc}} m. (\varphi(n) \rightarrow \varphi(n + m)) \rightarrow \varphi(n + m)$ ,  
 $m$ , and the continuation  $k : \varphi(n + 1) \rightarrow \varphi(n + m + 1)$ . We have to prove:

$$\varphi(n + m + 1)$$

Apply  $p$  to  $(m + 1)$  obtaining  $(p(m + 1)) : (\varphi(n) \rightarrow \varphi(n + m + 1)) \rightarrow \varphi(n + m + 1)$ . So, if we are able to prove the formula  $\varphi(n) \rightarrow \varphi(n + m + 1)$ , by some proof  $t$ , we can just apply  $(p(m + 1))$  to  $t$  and we are done.

So let's prove

$$\varphi(n) \rightarrow \varphi(n + m + 1)$$

Assume  $v : \varphi(n)$ . We apply  $k$  to  $(f n v)$ .  $\square$

For a formal proof in Natural Deduction style of Proposition 3.1 the reader is invited to consult the Appendix C. Now we are able to provide a new proof of the induction principle on natural numbers:

**Proposition 3.2**  $\varphi(0) \rightarrow (\forall n. (\varphi(n) \rightarrow \varphi(n + 1))) \rightarrow \forall n. \varphi(n)$ .

*Proof.* Assume  $b : \varphi(0)$ ,  $f : \forall n. (\varphi(n) \rightarrow \varphi(n + 1))$  and  $n$ . To prove  $\varphi(n)$ , instantiate the formula proved in Proposition 3.1 on  $b$ ,  $f$ ,  $n$ ,  $0$  and  $\varphi(n) \rightarrow \varphi(n)$ .  $\square$

The term extracted from the previous proof is the following:

```
[b,f,n]. (Rec nat => (sigma => sigma') => sigma')
          [k] (k b)
          [n,p,k] p([u] k(f n u))
          n ([x] x)
```

Here  $([x_1, \dots, x_N] t)$  stands for  $(\lambda x_1, \dots, x_N. t)$  and  $((\text{Rec nat} \Rightarrow \text{sigma}) \ r \ s)$  for  $(\mathcal{R}_N^\sigma r s)$ .

**Remark:** Although the functional parameter in  $A$  is named as *continuation*,  $A$  is not the CPS-transformed scheme of the recursion over naturals. In fact  $f$  and  $b$  are not altered in our transformation and they could contain *bad* expressions, like not tail calls. The formula (1) could be substituted by the more general  $\forall n. (\varphi(n) \rightarrow \perp) \rightarrow \perp$ , but the author think that the approach proposed in this paper supply a clearer idea of the logical property the continuation parameter is supposed to satisfy. Moreover, such approach represent a not trivial usage of the *not* computational quantifiers  $\forall^{\text{nc}}$ .

### 3.2 Accumulator based tail recursion

Here we present the essence of the Bauer's[1] original idea. Given the procedure  $(\mathcal{R}_N^\sigma b f)$  defined in the last section, let  $\Pi$  be:

$$\mathcal{R}_N^{\mathbf{N} \rightarrow \sigma \rightarrow \sigma} (\lambda m, y. y) (\lambda n, p, m, y. p(m + 1)(f m y))$$

In  $\Pi$  there are two accumulators parameters: a natural and parameter of type  $\sigma$  where are stored partial results. For each natural  $n$ , at the  $i$ -th ( $0 < i \leq n$ ) step of the computation of  $(\Pi n 0 b)$  the accumulator of the partial results will be equal to the expression  $(f(i-1)(\dots(f 0 b)\dots))$ . At the  $n$ -th step (base case of  $\Pi$ ) the accumulator of the partial results is returned and it correspond to  $(\mathcal{R}_N^\sigma b f)n$ . This fact is stated in the Theorem below.

**Definition 3.4** For all  $n, m$ , let  $\bar{f}^{\mathbf{N} \rightarrow \mathbf{N} \rightarrow \sigma \rightarrow \sigma}$  be a function such that:

$$\bar{f}_m n = f(n+m)$$

**Proposition 3.3** For all naturals  $n$  and  $m$ :

$$(\mathcal{R}_N^\sigma (\bar{f}_m 0 b) \bar{f}_{m+1}) n =_{\mathcal{R}\eta\beta} (\mathcal{R}_N^\sigma b \bar{f}_m) (n+1)$$

*Proof.* By induction on  $n$ . □

**Theorem 3.2** For all natural  $n$ ,

$$\Pi n =_{\mathcal{R}\eta\beta} \lambda m, y. (\mathcal{R}_N^\sigma y \bar{f}_m) n$$

*Proof.* Appendix B.

By Theorem 3.2,  $\lambda n.(\Pi n 0 b)$  and  $(\mathcal{R}_N^\sigma b f)$  are extensionally equals, moreover by definition 3.3,  $\Pi$  is tail recursive and the procedure application in  $\lambda n.(\Pi n 0 b)$  is a tail call being the body of a lambda abstraction. As done in the last section we address the following question: being

$$[F] = (\mathcal{R}_N^\sigma b f)$$

then how to turn  $F$  into  $F'$  such that:

$$[F'] = \lambda n.(\Pi n 0 b)?$$

We note that given a natural  $n$ :

$$\begin{aligned} (\lambda n.(\Pi n 0 b))n &=_{\mathcal{R}\eta\beta} (\Pi n 0 b) \\ &\vdots \\ &=_{\mathcal{R}\eta\beta} (f(n-1)(\dots(f 0 b)\dots)) \end{aligned}$$

So given two natural indexes  $i, j$ , with  $i + j = n$ ,  $(\Pi i j)$  is a function that takes the witness for  $\varphi(j)$  and returns the witness for  $\varphi(i+j)$ . So we expect  $\Pi$  to be the computational content of a proof with end formula:

$$\forall n, m. \varphi(m) \rightarrow \varphi(n+m)$$

that use the proofs terms  $M^{\varphi(0)}$  and  $N^{\forall n. \varphi(n) \rightarrow \varphi(n+1)}$  as assumptions. We prove this claim in the following,

**Proposition 3.4**  $\varphi(0) \rightarrow (\forall n. \varphi(n) \rightarrow \varphi(n+1)) \rightarrow \forall n, m. \varphi(m) \rightarrow \varphi(n+m)$

*Proof.* Assume  $b : \varphi(0)$  and  $f : \forall n. \varphi(n) \rightarrow \varphi(n+1)$ . By induction on  $n$ :

$n = 0$  We have to prove

$$\forall m. \varphi(m) \rightarrow \varphi(m)$$

this is trivially proved by  $(\lambda m. u. u)$ .

$n + 1$  Let's assume  $n$ , the recursive call  $p : \forall m. \varphi(m) \rightarrow \varphi(n+m)$ ,  $m$  and the accumulator  $y : \varphi(m)$ . We have to prove

$$\varphi(n+m+1)$$

Apply  $f$  to  $m$  and  $y$  obtaining  $(f m y) : \varphi(m+1)$ . Now apply  $p$  to  $(m+1)$  and  $(f m y)$ .  $\square$

See Appendix D for the formal proof in Natural Deduction style. The accumulator-based program transformation provides us with a new proof of the induction principle over natural numbers:

**Proposition 3.5**  $\varphi(0) \rightarrow (\forall n. \varphi(n) \rightarrow \varphi(n+1)) \rightarrow \forall n. \varphi(n)$ .

*Proof.* Assume  $b : \varphi(0)$ ,  $f : (\forall n. \varphi(n) \rightarrow \varphi(n+1))$  and  $n$ . To prove  $\varphi(n)$ : instantiate the formula proved in Proposition 3.4 on  $n$ ,  $0$  and  $b : \varphi(0)$   $\square$

The term extracted from the previous proof is the following:

```
[b,f,n].(Rec nat => nat => sigma => sigma)
  [m,y]y
  [n,p,m,y]p (m+1)(f m u))
  n 0 b
```

## 4 Conclusions and future work

The expression  $\Pi$  (section 3.2) represent a way to mimic a *let* expression just by a pure lambda calculus expression (original goal of Bauer in designing it). Moreover, while  $\Pi$  is tail recursive the recursive schema over natural written by a *let* would not. The terms  $\Lambda$  and  $\Pi$  compute, applied to opportune input parameters, the same function  $(\mathcal{R}_N^G b f)$ . But we note that  $\Lambda$  is in some way more *general* than  $\Pi$ . The modification of  $\Lambda$  in order to make it working on lists (let's name it  $\Lambda_{L(\rho)}$ ) instead of naturals is straightforward; but more important, the proof from which  $\Lambda_{L(\rho)}$  can be extracted is obtained by a slightly modification of the proof from which  $\Lambda$  is extracted. In the case of lists the end formula to prove should be:  $\forall l^{L(\rho)}.(P(l) \rightarrow \perp) \rightarrow \perp$ . Unfortunately we can not extend in the same way  $\Pi$  and its proof:  $\Pi$  looks intrinsically dependent from the algebra of natural numbers. Our current work regard the study of such extensions. Which is the formal connection between  $\Lambda$  and  $\Pi$ ? We claim is possible transform  $\Lambda$  into  $\Pi$  by the so called *defunctionalization* technique (Reynolds [8], Danvy and others [9]) a whole program transformation to turn higher-order into

first-order functional programs. But also this aspect need a deeper investigation. Possible applications of  $\Lambda$  and  $\Pi$  go beyond the *tail recursion*. We noted that there exists proofs from which are extracted programs that run in exponential time that can be turned (by the proofs transformations proposed here) in new proofs from which is possible to extract polynomial time algorithms. This can appear pretty amazing and we are currently working in order to state such result more precisely. Another application of the proofs transformations proposed here is an extension of the CPS-transformation over formal proofs (Schwichtenberg[10] and Griffin[11]) but this time concerning the induction axiom. The proposal is perform CPS over proofs in two stages: a preprocessing step where are transformed all the proofs by induction, and a second stage where the *canonical* CPS-transformation is applied. Currently we are studying also this aspect but it need a deeper investigation.

## Acknowledgments

I really thank Helmut Schwichtenberg, Philippe Audebaud, Amr Sabry Stefan Schimanski, Freiric Barral, Volker Heun, Andrej Bauer for the helpful discussions and feedback. Thank's to the anonymous referee for their useful comments.

## References

1. <http://math.andrej.com/2005/09/16/proof-hacking/>.
2. <http://www.minlog-system.de/>.
3. M.H. Sørensen and P.Urzyczyn. *Lectures on the Curry-Howard Isomorphism*, volume 149 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2006.
4. G. Kreisel. Interpretation of Analysis by means of Functionals of Finite Type. In A. Heyting, editor, *Constructivity in Mathematics*, 1959.
5. R. Kelsey, W. Clinger, and J. Rees (eds.). Revised<sup>5</sup> report on the algorithmic language scheme. *Higher-Order and Symbolic Computation*, 11(1), August, 1998.
6. K. Kent Dybvig. *The Scheme Programming Language*. Mit Press, 1996.
7. Helmut Schwichtenberg. Minimal Logic for Computable Functionals. <http://www.mathematik.uni-muenchen.de/~chiarabi/mlcf.pdf>, December 2008.
8. John C. Reynolds. Definitional interpreters for higher-order programming languages. *Higher-Order and Symbolic Computation*, 11(4):363–397, 1998. Reprinted from the proceedings of the 25th ACM National Conference (1972).
9. Olivier Danvy and Lasse R.Nielsen. Defunctionalization at work. In editor Harald Søndergaard, editor, *Proceedings of the Third International Conference of Principles and Practice of Declarative Programming*, pages 162–174, Firenze, Italy, September 2001. ACM Press. Extended version available as the technical report BRICS RS-01-23.
10. Helmut Schwichtenberg. Proofs, lambda terms and control operators. In *Logic of computation. Proceedings of the NATO ASI.Marktoberdorf, Germany*, 1995.
11. Timothy G. Griffin. A formulae-as-types notion of control. In *17th Annual ACM Symp. on Principles of Programming Languages, POPL'90, San Francisco, CA, USA*, 1990.

## A

**Theorem A.1** *For each natural n:*

$$\Lambda n =_{\mathcal{R}_{\eta\beta}} \lambda k^{\sigma \rightarrow \sigma'}. k((\mathcal{R}_N^\sigma b f)n)$$

*Proof.* By induction over n:

$$n = 0$$

$$\begin{aligned} \Lambda 0 &=_{\mathcal{R}_{\eta\beta}} \lambda k. kb \\ &=_{\mathcal{R}_{\eta\beta}} \lambda k. k((\mathcal{R}_N^\sigma b f)0) \end{aligned}$$

$$n + 1$$

$$\begin{aligned} \Lambda(n+1) &=_{\mathcal{R}_{\eta\beta}} (\lambda n, p, k. p \lambda u. k(f n u)) n (\Lambda n) \\ &=_{\mathcal{R}_{\eta\beta}} \lambda k. (\Lambda n) \lambda u. k(f n u) \\ &=_{\mathcal{R}_{\eta\beta}} \lambda k. (\lambda k. k((\mathcal{R}_N^\sigma b f)n)) \lambda u. k(f n u) \quad (\text{By IH}) \\ &=_{\mathcal{R}_{\eta\beta}} \lambda k. (\lambda u. k(f n u)) ((\mathcal{R}_N^\sigma b f)n) \\ &=_{\mathcal{R}_{\eta\beta}} \lambda k. k(f n ((\mathcal{R}_N^\sigma b f)n)) \\ &=_{\mathcal{R}_{\eta\beta}} \lambda k. k((\mathcal{R}_N^\sigma b f)(n+1)) \end{aligned}$$

□

## B

**Theorem B.1** *For all natural n,*

$$\Pi n =_{\mathcal{R}_{\eta\beta}} \lambda m, y. (\mathcal{R}_N^\sigma y \bar{f}_m) n$$

*Proof.* By induction on n:

$$n = 0$$

$$\begin{aligned} \Pi 0 &=_{\mathcal{R}_{\eta\beta}} \lambda m, y. y \\ &=_{\mathcal{R}_{\eta\beta}} \lambda m, y. (\mathcal{R}_N^\sigma y \bar{f}_m) 0 \end{aligned}$$

$$n + 1$$

$$\begin{aligned} \Pi(n+1) &=_{\mathcal{R}_{\eta\beta}} (\lambda n, p, m, y. p(m+1)(f m y)) n (\Pi n) \\ &=_{\mathcal{R}_{\eta\beta}} \lambda m, y. (\Pi n) (m+1) (f m y) \\ &=_{\mathcal{R}_{\eta\beta}} \lambda m, y. (\lambda m, y. (\mathcal{R}_N^\sigma y \bar{f}_m) n) (m+1) (f m y) \quad \text{By IH} \\ &=_{\mathcal{R}_{\eta\beta}} \lambda m, y. (\mathcal{R}_N^\sigma (f m y) \bar{f}_{m+1}) n \\ &=_{\mathcal{R}_{\eta\beta}} \lambda m, y. (\mathcal{R}_N^\sigma (\bar{f}_m 0 y) \bar{f}_{m+1}) n \quad \text{By Def.3.4} \\ &=_{\mathcal{R}_{\eta\beta}} \lambda m, y. (\mathcal{R}_N^\sigma y \bar{f}_m) (n+1) \quad \text{By Prop. 3.3} \end{aligned}$$

□

**C**

**Proposition C.1**  $\varphi(0) \rightarrow \forall n. \varphi(n) \rightarrow \varphi(n+1)) \rightarrow \forall n \forall^{\text{nc}} m. (\varphi(n) \rightarrow \varphi(n+m)) \rightarrow \varphi(n+m)$

*Proof.* Assume  $b : \varphi(0)$  and  $f : \forall n. \varphi(n) \rightarrow \varphi(n+1))$ . By induction on  $n$ :

$$n > 0$$

$$\begin{array}{c}
\frac{[k : \varphi(0) \rightarrow \varphi(m)] \quad [b : \varphi(0)]}{\frac{(kb) : \varphi(m)}{(\lambda k. kb) : (\varphi(0) \rightarrow \varphi(m)) \rightarrow \varphi(m)}} \\
\frac{}{(\lambda m, k. kb) : \forall^{\text{nc}} m. (\varphi(0) \rightarrow \varphi(m)) \rightarrow \varphi(m)}
\\
\hline
\frac{[f : \forall n. \varphi(n) \rightarrow \varphi(n+1)] \quad n}{\frac{(fn) : \varphi(n) \rightarrow \varphi(n+1)}{(fnv) : \varphi(n+1)}} \quad [v : \varphi(n)] \\
\frac{[k : \varphi(n+1) \rightarrow \varphi(n+m+1)]}{\frac{(k(fnv)) : \varphi(n+m+1)}{(\lambda v. k(fnv)) : \varphi(n) \rightarrow \varphi(n+m+1)}} \\
\hline
\frac{\frac{[p : \forall^{\text{nc}} m. (\varphi(n) \rightarrow \varphi(n+m)) \rightarrow \varphi(n+m)] \quad (m+1)}{(p(m+1)) : (\varphi(n) \rightarrow \varphi(n+m+1)) \rightarrow \varphi(n+m+1)}}{(p(m+1)(\lambda v. k(fnv))) : \varphi(n+m+1)} \\
\frac{}{(\lambda k. p(m+1)(\lambda v. k(fnv))) : (\varphi(n+1) \rightarrow \varphi(n+m+1)) \rightarrow \varphi(n+m+1)} \\
\frac{}{(\lambda m, k. p(m+1)(\lambda v. k(fnv))) : \forall^{\text{nc}} m. (\varphi(n+1) \rightarrow \varphi(n+m+1)) \rightarrow A(n+m+1)} \\
\frac{}{(\lambda p, m, k. p(m+1)(\lambda v. k(fnv))) : (\forall^{\text{nc}} m. (\varphi(n) \rightarrow \varphi(n+m)) \rightarrow \varphi(n+m)) \rightarrow} \\
\frac{}{\forall^{\text{nc}} m. (\varphi(n+1) \rightarrow \varphi(n+m+1)) \rightarrow \varphi(n+m+1)} \\
\hline
\frac{}{(\lambda n, p, m, k. p(m+1)(\lambda v. k(fnv))) : \forall n. (\forall^{\text{nc}} m. (\varphi(n) \rightarrow \varphi(n+m)) \rightarrow \varphi(n+m)) \rightarrow} \\
\frac{}{\forall^{\text{nc}} m. (\varphi(n+1) \rightarrow \varphi(n+m+1)) \rightarrow \varphi(n+m+1)}
\end{array}$$

**D**

**Proposition D.1**  $\varphi(0) \rightarrow (\forall n. \varphi(n) \rightarrow \varphi(n+1)) \rightarrow \forall n, m. \varphi(m) \rightarrow \varphi(n+m)$

*Proof.* Assume  $b : \varphi(0)$ ,  $f : (\forall n. \varphi(n) \rightarrow \varphi(n+1))$ . By induction on  $n$ :

$$n = 0$$

$$\frac{[u : \varphi(m)]}{\frac{(\lambda u. u) : \varphi(m) \rightarrow \varphi(m)}{(\lambda m, u. u) : \forall m. \varphi(m) \rightarrow \varphi(m)}}$$

$$n > 0$$

$$\begin{array}{c}
 \frac{[f : \forall n. \varphi(n) \rightarrow \varphi(n+1)] \quad m}{(f m) : \varphi(m) \rightarrow \varphi(m+1) \quad [y : \varphi(m)]} \\
 \frac{}{(f m y) : \varphi(m+1)} \\
 \searrow \\
 \frac{\frac{[p : \forall m. \varphi(m) \rightarrow \varphi(n+m)] \quad (m+1)}{(p(m+1)) : \varphi(m+1) \rightarrow \varphi(n+m+1)}}{\frac{\frac{(p(m+1))(f m y) : \varphi(n+m+1)}{(\lambda y. p(m+1)(f m y)) : \varphi(m) \rightarrow \varphi(n+m+1)}}{\frac{(\lambda m, y. p(m+1)(f m y)) : \forall m. \varphi(m) \rightarrow \varphi(n+m+1)}{(\lambda p, m, y. p(m+1)(f m y)) : (\forall m. \varphi(m) \rightarrow \varphi(n+m)) \rightarrow \\ (\forall m. \varphi(m) \rightarrow \varphi(n+m+1))}}}
 \end{array}$$

# Phase Shifts of LFSM as Pseudorandom Number Generators for BIST for VLSI \*

Sung-Jin Cho<sup>1</sup>, Un-Sook Choi<sup>2</sup>, Han-Doo Kim<sup>3</sup>, Yoon-Hee Hwang<sup>4</sup>, and Jin-Gyoung Kim<sup>5</sup>

<sup>1</sup> Division of Mathematical Sciences, Pukyong National University  
Busan 608-737, Korea, sjcho@pknu.ac.kr

<sup>2</sup> Department of Multimedia Engineering, Tongmyong University  
Busan 626-847, Korea, choies@tu.ac.kr

<sup>3</sup> Institute of Mathematical Sciences and School of Computer Aided Science  
Inje University, Gimhae 621-749, Korea, mathkhd@inje.ac.kr

<sup>4</sup> Department of Information Security, Graduate School  
Pukyong National University  
Busan 608-737, Korea, yhhwang@pknu.ac.kr

<sup>5</sup> Department of Applied Mathematics, Graduate School  
Pukyong National University  
Busan 608-737, Korea, 5892587@hanmail.net

**Abstract.** Large phase shifts are generally desirable as they result in less correlation and, therefore, higher fault coverage in the testing of VLSI. In this paper, we investigate the phase shifts of the sequences generated by companion matrices of primitive polynomials. Also we propose an algorithm for finding phase shifts of the sequences.

## 1 Introduction

The bit sequences generated by successive cells of built-in test pattern generators suffer in general from correlations and/or linear dependencies. This is problematic for pseudorandom and/or pseudoexhaustive generation of test patterns, structures commonly employed in a variety of test applications ([1] ~ [3]). A linear feedback shift register (LFSR) is usually used as an on-chip test pattern generator, which drives flip-flop chains in parallel. In the context of digital circuit testing, LFSRs have been used as very popular signature analyzers. As the number of gates or digital circuits that can be integrated into one chip of silicon becomes hundreds of million, the testing of VLSI has become a very important engineering problem. Since the number of states that a VLSI can assume is astronomically large, it is impossible to check all the states of each chip of VLSI before shipping. So we check it by some mechanism of random sampling. It was customary up to several years ago to use an LFSR as a test pattern generator for this purpose. A linear finite state machine (LFSM) is an external XOR LFSRs.

---

\* This work was supported by grant No. (R01-2006-000-10260-0) from the Basic Research Program of the Korea Science and Engineering Foundation.

Linear dependencies among bit positions in the patterns generated by an LFSM are to be avoided because they limit the number of different bit combinations seen by the circuit under test when that LFSM is used for pseudorandom testing of the circuit. The issue of linear dependencies has been studied by various researchers ([3] ~ [6]). In general, the bit sequences produced by an LFSM are shifted versions of each other. The difference in the number of shift positions that each bit sequence exhibits with respect to a reference sequence is referred to as the phase shift of that sequence. Large phase shifts are generally desirable as they result in less correlation and, therefore, higher fault coverage in the testing of VLSI. For many years, many researchers have been working on how to find phase shifts of maximum-length sequences. The phase shift analysis of 90/150 CA whose characteristic polynomials are primitive, has been investigated by Bardell [7], Das et al. [8], Sarkar ([9], [10]) and Cho et al. ([11], [12]). Also the phase shift analysis of LFSMs whose characteristic polynomials are primitive has been investigated by Bardell et al. [2] and Kagaris et al. ([4], [13] ~ [17]).

In this paper, we investigate the phase shifts of the sequences generated by companion matrices of primitive polynomials. Also we propose an algorithm for finding phase shifts of the sequences.

## 2 Preliminaries

Let  $GF(2)$  be the Galois field with cardinality 2. In this section, we investigate some properties of sequences generated by companion matrices of primitive polynomials.

**Definition 2.1.** Let  $f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1x + 1$ , where  $c_i \in GF(2)$ . Then the following  $n \times n$  matrix  $T$  is said to be the *companion matrix* of  $f(x)$ .

$$T = \left( \begin{array}{c|cccc} c_{n-1} & 1 & 0 & \cdots & 0 \\ c_{n-2} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_1 & 0 & 0 & \cdots & 1 \\ \hline 1 & 0 & 0 & \cdots & 0 \end{array} \right) = \left( \begin{array}{c|c} C_{(n-1) \times 1} & I_{n-1} \\ \hline 1 & O_{1 \times (n-1)} \end{array} \right)$$

, where  $I_{n-1}$  is an  $(n-1) \times (n-1)$  identity matrix and  $C = (c_{n-1}, c_{n-2}, \dots, c_1)^t$ . Since  $|T| = |I_{n-1}| = 1$ ,  $T$  is nonsingular. The inverse  $T^{-1}$  of  $T$  is

$$T^{-1} = \left( \begin{array}{c|c} 0 & 0 & \cdots & 0 & 0 & 1 \\ \hline 1 & 0 & \cdots & 0 & 0 & c_{n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & c_2 \\ 0 & 0 & \cdots & 0 & 1 & c_1 \end{array} \right) = \left( \begin{array}{c|c} O_{1 \times (n-1)} & 1 \\ \hline I_{n-1} & C_{(n-1) \times 1} \end{array} \right)$$

**Definition 2.2.** ([18], [19]) Let  $f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1x + 1$  be an  $n$ -degree primitive polynomial, where  $c_1, \dots, c_{n-1} \in GF(2)$ . Then

$f(x)$  generates a periodic sequence whose period is  $2^n - 1$ . This sequence is called a *pseudo-noise(PN) sequence*.

PN sequences are of fundamental importance in computer science, cryptology and engineering. The following are some basic properties of PN sequences of degree  $n$  [18].

(P1) The period of a PN sequence of degree  $n$  is  $2^n - 1$ .

(P2) In a PN sequence of period  $2^n - 1$ , the 0-run of length  $n - 1$  occurs exactly once.

### 3 Analysis of sequences generated by companion matrices of primitive polynomials

In this section, the method for finding phase shifts of the sequences generated by companion matrices of primitive polynomials is presented. The following theorems give various methods to compute phase shifts.

Let  $S$  be the  $(2^n - 1) \times n$  matrix that has as rows  $\mathbf{x}_0, T\mathbf{x}_0, \dots, T^{2^n - 2}\mathbf{x}_0$ , where  $\mathbf{x}_0 = (\overbrace{0, \dots, 0}^{n-1}, 1)^t$ . Then  $S$  is the  $(2^n - 1) \times n$  matrix consisting of  $n$  independent PN sequences as its columns. Denote  $T^i\mathbf{x}_0$  by  $\mathbf{x}_i$  for each  $i (1 \leq i \leq 2^n - 2)$ .

$$S = (s_{ij})_{(2^n - 1) \times n} = \begin{pmatrix} \mathbf{x}_0^t \\ \mathbf{x}_1^t \\ \vdots \\ \mathbf{x}_{n-1}^t \\ \mathbf{x}_n^t \\ \vdots \\ \mathbf{x}_{2^n - 2}^t \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ c_{n-1} & c_{n-2} & c_{n-3} & \cdots & c_2 & c_1 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & c_{n-1} & c_{n-2} & \cdots & c_3 & c_2 & c_1 \end{pmatrix}$$

Let  $ps(i)$  be the phase shift of the  $i$ th column with respect to the 1st column of  $S$ . From the definition of the phase shift we obtain the following theorems.

**Theorem 3.1.** Let  $T$  be the companion matrix of the  $n$ -degree primitive polynomial  $f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1x + 1$ . Then  $ps(1) = 0$  and  $ps(n) = 1$ .

**Theorem 3.2.** Let  $T$  be the companion matrix of the  $n$ -degree primitive polynomial  $f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1x + 1$  and let  $c_{n-1} = c_{n-2} = \cdots = c_{n-m} = 0$  and  $c_{n-m-1} = 1$ . Then  $ps(i) = 2^n - i$  ( $2 \leq i \leq m + 1$ ).

**Example 3.3.** We can find the phase shifts of the primitive polynomial  $x^6 + x + 1$  by Theorem 3.1 and Theorem 3.2.  $ps(1) = 0, ps(6) = 1$  by Theorem

3.1.  $ps(2) = 2^6 - 2 = 62$ ,  $ps(3) = 2^6 - 3 = 61$ ,  $ps(4) = 2^6 - 4 = 60$  and  $ps(5) = 2^6 - 5 = 59$  by Theorem 3.2.

**Lemma 3.4.** Let  $T$  be the companion matrix of the  $n$ -degree primitive polynomial  $f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1x + 1$ . And let  $T^{k+i} + T^{k+(i+m)} = T^i$  ( $k \in \mathbf{N}$ ,  $0 \leq i \leq n-2$  and  $1 \leq m \leq n-1$ ), where  $\mathbf{N}$  is the set of all positive integers. Then  $\mathbf{x}_{\mathbf{k+i}} + \mathbf{x}_{\mathbf{k+(i+m)}} = \mathbf{x}_i$ .

**Lemma 3.5.** Let  $f(x)$  be the  $n$ -degree primitive polynomial  $f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1x + 1$ . And let the  $(j+1)$ th row vector of  $S$  be  $(x_1, x_2, \dots, x_n)$ . Then the  $(j+2)$ th row vector  $\mathbf{x}_{j+1}^t$  of  $S$  is  $(x_2, x_3, \dots, x_n, 0) + x_1(c_{n-1}, \dots, c_1, 1)$ .

**Theorem 3.6.** Let  $T$  be the companion matrix of the  $n$ -degree primitive polynomial  $f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1x + 1$ . And let  $c_{n-1} = 1$  and  $T^k + T^{k+1} = I$  for some  $k \in \mathbf{N}$ . Then  $ps(2) = k$ .

*Proof.* Since  $T^k + T^{k+1} = I$ ,  $T^{k+i} + T^{k+(i+1)} = T^i$  ( $0 \leq i \leq n-1$ ). By Lemma 3.4,  $\mathbf{x}_{\mathbf{k+i}} + \mathbf{x}_{\mathbf{k+i+1}} = \mathbf{x}_i$  ( $0 \leq i \leq n-1$ ). That is,

$$\begin{aligned} s_{k+1,n} + s_{k+2,n} &= 1, \\ s_{k+2,n-1} + s_{k+3,n-1} &= 1, \\ &\vdots \\ s_{k+n,1} + s_{k+n+1,1} &= 1. \end{aligned}$$

Hence the 1st (resp. 2nd) elements of the  $(n-1)$  vectors  $\mathbf{x}_k^t, \dots, \mathbf{x}_{k+n-2}^t$  are the same. Suppose that the 1st and the 2nd elements of the  $(n-1)$  vectors  $\mathbf{x}_k^t, \dots, \mathbf{x}_{k+n-2}^t$  are of the form "0 \*\*". In the 1st column of  $S$  there are two vectors  $(s_{11}, \dots, s_{n-1,1})^t = (0, \dots, 0)^t$  and  $(s_{k+1,1}, \dots, s_{k+n-1,1})^t = (0, \dots, 0)^t$ . Since the 0-run of length  $n-1$  occurs exactly once in a PN sequence of period  $2^n - 1$  by (P2), this is a contradiction. This means that the 1st and the 2nd elements of the  $(n-1)$  vectors  $\mathbf{x}_k^t, \dots, \mathbf{x}_{k+n-2}^t$  must be of the form "1 \*\*". Suppose that the 1st and the 2nd elements of the  $(n-1)$  vectors  $\mathbf{x}_k^t, \dots, \mathbf{x}_{k+n-2}^t$  are of the form "11". Since  $\mathbf{x}_k = (1, 1, *, \dots, *)^t$ ,  $\mathbf{x}_{k+1} = (1, *, \dots, *, 0)^t + (c_{n-1}, c_{n-2}, \dots, c_1, 1)^t = (0, *, \dots, *, 1)^t$  by Lemma 3.5. But  $\mathbf{x}_k + \mathbf{x}_{k+1} = (0, \dots, 0, 1)^t$ . This is a contradiction. Thus the 1st and the 2nd elements of the  $(n-1)$  vectors  $\mathbf{x}_k^t, \dots, \mathbf{x}_{k+n-2}^t$  must be all "10". Since  $(n-1)$  consecutive 0's occur from the  $(k+1)$ th element to the  $(k+n-1)$ th element in the 2nd column of  $S$ ,  $ps(2) = k$ .

**Example 3.7.** We can find the phase shift  $ps(2)$  of the primitive polynomial  $x^7 + x^6 + x^5 + x^4 + 1$  by Theorem 3.6. Since  $T^{86} + T^{87} = I$ ,  $ps(2) = 86$ .

**Theorem 3.8.** Let  $T$  be the companion matrix of the  $n$ -degree primitive polynomial  $f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1x + 1$ . And let  $c_{i+1} = 1$  and  $c_i = c_{i-1} = \cdots = c_1 = 0$  for some  $i$  ( $1 \leq i \leq n-2$ ). Then  $ps(n-j) = j+1$  ( $1 \leq j \leq i$ ).

*Proof.* By Lemma 3.5, the  $(n+2)$ th row vector  $\mathbf{x}_{n+1}^t$  of  $S$  is of the following form:

$$\begin{aligned}\mathbf{x}_{n+1}^t &= c_{n-1}(c_{n-1}, \dots, 1, \underbrace{0}_{\text{n-i}}, \dots, 0, 0, 1) + (c_{n-2}, \dots, \underbrace{0}_{\text{n-i-1}}, 0, \dots, 0, 1, 0) \\ &= (c_{n-1} + c_{n-2}, \dots, c_{n-1}, \underbrace{0}_{\text{n-i}}, \dots, 0, 1, c_{n-1}).\end{aligned}$$

By the similar method,  $S$  is of the following form:

$$\left( \begin{array}{c} \mathbf{x}_0^t \\ \mathbf{x}_1^t \\ \vdots \\ \mathbf{x}_i^t \\ \mathbf{x}_{i+1}^t \\ \vdots \\ \mathbf{x}_{n-1}^t \\ \mathbf{x}_n^t \\ \mathbf{x}_{n+1}^t \\ \mathbf{x}_{n+2}^t \\ \vdots \\ \mathbf{x}_{n+i-1}^t \\ \mathbf{x}_{n+i}^t \\ \vdots \end{array} \right) = \left( \begin{array}{ccccccccc} 0 & 0 & \cdots & 0 & \underbrace{0}_{\text{n-i}} & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ c_{n-1} & c_{n-2} & \cdots & 1 & & & & & & \\ * & * & \cdots & c_{n-1} & & & & & & \\ * & * & \cdots & * & & & & & & \\ \vdots & \vdots & \ddots & \vdots & & & & & & \\ * & * & \cdots & * & & & & & & \\ * & * & \cdots & * & & & & & & \\ \vdots & \vdots & \ddots & \vdots & & & & & & \\ \boxed{B} & & & & & & & & & \end{array} \right)$$

, where  $B$  is the following  $(i+1) \times (i+1)$  submatrix of  $S$ :

$$\begin{aligned}B &= \begin{pmatrix} s_{n+1,n-i} & s_{n+1,n-i+1} & \cdots & s_{n+1,n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n+i+1,n-i} & s_{n+i+1,n-i+1} & \cdots & s_{n+i+1,n} \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & \cdots & 0 & 1 & * \\ 0 & 0 & \cdots & 1 & * & * \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 1 & \cdots & * & * & * \\ 1 & * & \cdots & * & * & * \end{pmatrix}\end{aligned}$$

For  $1 \leq j \leq i$ , since  $s_{j+1,n-j} = 1, s_{j+2,n-j} = s_{j+3,n-j} = \cdots = s_{j+n,n-j} = 0$  and  $s_{j+n+1,n-j} = 1$ , the number of consecutive 0's in the  $(n-j)$ th column of  $S$  is  $n-1$ . Thus  $ps(n-j) = j+1$ .

**Example 3.9.** We can find the phase shifts  $ps(3) = 3$  and  $ps(4) = 2$  of the primitive polynomial  $x^5 + x^3 + 1$  by Theorem 3.8.

**Corollary 3.10.** Let  $T$  be the companion matrix of the  $n$ -degree primitive polynomial  $f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1x + 1$ . And let  $c_{i+1} = 1$  and  $c_i = c_{i-1} = \cdots = c_1 = 0$ . Then  $ps(k) = n - k + 1 (n - i \leq k \leq n - 1)$ .

**Theorem 3.11.** Let  $T$  be the companion matrix of the  $n$ -degree primitive polynomial  $f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1x + 1$ . And let  $c_{i+1} = 0$ . Then  $ps(n-i-1) = ps(n-i) + 1 (0 \leq i \leq n-2)$ .

*Proof.* Let  $ps(n-i) = p$ . Then  $s_{p+1,n-i} = s_{p+2,n-i} = \cdots = s_{p+(n-1),n-i} = 0$  and  $s_{p+n,n-i} = 1$  by (P2). Since  $s_{k+1,n-i-1} = s_{k,n-i} + c_{i+1}s_{k,1} (1 \leq k \leq 2^n - 2)$  by Lemma 3.5 and  $c_{i+1} = 0$ ,

$$\begin{aligned} s_{p+2,n-i-1} &= s_{p+1,n-i} = 0, \\ s_{p+3,n-i-1} &= s_{p+2,n-i} = 0, \\ &\vdots \\ s_{p+n,n-i-1} &= s_{p+n-1,n-i} = 0, \\ s_{p+n+1,n-i-1} &= s_{p+n,n-i} = 1. \end{aligned}$$

Thus  $ps(n-i-1) = ps(n-i) + 1$ .

**Corollary 3.12.** Let  $T$  be the companion matrix of the  $n$ -degree primitive polynomial  $f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1x + 1$ . And let  $c_{i+k} = c_{i+(k-1)} = \cdots = c_{i+1} = 0$  and  $ps(n-i) = p$ . Then  $ps(n-i-j) = ps(n-i) + j = p + j (1 \leq j \leq k)$ .

**Theorem 3.13.** Let  $T$  be the companion matrix of the  $n$ -degree primitive polynomial  $f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1x + 1$ . And let  $c_{n-1} = \cdots = c_{n-m+1} = 0$  and  $c_{n-m} = 1$ . If  $T^k + T^{k+m} = I$  for some  $k \in \mathbf{N}$ , then  $ps(m+1) = k$ .

*Proof.* Since  $T^k + T^{k+m} = I$ ,  $T^{k+i} + T^{k+m+i} = T^i (0 \leq i \leq n-1)$ . By Lemma 3.4,  $\mathbf{x}_{\mathbf{k}+\mathbf{i}} + \mathbf{x}_{\mathbf{k}+\mathbf{m}+\mathbf{i}} = \mathbf{x}_i (0 \leq i \leq n-1)$ . Since  $\mathbf{x}_k + \mathbf{x}_{k+m} = \mathbf{x}_0$ ,  $s_{k+1,m+1} = \cdots = s_{k+m+2,m+1} = 0$ . From  $\mathbf{x}_{\mathbf{k}+\mathbf{i}} + \mathbf{x}_{\mathbf{k}+\mathbf{m}+\mathbf{i}} = \mathbf{x}_i (1 \leq i \leq n-1)$ , we obtain  $s_{k+m+3,m+1} = \cdots = s_{k+(n-1),m+1} = 0$ . Thus  $s_{k+1,m+1} = s_{k+2,m+1} = \cdots = s_{k+(n-1),m+1} = 0$  and  $s_{k+n,m+1} = 1$ . Hence by (P2)  $ps(m+1) = k$ .

**Example 3.14.** We can find the phase shift  $ps(3) = 14$  of the primitive polynomial  $x^6 + x^4 + x^3 + x + 1$  by Theorem 3.13.

**Theorem 3.15.** Let  $T$  be the companion matrix of the  $n$ -degree primitive polynomial  $f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1x + 1$ . And let  $c_i = 1$  and  $c_{i-1} = \cdots = c_1 = 0$ . If  $T^k + T^{k-i} = I$  for some  $k \in \mathbf{N}$ , then  $ps(n-i) = k+1$ .

*Proof.* The matrix obtained by  $(i+1)$  row vectors  $\mathbf{x}_{\mathbf{k}-\mathbf{i}}^t, \mathbf{x}_{\mathbf{k}-\mathbf{i}+1}^t, \dots, \mathbf{x}_{\mathbf{k}}^t$  of  $S$  is of the following form:

$$\begin{pmatrix} \mathbf{x}_{\mathbf{k}-\mathbf{i}}^t \\ \mathbf{x}_{\mathbf{k}-\mathbf{i}+1}^t \\ \mathbf{x}_{\mathbf{k}-\mathbf{i}+2}^t \\ \vdots \\ \mathbf{x}_{\mathbf{k}-1}^t \\ \mathbf{x}_{\mathbf{k}}^t \end{pmatrix} =$$

$$\begin{pmatrix} s_{k-i+1,1} & \cdots & s_{k-i+1,n-i}^{n-i} & s_{k-i+1,n-i+1}^{n-i+1} & \cdots & s_{k-i+1,n} \\ s_{k-i+2,1} & \cdots & s_{k-i+1,n-i+1} + c_i \cdot s_{k-i+1,1} & * & \cdots & s_{k-i+1,1} \\ s_{k-i+3,1} & \cdots & s_{k-i+1,n-i+2} + c_i \cdot s_{k-i+2,1} & * & \cdots & s_{k-i+2,1} \\ \vdots & & & & & \\ s_{k,1} & \cdots & s_{k-i+1,n-1} + c_i \cdot s_{k-1,1} & s_{k-i+1,n} & s_{k-i+1,1} & \cdots \\ s_{k-i+1,1} & \cdots & s_{k-i+1,n-1} & s_{k-i+1,1} & \cdots & \overline{s_{k-i+1,n}} \end{pmatrix}$$

Since  $c_{i-1} = c_{i-2} = \cdots = c_1 = 0$ ,  $s_{k+1,n-i+1} = s_{k-i+1,1}$ . Since  $T^{k-i} + T^k = I$ , by Lemma 3.4,  $\mathbf{x}_{\mathbf{k}-\mathbf{i}} + \mathbf{x}_{\mathbf{k}} = \mathbf{x}_0$  and thus  $s_{k-i+1,n-i+1} + s_{k+1,n-i+1} = 0$ . Since  $s_{k+1,n-i+1} = s_{k-i+1,1}$ ,  $s_{k-i+1,n-i+1} = s_{k-i+1,1}$ . Therefore  $s_{k-i+1,n-i+1} + c_i \cdot s_{k-i+1,1} = 0$  because  $c_i = 1$ . By the similar method we can show that  $s_{k-i+3,n-i} = s_{k-i+4,n-i} = \cdots = s_{k,n-i} = 0$ . Also  $s_{k-i+1,n-i} = s_{k-i+1,n} + c_i \cdot s_{k,1} = s_{k-i+1,n} + \overline{s_{k-i+1,n}} = 1$ . Thus  $s_{k+1,n-i} = 1$  and  $s_{k+2,n-i} = \cdots = s_{k+n,n-i} = 0$ . Hence  $ps(n-i) = k+1$ .

**Theorem 3.16.** Let  $T$  be the companion matrix of the  $n$ -degree primitive polynomial  $f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1x + 1$ . Let  $c_1 = c_2 = \cdots = c_m = 1$  and  $c_{m+1} = 0$ . And let  $T^{k_i} + T^{k_i+1} + \cdots + T^{k_i+i} = I$  ( $1 \leq i \leq m$ ). Then  $ps(n-i) = k_i + i + 1$ .

*Proof.* For  $k_i$  such that  $T^{k_i} + T^{k_i+1} + \cdots + T^{k_i+i} = I$ , the matrix obtained by  $(i+1)$  vectors  $\mathbf{x}_{\mathbf{k}_i}, \mathbf{x}_{\mathbf{k}_i+1}, \dots, \mathbf{x}_{\mathbf{k}_i+i}$  of  $S$  is of the following form:

$$\begin{pmatrix} \mathbf{x}_{\mathbf{k}_i}^t \\ \mathbf{x}_{\mathbf{k}_i+1}^t \\ \vdots \\ \mathbf{x}_{\mathbf{k}_i+i-2}^t \\ \mathbf{x}_{\mathbf{k}_i+i-1}^t \\ \mathbf{x}_{\mathbf{k}_i+i}^t \end{pmatrix} = \begin{pmatrix} * & * & \cdots & 0 & * & * & \cdots & * \\ * & * & \cdots & 0 & * & * & \cdots & * \\ \vdots & \vdots \\ & & & & 0 & & & \\ & & & & 1 & & & \\ s_{k_i+(i+1),1} & s_{k_i+(i+1),2} & \cdots & 1 & * & \cdots & \cdots & s_{k_i+(i+1),n} \end{pmatrix}^{n-i}$$

Since  $c_1 = c_2 = \cdots = c_i = 1$ ,  $s_{k_i+1,n-i} = \cdots = s_{k_i+(i-1),n-i} = 0$  and  $s_{k_i+i,n-i} = s_{k_i+(i+1),n-i} = 1$ , so that  $s_{k_i+(i+2),n-i} = \cdots = s_{k_i+(i+n),n-i} = 0$ . Thus  $ps(n-i) = k_i + i + 1$ .

**Example 3.17.** For the primitive polynomial  $x^8 + x^6 + x^5 + x^4 + 1$ ,  $ps(1) = 0$  and  $ps(8) = 1$  by Theorem 3.1. By Theorem 3.2,  $ps(2) = 254$ . By Theorem 3.11,  $ps(5) = 4$ ,  $ps(6) = 3$  and  $ps(7) = 2$ . Since  $T^{48} + T^{50} = I$ ,  $ps(3) = 48$  by Theorem 3.16. Since  $T^{100} + T^{96} = I$ ,  $ps(4) = 101$  by Theorem 3.15.

The following theorem is the main theorem in this section.

**Theorem 3.18.** Let  $T$  be the companion matrix of the  $n$ -degree primitive polynomial  $f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_1x + 1$ . Let  $c_j = 1$  for some  $j$  ( $1 < j < n$ ). And let  $T^k + c_{j+1}T^{k+1} + \cdots + c_{n-1}T^{k+(n-j-1)} + T^{k+(n-j)} = I$ . Then  $ps(n-j+1) = k$ .

**Example 3.19.** Phase shifts for the primitive polynomial  $f(x) = x^{12} + x^{10} + x^9 + x^8 + x^6 + x^2 + 1$  are in Table 1.

**Table 1.** Phase shifts for  $f(x)$ 

Phase shifts	Theorem name
$ps(1) = 2^{12} - 1$	Theorem 3.1
$ps(2) = 4094$	Theorem 3.2
$ps(3) = 1156$	Theorem 3.18
$ps(4) = 2511$	Theorem 3.18
$ps(5) = 935$	Theorem 3.18
$ps(6) = 934$	Theorem 3.11
$ps(7) = 1162$	Theorem 3.11
$ps(8) = 1161$	Theorem 3.11
$ps(9) = 1160$	Theorem 3.11
$ps(10) = 1159$	Theorem 3.15
$ps(11) = 2$	Theorem 3.10
$ps(12) = 1$	Theorem 3.1

## 4 Algorithms to compute Phase shifts

Using the theorems outlined above, we give an algorithm to find the phase shifts of the sequences generated by companion matrices of primitive polynomials.

### Algorithm FindPhaseShiftsOfLFSM

**Input:**  $n$ , The 1st column  $(c_{n-1}, c_{n-2}, \dots, c_1, 1)$  of the companion matrix  $T$  of an  $n$ -degree primitive polynomial  $f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \dots + c_1x + 1$ .

**Output:** The phase shifts  $ps(i)(1 \leq i \leq n)$  of the  $i$ th column of  $S$ .

BEGIN

$ps(1) \leftarrow 2^n - 1$

$ps(n) \leftarrow 1$

count  $\leftarrow 0$

**Step 1.** Find  $m = \min\{j | c_j = 1, 1 \leq j < n\}$ ;

**Step 2. for**  $h \leftarrow 1$  **to**  $m - 1$

do {

$ps(n - h) \leftarrow h + 1$

count  $\leftarrow$  count + 1

}

if count  $> n - 3$  **then** Stop.

**Step 3.**  $ps(n - m) \leftarrow k$  subject to  $T^k + T^{k-m} = I$

count  $\leftarrow$  count + 1

if count  $> n - 3$  **then** Stop.

**Step 4.** Find  $M = \max\{j | c_j = 1, m \leq j \leq n - 1\}$ ;

**Step 5. for**  $l \leftarrow n - 1$  **downto**  $M + 1$

do {

$ps(n - l + 1) \leftarrow ps(n - l) - 1$

**Table 2.** Phase shifts for  $f(x)$   
(In this table, 86540 stands for the polynomial  $x^8 + x^6 + x^5 + x^4 + 1$ .)

$n$	$f(x)$	phase shifts
8	86540	0,254,48,101,4,3,2,1
8	87530	0,242,241,69,68,3,2,1
8	8765420	0,121,179,108,246,245,2,1
9	94310	0,510,509,508,507,24,317,316,1
9	9643210	0,510,509,275,274,356,331,460,1
9	986543210	0,107,106,92,459,325,248,109,1
10	10,4,3,1,0	0,1022,1021,1020,1019,1018,966,533,532,1
10	10,6,5,3,2,1,0	0,1022,1021,1020,769,326,325,856,450,1
10	10,7,6,5,4,3,2,1,0	0,1022,1021,479,527,315,292,45,439,1
11	11,10,8,1,0	0,308,309,316,315,314,313,312,311,310,1
12	12,10,2,1,0	0,4094,639,640,641,642,643,644,645,646,2369,1
17	17,3,0	0, $2^{17} - 2, 2^{17} - 3, \dots, 2^{17} - 14, 3, 2, 1$

```

        count ← count + 1
    }
    if count >  $n - 3$  then Stop.
Step 6. while count  $\leq n - 3$ ,
    do {
        if  $C_M = 1$ , then
             $\{ps(n - M + 1) = k$ 
            subject to  $T^k + c_{M+1}T^{k+1} + \dots + c_{n-1}T^{k+(n-M-1)} + T^{k+(n-M)} = I$ 
            count ← count + 1
             $M \leftarrow M - 1$ 
        }
        else
        {
             $ps(n - M + 1) \leftarrow ps(n - M) - 1$ 
             $M \leftarrow M - 1$ 
        }
    }
END

```

Phase shifts with respect to the 1st column of the given primitive polynomial are in Table 2.

## 5 Conclusion

In this paper, we investigated the phase shifts of the sequences generated by companion matrices of primitive polynomials. And we proposed an algorithm for finding phase shifts of the sequences.

## References

1. P.H. Bardell, Calculating the effects of linear dependencies on  $m$ -sequences used as test stimuli, IEEE Trans. CAD of Integrated Circuits and Systems, **11** (1992) 83-86
2. P.H. Bardell, W.H. McAnney and J. Savir, Built-In test for VLSI: pseudorandom techniques, John Wiley & Sons, 1987
3. C.L. Chen, Linear dependencies in linear feedback shift registers, IEEE Trans. Comput., **32** (1986) 1086-1088
4. J. Kakade and D. Kagaris, Phase shifts and linear dependencies, in Proc. IEEE Int. Symp. Circuits Syst., (2006) 1595-1598
5. A. Lempel, Analysis and synthesis of polynomials and sequences over  $GF(2)$ , IEEE Trans. Inform. Theory, **IT-17** (1971) 297-303
6. J. Rajski and J. Tyszer, On linear dependencies in subspaces of LFSR-generated sequences, IEEE Trans. Computers, **45** (1996) 1212-1221
7. P.H. Bardell, Analysis of cellular automata used as pseudorandom pattern generators, Proc. IEEE int. Test. Conf. (1990) 762-767
8. A.K. Das and P.P. Chaudhuri, Vector space theoretic analysis of additive cellular automata and its application for pseudo-exhaustive test pattern generation, IEEE Trans. Comput. **42** (1993) 340-352
9. P. Sarkar, Computing Shifts in 90/150 cellular automata sequences, Finite Fields Their Appl. **42** (2003) 340-352
10. P. Sarkar, The filter-combiner model for memoryless synchronous stream ciphers, in Proceedings of Crypto 2002, LNCS, **2442** (2002) 533-548
11. S.J. Cho, U.S. Choi, Y.H. Hwang, Y.S. Pyo, H.D. Kim and S.H. Heo, Computing phase shifts of maximum-length 90/150 cellular automata sequences, LNCS, **3305** (2004) 31-39
12. S.J. Cho, U.S. Choi, H.D. Kim, Y.H. Hwang, J.G. Kim and S.H. Heo, New synthesis of one-dimensional 90/150 linear hybrid group cellular automata, IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., **26** (2007) 1720-1724
13. D. Kagaris, F. Makedon and S. Tragoudas, A method for pseudoexhaustive test pattern generation, IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., **13** (1994) 1170-1178
14. D. Kagaris and S. Tragoudas, Avoiding linear dependencies in LFSR test pattern generators, J. Electron. Testing: Theory Applicat. **6** (1995) 229-241
15. D. Kagaris, Linear dependencies in extended LFSMs, IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., **21** (2002) 852-859
16. D. Kagaris, A unified method for phase shifter computation, ACM Trans. Des. Autom. Electron. Syst., **10** (2005) 157-167
17. J. Kakade and D. Kagaris, Minimization of linear dependencies through the use of phase shifters, IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., **26** (2007) 1877-1882
18. S.W. Golomb, Shift register sequences, Holden Day, 1967
19. R. Lidl and H. Niederreiter, Finite fields, Cambridge University Press, 1997

# Introducing Service Schemes and Systems Organization in the Theory of Interactive Computation

Antônio Carlos da Rocha Costa and Graçaliz Pereira Dimuro

Programa de Pós-Graduação em Informática, ESIN, Universidade Católica de Pelotas, Rua Felix da Cunha 412, 96010-000 Pelotas, Brazil.  
e-mail: {rocha,liz}@ucpel.tche.br

**Abstract.** This paper introduces the notion of *service scheme* to allow for a formal approach to the study of the realizability of *services* by *interactive systems*. It shows how the organization-theoretic conceptual framework required by the notion of system service can be formally introduced, and shows examples of its application that allow the identification of some basic service schemes, that seem to be common to every interactive system. Finally, it briefly considers the conceptual framework of the Theory of Interactive Computation proposing that its central problem is the formal characterization of the *realizability* of service schemes, that is, the identification of the class of all service schemes realizable by interactive computational means.

**Key words:** Interactive computation, service schemes, systems organization.

## 1 Introduction

This paper aims to contribute to the foundations of the Theory of Interactive Computation (TIC) [1, 2] through the formalization of the notion of service. *Services* are understood in their usual intuitive sense, but are introduced as an organizational notion distinct from other notions such as the notion of *organizational role* that a component may play in an interactive system (IS), or of the notion of a *behavior* that a component may perform in such system.

The paper adopts the common terminology of the area of Organization of Multiagent Systems (MAS) [3–7] to express the concepts necessary for the formalization of the notion of service, but it is not essentially dependent on the conceptual framework of that area: an agent, as considered here, is simply a system component, i.e., a computing element integrated in a computing system where organizational features (roles, services, behaviors, relationships, interactions, etc.) are present. No other implication on the concept of agent is assumed.

We have refrained from adopting any of the current formalisms for describing interactive systems, like the various versions of process algebras [8, 9], or equivalent formalisms. Instead, we have adopted an intuitive trace-based framework for the presentation of our concepts, a choice that has led us to conceive

system processes (i.e., organized sets of actions performed by sets of agents) as sets of *interleaved action streams* (IAS). Wherever possible, we have remained as much informal as we could, in order not to overwhelm the presentation of the conceptual framework with technical details that are not mandatory at the current stage of our work. Of course, this informality determines that the results presented in this paper can not be seen as completely accomplished.

The two main results of the paper are: (1) a definition of the notion of *service scheme*, a formal construct that contributes to methodology by allowing for suitable formal organizational specifications of IS; (2) the suggestion (of a foundational character) that the true object of the TIC is the mathematical determination of the class of *computable service schemes*, i.e., *the class of services that can be effectively realized by interactive computational means*.

## 2 Services and the Theory of Interactive Computation

*Interactive computation* (IC) is computation where data is continually exchanged between the computing system and its environment. In an IC, the stream of input information does not need to be completely specified before the computation starts, and since the specification of the content of the input information may happen along with the performance of the computation, such specification may depend on the very stream of output information that it helps to generate.

Circular constructions like that, making the specification of the system's input depend on the system's own output, traditionally called "feedback loop" in the context of Cybernetics, is neatly out of the scope of the classical models of algorithmic computation, such as (any non-extended variant of) the Turing machine model (see [10, 2], for further elaboration of this idea).

Peter Wegner and Dina Golding ([1] and other papers by them, cited there) have long claimed that interaction takes computing power beyond the level attained by algorithms (i.e., computing based on the classical Turing machine model). They have stressed the importance of interaction for the right conception of software structures, and emphasized that interactive computations are *service-oriented*, not question answering-oriented.

We have previously argued [2, 10, 11] that the conceptual framework needed for the explication of service-oriented computations should take as its foundation the general conceptual framework of the cybernetic-oriented theories of systems dynamics and organization. This requires taking into account both the concept of *behavior*, which Theoretical Computer Science has already formalized in its various theories of concurrent processes (e.g. [8, 9, 12]), and the concept of *role* played by a component within a system, which is currently being formalized following various alternative methodological approaches in the area of MAS (e.g., [4–7]). Besides that, however, that foundational proposal also requires the formalization of the concept of *service* (or, *systemic function*) performed by a system component to other system components and/or to the system as a whole, such formalization demanding that the concept of service be considered distinct from the other two concepts of role and behavior [10, 11].

In this paper we formalize the concept of service by introducing the formal construct of a *service scheme*, aiming to further the TIC in this respect.

On the other hand, and as a consequence of this preliminary formalization of the concept of service, the paper also addresses the foundational question: “What object is computed when an IC is carried on?” Such question, which got the definite answer of “a mathematical function”, in the Classical Theory of Computation, seems to have not yet been adequately settled in the case of the TIC. We offer it the answer: “a service scheme”.

### 3 Interleaved Action Streams

In the following, let  $T = (t_0, t_1, \dots)$  be a discrete linear time structure and  $\Delta = [t_i, t_j] \subseteq T$ , with  $t_i \leq t_j$ , be a time interval. We take a trace-based approach to computational behavior, and formally model system processes as (sets of) interleaved action streams (IAS):

**Definition 1.** Let  $Acts$  be a set of actions that can be performed during  $\Delta$ . An action stream performed during  $\Delta$  is a partial function  $as^\Delta : \Delta \rightarrow Acts$ . For any  $t \in \Delta$ , if  $as^\Delta(t)$  is defined, we write  $as^\Delta(t)\downarrow$ , and  $as^\Delta(t)\uparrow$  if  $as^\Delta(t)$  is undefined.

Let  $Ag$  be a set of agents, and  $ag \in Ag$ . Let  $Acts_{ag}$  be the set of actions that agent  $ag$  may possibly perform. We denote by  $as_{ag}^\Delta$  any action stream that agent  $ag$  may perform during the time interval  $\Delta$ , employing actions of  $Acts_{ag}$ .

**Definition 2.** Let  $a_1, a_2 \in Ag$  be two agents. Let  $as_{a_1}^\Delta$  and  $as_{a_2}^\Delta$  be the two action streams performed by the agents  $a_1$  and  $a_2$  during the time interval  $\Delta$ , so that for any  $t \in \Delta$ ,  $as_{a_1}^\Delta(t)\downarrow$  implies that  $as_{a_2}^\Delta(t)\uparrow$ , and  $as_{a_2}^\Delta(t)\downarrow$  implies that  $as_{a_1}^\Delta(t)\uparrow$ . The interleaved action stream of agents  $a_1$  and  $a_2$  during  $\Delta$  is defined as the function  $ias_{a_1, a_2}^\Delta : \Delta \rightarrow T$ , given by

$$ias_{a_1, a_2}^\Delta(t) = \begin{cases} as_{a_1}^\Delta(t) & \text{if } as_{a_1}^\Delta(t)\downarrow, \\ as_{a_2}^\Delta(t) & \text{if } as_{a_2}^\Delta(t)\downarrow, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

In the following, we will often let implicit the reference to the interval  $\Delta$ .

An *alternating action stream* established by two agents  $a_1$  and  $a_2$  is a kind of interleaved action stream  $ias_{a_1, a_2}$  where actions of  $a_1$  and  $a_2$  alternate:

**Definition 3.** An alternating action stream established by two agents  $a_1$  and  $a_2$  is an interleaved action stream  $ias_{a_1, a_2}^\Delta(t)$  where, for any sub-interval  $[t_i, t_k] \subseteq \Delta$  such that  $ias_{a_1, a_2}(t_i) \downarrow$ ,  $ias_{a_1, a_2}(t_k) \downarrow$  and, for any  $t_j$  with  $t_i < t_j < t_k$ ,  $ias_{a_1, a_2}(t_j)\uparrow$ :

- if  $ias_{a_1, a_2}(t_i) = as_{a_1}(t_i)$  then  $ias_{a_1, a_2}(t_k) = as_{a_2}(t_k)$ ;
- if  $ias_{a_1, a_2}(t_i) = as_{a_2}(t_i)$  then  $ias_{a_1, a_2}(t_k) = as_{a_1}(t_k)$ .

The concept of interleaved action stream can be extended in the obvious way to *sets of agents*  $\{a_1, a_2, \dots, a_n\}$ , where  $n \geq 2$ . The concept of alternating action stream can then be correspondingly extended in a natural way to the concept of *cyclicly interleaved action stream*. The various sequences of actions performed during the intervals  $[t_i, t_k]$  which structure the alternation of actions are informally called the *cycles* that characterize the action stream.

In an interleaved action stream  $ias_{a_1, \dots, a_n}$  performed by a set of agents  $\{a_1, \dots, a_n\}$ , the action  $\alpha$  performed at time  $t$  by agent  $a_i$  is denoted by  $\langle a_i : \alpha \rangle^t$ , or by  $\langle a_i : \alpha \rangle$ , when we are not interested in the time at which the action occurred. A segment (with length  $k$ ) of an interleaved action stream  $ias_{a_1, \dots, a_n}$  can be denoted by  $[\langle a_{i_1} : t_1 \rangle \alpha_1, \dots, \langle a_{i_k} : t_k \rangle \alpha_k]$ , or simply by  $[\langle a_{i_1} : \rangle \alpha_1, \dots, \langle a_{i_k} : \rangle \alpha_k]$ .

## 4 Interactions Modelled with Interleaved Action Streams

Let's model the behavior of a simple client-server system with the help of IAS. We assume that the system operates through cycles of requests and service performances, each cycle servicing a single request from a single client, the server serving possibly many different clients in a first-come first-served basis. Also, every performance of the service is initiated by a request from a client.

To structure the notion of *request-service cycle*, a relation is needed between request actions and the corresponding response actions. We capture this with the relation  $\triangleright \subseteq \bigcup_i Acts_{cl_i} \times Acts_s$ , called an *activation relation*, relating each possible response of the server to the request by a client that may initiated it.

*Example 1.* Consider a server  $s$  and a set of clients  $Cl = \{cl_1, cl_2, \dots, cl_n\}$ . Let  $Acts_s$  be the possible actions of the server, and  $Acts_{cl_i}$  the possible actions of the client  $cl_i$ . Let  $ias_{s, Cl}$  be any interleaved action stream involving the server  $s$  and the clients in the set  $Cl$  during the time interval  $\Delta$ . Then, the *client-service cycle* occurs repeatedly at some sub-intervals  $[t_i, t_k] \subseteq \Delta$  as the server performs services for its clients. It goes on according to the following rules:

- for any time interval  $[t_i, t_k]$  such that  $ias_{s, Cl}(t_i) \downarrow, ias_{s, Cl}(t_k) \downarrow$  and, for any  $t_i < t_j < t_k$ ,  $ias_{s, Cl}(t_j) \uparrow$ , then: (1) if  $ias_{s, Cl}(t_i) = as_{cl}(t_i)$ , for some  $cl \in Cl$ , then  $ias_{s, Cl}(t_k) = as_s(t_k)$ ; (2) if  $ias_{s, Cl}(t_i) = as_s(t_i)$ , then  $ias_{s, Cl}(t_k) = as_{cl}(t_k)$ , for some  $cl \in Cl$ ;
- for any time interval  $[t_i, t_k]$  in which it holds that  $ias_{s, cl_1, \dots, cl_n}(t_i) = as_{cl}(t_i)$  and  $ias_{s, cl_1, \dots, cl_n}(t_k) = as_s(t_k)$ , with no defined intervening action between  $t_i$  and  $t_k$ , it happens that  $as_{cl}(t_i) \triangleright as_s(t_k)$ .

*Example 2.* Let  $db$  be a database server and *supplier* and *producer* be two of its clients, and the set of all possible service performances be specified as follows ( $Actv$  is the activation relation):

- $Acts_{db} = \{confirm, reply\}; Acts_{sup} = \{update\}; Acts_{prod} = \{query\};$
- $Actv_{db, sup, prod} = \{update \triangleright confirm, query \triangleright reply\};$
- if at time  $t$  it holds that  $ias_{db, sup, prod}(t) \downarrow$  and  $ias_{db, sup, prod}(t) = as_{sup}(t) = update$  then at the next time  $t'$  where  $ias_{db, sup, prod}(t') \downarrow$  it should also hold that  $ias_{db, sup, prod}(t') = as_{db}(t') = confirm$ ;

– if at time  $t$  it holds that  $\text{ias}_{db,sup,prod}(t) \downarrow$  and  $\text{ias}_{db,sup,prod}(t) = \text{as}_{prod}(t) = \text{query}$  then at the next time  $t'$  where  $\text{ias}_{db,sup,prod}(t') \downarrow$  it should also hold that  $\text{ias}_{db,sup,prod}(t') = \text{as}_{db}(t') = \text{reply}$ .

Then, for instance, the following are possible interactions between the agents:

- $[\langle sup : update \rangle, \langle db : confirm \rangle, \langle prod : query \rangle, \langle db : reply \rangle]$ , and
- $[\langle prod : query \rangle, \langle db : reply \rangle, \langle prod : query \rangle, \langle db : reply \rangle, \langle sup : update \rangle, \langle db : confirm \rangle, \langle prod : query \rangle, \langle db : reply \rangle]$ .

The activation relation between the actions of the server and the clients captures the interaction constraints imposed by the nature of the service being performed by the server. In fact, such activation relation can be taken as the basis of the formal notion of *service scheme*, as is done in the next section.

## 5 Interaction Constraints as Services Schemes

The concept of a *service scheme* as a constraint imposed on the interactions that may happen between a server and its clients is established here, in a tentative way, under the simplifying assumption that the server is able to serve just one client at a time. This allows us to abstract the set of clients of the service into one single *generalized client*, able to represent any client in such set, thus establishing the concept of *generalized interleaved action stream*.

**Definition 4.** Let  $a_1$  and  $a_2$  be two agents, the server and the (generalized) client. Let  $\text{Acts}_{a_1}$  and  $\text{Acts}_{a_2}$  be their respective set of possible actions. The service scheme performed by the agent  $a_1$  to the agent  $a_2$  during the time interval  $\Delta$  is given by the structure  $\text{Srv}_{a_1,a_2}^\Delta = (\text{Ias}_{a_1,a_2}^\Delta, \triangleright)$  where  $\triangleright \subseteq \text{Acts}_{a_2} \times \text{Acts}_{a_1}$  is an activation relation between the actions of  $a_1$  and  $a_2$ , and  $\text{Ias}_{a_1,a_2}^\Delta$  is the set of all possible alternating action streams  $\text{ias}_{a_1,a_2} : \Delta \rightarrow \text{Acts}_{a_1} \cup \text{Acts}_{a_2}$ , so that the following statements hold, for any  $\text{ias}_{a_1,a_2} \in \text{Ias}_{a_1,a_2}^\Delta$ :

- if  $t$  is the least time at which  $\text{ias}_{a_1,a_2}(t) \downarrow$  then  $\text{ias}_{a_1,a_2}(t) = \text{as}_{a_2}(t)$  (that is, any alternating action stream initiates with a client request);
- in any alternating action stream  $\text{ias}_{a_1,a_2} \in \text{Ias}_{a_1,a_2}^\Delta$ , if at  $t$  and  $t'$  there are  $\text{ias}_{a_1,a_2}(t) = \text{as}_{a_2}(t)$  and  $\text{ias}_{a_1,a_2}(t') = \text{as}_{a_1}(t')$ , with no intervening action between them, then the activation relation is respected,  $\text{as}_{a_2}(t) \triangleright \text{as}_{a_1}(t')$ , i.e., the server's response is coherent with the (generalized) client's request.

$\text{Ias}_{a_1,a_2}^\Delta$  is said to be the *generalized alternating action stream* underlying the service scheme  $\text{Srv}_{a_1,a_2}^\Delta$ . We say that the agent  $a_1$  performs the service scheme  $\text{Srv}_{a_1,a_2}^\Delta$  for  $a_2$ . Also, we say that the *interactive system* composed of the agents  $a_1$  and (all agents represented by)  $a_2$  realizes the service scheme  $\text{Srv}_{a_1,a_2}^\Delta$ .

**Definition 5.** Given the service scheme  $\text{Srv}_{a_1,a_2}^\Delta = (\text{Ias}_{a_1,a_2}^\Delta, \triangleright)$  performed by agent  $a_1$  to agent  $a_2$ , the interaction constraint on  $\text{Ias}_{a_1,a_2}^\Delta$  induced by the activation relation  $\triangleright$  is the relation  $\triangleright \subseteq \text{Acts}_{a_2} \times \text{Acts}_{a_1}$ , such that, for any  $\text{ias}_{a_1,a_2}^\Delta \in \text{Ias}_{a_1,a_2}^\Delta$  and  $t, t' \in \Delta$ , if  $t$  is such that  $\text{ias}_{a_1,a_2}^\Delta(t) = \text{as}_{a_2}^\Delta(t) = \alpha_2$  and  $t'$  is the

next time after  $t$  at which  $\text{ias}_{a_1,a_2}^\Delta(t') \downarrow$  such that  $\text{ias}_{a_1,a_2}^\Delta(t') = \text{as}_{a_1}^\Delta(t') = \alpha_1$  and  $\alpha_2 > \alpha_1$ , then  $\langle a_2 : \alpha_2 \rangle^t \triangleright \langle a_1 : \alpha_1 \rangle^{t'}$ . If times  $t$  and  $t'$  can be understood from the context, the interaction constraint on  $\text{Ias}_{s,\text{cl}}$  induced by the action  $\alpha_2$  of a client  $a_2$  and the action  $\alpha_1$  of the server  $a_1$  can be denoted by  $\langle a_2 : \alpha_2 \rangle \triangleright \langle a_1 : \alpha_1 \rangle$ .  $\text{Srv}_{a_1,a_2} = (\text{Ias}_{a_1,a_2}, \triangleright)$  can then be denoted by  $\text{Srv}_{a_1,a_2} = (\text{Ias}_{a_1,a_2}, \triangleright)$ .

**Definition 6.** Two interaction constraints  $\triangleright_1$  and  $\triangleright_2$  are said to be equivalent, denoted by  $\triangleright_1 \equiv \triangleright_2$ , if and only if the following condition holds: whenever  $a_2$  and  $a_4$  are clients and  $a_1$  and  $a_3$  are servers, it holds that  $\langle a_2 : \alpha_2 \rangle \triangleright_1 \langle a_1 : \alpha_1 \rangle$  and  $\langle a_4 : \alpha_2 \rangle \triangleright_2 \langle a_3 : \alpha_3 \rangle$  if and only if  $\alpha_1 = \alpha_3$ .

**Definition 7.** Service schemes  $\text{Srv}_{a_1,a_2} = (\text{Ias}_{a_1,a_2}, \triangleright_{a_1,a_2})$  and  $\text{Srv}_{a_3,a_4} = (\text{Ias}_{a_3,a_4}, \triangleright_{a_3,a_4})$  are said to be equivalent, denoted by  $\text{Srv}_{a_1,a_2} \equiv \text{Srv}_{a_3,a_4}$ , if and only if: (1) the repertoires of actions of their corresponding agents are equal:  $\text{rep}(a_1) = \text{rep}(a_3)$  and  $\text{rep}(a_2) = \text{rep}(a_4)$ ; (2) their corresponding interaction constraints are equivalent:  $\triangleright_{a_1,a_2} \equiv \triangleright_{a_3,a_4}$ .

## 6 Composition of Service Schemes

Here, we introduce a notation to represent compositions of service schemes.

**Definition 8.** The sequential composition of two services  $\text{Srv}_1$  and  $\text{Srv}_2$  is the service scheme denoted by  $\text{Srv}_1 \Rightarrow \text{Srv}_2$ . An interleaved action stream  $\text{ias}$  satisfies  $\text{Srv}_1 \Rightarrow \text{Srv}_2$  during an interval  $[t_i, t_k]$  if and only if there is a time  $t_j$ , with  $t_i < t_j < t_k$ , such that  $\text{ias}$  satisfies  $\text{Srv}_1$  in  $[t_i, t_j]$  and  $\text{Srv}_2$  in  $[t_j, t_k]$ .

**Definition 9.** The  $n$ -fold repetition of a service  $\text{Srv}$  (for  $n \geq 1$ ) is the service scheme denoted by  $\text{Srv}^n$ . An interleaved action stream  $\text{ias}$  satisfies  $\text{Srv}^n$  during an interval  $[t_i, t_k]$  if and only if there is an  $n$ -fold partition of  $[t_i, t_k]$ , in the form  $[t_{j_0}, t_{j_1}], [t_{j_1}, t_{j_2}], \dots, [t_{j_{n-1}}, t_{j_n}]$ , where  $t_{j_0} = t_i$  and  $t_{j_n} = t_k$ , such that  $\text{ias}$  satisfies  $\text{Srv}$  in each interval  $[t_{j_i}, t_{j_{i+1}}]$ .

Of course,  $\text{Srv}^n = \text{Srv} \Rightarrow \dots \Rightarrow \text{Srv}$  ( $n$  times). Also, we denote by  $\text{Srv}^+$  an undefined number of repetitions of  $\text{Srv}$ , but including at least one repetition.

**Definition 10.** The non-deterministic choice between two service schemes  $\text{Srv}_1$  and  $\text{Srv}_2$  is denoted by  $\text{Srv}_1 \sqcup \text{Srv}_2$ . An interleaved action scheme  $\text{ias}$  satisfies the service scheme  $\text{Srv}_1 \sqcup \text{Srv}_2$  during an interval  $[t_i, t_k]$  if and only if it satisfies service scheme  $\text{Srv}_1$  or it satisfies service scheme  $\text{Srv}_2$ .

## 7 Basic Service Schemes

*Example 3.* Let  $\text{prod}$  be the producer agent and  $\text{con}$  represent any of its consumer agents. Consider that  $\text{Acts}_{\text{prod}} = \{\text{produce}, \text{accept\_delivery\_request}, \text{deliver}, \text{deny\_delivery\_request}, \text{delivery\_denied}\}$  and  $\text{Acts}_{\text{cons}} = \{\text{delivery\_request}, \text{receive}, \text{consume}\}$ . Then, the service is characterized by the interaction constraint:  $[\langle \text{cons} : \text{delivery\_request} \rangle \triangleright [\langle \text{prod} : \text{deliver} \rangle \sqcup \langle \text{prod} : \text{delivery\_denied} \rangle]]^+$ .

We note that characterizing a service is not the same as characterizing the behaviors of the server and its clients. It's just characterizing how the server and the client interact. Thus, in Ex. 3, the service performed by the producer is the delivery of the product to the client, not the full working necessary for the production of the product (which includes actions like *produce* and *accept\_delivery\_request*). The latter is the behavior that an agent has to perform in order to fulfill the *role* of producer. A similar conclusion can be drawn with respect to the consumer.

*Example 4.* Now, we model both the service scheme performed by an *intermediary* agent *int* that delivers products to a generalized *consumer* agent *cl* (the delivery of the products that it will consume) and the service performed by the *producer* agent *prod* to the intermediary agent *int* (the delivery of the products that the latter will deliver to the consumers). Let  $Acts_{cons} = \{delivery\_request, receive, consume\}$ ,  $Acts_{prod} = \{produce, accept\_delivery\_request, deliver, deny\_delivery\_request, delivery\_denied\}$ ,  $Acts_{int} = \{delivery\_request, receive, accept\_delivery\_request, deliver, deny\_delivery\_request, delivery\_denied\}$ . The service schemes may be characterized by  $[(int : delivery\_request) \triangleright [(prod : deliver) \sqcup (prod : delivery\_denied)]]^+$ , and  $[(cons : delivery\_request) \triangleright [(int : deliver) \sqcup (int : delivery\_denied)]]^+$ .

The interaction constraint that characterizes the service performed by the producer to the consumer in Ex. 3 has the same structure as the interaction constraint which characterizes the service performed by the producer to the intermediary, and that of the intermediary to the consumer, in Ex. 4. Thus:

**Definition 11.** Two agents  $a_1$  (the *deliverer*) and  $a_2$  (the *generalized receiver*), realize the delivery service scheme if their generalized alternating action stream  $Ias_{a_1, a_2}$  satisfies the interaction constraint:

$$[(a_2 : delivery\_request) \triangleright (a_1 : deliver) \sqcup (a_1 : delivery\_denied)]^+.$$

Let's now examine the case of agents that receive objects delivered to them by other agents. Let *rec* be an agent that is able to receive objects sent to it by agent *sen*. Let  $Acts_{rec} = \{accept\_reception\_request, receive, deny\_reception\_request, reception\_denied\}$  and  $Acts_{sen} = \{reception\_request, deliver\}$ . The service scheme realized by *rec* and *sen* can then be characterized by:

$$[(sen : reception\_request) \triangleright [(rec : receive) \sqcup (rec : reception\_denied)]]^+.$$

This motivates the general definition:

**Definition 12.** Two agents,  $a_1$  (the *receiver*) and  $a_2$  (the *generalized sender*), realize the reception service scheme if their generalized alternating action stream  $Ias_{a_1, a_2}$  satisfies the interaction constraint:

$$[(a_2 : reception\_request) \triangleright [(a_1 : receive) \sqcup (a_2 : reception\_denied)]]^+.$$

We note that in the *reception service scheme* the initiative of realization of the service scheme is taken by the agent that delivers the object, while in the *delivery service scheme* the initiative is with the agent that receives the object. This is so because in any service scheme the initiative towards the realization of the scheme is always taken by a client of the service, not the server.

*Example 5.* Consider the case of agents that store objects for other agents, so that objects can be sent for storage at some time and got back later, when requested. Let  $sto$  be an agent that is able to store objects for a generalized client  $cl$ . Let  $Acts_{sto} = \{accept\_reception\_request, receive, accept\_delivery\_request, deliver, deny\_reception\_request, reception\_denied\}$  and  $Acts_{cl} = \{reception\_request, deliver, delivery\_request, receive\}$ . Then, the service performed by  $sto$  to  $cl$  is characterized by (using exponents as repetition operators, with  $m \geq n$ ):

$$[[\langle cl : reception\_request \rangle \triangleright [\langle sto : receive \rangle \sqcup \langle sto : reception\_denied \rangle]]^m \Rightarrow [\langle cl : delivery\_request \rangle \triangleright \langle sto : deliver \rangle]^n]^+$$

This motivates the general definition:

**Definition 13.** Two agents,  $a_1$  (the store) and  $a_2$  (the accessor), realize the storage service scheme if their generalized alternating action stream satisfies the service scheme, for  $m \geq n$ :

$$[[\langle a_2 : reception\_request \rangle \triangleright \langle a_1 : receive \rangle]]^m \Rightarrow [[\langle a_2 : delivery\_request \rangle \triangleright \langle a_1 : deliver \rangle]]^n]^+$$

The following proposition illustrates the equivalence of service schemes:

**Proposition 1.** Let

$$1) Storage_{a_1,a_2} = (Ias_{a_1,a_2}, \triangleright_{a_1,a_2}^{Sto}), \text{ where } \triangleright_{a_1,a_2}^{Sto} \text{ is given by (for } m \geq n\text{):}$$

$$[[\langle a_2 : reception\_request \rangle \triangleright \langle a_1 : receive \rangle]]^m \Rightarrow [[\langle a_2 : delivery\_request \rangle \triangleright \langle a_1 : deliver \rangle]]^n]^+$$

$$2) Reception_{a_1,a_2} = (Ias_{a_1,a_2}, \triangleright_{a_1,a_2}^{Rec}), \text{ where } \triangleright_{a_1,a_2}^{Rec} \text{ is given by:}$$

$$\langle a_2 : reception\_request \rangle \triangleright \langle a_1 : receive \rangle$$

$$3) Delivery_{a_1,a_2} = (Ias_{a_1,a_2}, \triangleright_{a_1,a_2}^{Del}), \text{ where } \triangleright_{a_1,a_2}^{Del} \text{ is given by:}$$

$$\langle a_2 : delivery\_request \rangle \triangleright \langle a_1 : deliver \rangle$$

Then, one notice that  $\triangleright_{a_1,a_2}^{Sto} \equiv [[\triangleright_{a_1,a_2}^{Rec}]^m \Rightarrow [\triangleright_{a_1,a_2}^{Del}]^n]^+$ , and so:

$$Storage_{a_1,a_2} \equiv [[Reception_{a_1,a_2}]^m \Rightarrow [Delivery_{a_1,a_2}]^n]^+.$$

*Example 6.*  $Lending_{a_1,a_2} = [Delivery_{a_1,a_2} \Rightarrow Reception_{a_1,a_2}]^+$ , the lending service scheme, is a compound service scheme where agent  $a_1$  (the lender) lends an object to the agent  $a_2$  (the borrower), which later returns the lent object.

With the help of the above examples, we claim that the reception and delivery service schemes are two of the basic service schemes that can be found in any interactive computing system.

## 8 Organizing Interactive Systems with Service Schemes

As stated in the Introduction, and stressed in Sect. 7, we think that interactive computations cannot be fully explained by simply adding the concept of interaction to the conceptual framework usually employed in the understanding of classical computations. As established in [10] and [11], and further elaborated in [2], it seems to us that the explication of interactive computing requires the framing of interactive systems in a more complex conceptual framework, where the concept of organization [2, 10, 13] should have a preeminent role.

Organization is a concept highly worked out in areas like Biology and Sociology. It involves many component concepts, such as:

1. *organizational role* that, as explained in Sect. 7, concerns the full behavior that an agent may have to perform when participating in an adequate way in the organization;
2. *organizational function* [11], which can be equated to the notion of *service* that we have been using here;
3. *organizational process* [3], which can be equated to the notion of generalized IAS that was formally introduced here;
4. *organizational link*, which concerns the way organizational roles interact [3], and which has a strong relationship to the notion of *service scheme*, also formally introduced here.

Disentangling the concept of *service* from the concepts of *role* and *behavior* should help the methodology of interactive systems design, as the different concerns of those organizational concepts should allow for their different methodological imports. It also allows the advancing of issues related to the *structural dynamics* of systems, and the consequent possibility of introducing into the dynamics of the system a notion of *system development* [2, 3, 10, 13, 14].

## 9 Related Work

A connection can be established between the work presented here and WSCL (Web Service Conversation Language), a language defined by the W3C (World Wide Web Consortium) for the abstract specification of *service interfaces*, whose definition is available at <http://www.w3.org/TR/wscl10/>.

In WSCL, a service interface is a *conversation* (i.e., an exchange of XML documents) that the server and its clients should realize to get the service performed. A conversation is essentially defined as a set of two types of elements: *interactions* (i.e., operations of exchange of documents) and *transactions* (i.e., pairwise temporal orderings of interactions). A conversation specifies all the possible sequences of document exchanges that allow for the successful performance of the service (as well as the sequences that lead to failure in such performance).

The way to connect WSCL and service schemes seems to be immediate, by making service schemes serve as statements of formal behavioral requirements for WSCL conversations: an *interaction constraint* of a service scheme states a behavioral requirement for a WSCL *transaction*, and a *service composition* of a service scheme states a behavioral requirement for a WSCL *interaction*.

## 10 Conclusion: On the Object of the TIC

With the formalization of the concept of service scheme, this paper also aims to contribute to a foundational issue: it proposes that the object computed by an interactive system is, simply, a service scheme. Computability, in the context of interactive computations, is not a property of mathematical functions or relations, as it was in the context of classical computations, but is a property of schemes of system services: *computability in the context of interactive computations is synonymous of realizability of service schemes*.

The fundamental question of the TIC seems to be, thus: *What is the formal characterization of the class of all service schemes that are effectively realizable by interactive systems?*

**Acknowledgements.** This work was supported by FAPERGS and CNPq. We are grateful to the referees for the comments that helped us to improve the paper.

## References

1. Goldin, D., Smolka, S., Wegner, P., eds.: *Interactive Computation: The New Paradigm*. Springer-Verlag, New York (2006)
2. Costa, A.C.R., Dimuro, G.P.: Interactive computation: Stepping stone in the pathway from classical to developmental computation. *ENTCS* **141**(5) (2005) 5–31
3. Demazeau, Y., Costa, A.C.R.: Populations and organizations in open multi-agent systems. In: Proc. 1st. Symp. Parallel and Distributed AI, Hyderabad, India (1996)
4. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: the Gaia methodology. *ACM Transactions on Software Engineering and Methodology* **12**(3) (2003) 317–370
5. Hübner, J.F., Sichman, J.S., Boissier, O.: A model for the structural, functional, and deontic specification of organizations in multiagent systems. In Bittencourt, G., Ramalho, G.L., eds.: *Advances in Artificial Intelligence: 16th Braz. Symp. on Artificial Intelligence*. Volume 2507 of LNAI., Berlin, Springer (2002) 118–128
6. Dignum, V., Vázquez-Salceda, J., Dignum, F.: Omni: Introducing social structure, norms and ontologies into agent organizations. In Bordim, R.H., Dastani, M., Dix, J., Seghrouchni, A.E.F., eds.: *Programming Multi-Agent Systems: 2nd. Intl. Work. ProMAS 2004*. Volume 3346 of LNCS., New York, Springer (2004) 181–198
7. Ferber, J., Gutknecht, O., Michel, F.: From agents to organizations: an organizational view of multi-agent systems. In Giorgini, P., Müller, J.P., Odell, J., eds.: *Agent-Oriented Software Engineering IV*. Volume 2935 of LNCS., Berlin, Springer (2004) 214–230
8. Milner, R.: *Communication and Concurrency*. Prentice-Hall, New Jersey (1989)
9. Hoare, C.A.R.: *Communicating Sequential Processes*. Prentice-Hall, NJ (1985)
10. Costa, A.C.R.: Machine Intelligence: sketch of a constructive approach. PhD thesis, CPGCC/UFRGS, Porto Alegre (1993) (in Portuguese).
11. Costa, A.C.R., Castilho, J.M.V., Claudio, D.M.: Toward a constructive notion of functionality. *Cybernetics and Systems* **26**(4) (1995) 443–480
12. Reisig, W.: *Petri Nets, An Introduction*. Springer-Verlag, Berlin (1985)
13. Costa, A.C.R., Dimuro, G.P.: Semantical concepts for a formal structural dynamics of situated multiagent systems. In Sichman, J., Noriega, P., Padget, J., Ossowski, S., eds.: *Coordination, Organizations, Institutions, and Norms in Agent Systems III*. Number 4870 in LNAI, Berlin (2008) 139–154
14. Dimuro, G.P., Costa, A.C.R.: Toward a domain-theoretic model of developmental machines. In Cooper, S.B., Kent, T.F., Löwe, B., Sorbi, A., eds.: *CiE 2007: Computation and Logic in the Real World*. Quaderni del Dipartimento di Scienze Matematiche e Informatiche Roberto Magari of the Univ. of Siena (2007) 114–122

# Online-division with Periodic Rational Numbers

G. de Miguel Casado<sup>1\*</sup>, J.M. García Chamizo<sup>2</sup>, and H. Mora Mora<sup>2</sup>

<sup>1</sup> GISED, University of Zaragoza, Spain

{gmiguel}@unizar.es

<sup>2</sup> I2RC-SPALab, University of Alicante, Spain

{juanma,hmora}@dtic.ua.es

**Abstract.** This paper discusses different approaches for exact rational arithmetic and proposes a novel representation for periodic rational numbers with double mantissa (fixed and periodic) based on signed-digit arithmetic. The representation is proposed under the scope of Type-2 Theory of Effectivity (TTE) and the extension of an online-arithmetic algorithm for division is analyzed.

**Key words:** computer arithmetic algorithms; Type-2 Theory of Effectivity

## 1 Introduction

Data codification formats in digital computers define the features of the numeric sets and the arithmetic operations which support higher abstraction level operations required by computer applications [19]. In this context, positional fractional codifications provide a direct way to express rational numbers by means of both integer and fractional parts. One of the most representative formats is the floating point specification IEEE754, which has been adopted as standard in most of general purpose computer systems and is currently under revision. After some years, the lack of reliability of this standard for scientific computing applications [16, 20] has motivated the development of alternative methods for numerical data codification and operation [2, 10]. Among the whole research in the topic, some researchers in computer arithmetics have focused their efforts in symbolic computing with algebraic expressions [17]. This approach has led to proposals in which rational numbers are represented by means of fractions [13, 14]. However, its main drawback is the fact that there are no low complexity algorithms to obtain irreducible fractions (Euclid's Greatest Common Divisor Algorithm) and therefore the codification redundancy cannot be easily restrained [4]. In addition, some operators such as comparators, outline a high complexity when compared with those based on fractional positional notations.

The continuous fractions approach also offers an exact representation method for rational numbers. In this case, the codification of numbers is performed by successive fractions [5]. However, early hardware designs [17] outlined a high

---

\* The authors thank the Spanish Education Ministry for financial support (Project TIN2005-08832-C03-02)

complexity of the arithmetic operations involved [21]. As it happened in symbolic computing approaches, if a numeric result in fractional positional notation is required, some additional operations have to be performed and then, imprecisions in expression translations may appear. Another interesting proposal for error-free codification of rational numbers is based on the explicit representation of the periodic development of fractional numbers [12]. Nevertheless, as this research was limited to the theoretical formulation and no suitable procedures and architectures were developed at that time, no interest was shown by the scientific community.

The online arithmetic approach, based on signed digit arithmetic, deals with the hardware implementation of digit-serial left-to-right (online or Most Significant Bit First) arithmetic operators for signed digit numbers [9]. As the operation dynamics resembles that of the Turing Machine model, a conceptual convergence with Type-2 Theory of Effectivity (TTE) can be realized [22, 6]. Under this scope, partial implementation of Turing Machines can be done (according to the limited memory resources) and therefore, hardware design support for scientific computing tasks can be developed.

Let us consider the set-builder notation of the set of rational numbers  $\mathbb{Q} := \left\{ \frac{a}{b} \mid a, b \in \mathbb{Z}, b \neq 0 \right\}$ <sup>3</sup>. A rational number can be expressed as a fraction  $\frac{a}{b}$  where  $a$  and  $b$  are integers and  $b \neq 0$ . The set  $\mathbb{Q}$  is countable, continuous and dense in  $\mathbb{R}$ . In Type-2 Theory of Effectivity,  $\mathbb{Q}$  stands for the basis for higher abstraction level representations: the Cauchy representation of real numbers, some computable topological spaces, continuous functions in  $C[0; 1]$ ,  $L_p$  Spaces and Sobolev Spaces, to name a few [22, 15, 23]. There are, at least, two standard notations for the rational numbers:  $\nu_{\mathbb{Q}}$  [22, Def. 3.1.2] for a rational representation derived from the set-builder notation and  $\nu_{sd}$  [22, Def. 7.2.4] for a fractional positional notation for non periodic binary rational numbers. A normalized notation for these numbers is also proposed in [7].

This paper is focused on the proposal of a novel codification of rational numbers which aims for a feasible hardware implementation of operators based on online-arithmetic. We propose the extension of the fractional positional notation  $\nu_{sd}$  for non periodic binary rational numbers so that to include the periodic ones. In addition, normalization criteria is proposed for the representation so that to match the digital computer design criteria standards, which aim for simplifying hardware designs as well as for improving operand memory packaging and storage. An extension of online algorithms for the basic rational number operations is proposed.

The paper is arranged in the following sections: after the introduction and the preliminary section, Section 3 develops a fractional positional notation for the rational numbers with double mantissa  $\nu_{\mathbb{Q}}^{dm}$ , Section 4 deals with the modified online-division algorithm for these periodic rational numbers in fractional positional notation and, finally, Section 5 summarizes the conclusions.

---

<sup>3</sup> The symbol  $\mathbb{Z}$  derives from the German word *Zahl* (number) [8] and  $\mathbb{Q}$  derives from the German word *Quotient* (ratio) [3, p. 671].

## 2 Preliminaries

A fractional positional representation provides a direct value number, which is a desirable feature for a numeric codification format. The codification of rational numbers can be effectively performed with an exact positional representation, as it can be expressed by means of a finite concatenation of digits such as:

$$\forall w \in \mathbb{Q} \Leftrightarrow \exists i, j, k | w = w_0 \dots w_i \cdot w_{i+1} \dots w_j w_{j+1} \dots w_k w_{j+1} \dots w_k \dots, \quad (1)$$

where  $i, j, k \in \mathbb{N}$ , such that  $\forall w_l \in w, w_l \in \Gamma_n$ ,  $\Gamma_n = \{0, \dots, n-1\}$  and  $n \geq 2$  is the number base.

This codification is based on the concatenation of a sequence of digits around the fractional point (fixed mantissa  $w_0 \dots w_i \cdot w_{i+1} \dots w_j$ ) and a periodic series of digits (periodic mantissa  $w_{j+1} \dots w_k$ ). The shortest periodic sequence which is repeated is called the period. Conventionally, this periodic part is represented only once, embraced by an upper circumference arch, which permits the exact codification of a rational number with a finite number of symbols:

$$w = w_0 \dots w_i \cdot w_{i+1} \dots w_j \widehat{w_{j+1} \dots w_k}. \quad (2)$$

According to the features of the period, the following cases can be considered:

- All the periodic digits are zero ( $w_{j+1}, \dots, w_k = 0$ ). Then, the fixed mantissa codifies the exact rational number:  $w = w_0 \dots w_i \cdot w_{i+1} \dots w_j$ .
- All the periodic digits are equal to the highest digit of the number base  $n$ . Then, the number is codified by adding one unit in the last position (*ulp*) to the less significative digit of the fixed mantissa and removing the periodic digit  $w = w_0 \dots w_i \cdot w_{i+1} \dots w_j + ulp$ , with  $ulp = v_0 \dots v_i \cdot v_{i+1} \dots v_{j-1} v_j$ ,  $v_0, \dots, v_i, v_{i+1}, \dots, v_{j-1} = 0$  and  $v_j = n - 1$ .

The length of the periodic mantissa  $p = k - (j + 1)$  is tightly related to the denominator of the irreducible fraction with respect to the set-builder based representation of a rational number. Particularly is less than the denominator value:

$$\begin{aligned} \forall w \in \mathbb{Q}, w = \frac{a}{b}, a, b \in \mathbb{Z}, b \neq 0, \gcd(a, b) = 1 \Rightarrow \\ w = m_f \widehat{m_p} \wedge p = \Psi(b) < b, \end{aligned} \quad (3)$$

where  $m_f = w_0 \dots w_i \cdot w_{i+1} \dots w_j$ ,  $m_p = w_{j+1} \widehat{w_k}$ ,  $\gcd$  is the greatest common divisor of the numerator  $a$  and the denominator  $b$ , and  $\Psi(b)$  is a function based on the congruence relationship ( $\equiv$ ) [1, 11]:

$$\Psi(b) \equiv \begin{cases} 1 \bmod(b) & \text{if } \gcd(n, b) = 1, \\ 1 \bmod(b') & \text{if } \gcd(n, b) > 1, \end{cases} \quad (4)$$

where  $\bmod$  is the modular operator (integer division). Notice that when  $\gcd(n, b) > 1$  follows that  $\exists b', m \in \mathbb{N} | b = m \cdot b' \wedge \gcd(n, b') = 1 \wedge m = \prod_{i=1}^p z_i^j$ ,  $i > 0 \in \mathbb{N}$ ,  $z_1 \dots z_p \in \mathbb{N}$  is the set of prime factors of the number base  $n$  and  $j \in \mathbb{N}$  such as  $\frac{a}{b} = \frac{1}{m} \frac{a}{b'} \wedge \gcd(n, b') = 1 \wedge \Psi(b) \equiv 1 \bmod(b')$ .

According to the modular arithmetic rules, as  $\frac{1}{m}$  is composed by prime factors  $z_1 \dots z_p \in \mathbb{N}$  of the base  $n$  it makes  $p = 0$ . There exist infinitely many values which fulfill the previous congruence relationship, as every multiple of  $p$  satisfies the equation:

$$\Psi(b) \equiv 1 \pmod{b} \implies \forall j \in \mathbb{N}, n^{j \cdot p} \equiv 1 \pmod{b} \quad (5)$$

This fact justifies that every digit group which includes the period several times and whose length is a multiple of  $p$  is itself a period. In addition, all the rational numbers are periodic, despite the fact that the period is not codified when all the digits are 0. The amount of periodic numbers with  $p > 0$  is related with the base of the rational number to be codified.

There exist rational numbers in decimal with  $p = 0$  which in binary base produce a number with  $p > 0$ , whereas 2 is a prime factor of 10 and then all the binary number with  $p = 0$  produces, at the same time, a decimal number with  $p = 0$ . This has a great impact in numerical codification, because in the domain of the binary numbers there are fractional numbers with period 0 which can generate errors in the binary codification because the impossibility of codifying infinite fractional digits.

**Definition 1. (Periodic number)** A periodic number is a rational number whose periodic mantissa has length greater than 0 ( $p > 0$ ) such that

$$w \in \mathbb{Q}, \exists f, p \in \mathbb{N} \mid w = m_f \widehat{m_p}, \quad (6)$$

where  $m_f = w_0 \dots w_i \cdot w_{i+1} \dots w_j$ ,  $\widehat{m_p} = w_{j+1} \dots w_k w_{j+1} \dots w_k \dots$ ,  $i, j, k \in \mathbb{N}$ , such that  $\forall w_l \in w, w_l \in \Gamma_n$ ,  $\Gamma_n = \{0, \dots, n-1\}$ ,  $n \geq 2$ , and  $n$  is the number base.

**Definition 2. (Non periodic number)** A non periodic number is a rational number whose periodic mantissa has length 0 ( $p = 0$ ) such that

$$w \in \mathbb{Q}, \exists f, p \in \mathbb{N} \mid w = m_f, \quad (7)$$

where  $m_f = w_0 \dots w_i \cdot w_{i+1} \dots w_j$ ,  $\widehat{m_p} = \lambda$  (empty string),  $i, j \in \mathbb{N}$ , such that  $\forall w_l \in w, w_l \in \Gamma_n$ ,  $\Gamma_n = \{0, \dots, n-1\}$ ,  $n \geq 2$ , and  $n$  is the number base.

**Definition 3. (Pure periodic number)** A pure periodic number is a rational number whose fixed mantissa is 0 ( $f = 1$ ) such that

$$w \in \mathbb{Q}, \exists f, p \in \mathbb{N} \mid w = m_f m_p, \quad (8)$$

where  $m_f = 0.$ ,  $\widehat{m_p} = w_{j+1} \dots w_k w_{j+1} \dots w_k \dots$ ,  $j, k \in \mathbb{N}$ , such that  $\forall w_l \in w, w_l \in \Gamma_n$ ,  $\Gamma_n = \{0, \dots, n-1\}$ ,  $n \geq 2$ , and  $n$  is the number base.

Now, consider the following notations and representations in TTE.

Let  $\Sigma = \{\bar{1}, 0, 1\}$  and let  $\Sigma^*$  and  $\Sigma^\omega$  denote the sets of finite and infinite sequences over  $\Sigma$ , respectively. We always assume that there exists an element  $\# \notin \Sigma$  and denote  $\Sigma_\# := \Sigma \cup \{\#\}$ . Let  $\iota_w : \Sigma^* \rightarrow \Sigma_\#^*$  be a wrapping function

that assigns to a word  $u = u_0 \dots u_k \in \Sigma^*$  with  $u_0, \dots, u_k \in \Sigma$  and  $k \in \mathbb{N}$  the word  $\iota_w(u) := \#\#u_0\#u_1\#\dots\#u_k\#\#$  and let  $\iota_u : \Sigma_\#^* \rightarrow \Sigma^*$  be the corresponding unwrapping function that obtains from a word  $v \in \Sigma_\#^*$  the word  $\iota_u(v) := w$  with  $w = w_0 \dots w_k \in \Sigma^*$ , and  $k \in \mathbb{N}$ .

**Definition 4.** (*Some standard notations and representations*) [22, Def. 3.1.2 (1-4)].

1. The identity  $\text{id}_{\Sigma^*} : \Sigma^* \rightarrow \Sigma^*$  is a notation of  $\Sigma^*$  and the identity  $\text{id}_{\Sigma^\omega} : \Sigma^\omega \rightarrow \Sigma^\omega$  is a representation of  $\Sigma^\omega$ .
2. Define the binary notation  $\nu_{\mathbb{N}} : \subseteq \Sigma^* \rightarrow \mathbb{N}$  of the natural numbers by:  

$$\text{dom}(\nu_{\mathbb{N}}) := \{0\} 1 \{0, 1\}^* \text{ and } \nu_{\mathbb{N}}(a_k, \dots, a_0) = \sum_{i=0}^k a_i \cdot 2^i, (a_0, \dots, a_k \in \{0, 1\}).$$
3. Define a notation  $\nu_{\mathbb{Z}} : \subseteq \Sigma^* \rightarrow \mathbb{Z}$  of the integers by:  $\text{dom}(\nu_{\mathbb{Z}}) := \{0\} 1 \{0, 1\}^* \cup -1 \{0, 1\}^*$ ,  $\nu_{\mathbb{Z}}(w) = \nu_{\mathbb{N}}(w)$  and  $\nu_{\mathbb{Z}}(-w) = -\nu_{\mathbb{N}}(w)$  for  $w \in \text{dom}(\nu_{\mathbb{N}}) \setminus \{0\}$ .
4. Define a notation  $\nu_{\mathbb{Q}} : \subseteq \Sigma^* \rightarrow \mathbb{Q}$  by:  

$$\text{dom}(\nu_{\mathbb{Q}}) := \{u/v \mid u \in \text{dom}(\nu_{\mathbb{Z}}), v \in \text{dom}(\nu_{\mathbb{N}}), v_{\mathbb{N}}(v) \neq 0\} \text{ and } \nu_{\mathbb{Q}}(u/v) := \nu_{\mathbb{Z}}(u) / \nu_{\mathbb{N}}(v).$$
 Later we will abbreviate  $\nu_{\mathbb{Q}}(w)$  by  $\bar{w}$ .

**Definition 5.** (*Signed digit notation and representation*) [22, Def. 7.2.4]. Let  $\bar{1} \in \Sigma$  abbreviate  $-1$ . Define the signed digit notation  $\nu_{sd}$  of the binary rational numbers and the signed digit representation  $\rho_{sd} : \subseteq \Sigma^* \rightarrow \mathbb{R}$  of the real numbers as follows:

$$\begin{aligned} \text{dom}(\rho_{sd}) &:= \begin{cases} \text{all } a_n \dots a_0 \bullet a_{-1} a_{-2} \dots \in \Sigma^\omega \text{ such that } n \geq -1, \\ a_i \in \{\bar{1}, 0, 1\} \text{ for } i \leq n, \\ a_n \neq 0, \text{ if } n \geq 0 \text{ and } a_n a_{n-1} \notin \{1\bar{1}, \bar{1}1\} \text{ if } n \geq 1, \end{cases} \\ \text{dom}(\nu_{sd}) &:= \{u \bullet v \mid u, v \in \Sigma^*, u \bullet v 0^w \in \text{dom}(\rho_{sd})\}, \\ \rho_{sd}(a_n \dots a_0 \bullet a_{-1} a_{-2} \dots) &:= \sum_{i=n}^{-\infty} a_i \cdot 2^i, \\ \nu_{sd}(u \bullet v) &:= \rho_{sd}(u \bullet v 0^w). \end{aligned}$$

For  $p := a_n \dots a_0 \bullet a_{-1} a_{-2} \dots \in \text{dom}(\rho_{sd})$  and  $k \in \mathbb{N}$  define  $p[k] := a_n \dots a_0 \bullet a_{-1} \dots a_{-k} \in \Sigma^*$ .

### 3 Fractional Positional Notation for Periodic Rational Numbers

Define the notation  $\nu_{\mathbb{Z}}^{sd}$  for codifying integer numbers in signed digit notation:

$$\begin{aligned} \nu_{\mathbb{Z}}^{sd} &: \subseteq \Sigma^* \rightarrow \mathbb{Z}, \\ \text{dom}(\nu_{\mathbb{Z}}^{sd}) &:= \begin{cases} \text{all } a_n \dots a_0 \in \Sigma^* \text{ for } n \geq 0, \\ a_i \in \Sigma \text{ for } i \leq n, \\ a_n \neq 0, \text{ if } n \geq 0 \text{ and } a_n a_{n-1} \notin \{1\bar{1}, \bar{1}1\}, \text{ if } n \geq 1, \end{cases} \\ \nu_{\mathbb{Z}}^{sd}(e_n \dots e_0) &:= \rho_{sd}(e_n \dots e_0 0^w). \end{aligned} \tag{9}$$

Now, define a fractional positional notation  $\nu_{\mathbb{Q}}^{nsd}$  of signed digit rational numbers, such as for  $a \in \mathbb{Q}$  then  $0 \leq a < 1$ .

$$\begin{aligned} \nu_{\mathbb{Q}}^{nsd} : \subset \Sigma^* &\longrightarrow A = \{a \mid 0 \leq a < 1\} \cap \mathbb{Q} \\ \text{dom}(\nu_{\mathbb{Q}}^{nsd}) &:= \{u \cdot v \mid u = 0, v \in \Sigma^*, u \cdot v 0^w \in \text{dom}(\rho_{sd})\}, \\ \nu_{\mathbb{Q}}^{nsd}(u \cdot v) &:= \rho_{sd}(u \cdot v 0^w). \end{aligned} \quad (10)$$

Finally, define the notation  $\nu_{\mathbb{Q}}^{dm}$  for normalized fractional positional periodic rational numbers with double mantissa (fixed and periodic) as:

$$\begin{aligned} \nu_{\mathbb{Q}}^{dm} : \subset \Sigma^* &\longrightarrow \mathbb{Q} \\ \text{dom}(\nu_{\mathbb{Q}}^{dm}) &:= \{\iota_w(e) \iota_w(m_f) \iota_w(m_p) \mid e, m_p \in \text{dom}(\nu_{\mathbb{Z}}^{sd}) \text{ and} \\ &m_f \in \text{dom}(\nu_{\mathbb{Q}}^{nsd})\}, \\ \nu_{\mathbb{Q}}^{dm}(uvw) &:= \left\{ q = \frac{a}{b} \mid a = \nu_{sd}(2^e) \cdot \nu_{sd}(m_f m_p - m_f) \text{ with} \right. \\ &e = \iota_u(u), m_f = \iota_u(v), m_p = \iota_u(w), \text{and } b = \{1\}^p \{0\}^f \text{ with} \\ &f = |m_f|, p = |m_p| \}, \end{aligned} \quad (11)$$

for a signed digit notation in radix-2.

**Theorem 1.** *The notation  $\nu_{\mathbb{Q}}^{dm}$  induces the same concept of computability as the standard notation  $\nu_{\mathbb{Q}}$  ( $\nu_{\mathbb{Q}}^{dm} \equiv \nu_{\mathbb{Q}}$ )*

*Proof.* ( $\nu_{\mathbb{Q}} \leq \nu_{\mathbb{Q}}^{dm}$ ). The translation from the notation  $\nu_{\mathbb{Q}}$  into  $\nu_{\mathbb{Q}}^{dm}$  notation can be done straight away by applying a modified version of the conventional school division algorithm in which, when the fractional part is being obtained, we check the repetition of partial remainders. Notice that this key issue is used for the online-division algorithm proposed in Section 4. Therefore, we wrap the partial remainders  $pr \in \text{dom}(\nu_{\mathbb{Z}})$  and then successively concatenate them with a given word  $r \in \Sigma_{\#}^* \cup \{\lambda\}$ , which we initialize to  $\lambda$  ( $r = r \iota_w(pr)$ ). On every iteration related with the calculation of the fractional part, we check if the wrapped partial remainder  $\iota_w(pr)$ . In that case, we find the position of the subword by means of a subword count function  $c : \Sigma_{\#}^* \cup \{\lambda\} \times \Sigma_{\#}^* \rightarrow \mathbb{Z}$ , which outputs the number of words counted  $i \in \mathbb{N}$  when the partial remainder subword was found and then the algorithm stops. Notice that, as usual, the algorithm can also stop when the partial remainder is zero ( $pr = 0$ ). If the partial remainder is not found ( $pr$  is not a subword of  $r$ ), then output  $-1$ .

Now, let  $q \in \text{dom}(\nu_{\mathbb{Q}})$  be the quotient obtained from the division,  $m_f \in \text{dom}(\nu_{sd})$  the fixed mantissa and  $m_p \in \text{dom}(\nu_{\mathbb{Z}}^{sd})$  the periodic mantissa. Then  $m_f$  and  $m_p$  are the following prefix and suffix from  $q$ , which are now translated into signed digit based notations:

$$\begin{aligned} m_f &= \nu_{sd}(q_0 \dots q_i), \\ m_p &= \nu_{\mathbb{Z}}^{sd}(q_{i+1} \dots q_j), \end{aligned} \quad (12)$$

where  $j = |q| - 1$ .

Now, in order to normalize the rational number obtained:

Define the count function  $c : \text{dom}(\nu_{sd}) \rightarrow \text{dom}(\nu_{\mathbb{N}})$ , which outputs a natural number corresponding to the position of the dot “.” in the input word. If the dot “.” is not found, then the function outputs the length of the word.

Define the function  $a : \Sigma^* \times \text{dom}(\nu_{\mathbb{N}}) \rightarrow \Sigma^* \times \text{dom}(\nu_{\mathbb{Z}})$  which removes the 0s at the beginning of an input string  $u \in \Sigma^*$  and successively decrements the input  $n \in \text{dom}(\nu_{\mathbb{N}})$ . This function outputs the remaining string  $m \in \Sigma^*$  and the decremented input  $n$ , which can be negative ( $z \in \text{dom}(\nu_{\mathbb{Z}})$ ).

Finally, define the function  $i : \Sigma^* \times \text{dom}(\nu_{\mathbb{Z}}) \rightarrow \text{dom}(\nu_{sd})$  as

$$i(u, z) := \begin{cases} 0 \cdot \{0\}^z u & \text{if } z - 1 \leq 0, \\ u_0 \dots u_{z-1} \cdot u_z \dots u_k & \text{if } 0 < z - 1 < \nu_{\mathbb{Z}}(k), \\ u_0 \dots u_z \cdot \{0\}^{z - \nu_{\mathbb{Z}}(k) - 1} & \text{if } z - 1 \geq \nu_{\mathbb{Z}}(k), \end{cases} \quad (13)$$

where  $u = u_0 \dots u_k \in \Sigma^*$ ,  $z \in \text{dom}(\nu_{\mathbb{Z}})$  and  $k \in \mathbb{N}$ .

Then, the exponent is obtained in the following way:

1. Count the number of positions from the beginning of the string until the dot character “.” is found with the function  $c : \text{dom}(\nu_{sd}) \rightarrow \text{dom}(\nu_{\mathbb{N}})$ . This will provide the initial exponent.
2. Remove the dot character “.” from the string and then remove the 0s at the beginning of it. At the same time, while removing the 0s, successively decrement the exponent by applying the function  $a : \Sigma^* \times \text{dom}(\nu_{\mathbb{N}}) \rightarrow \Sigma^* \times \text{dom}(\nu_{\mathbb{Z}})$ . The mantissa and the exponent at the output are the remaining chunk of the string and the decremented exponent, respectively.
3. Apply the wrapping function in order to wrap the exponent the fixed mantissa and the periodic mantissa:

$$\begin{aligned} \iota_w(e) &:= u = \# \# e_0 \# e_1 \# \dots \# e_k \# \#, \\ \iota_w(m_f) &:= v = \# \# m_{f,0} \# m_{f,1} \# \dots \# m_{f,k} \# \#, \\ \iota_w(m_p) &:= w = \# \# m_{p,0} \# m_{p,1} \# \dots \# m_{p,k} \# \#, \end{aligned}$$

and then concatenate  $u$  and  $v$  such as  $w = uv \in \text{dom}(\nu_{nsd})$ .

*Proof.* ( $\nu_{\mathbb{Q}}^{dm} \leq \nu_{\mathbb{Q}}$ ). The translation from  $\nu_{\mathbb{Q}}^{dm}$  into  $\nu_{\mathbb{Q}}$  can be directly achieved by using TTE-computable string manipulation functions.

Define the search function  $s : \Sigma_{\#}^* \times \mathbb{N} \rightarrow \Sigma^*$  such as for a word  $w \in \Sigma_{\#}^*$  and  $j \in \mathbb{N}$  it obtains the corresponding unwrapped subword from  $w$ :

$$s(w, j) := \{ \iota_u(u) \mid u \text{ is the subword } j \text{ in } w \text{ with } u \in \Sigma_{\#}^* \text{ and } j \in \mathbb{N} \}. \quad (14)$$

1. For  $w \in \text{dom}(\nu_{\mathbb{Q}}^{dm})$ , apply the search function  $s : \Sigma_{\#}^* \times \mathbb{N} \rightarrow \Sigma^*$  to obtain the exponent  $e = s(w, 0)$ , the fixed mantissa  $m_f = s(w, 1)$  and the periodic mantissa  $s(w, 2)$ .

2. Define  $q = \frac{\nu_{\mathbb{Z}}(a)}{\nu_{\mathbb{N}}(b)}$  where  $a = \nu_{sd}(2^e) \cdot \nu_{sd}(m_f m_p - m_f) \in \text{dom}(\nu_{sd})$ , and  $b = \{1\}^p \{0\}^f$  with  $f = |m_f|$ ,  $p = |m_p|$ .

Then, as  $\nu_{\mathbb{Q}}$  is reducible to  $\nu_{\mathbb{Q}}^{dm}$  ( $\nu_{\mathbb{Q}} \leq \nu_{\mathbb{Q}}^{dm}$ ) and also  $\nu_{\mathbb{Q}}^{dm}$  is reducible to  $\nu_{\mathbb{Q}}$  ( $\nu_{\mathbb{Q}}^{dm} \leq \nu_{\mathbb{Q}}$ ) it can be concluded that  $\nu_{\mathbb{Q}}^{dm}$  is equivalent to  $\nu_{\mathbb{Q}}$  ( $\nu_{\mathbb{Q}}^{dm} \equiv \nu_{\mathbb{Q}}$ ).

## 4 Online Division with Periodic Rational Numbers

The computability of the general algorithm for developing online-arithmetic operators is absolutely concerned with the possibility to reduce the residual recurrence equation to additions and shifts [9, Chap. 9]. Among all the basic algorithms (addition, subtraction, multiplication and division) the case of the division outlines a great variety of cases. The extended online division algorithm for normalized signed digit operands  $X, D$  ( $0 < X, D < 1$ ) with 4 delay cycles for the initialization is the following:

### Online Division Algorithm

```

 $REGQ, REGD, v, w := 0$  'Initialize
 $(start\_CP, modul) := MaxOverlap(X, D)$ 
For  $it := 0$  To precision
     $x := ObtainDigit(\dots); d := ObtainDigit(\dots)$ 
     $REGD := AppendOnline(REGD, d)$ 
    If  $it < kDELAY\_DIVSD$  Then 'Initialization stage
         $v := 2w + x \cdot 2^{-kDELAY\_DIVSD}$ 
         $w := v$ 
    Else 'For the rest of iterations
         $v := 2w + x \cdot 2^{-kDELAY\_DIVSD} - REGQ \cdot d \cdot 2^{-kDELAY\_DIVSD}$ 
         $q := SELD(v)$ 
         $w := v - q \cdot REGD$ 
         $exit := CheckPeriod(\dots)$ 
        If  $exit = True$  Then
            Exit For;
        Else
             $REGQ := AppendOnline(REGQ, q)$ 
        EndIf
    EndIf
Next it
```

The extension of the algorithm is based on two functions. The function  $MaxOverlap(X, D)$  calculates the iteration ( $start\_CP$ ) at which the residual is stored for a later identification of the periodic mantissa with a predictable iteration modulus ( $modul$ ). The function  $CheckPeriod(\dots)$  evaluates every matching residue  $w$  at a given iteration so that to identify the repetition of a periodic mantissa. For the latter function, the following cases have to be considered:

1.  $X, D$  are non-periodic. After storing the residual value  $res := w$  at the iteration given ( $it = start\_CP$ ), the residual is checked iteration after iteration so that to find either  $w = 0$  (non-periodic result) or  $res = w$  (periodic result of with mantissa length  $lmp = it - start\_CP$ ).
2.  $X$  is periodic and  $D$  is non-periodic. After storing the residual value  $res := w$  at the iteration given ( $it = start\_CP$ ), the residual is checked at the iterations which fulfill the condition  $mod(it, modul) = 0$  so that to find  $res = w$  (periodic result of with mantissa length  $lmp = it - start\_CP$ ). The value of

*modul* assigned in  $\text{MaxOverlap}(X, D)$  is the length of the periodic mantissa of  $X$ .

3.  $X$  is non-periodic and  $D$  is periodic. After storing the residual value  $res := w$  at the iteration given ( $it = start\_CP$ ), the residual is checked at the iterations which fulfill the condition  $\text{mod}(it, modul) = 0$  so that to find  $res - w < ulp_{res}$  (periodic result of with mantissa length  $lmp = it - start\_CP$ ). The value of *modul* assigned in  $\text{MaxOverlap}(X, D)$  is the length of the periodic mantissa of  $D$ .
4.  $X, D$  are periodic. The case is similar to the previous one, except that the value of *modul* assigned in  $\text{MaxOverlap}(X, D)$  is the maximum of length of the periodic mantissas  $X$  and  $D$ , and the result is periodic with mantissa length  $lmp = it - start\_CP$ .

The algorithm initializes the internal variables  $REGQ$ ,  $REGD$ ,  $v$  and  $w$  which hold the quotient, the divisor as well as the estimator and residual, respectively. In the main loop, the following leftmost input bits from the operands  $X$  and  $D$  are obtained ( $x$  and  $d$ , respectively). The divisor digit obtained is concatenated to the divisor register  $REGD$ . Then, two stages are considered: first, initialization of the residual ( $it < kDELAY\_DIVSD$ , the first 4 cycles) and second, period check and/or digit output estimation. For the second stage, the resulting digit at this iteration is obtained with the selection function  $SEL(.)$  applied to the estimation value  $v$  and the new value of the residual is calculated. Then the conditions for checking the period are analyzed in  $\text{CheckPeriod}(\dots)$ . If the residual is not repeated, the resulting digit is appended to the output quotient register  $REGQ$  with the function  $\text{AppendOnline}(\dots)$  and the algorithm performs a new iteration, unless it reaches the maximum iterations fixed.

## 5 Conclusions

A computable fractional positional notation for periodic rational numbers has been proposed. It aims to improve the comparison operations as well as to reduce memory storage by managing periodic mantissas. With this approach, the space of representation for a fixed precision is increased, as it is possible to emulate the infinite digits of a periodic rational number with a fixed length periodic mantissa. The algorithm proposed for the division identifies the periodic mantissa of the result before its first repetition by exploiting the concept of the residual  $w$  as the inner state of the algorithm. Similar results can be obtained with the addition, subtraction and multiplication. The complexity of the operations introduced implies: obtaining the algorithm iteration at which the singular residual has to be stored, calculating the iteration modulus step at which the next residuals have to be checked and subtracting and comparing at the iteration check established.

The encouraging experimental results obtained outline the interest of formalizing the periodic mantissa identification using the congruence relationship based on the Fermat's Little Theorem (generalized by Euler) and also of extending the algorithms to high-radix number bases as well as to other arithmetic operations.

## References

1. Belski, A.A. and Kaluzhnin L.A.: División inexacta, Lecciones populares de matemáticas. Ed. Mir, Moscow (Spanish Translation), (1980)
2. Blanck, J.: Exact real arithmetic systems: Results of competition, Computability and Complexity in Analysis, LNCS, Vol. 2064, (2001) 390–394
3. Bourbaki, N.: Éléments de mathématique: Algèbre. Reprinted as Elements of Mathematics: Algebra I. Berlin: Springer-Verlag, (1998)
4. Buchberger, B.: Groebner Bases in MATHEMATICA: Enthusiasm and Frustation, Programming Environments for High-level Scientific Problem Solving, (1991) 80–91
5. Brezinski, C.: History of Continued Fractions and Pade Approximants. Springer-Verlag, (1980)
6. de Miguel Casado, G. and García Chamizo, J.M.: The Role of Algebraic Models and Type-2 Theory of Effectivity in Special Purpose Processor Design, LNCS, Vol. 3988, (2006) 137–146
7. de Miguel Casado, G.; García Chamizo, J.M. and Signes Pont, M.T.: Algebraic Model of an Arithmetic Unit for TTE-Computable Normalized Rational Numbers, LNCS, Vol. 4497, (2007) 218–227
8. Dummit, D. S. and Foote, R. M., "Abstract Algebra, 2nd ed.", Englewood Cliffs, NJ: Prentice-Hall, 1998.
9. Ercegovac, M.D. and Lang, T.: Digital Arithmetic. M. Kaufmann, (2004)
10. Gowland, P. and Lester, D.: A Survey of Exact Arithmetic Implementations, LNCS, Vol. 2064, (2001) 30–47
11. Guelfond, A.O.: Resolución de ecuaciones en números enteros, Lecciones populares de matemáticas- Ed. Mir, Moscow (Spanish Translation), (1979)
12. Hehner, E.C.R. and Horspool, R.N.S.: A New Representation of the Rational Numbers for fast Easy Arithmetic, SIAM J. Computing, Vol. 8(2), (1979)
13. Matula, D. and Kornerup, P.: Finite Precision Rational Arithmetic: An Arithmetic Unit, IEEE Transactions on Computers, Vol. C-32, (1983) 378–387
14. Kornerup, P. and Matula, D.: Algorithms for Arbitrary Precision Floating Point Arithmetic, Proceedings of the 10th IEEE Symposium on Computer Arithmetic, (1991)
15. Kunkle, D.: Type-2 computability on spaces of integrable functions, Math. Log. Quart., Vol. 50, (2004) 417–430
16. Lynch, T. and Schulte, M.: A High Radix On-line Arithmetic for Credible and Accurate Computing, Journal of UCS, Vol. 1, (1995) 439–453
17. Mencer, O: Rational Arithmetic Units in Computer Systems, PhD Thesis, Stanford University, (2000)
18. Mora, H.: Procesadores Aritméticos Especializados. Computación Racional Exacta, Ph.D Dissertation, University of Alicante, (2003)
19. Patterson, D.A. and Hennessy J.L.: Computer Architecture a quantitative approach. M. Kaufmann, (2002)
20. Schröder, M.: Admissible Representations in Computable Analysis, LNC, Vol. 3998, (2006) 471–480
21. Vuillemin, J.E.: Exact Real Computer Arithmetic with Continued Fractions, IEEE Transactions on Computers, Vol. 39, (1990) 1087–1105
22. Weihrauch, K.: Computable Analysis. Springer-Verlag, (2000)
23. Zhong, N. and Weihrauch, K.: Computability Theory of Generalized Functions, Journal of the ACM, Vol. 50, 2003 (469–505)

# Abstract Geometrical Computation with Accumulations: Beyond the Blum, Shub and Smale Model

Jérôme Durand-Lose

Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans,  
B.P. 6759, F-45067 ORLÉANS Cedex 2.

<http://www.univ-orleans.fr/lifo/Members/Jerome.Durand-Lose>,  
Jerome.Durand-Lose@univ-orleans.fr

**Abstract.** Abstract geometrical computation (AGC) naturally arises as a continuous counterpart of cellular automata. It relies on signals (dimensionless points) traveling and colliding. It can carry out any Turing computation, but since it works with continuous time and space, some analog computing capability exists. In *Abstract Geometrical Computation and the Linear BSS Model* (CiE 2007, LNCS 4497, p. 238-247), it is shown that AGC without any accumulation has the same computing capability as the linear BSS model.

An accumulation brings infinitely many time steps in a finite duration. This has been used to implement the black-hole model of computation (*Fundamenta Informaticae* 74(4), p. 491-510). It also makes it possible to multiply two variables, thus simulating the full BSS. Nevertheless a BSS uncomputable function, the square root, can also be implemented, thus proving that the computing capability of AGC with isolated accumulations is strictly beyond the one of BSS.

**Key words:** Abstract geometrical computation, Accumulations, Analog computation, BSS model, Signal machine.

## 1 Introduction

There is no agreed continuous/analog/ $\mathbb{R}$  counterpart of the Church-Turing thesis. Relating the numerous models is crucial to understand the differences between their computing capabilities. For example, Bournez *et al.* [BH04,BCGH06] as well as others [Kaw05] have related Moore's recursion theory on  $\mathbb{R}$  [Moo96], computable analysis [Wei00] and the general purpose analog computer [PE74]. The aim of the present paper is to relate two models. One, abstract geometrical computation (AGC) deals with regular and automatic drawing on the Euclidean plane, while the second, the Blum, Shub and Smale model [BCSS98] relies on algebraic computations over  $\mathbb{R}$ . Bournez [Bou99] has already provided some relations between linear BSS and Piecewise constant derivative systems which also generates Euclidean drawings. In [DL07], AGC without accumulation was proved equivalent to the linear BSS model (*i.e.* BSS restricted to multiplication only by constants) with an unbounded number of variables. In the present paper, the full BSS is related to AGC with isolated accumulations.

Let us note that AGC relies on intersection of lines (to find the collisions), another model considers also intersections with circles [Huc89,Huc91]; AGC considers all collisions while in Huckenbeck's geometrical machines the programs chooses which geometric object to build and to consider.

Abstract geometrical computation (ACG) arises from the common use in cellular automata (CA) literature of Euclidean settings to explain an observed dynamics or to design a CA for a particular purpose. While CA operate in discrete time over discrete space, Euclidean geometry deals with both continuous time and space. This switch of context is justified by the scaling invariance of CA and relates to our preference and ability for thinking in continuous rather than discrete terms. Abstract geometrical computation works in a continuous setting: discrete signals/particles are dimensionless points; the local function of CA – computing the next state of a cell according to the states of neighbouring cells – is replaced by collision rules: which signals emerge from a collision of signals. Signals and rules define *signal machines* (SM).

This recent model, restricted to rational numbers, is able to carry out any (discrete) Turing-computation [DL05b]; even with the additional restriction of reversibility and conservativeness [DL06c]. With continuous time, comes Zeno effect (infinitely many discrete steps during a finite duration): not only are accumulations possible, but they can be used to decide recursively enumerable problems by using the black-hole scheme [DL05a,DL06a]. Although accumulations can easily be generated, they can hardly be foreseen [DL06b].

In the Blum, Shub and Smale model (BSS), machines compute over any ring. Roughly speaking, polynomial functions can be performed on variables as well as tests (according to some order) for branching. The dimension of the input is not bounded, a *shift* operator is provided in order to access any variable (finitely many variables are considered since only finitely many are accessed in finite time).

In [DL07], AGC (without accumulation) and linear (multiplying two variables is forbidden) BSS over  $\mathbb{R}$  with an unbounded number of variables are proved equivalent. This is done through linear real number unlimited register machines (the arguments of [Nov95] for the equivalence of URM and BSS translate to the linear case).

With a reasonable handling of accumulations, restrictions on multiplication can be lifted thus achieving the full BSS computing capability. They are handled in the following way: there is no second (or higher) order accumulation; only finitely many signals leave any accumulation; and one signal appears where an accumulation takes place. Multiplication is embedded in the same context and the same encoding of real numbers as in [DL07], so that addition, multiplication by constants and test do not have to be implemented again. Only the basis is recalled: a real number is encoded as the distance between two signals.

Multiplication of two real numbers,  $x$  and  $y$ , is done by producing the binary extension of  $y$  and according to  $y_n$  adding or not  $x2^n$ . With integers,  $n$  is increasing from 0, with real numbers,  $n$  goes down to  $-\infty$  which explains the use of an accumulation. Each iteration divides  $x$  by 2, computes the next bit of

$y$  and updates the partial product accordingly. All the values are geometrically decreasing to zero and so are the time and space used by an iteration, so that there is an accumulation. The signal left by the accumulation is located exactly at the value of the product.

To prove that AGC with accumulation is, as expected, strictly more powerful than the BSS model, it is explained how to implement the square root. Each iteration only uses addition and multiplication by constants.

Since the reader might be more familiar with BSS than with ACG, more care and illustrations are given to ACG. Section 2 provides all the needed definitions. Section 3 recalls basic encoding and geometric constructions for BSS simulation. Section 4 provides the multiplication between variables. Section 5 explains how to build a signal machine able to compute the square root with an accumulation. Conclusion, remarks and perspectives are gathered in Section 6.

## 2 Definitions

**Abstract geometrical computation.** In this model, dimensionless objects are moving on the real axis. When a collision occurs they are replaced according to rules. This is defined by the following machines:

**Definition 1** A *signal machine with accumulation* is defined by  $(M, S, R, \mu_a)$  where  $M$  (*meta-signals*) is a finite set,  $S$  (*speeds*) a function from  $M$  to  $\mathbb{R}$ ,  $R$  (*collision rules*) a partial function from the subsets of  $M$  of cardinality at least two into subsets of  $M$  (all these sets are composed of signals of distinct speed) and  $\mu_a$  is a meta-signal, the one that comes out of any accumulation.

Each instance of a meta-signal is a *signal*. The function  $S$  assigns *speeds* to meta-signals. They correspond to the inverse slopes of the segments in space-time diagrams. The *collision rules*, denoted  $\rho^- \rightarrow \rho^+$ , define what emerge ( $\rho^+$ ) from the collision of two or more signals ( $\rho^-$ ). Since  $R$  is a function, signal machines are deterministic. The *extended value set*,  $V$ , is the union of  $M$  and  $R$  plus two symbols: one for void,  $\emptyset$ , and one for accumulation  $*$ . A *configuration*,  $c$ , is a total function from  $\mathbb{R}$  to  $V$  such that the set  $\{x \in \mathbb{R} \mid c(x) \neq \emptyset\}$  is finite.

A signal corresponding to a meta-signal  $\mu$  at a position  $x$ , i.e.  $c(x) = \mu$ , is moving uniformly with constant speed  $S(\mu)$ . A signal can only start in a collision, except for  $\mu_a$  that can also be generated by an accumulation. A signal can only end in a collision or an accumulation. This corresponds to condition 2 in Def. 2. At a  $\rho^- \rightarrow \rho^+$  collision signals corresponding to the meta-signals in  $\rho^-$  (resp.  $\rho^+$ ) must end (resp. start) and no other signal should be present (condition 3). Condition 4 deals with accumulations, the first line implies that sufficiently close to the accumulation, outside of the light cone, there is nothing but a  $\mu_a$  signal leaving the accumulation. The second line expresses that there is indeed an accumulation (this is not formalized since it would be too ponderous).

Let  $S_{min}$  and  $S_{max}$  be the minimal and maximal speeds. The *causal past*, or backward *light-cone*, arriving at position  $x$  and time  $t$ ,  $J^-(x, t)$ , is defined by all the positions that might influence the information at  $(x, t)$  through signals, formally:

$$J^-(x, t) = \{ (x', t') \mid x - S_{max}(t-t') \leq x' \leq x - S_{min}(t-t') \} .$$

**Definition 2** The *space-time diagram* issued from an initial configuration  $c_0$  and lasting for  $T$ , is a function  $c$  from  $[0, T]$  to configurations (i.e. a function from  $\mathbb{R} \times [0, T]$  to  $V$ ) such that,  $\forall (x, t) \in \mathbb{R} \times [0, T]$  :

1.  $\forall t \in [0, T], \{x \in \mathbb{R} \mid c_t(x) \neq \emptyset\}$  is finite,
2. if  $c_t(x) = \mu$  then  $\exists t_i, t_f \in [0, T]$  with  $t_i < t < t_f$  or  $0 = t_i = t < t_f$  or  $t_i < t = t_f = T$  s.t.:
  - $\forall t' \in (t_i, t_f), c_{t'}(x + S(\mu)(t' - t)) = \mu$ ,
  - $t_i = 0$  or ( $c_{t_i}(x_i) = \rho^- \rightarrow \rho^+$  and  $\mu \in \rho^+$ ) or ( $c_{t_i}(x_i) = \ast$  and  $\mu = \mu_a$ ) where  $x_i = x + S(\mu)(t_i - t)$ ,
  - $t_f = T$  or ( $c_{t_f}(x_f) = \rho^- \rightarrow \rho^+$  and  $\mu \in \rho^-$ ) or  $c_{t_f}(x_f) = \ast$  where  $x_f = x + S(\mu)(t_f - t)$ ;
3. if  $c_t(x) = \rho^- \rightarrow \rho^+$  then  $\exists \varepsilon, 0 < \varepsilon, \forall t' \in [t - \varepsilon, t + \varepsilon] \cap [0, T], \forall x' \in [x - \varepsilon, x + \varepsilon]$ ,
  - $(x', t') \neq (x, t) \Rightarrow c_{t'}(x') \in \rho^- \cup \rho^+ \cup \{\emptyset\}$ ,
  - $\forall \mu \in M, c_{t'}(x') = \mu \Leftrightarrow$  or  $\begin{cases} \mu \in \rho^- \text{ and } t' < t \text{ and } x' = x + S(\mu)(t' - t), \\ \mu \in \rho^+ \text{ and } t < t' \text{ and } x' = x + S(\mu)(t' - t). \end{cases}$
4. if  $c_t(x) = \ast$  then
  - $\exists \varepsilon > 0, \forall t' \in [t - \varepsilon, t + \varepsilon] \cap [0, T], \forall x' \in [x - \varepsilon, x + \varepsilon]$ ,
  - $(x', t') \notin J^-(x, t) \Rightarrow$  or  $\begin{cases} c_{t'}(x) = \emptyset \text{ and } x' \neq x + S(\mu_a)(t' - t), \\ c_{t'}(x) = \mu_a \text{ and } x' = x + S(\mu_a)(t' - t), \end{cases}$
  - $\forall \varepsilon > 0$ , there are infinitely many collisions in  $J^-(x, t) \cap \mathbb{R} \times [t - \varepsilon, t]$ .

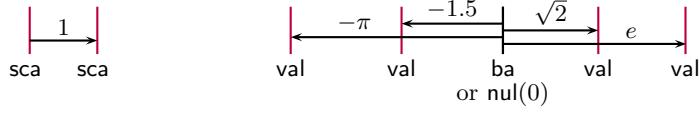
On space-time diagrams, the traces of signals are line segments whose directions are defined by  $(S(\cdot), 1)$  (1 is the temporal coordinate). Collisions correspond to the extremities of these segments. This definition can easily be extended to  $T = \infty$ . In the space-time diagrams, time increases upwards. To simplify, the same name is used for a signal throughout the computation, in fact, there is a different meta-signal for each speed. As a computing device, the input is the initial configuration and the output is the final configuration.

**Blum, Shub and Smale model.** (The reader is expected to be more familiar with the BSS model than with AGC, so this part is not very detailed.) BSS machines operate on an unbounded array containing real numbers in exact precision. The input/output is the content of the array. Apart from start and stop, the available instructions are: compute a polynomial function (and store the result), branch according to a sign test and shift. The machine can only access a finite part of the array at any time, the shift operator allows it to move on the array (like the head of a Turing machine).

### 3 Basic construction

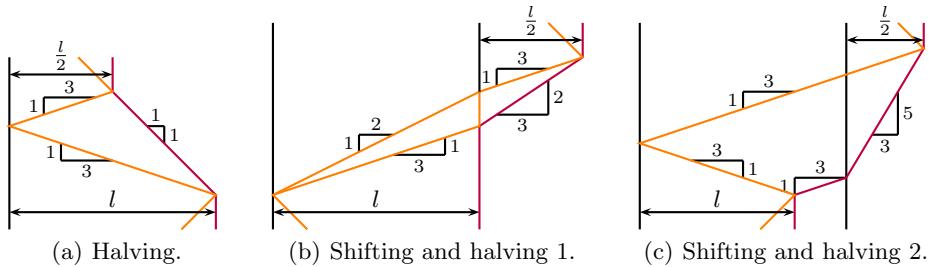
**Real number encoding.** A Real number is encoded as the distance from a **ba** signal<sup>1</sup> to its pairing **val** signal. Since signal machines are scaleless, two **sca** signals whose distance amounts for a scale are provided as depicted on Fig. 1. All numbers use the same scale. For the value 0, the superposition of **ba** and **val** is encoded as a single signal **nul**. This value is never considered in the rest of the paper; the reader is invited to check that it can be easily covered.

<sup>1</sup> **xx** signal always means that it is an instance of the meta-signal **xx**.



**Fig. 1.** Encoding: scale and positions of  $\text{val}$  for values  $-\pi$ ,  $-1.5$ ,  $0$ ,  $\sqrt{2}$  and  $e$ .

**Geometric constructions.** The construction of the multiplication, like the addition and the multiplication by a constant, relies on some basic geometric constructions. In each picture, the slopes of the line segments are indicated. It can easily be checked that it is scale invariant and provides the desired effect. Signals with equal speeds result in parallel segments, the key to many geometric properties. Figure 2(a) shows how a distance can be halved. This is done by starting two signals. The first one is going three times slower than the second one. The time the first one crosses half the way, the second one goes one full way and half way back, so that they meet exactly in the middle. Doubling is done the other way round. Figures 2(b) and 2(c) show two ways to halve a distance while shifting it. They also work with two signals emitted that change speed or direction at some point and meet at the desired location. Generating simple shifts is done by modifying only one slope ( $\frac{2}{3}$  by  $\frac{1}{2}$  for Fig. 2(b) and  $\frac{5}{3}$  by 1 for Fig. 2(b)).



**Fig. 2.** Basic geometric constructions.

## 4 Multiplication

### 4.1 Algorithm

The multiplication of two real numbers  $x$  and  $y$  uses the (possibly) infinite binary extension of  $y$ . This is done in two steps: normalization and then an infinite loop.

The first step starts by considering signs and special cases (i.e. multiplication by zero). Then, while  $y$  is greater than 2,  $x$  is multiplied by 2 and  $y$  divided by 2 (so that the product remains constant).

The second step carries out the multiplication. The binary extension  $y = y_0.y_1y_2y_3\dots$  (the initialization ensures that  $0 < y < 2$ ) is generated bit by bit.

The underlying formula is:

$$xy = \sum_{0 \leq i} y_i \left( \frac{x}{2^i} \right) .$$

This is computed iteratively with the updating on Table 1. The two last cases correspond to  $y_n = 1$  and  $y_n = 0$  respectively. It is started with:  $p_0 = 0$  (product),  $b_0 = 1$  (bit test),  $x_0 = x$  and  $0 < y_0 = y < 2b_0 = 2$ . The following invariants are satisfied:

- $b_n = 2^{-n}$ ,
- $0 \leq y_n < 2b_n$ ,
- $x_n = x2^{-n}$ , and
- $xy = p_n + \frac{x_n y_n}{b_n}$ .

The last invariant is trivially true for  $n = 0$  and preserved by the loop. Since  $x_n \frac{y_n}{b_n} < 2x_n$  and  $x_n = x2^{-n}$ , from the last invariant comes that  $\lim_{n \rightarrow \infty} p_n = xy$ .

**Table 1.** Potentially infinite loop to compute the product.

	$p_{n+1}$	$x_{n+1}$	$y_{n+1}$	$b_{n+1}$
if $y_n = 0$	stop			
else if $b_n < y_n$	$p_n + x_n$	$x_n/2$	$y_n - b_n$	$b_n/2$
else	$p_n$	$x_n/2$	$y_n$	$b_n/2$

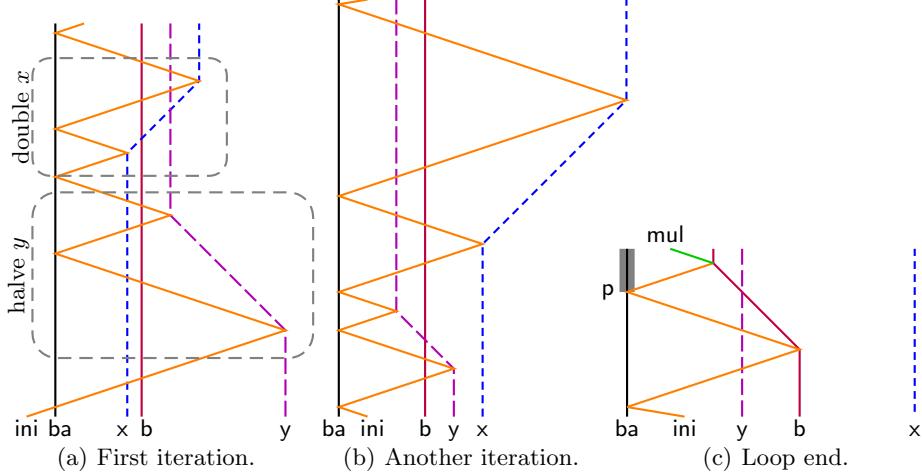
## 4.2 Initialisation

With our encoding, detecting whether  $x$  or  $y$  is zero is trivial. Detecting the signs and computing the sign of the product is also very easy. The following only deals with multiplication of positive values, the other cases are generated using absolute values and symmetry if the product is negative.

The above algorithm should be started with  $0 < y_0 < 2b_0 = 2$ . So that there is a loop that multiplies  $x$  by 2 and divides  $y$  by 2 until  $y$  is small enough. This is illustrated on Fig. 3 (the first two space-time diagrams go one above the other).

The algorithm is sequential: signals for base (**ba** ↗), bit testers (**b** ↗),  $x$  (x ↘) and  $y$  ( $y$  ↘) are fix unless set on movement by some bouncing initialization signals (ini). All distances are from **ba** and are measured according to the “official” scale (not represented). The **b** signal stays at position 2 for testing the end of the loop (i.e.  $y < 2$ ). The signal **ini** comes from the left. If it meets **y** before **b**, this means that the loop is finished (Fig. 3(c)). Otherwise it has to half  $y$ , to double  $x$  and to test again (figures 3(a) and 3(b)).

At the end of initialization (which is always achieved in finite time and finitely many collisions), the **b** at position 2 is set at position 1 (halved as usual). The signal **p** amounting for  $p$  is generated. Since  $p$  is 0 at start, it is set on **ba** (this corresponds to a different meta-signal). And finally, **ini** turns to **mul** that handles the second step of the multiplication. Everything is depicted on Fig. 3(c).



**Fig. 3.** Preparing the data for multiplication.

### 4.3 Main loop

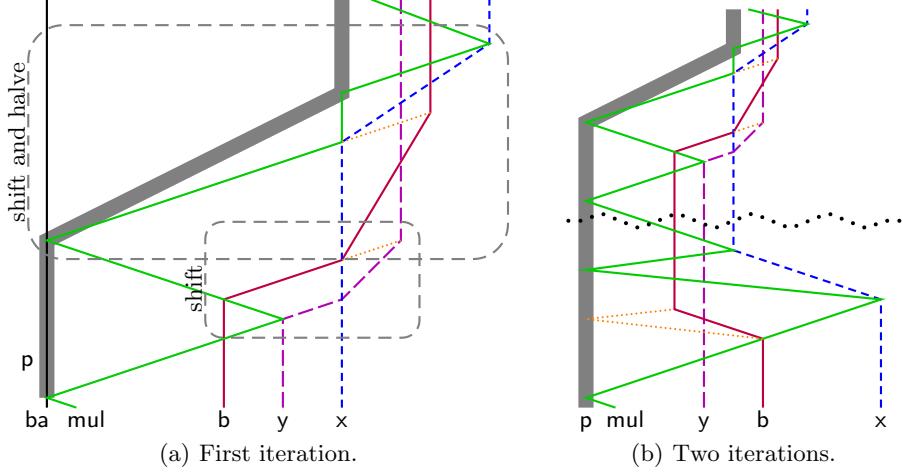
The multiplication follows the (possibly) infinite loop defined in Table 1. Basically, the things to do are additions, subtractions, divisions by 2 and tests. These are easy implementable inside signal machines. But, since the loop is infinite, a *correct* accumulation have to be generated. By correct, it is understood as at the right location and there is indeed an accumulation. The second point is not to be underestimated since, for example, if at each iteration some constant distance would have to be crossed, then there would be an infinite duration process but no accumulation.

The algorithm is driven by a **mul** signal that bounces between **p** and other signals. First of all, **mul** has to test to know which case to consider. This is done easily: going away from **ba**, if **b** is encountered before **y** this means that  $b_n < y_n$  otherwise  $y_n < b_n$  (when they are met simultaneously, i.e. they are equal, this leads to the end of the loop at the next iteration).

The simpler case is when  $y_n < b_n$ . There is nothing to do but to halve  $b_n$  and  $x_n$ , that is halve the distance from **b** and **x** to **ba**. Signals **p** and **y** are left untouched. This is done as depicted on the lower part of Fig. 4(b).

The other case  $b_n < y_n$  is depicted on Fig. 4(a) and on the upper part of Fig. 4(b). The addition of  $x_n$  to  $p_n$  is done by moving **p** to the location of **x**, meanwhile **x** is moved on the right by half the distance it has from **p**. The signal **b** is also moved on the right, at a distance from the new **p** that is half the previous distance. The signal **y** is moved to the right, at a distance from the new **p** that is equal to its original distance from **b**.

To ensure accumulation, all lengths are geometrically scaled down and all the computations take place by **p** and are shifted according to the moves of **p**. This can be seen on Fig. 4. Each iteration leads to halving the distance between **ba**

**Fig. 4.** Multiplication.

and both  $x$  and  $b$ . Since the distance from  $ba$  to  $y$  is at most twice the distance to  $b$ , this ensures that all distances are bounded by some  $\frac{M_s}{2^n}$  at the  $n$ th iteration. Clearly the duration of an iteration is bounded proportionally to the maximum distance between the signals. Thus it is also bounded by some  $\frac{M_t}{2^n}$  at the  $n$ th iteration. There is no time lag between two consecutive iterations, so that there is indeed an accumulation.

## 5 Square root

In this section, it is briefly argued why it is possible to compute the square root with an accumulation and how. Let  $a$  be any positive real number. The construction follows the digit by digit approximation algorithm constructing a sequence  $b_n$  such that:

$$b_n^2 \leq a < \left(b_n + \frac{1}{2^n}\right)^2 .$$

At each stage it should be tested whether  $\left(b_n + \frac{1}{2^{n+1}}\right)^2 \leq a$ . If it is the case then  $b_{n+1} = b_n + \frac{1}{2^{n+1}}$  otherwise  $b_{n+1} = b_n$ . Using the sequences:  $d_n = a - b_n^2$ ,  $e_n = \frac{b_n}{2^n}$  and  $f_n = \frac{1}{4^{n+1}}$ , the test corresponds to computing the sign of

$$a - \left(b_n + \frac{1}{2^{n+1}}\right)^2 = a - b_n^2 - \frac{b_n}{2^n} - \frac{1}{4^{n+1}} = d_n - e_n - f_n .$$

The updating of all these sequences is shown on Table 2 (plus another helpful sequence  $g_n = \frac{1}{2^{n+1}}$ ). The only operations used are additions, multiplications by constants and tests. Again the size of the computing part is decreasing geometrically and computation can be done by the signal encoding the value of  $b_n$  and shifted with it.

The algorithm starts by a pretreatment that finds the initial value for  $n$ . This value might be negative (e.g.  $n = -11$  for  $a = 2^{20} + 1$ ).

**Table 2.** Infinite loop to compute the square root.

	$b_{n+1}$	$d_{n+1}$	$e_{n+1}$	$f_{n+1}$	$g_{n+1}$
if $d_n - e_n - f_n = 0$	stop				
else if $0 < d_n - e_n - f_n$	$b_n + g_n$	$d_n - e_n - f_n$	$e_n/2 + f_n$	$f_n/4$	$g_n/2$
else	$b_n$	$d_n$	$e_n/2$	$f_n/4$	$g_n/2$

## 6 Conclusion

In the present paper, AGC with accumulation is proved to be strictly more powerful than basic BSS. This is not very surprising because it is already known to decide in finite time any recursively enumerable problem (in the classical discrete setting). It would be natural to extends the structure BSS works with (i.e.  $\mathbb{R}$  as a ring) with square rooting but many functions are computable with an accumulation. It would be interesting to identify them. Considering  $\sqrt{2}$ , an accumulation point of a rational signal machine can be irrational.

If the computation is stopped before the accumulation happens, then an approximation is generated. Computable analysis relies on the idea of an infinite approximating sequence both for representing real numbers and for computing (type-2 Turing machine needs an infinite number of iterations to compute a function on real numbers). The next step would be to relate these two models (the spirit of [CH99]). One problem would be to miniaturize and to ensure the generation of an accumulation. Another one is that computable analysis only provides continuous functions, while in ACG, there is, for example, the sign function which is clearly not continuous. On the other side, Moore's recursion theory allows non continuous functions (even the characteristic function of rational numbers).

There might be many accumulations to simulate BSS, but none is of order two. Another issue is to consider  $n$ th order accumulation and connect with infinite Turing machines and ordinals [Ham07].

## References

- [BCGH06] O. Bournez, M. L. Campagnolo, D. S. Graça, and E. Hainry. The general purpose analog computer and computable analysis are two equivalent paradigms of analog computation. In J.-Y. Cai, S. B. Cooper, and A. Li, editors, *Theory and Applications of Models of Computations (TAMC '06), Beijing, China*, number 3959 in LNCS, pages 631–643. Springer, 2006.
- [BCSS98] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and real computation*. Springer, New York, 1998.
- [BH04] O. Bournez and E. Hainry. An analog characterization of elementarily computable functions over the real numbers. In J. Diaz, J. Karhumäki, A. Lepistö, and D. T. Sannella, editors, *31st Int. Col. on Automata, Languages and Programming (ICALP '04), Turku, Finland*, number 3142 in LNCS, pages 269–280. Springer, Jul 2004.
- [Bou99] O. Bournez. Some bounds on the computational power of piecewise constant derivative systems. *Theory of Computing Systems*, 32(1):35–67, 1999.

- [CH99] T. Chadzelek and G. Hotz. Analytic machines. *Theoret. Comp. Sci.*, 219(1-2):151–167, 1999.
- [DL05a] J. Durand-Lose. Abstract geometrical computation for black hole computation (extended abstract). In M. Margenstern, editor, *Machines, Computations, and Universality (MCU '04)*, number 3354 in LNCS, pages 176–187. Springer, 2005.
- [DL05b] J. Durand-Lose. Abstract geometrical computation: Turing-computing ability and undecidability. In B. S. Cooper, B. Löwe, and L. Torenvliet, editors, *New Computational Paradigms, 1st Conf. Computability in Europe (CiE '05)*, number 3526 in LNCS, pages 106–116. Springer, 2005.
- [DL06a] J. Durand-Lose. Abstract geometrical computation 1: embedding black hole computations with rational numbers. *Fund. Inf.*, 74(4):491–510, 2006.
- [DL06b] J. Durand-Lose. Forecasting black holes in abstract geometrical computation is highly unpredictable. In J.-Y. Cai, S. B. Cooper, and A. Li, editors, *Theory and Applications of Models of Computations (TAMC '06)*, number 3959 in LNCS, pages 644–653. Springer, 2006.
- [DL06c] J. Durand-Lose. Reversible conservative rational abstract geometrical computation is turing-universal. In A. Beckmann and J. V. Tucker, editors, *Logical Approaches to Computational Barriers, 2nd Conf. Computability in Europe (CiE '06)*, number 3988 in LNCS, pages 163–172. Springer, 2006.
- [DL07] J. Durand-Lose. Abstract geometrical computation and the linear Blum, Shub and Smale model. In S. Cooper, B. Löwe, and A. Sorbi, editors, *Computation and Logic in the Real World, 3rd Conf. Computability in Europe (CiE '07)*, number 4497 in LNCS, pages 238–247. Springer, 2007.
- [Ham07] J. D. Hamkins. A survey of infinite time turing machines. In J. Durand-Lose and M. Margenstern, editors, *Machines, Computations and Universality (MCA '07)*, number 4664 in LNCS, pages 62–71. Springer, 2007.
- [Huc89] U. Hockenbeck. Euclidian geometry in terms of automata theory. *Theor. Comput. Sci.*, 68(1):71–87, 1989.
- [Huc91] U. Hockenbeck. A result about the power of geometric oracle machines. *Theor. Comput. Sci.*, 88(2):231–251, 1991.
- [Kaw05] A. Kawamura. Type-2 computability and moore’s recursive functions. *Electr. Notes Theor. Comput. Sci.*, 120:83–95, 2005.
- [Moo96] C. Moore. Recursion theory on the reals and continuous-time computation. *Theoret. Comp. Sci.*, 162(1):23–44, 1996.
- [Nov95] E. Novak. The real number model in numerical analysis. *J. Complex.*, 11(1):57–73, 1995.
- [PE74] M. B. Pour-El. Abstract computability and its relation to the general purpose analog computer (some connections between logic, differential equations and analog computers). *Trans. Amer. Math. Soc.*, 199:1–28, 1974.
- [Wei00] K. Weihrauch. *Introduction to computable analysis*. Texts in Theoretical computer science. Springer, Berlin, 2000.

# Notions of Bisimulation for Heyting-Valued Modal Languages

Pantelis E. Eleftheriou<sup>1</sup>, Costas D. Koutras<sup>2</sup>, and Christos Nomikos<sup>3\*</sup>

<sup>1</sup> Institut de Matemàtica, Universitat de Barcelona,  
Gran Via de les Corts Catalanes 585, 08007 Barcelona, Spain  
[pelefthe@gmail.com](mailto:pelefthe@gmail.com)

<sup>2</sup> Department of Computer Science and Technology, University of Peloponnese,  
end of Karaiskaki Street, 22100 Tripolis, Greece  
[ckoutras@uop.gr](mailto:ckoutras@uop.gr)

<sup>3</sup> Department of Computer Science, University of Ioannina,  
45110 Ioannina, Greece  
[cnomikos@cs.uoi.gr](mailto:cnomikos@cs.uoi.gr)

**Abstract.** We define notions of bisimulation for the family of Heyting-valued modal logics introduced by M. Fitting. In this family of logics, each modal language is built on an underlying space of truth values, a Heyting algebra  $\mathcal{H}$ . All the truth values are directly represented in the language, which is interpreted on relational frames with an  $\mathcal{H}$ -valued accessibility relation. We investigate the correct notion of bisimulation in this context: we define two variants of bisimulation relations and derive relative (to a truth value) modal equivalence results for bisimilar states. We further investigate game semantics for our bisimulation, Hennessy-Milner classes and other relevant properties. If the underlying algebra  $\mathcal{H}$  is finite, Heyting-valued modal models can be equivalently reformulated to a form relevant to epistemic situations with many interrelated experts. Our definitions and results draw from this formulation, which is of independent interest to Knowledge Representation applications.

**Key words:** Modal Logic, Many-Valued Logic, Bisimulations.

## 1 Introduction

Bisimulation is a very rich concept which plays an important role in many areas of Computer Science, Logic and Set Theory. Its origins can be found in the analysis of Modal Logic but it was independently rediscovered by computer scientists in their efforts to understand concurrency. In Modal Logic, bisimulations were introduced by Johan van Benthem, under the name of p-relations or zig-zag relations, in the course of his work on the correspondence theory of Modal Logic [vB83,vB84]. In Computer Science, bisimulations were introduced by Park in [Par91] and Hennessy and Milner in [HM85], in the course of investigating the notion of equivalence among processes (see [San07] for a tutorial on the history

---

\* Corresponding author

of bisimulations). In this context, bisimulations represent a fundamental notion of identity between process states and every language designed to capture the essential properties of processes should be blind for bisimilar states.

On the other hand, from the Modal Logic viewpoint, bisimulation is the correct notion of similarity between two modal models: modal formulas are unable to distinguish bisimilar points of the two models. But its importance lies far beyond; the celebrated van Benthem characterization theorem, published in the mid-'70s, states that invariance for bisimulation captures the essential property of the ‘modal fragment’ of first-order logic: *a first-order formula is invariant for bisimulation iff it is (equivalent to the) the syntactical translation of a modal formula* (see [BdRV01] for a nice exposition of this result and its consequences). The characterization theorem has generated an important stream of research in the analysis of logical languages. In particular, the bisimulation-based analysis of modal languages has been extensively studied in the Amsterdam school of modal logic [dR93, Ger99, MV03], and it has even been suggested that this notion is as important for modal logic as the notion of partial isomorphism has been for the model theory of classical logic (see the PhD thesis of M. de Rijke [dR93]). It is worth mentioning that bisimulations have been used beyond the realm of classical modal logic: see for instance the variant used to analyze since-until temporal languages [KdR97], M. Otto’s work related to Finite Model Theory [Ott99] and J. Gerbrandy’s dissertation [Ger99]. Bisimulations have been also used as a fundamental tool in the area of non-well founded set theory ([Acz88], see [BdRV01] for a few details and further references).

In this note, we address the question of what constitutes a suitable notion of bisimulation for the family of many-valued modal languages introduced by M. Fitting in the early ’90s [Fit92, Fit91]. Each language of this family is built on an underlying space of truth values, a Heyting algebra  $\mathcal{H}$ . There exist three features that give these logics their distinctive character. The first one is syntactic: the elements of  $\mathcal{H}$  are directly encoded in the language as special constants and this permits the formation of ‘weak’, uncertainty-oriented versions of the classical modal epistemic actions [Kou03, KNP02, KP02]. The second is semantic: the languages we discuss are interpreted on  $\mathcal{H}$ -labelled directed graphs which provide us a form of many-valued accessibility relation. Finally, the third one concerns the potential applicability of these logics in epistemic situations with multiple intelligent agents. More specifically, assuming that  $\mathcal{H}$  is a finite Heyting algebra, these logics can be formulated in a way that expresses the epistemic consensus of many experts, interrelated through a binary ‘dominance’ relation [Fit92]. It is worth mentioning that, model-theoretically, every complete Heyting algebra can serve as the space of truth values. However, apart from the equivalent multiple-expert formulation of the logics, the finiteness assumption for  $\mathcal{H}$  is essential for the elegant canonical model construction of [Fit92] which leads to a completeness theorem; note that this finiteness restriction seems to be also necessary for obtaining a many-valued analog of the ultrafilter extension construction [EK05].

We provide below two notions of bisimulation and derive modal equivalence results. The first one is a rather strong notion, that allows us to formulate sim-

ple, intuitive, Ehrenfeucht-Fraissé type bisimulation games through which one can easily define bounded bisimulations, as in the classical case. Also, an appropriate notion of unravelling is given, through which one gets a form of the celebrated *tree-model property*, considered to be critical for the ‘robust decidability’ of modal logics [Var97]. A second, rather involved notion of weak bisimulation is discussed which allows us to obtain an interesting notion of Hennessy-Milner class of Heyting-valued modal models. Both notions of bisimulations draw inspiration from the equivalent multiple-expert formulation of these logics, which is actually a mixture of Kripke modal and Kripke intuitionistic semantics. Due to space limitations, this semantics, along with the interpretation of our bisimulation relations in this context, is left for the full paper.

## 2 Many-Valued Modal Languages

In this section we provide the syntax and semantics of many-valued modal languages, as introduced in [Fit92], with only minor changes in the notation. To construct a modal language of this family, we first fix a Heyting algebra  $\mathcal{H}$  which will serve as the space of truth values. Thus, we first briefly expose the basic definitions and properties of Heyting algebras, fixing also notation and terminology. We assume that the reader already has some familiarity with the elements of lattice theory and universal algebra. For more details the reader is referred to the classical texts [RS70, BD74].

*Heyting Algebras* A *lattice*  $\mathcal{L}$  is a pair  $\langle L, \leq \rangle$  consisting of a non-empty set  $L$  equipped with a partial-order relation  $\leq$ , such that every two-element subset  $\{a, b\}$  of  $L$  has a *least upper bound* or *join*, denoted by  $a \vee b$ , and a *greatest lower bound* or *meet*, denoted by  $a \wedge b$ . A lattice  $\mathcal{L}$  is *complete* if a join and a meet exist for *every* subset of  $\mathcal{L}$ . A *least* (or *bottom*) element of a lattice is denoted by  $\perp$  and a *greatest* (or *top*) one by  $\top$ . An element  $x \in L$  is *join-irreducible* if  $x \neq \perp$  (in case  $L$  has a bottom element) and  $x = a \vee b$  implies  $x = a$  or  $x = b$ . We frequently use indexed sets and denote (possibly infinite) meets and joins by  $\bigwedge_{t \in T} a_t$  and  $\bigvee_{t \in T} a_t$ . Some fairly obvious properties of infinite joins and meets, such as  $\bigwedge_{t \in T} (a \wedge a_t) = a \wedge \bigwedge_{t \in T} a_t$  will be used, generally without comment.

A lattice  $\mathcal{H} = \langle H, \leq \rangle$  with the additional property that, for every pair of elements  $\langle a, b \rangle$ , the set  $\{x \mid a \wedge x \leq b\}$  has a greatest element, is called a *relatively pseudo-complemented lattice*. This element is denoted by  $a \Rightarrow b$  and is called the *pseudo-complement of a relative to b*. A relatively pseudo-complemented lattice is always a topped ordered set [RS70]. It is not always the case that a relatively pseudo-complemented lattice has a least element. A relatively pseudo-complemented lattice  $\mathcal{H}$  with a least element is called a **Heyting algebra** (HA) or a **pseudo-Boolean algebra**. It is known that the class of HAs includes the class of Boolean algebras and is included in the class of distributive lattices; both inclusions are proper. For finite lattices, the second inclusion becomes an equality: the class of finite HAs coincides with the class of finite distributive lattices [RS70]. The following lemma gathers some useful properties of relatively

pseudocomplemented lattices that will be used in Section 3; whenever a possibly infinite join or meet is involved, it is assumed that it exists. The proof of its items can be found in [RS70, BD74]. Note also that the first item of the lemma can be equivalently considered as a definition of relative pseudo-complementation.

- Lemma 1.**
1.  $x \leq (a \Rightarrow b) \text{ iff } (x \wedge a) \leq b$
  2.  $(a \Rightarrow b) = \top \text{ iff } a \leq b$
  3. If  $a_1 \leq a_2$  then  $(a_2 \Rightarrow b) \leq (a_1 \Rightarrow b)$
  4.  $c \wedge (a \Rightarrow b) = c \wedge ((c \wedge a) \Rightarrow (c \wedge b))$
  5.  $\bigvee_{t \in T} (a \wedge b_t) = a \wedge \bigvee_{t \in T} b_t$
  6.  $(a \vee b) \Rightarrow c = (a \Rightarrow c) \wedge (b \Rightarrow c)$

*Syntax of Many-Valued Modal Languages* Having fixed a complete Heyting algebra  $\mathcal{H}$  we proceed to define the syntax of the modal language. The elements of  $\mathcal{H}$  are directly represented in the language by special constants, called *propositional constants*, and we reserve lowercase letters (along with  $\perp, \top$ ) to denote them. To facilitate notation, we use the same letter for the element of  $\mathcal{H}$  and the constant which represents it in the language; context will clarify what is meant. Assuming also a set  $\Phi$  of *propositional variables* (also called propositional letters) we define the many-valued modal language  $L_{\square\Diamond}^{\mathcal{H}}(\Phi)$  with the following BNF specification, where  $t$  ranges over elements of  $\mathcal{H}$ ,  $P$  ranges over elements of  $\Phi$  and  $A$  is a formula of  $L_{\square\Diamond}^{\mathcal{H}}(\Phi)$ .

$$A ::= t \mid P \mid A_1 \vee A_2 \mid A_1 \wedge A_2 \mid A_1 \supset A_2 \mid \Box A \mid \Diamond A$$

As we will see below, the modal logics defined are in general bimodal, thus we need both modal operators. Note also that  $\vee$  and  $\wedge$  serve both as logical connectives, as well as lattice operation symbols but it should be clear by context what is meant. In the rest of the paper, we shall often omit  $\Phi$  when possible and speak of the language  $L_{\square\Diamond}^{\mathcal{H}}$ . A (non-classical) negation  $\neg X$  can be defined as  $(X \supset \perp)$ .

*Semantics of Many-Valued Modal Languages*  $L_{\square\Diamond}^{\mathcal{H}}$  is interpreted on an interesting variant of a relational frame, which possesses a kind of Heyting-valued accessibility relation. Note that there have been other approaches in the literature for defining many-valued modal logics, but all of them have kept the essence of classical relational semantics intact (see [Fit92] for references). Given  $L_{\square\Diamond}^{\mathcal{H}}(\Phi)$ , we define  $\mathcal{H}$ -modal frames and  $\mathcal{H}$ -modal models as follows:

**Definition 1.** An  $\mathcal{H}$ -modal frame for  $L_{\square\Diamond}^{\mathcal{H}}(\Phi)$  is a pair  $\mathfrak{F} = \langle \mathfrak{S}, \mathfrak{g} \rangle$ , where  $\mathfrak{S}$  is a non-empty set of states and  $\mathfrak{g} : \mathfrak{S} \times \mathfrak{S} \rightarrow \mathcal{H}$  is a total function mapping pairs of states to elements of  $\mathcal{H}$ .

An  $\mathcal{H}$ -modal model  $\mathfrak{M} = \langle \mathfrak{S}, \mathfrak{g}, v \rangle$  is built on  $\mathfrak{F}$  by providing a valuation  $v$ , that is a function  $v : \mathfrak{S} \times (\mathcal{H} \cup \Phi) \rightarrow \mathcal{H}$  which assigns a  $\mathcal{H}$ -truth value to atomic formulae in each state, such that  $v(s, t) = t$ , for every  $s \in \mathfrak{S}$  and  $t \in \mathcal{H}$ . In other words, the propositional constants are always mapped to ‘themselves’.

In the sequel, we shall often omit the adjective ‘modal’ and talk simply of  $\mathcal{H}$ -frames and  $\mathcal{H}$ -models.

The valuation  $v$  extends to all the formulae of  $L_{\square\Diamond}^{\mathcal{H}}(\Phi)$  in a standard recursive fashion:

**Definition 2.** Let  $\mathfrak{M} = \langle \mathfrak{S}, \mathbf{g}, v \rangle$  be an  $\mathcal{H}$ -model and  $\mathbf{s}$  a state of  $\mathfrak{S}$ . The extension of the valuation  $v$  to the whole language  $L_{\square\Diamond}^{\mathcal{H}}(\Phi)$  is given by the following clauses;

$$\begin{aligned} - v(\mathbf{s}, A \wedge B) &= v(\mathbf{s}, A) \wedge v(\mathbf{s}, B) \\ - v(\mathbf{s}, A \vee B) &= v(\mathbf{s}, A) \vee v(\mathbf{s}, B) \\ - v(\mathbf{s}, A \supset B) &= v(\mathbf{s}, A) \Rightarrow v(\mathbf{s}, B) \\ - v(\mathbf{s}, \Box A) &= \bigwedge_{\mathbf{t} \in \mathfrak{S}} (\mathbf{g}(\mathbf{s}, \mathbf{t}) \Rightarrow v(\mathbf{t}, A)) \\ - v(\mathbf{s}, \Diamond A) &= \bigvee_{\mathbf{t} \in \mathfrak{S}} (\mathbf{g}(\mathbf{s}, \mathbf{t}) \wedge v(\mathbf{t}, A)) \end{aligned}$$

The operators of *necessity* ( $\Box$ ) and *possibility* ( $\Diamond$ ) are not each other’s dual, unless  $\mathcal{H}$  is a Boolean algebra [Fit92]. Note also that all the definitions above collapse to the familiar ones from the classical case, in the case of the classical language  $L_{\square\Diamond}^{\mathbf{2}}$ , where  $\mathbf{2}$  is the lattice of two-valued classical logic.

### 3 Bisimulations for Many-Valued Modal Languages

In this section, we define two suitable general notions of bisimulation for a language  $L_{\square\Diamond}^{\mathcal{H}}$  of the family defined in the previous section. Before proceeding, we have to define a refined notion of modal truth invariance which fits our aims and which also has an interesting interpretation in the multiple-expert context. Note that the following notion is trivial for  $t = \perp$ .

**Definition 3 ( $t$ -invariance).** Let  $\mathfrak{M} = \langle \mathfrak{S}, \mathbf{g}, v \rangle$  and  $\mathfrak{M}' = \langle \mathfrak{S}', \mathbf{g}', v' \rangle$  be  $\mathcal{H}$ -models for  $L_{\square\Diamond}^{\mathcal{H}}(\Phi)$ ,  $\mathbf{s} \in \mathfrak{S}$  and  $\mathbf{s}' \in \mathfrak{S}'$  two states and  $t \in \mathcal{H}$  a truth value ( $t \neq \perp$ ). We say that modal truth is  *$t$ -invariant for the transition between  $\mathbf{s}$  and  $\mathbf{s}'$*  if for every  $X \in L_{\square\Diamond}^{\mathcal{H}}(\Phi)$

$$t \wedge v(\mathbf{s}, X) = t \wedge v'(\mathbf{s}', X)$$

#### 3.1 Strong Bisimulation for Many-Valued Modal Languages

The following definition captures the idea of moving ‘back and forth’ between two  $\mathcal{H}$ -models by matching steps (‘modulo’  $t$ ) in both directions.

**Definition 4 ( $t$ -bisimulations).** Given two  $\mathcal{H}$ -models  $\mathfrak{M} = \langle \mathfrak{S}, \mathbf{g}, v \rangle$  and  $\mathfrak{M}' = \langle \mathfrak{S}', \mathbf{g}', v' \rangle$  and a truth value  $t \in \mathcal{H}$  ( $t \neq \perp$ ), a non-empty relation  $Z \subseteq \mathfrak{S} \times \mathfrak{S}'$  is a  *$t$ -bisimulation* between  $\mathfrak{M}$  and  $\mathfrak{M}'$  if for any pair  $\langle \mathbf{s}, \mathbf{s}' \rangle \in Z$

- (base)  $t \wedge v(\mathfrak{s}, P) = t \wedge v'(\mathfrak{s}', P)$  for every  $P \in \Phi$
  - (forth) for every  $\mathfrak{r} \in \mathfrak{S}$  such that  $t \wedge g(\mathfrak{s}, \mathfrak{r}) \neq \perp$ ,
    - there exists an  $\mathfrak{r}' \in \mathfrak{S}'$  such that  $t \wedge g(\mathfrak{s}, \mathfrak{r}) = t \wedge g'(\mathfrak{s}', \mathfrak{r}')$  and  $\mathfrak{r} Z \mathfrak{r}'$
  - (back) for every  $\mathfrak{r}' \in \mathfrak{S}'$  such that  $t \wedge g'(\mathfrak{s}', \mathfrak{r}') \neq \perp$ ,
    - there exists an  $\mathfrak{r} \in \mathfrak{S}$  such that  $t \wedge g'(\mathfrak{s}', \mathfrak{r}') = t \wedge g(\mathfrak{s}, \mathfrak{r})$  and  $\mathfrak{r} Z \mathfrak{r}'$
- Two states  $\mathfrak{s}$  and  $\mathfrak{s}'$  are called *t-bisimilar* (notation  $\mathfrak{s} \sqsubseteq_t \mathfrak{s}'$  or  $\mathfrak{M}, \mathfrak{s} \sqsubseteq_t \mathfrak{M}', \mathfrak{s}'$ ) if there is a *t*-bisimulation  $Z$  between  $\mathfrak{M}$  and  $\mathfrak{M}'$  such that  $\mathfrak{s} Z \mathfrak{s}'$ .

We can now prove the basic theorem which states that *t*-bisimulation implies *t*-invariance.

**Theorem 1.** *If  $\mathfrak{M}, \mathfrak{s} \sqsubseteq_t \mathfrak{M}', \mathfrak{s}'$ , then  $t \wedge v(\mathfrak{s}, X) = t \wedge v'(\mathfrak{s}', X)$  for every  $X \in L_{\square \diamond}^{\mathcal{H}}$ .*

PROOF. The proof is left for the full version of the paper. ■

*EF-type games for t-bisimulation* The *t*-bisimulation game is a simple variant of the Ehrenfeucht-Fraissé game played in First-Order Logic. For the purposes of the rest of this section call a state  $\mathfrak{r}$  a *t-compatible successor* state of  $\mathfrak{s}$  if  $t \wedge g(\mathfrak{s}, \mathfrak{r}) \neq \perp$ . Two elements  $a, a'$  of  $\mathcal{H}$  are called *t<sub>Λ</sub>-equivalent* if  $t \wedge a = t \wedge a'$ . We call *labels* the  $\mathcal{H}$ -truth values attached to the graph's edges and to the propositional letters of the language in each possible world. The *t*-bisimulation game is played on two *pointed*  $\mathcal{H}$ -models (models with a single distinguished state)  $\mathfrak{M}, \mathfrak{s}_0$  and  $\mathfrak{M}', \mathfrak{s}'_0$ . There exists one marked element in each  $\mathcal{H}$ -model; initially, the marked elements are the distinguished nodes  $\mathfrak{s}_0$  and  $\mathfrak{s}'_0$ . In each *round* of the game

- **Player I** selects one of the  $\mathcal{H}$ -models, chooses a *t*-compatible successor of the marked element and moves the marker along the edge (labelled by  $a_I$ ) to its target
- **Player II** responds with a move of the marker in the other  $\mathcal{H}$ -model in a corresponding *t*-compatible transition (labelled by  $a_{II}$ ) such that  $a_I$  and  $a_{II}$  are *t<sub>Λ</sub>*-equivalent and the labels of the propositional letters in the marked elements (states) of the models are also *t<sub>Λ</sub>*-equivalent

The *length* of the game is the (finite or infinite) number of rounds and Player II loses the *match* if at a certain round cannot respond with an appropriate move. It is obvious that Player I is trying to spoil a *t*-bisimulation while Player II is trying to reveal one. Player II has a *winning strategy* in a game of  $n$  rounds if she can win every  $n$ -round game played on  $\mathfrak{M}, \mathfrak{s}_0$  and  $\mathfrak{M}', \mathfrak{s}'_0$ . In a classical fashion, we can proceed to a finer analysis of *t*-bisimulations using the inductively defined notion of the modal depth of a formula (the maximum number of modal operators encountered in a subformula, [BdRV01,MV03]). The notion of a *t-bisimulation bounded by a positive integer n*, or any ordinal actually, can easily be defined (see [Ger99, Chapter 2.1] for the classical case), but we will not give further details here, since the whole construction is identical to the classical one. We only provide the following proposition which generalizes the known classical results from two-valued modal logic:

**Proposition 1.** 1. Player II has a winning strategy in the  $n$ -round game played on  $\mathfrak{M}, \mathfrak{s}_0$  and  $\mathfrak{M}', \mathfrak{s}'_0$  iff modal truth is  $t$ -invariant in  $\mathfrak{s}_0$  and  $\mathfrak{s}'_0$  for every formula up to modal depth  $n$ .  
 2. Player II has a winning strategy for the infinite game played on  $\mathfrak{M}, \mathfrak{s}_0$  and  $\mathfrak{M}', \mathfrak{s}'_0$  iff  $\mathfrak{s}_0 \leftrightharpoons_t \mathfrak{s}'_0$ .

PROOF. The proof of the first item runs by induction on  $n$  and is actually a restatement of the proof of Theorem 1. The second item follows by the definitions above. ■

The  $t$ -bisimulation games can be formulated in a simple way for the class of languages built on finite linear orders. Assuming further that truth values are colours, linearly ordered, and given that the meet operation is simple in finite chains ( $a \wedge b = \min(a, b)$ ) the game can be described in an easy way that provides also an element of fun.

*t-unravellings and the tree-model property* The idea of unravelling a model into a modally-equivalent tree model is known both from modal logic and the theory of processes. In the latter field, the states of the unravelled model represent *traces (histories)* of processes, starting from a state  $\mathfrak{s}$ . The following definition provides the many-valued analog of this notion.

**Definition 5.** Given a pointed model  $\mathfrak{M} = \langle \mathfrak{S}, \mathfrak{g}, v \rangle, \mathfrak{s}_1$ , its *t-unravelling* is the model  $\mathfrak{M}_{\mathfrak{s}_1}^u = \langle \mathfrak{S}_{\mathfrak{s}_1}^u, \mathfrak{g}_{\mathfrak{s}_1}^u, v_{\mathfrak{s}_1}^u \rangle$ , where

1.  $\mathfrak{S}_{\mathfrak{s}_1}^u$  consists of all tuples  $\langle \mathfrak{s}_1, \dots, \mathfrak{s}_k \rangle$  where  $\mathfrak{s}_{i+1}$  is a  $t$ -compatible successor of  $\mathfrak{s}_i$
2.  $\mathfrak{g}_{\mathfrak{s}_1}^u(\langle \mathfrak{s}_1, \dots, \mathfrak{s}_k \rangle, \langle \mathfrak{s}_1, \dots, \mathfrak{s}_{k+1} \rangle) = t \wedge \mathfrak{g}(\mathfrak{s}_k, \mathfrak{s}_{k+1})$ , and  $\perp$  for any other pair of tuples
3.  $v_{\mathfrak{s}_1}^u(\langle \mathfrak{s}_1, \dots, \mathfrak{s}_k \rangle, P) = t \wedge v(\mathfrak{s}_k, P)$ , ( $P \in \Phi$ )

Obviously  $\mathfrak{M}_{\mathfrak{s}_1}^u$  is a tree model and the following proposition can be proved by a careful inspection on the definition of a  $t$ -bisimulation.

**Proposition 2.** The graph of the function from  $\mathfrak{S}_{\mathfrak{s}_1}^u$  to  $\mathfrak{S}$ , which maps every tuple to its last component (and  $\langle \mathfrak{s}_1 \rangle$  to  $\mathfrak{s}_1$ ) is a  $t$ -bisimulation.

Thus, modal truth is  $t$ -invariant for the transition from  $\mathfrak{s}_1$  to the root  $\langle \mathfrak{s}_1 \rangle$  of the tree and this is a generalized version of the *tree model property* [Var97].

*Satisfiability in Many-Valued Modal Languages* The general satisfiability problem in this context can be phrased as follows: given  $X \in L_{\square}^{\mathcal{H}}$  and  $t \in \mathcal{H}$ , is there a state  $\mathfrak{s}$  of an  $\mathcal{H}$ -model  $\mathfrak{M}$  in which  $v(\mathfrak{s}, X) \geq t$ ? This is equivalent (by Lemma 1(2)) to  $t \Rightarrow v(\mathfrak{s}, X) = \top$  which is equivalent (by Def. 2) to  $v(\mathfrak{s}, t \supset X) = \top$ . Thus, the general satisfiability problem is subsumed by the question of finding a state in which a formula is mapped to the top element of the lattice. By the previous paragraph, if such a state/model exists, then this formula can be also satisfied at the root of a ( $\top$ -unravelled) tree. Imitating the

classical arguments ([MV03]), it is easy to prove that, if  $\mathcal{H}$  is finite, every formula can be satisfied in a finite tree whose size is bounded: its depth is bounded by the modal depth of  $X$  and its branching degree is bounded by the number of box and diamond subformulas of  $x$ . This leads to a simple way of proving the fact that the many-valued analog of the system **K** (which is determined by the class of all  $\mathcal{H}$ -models [Fit92]) has a decidable general satisfiability problem.

### 3.2 Weak Bisimulations for Many-Valued Modal Languages

We proceed now to define, a weaker, more fine-grained notion of bisimulation that is directly inspired from (and can be better explained in the context of) the multiple-expert semantics of these languages. We first fix some notation.

Let  $I_{\mathcal{H}}$  denote the set of join-irreducible elements of  $\mathcal{H}$ . For the rest of this section, we fix a complete Heyting algebra  $\mathcal{H}$  that has the following property:

Every  $t \in \mathcal{H} - I_{\mathcal{H}}$  is equal to the join of a finite number of elements in  $I_{\mathcal{H}}$  (1)

Define the function  $D_{\mathcal{H}}$  from  $\mathcal{H} - \{\perp\}$  to  $2^{I_{\mathcal{H}}}$ , such that  $D_{\mathcal{H}}(t) = \{c \in I_{\mathcal{H}} \mid c \leq t\}$ . Using Property (1), we see that  $t = \bigvee_{c \in D_{\mathcal{H}}(t)} c$ . Intuitively,  $D_{\mathcal{H}}$  provides a decomposition of a value  $t \in \mathcal{H}$  into join-irreducible values. In the next definition, a bisimulation relation is defined for every truth value, but in a way that it is “upwards (with respect to the lattice of truth values) consistent”.

**Definition 6 (Weak bisimulation).** Given two  $\mathcal{H}$ -models  $\mathfrak{M} = \langle \mathfrak{S}, \mathfrak{g}, v \rangle$  and  $\mathfrak{M}' = \langle \mathfrak{S}', \mathfrak{g}', v' \rangle$ , a function  $Z$  from  $\mathcal{H} - \{\perp\}$  to  $2^{\mathfrak{S} \times \mathfrak{S}'}$  is a *weak bisimulation between  $\mathfrak{M}$  and  $\mathfrak{M}'$*  if it satisfies the following properties:

- for every  $t_1, t_2 \in \mathcal{H}$ 
  - (*consistency*)  $Z(t_1 \vee t_2) = Z(t_1) \cap Z(t_2)$
- for every join-irreducible value  $t \in I_{\mathcal{H}}$  and any pair  $\langle \mathfrak{s}, \mathfrak{s}' \rangle \in Z(t)$ 
  - (*base*)  $t \wedge v(\mathfrak{s}, P) = t \wedge v'(\mathfrak{s}', P)$  for every  $P \in \Phi$
  - (*forth*) for every  $\mathfrak{r} \in \mathfrak{S}$  such that  $t \wedge \mathfrak{g}(\mathfrak{s}, \mathfrak{r}) \neq \perp$ 
    - and for every  $c \in D_{\mathcal{H}}(t \wedge \mathfrak{g}(\mathfrak{s}, \mathfrak{r}))$ ,
    - there exists an  $\mathfrak{r}' \in \mathfrak{S}'$  such that  $c \leq \mathfrak{g}'(\mathfrak{s}', \mathfrak{r}')$  and  $\langle \mathfrak{r}, \mathfrak{r}' \rangle \in Z(c)$
  - (*back*) for every  $\mathfrak{r}' \in \mathfrak{S}'$  such that  $t \wedge \mathfrak{g}'(\mathfrak{s}', \mathfrak{r}') \neq \perp$ 
    - and for every  $c \in D_{\mathcal{H}}(t \wedge \mathfrak{g}'(\mathfrak{s}', \mathfrak{r}'))$ ,
    - there exists an  $\mathfrak{r} \in \mathfrak{S}$  such that  $c \leq \mathfrak{g}(\mathfrak{s}, \mathfrak{r})$  and  $\langle \mathfrak{r}, \mathfrak{r}' \rangle \in Z(c)$

Two states  $\mathfrak{s}$  and  $\mathfrak{s}'$  are called *weakly  $t$ -bisimilar* (notation  $\mathfrak{s} \rightsquigarrow_t \mathfrak{s}'$  or  $\mathfrak{M}, \mathfrak{s} \rightsquigarrow_t \mathfrak{M}', \mathfrak{s}'$ ) if there is a weak bisimulation  $Z$  between  $\mathfrak{M}$  and  $\mathfrak{M}'$  such that  $\langle \mathfrak{s}, \mathfrak{s}' \rangle$  belongs to  $Z(t)$ .

The reader can check that we have indeed defined a weaker notion than that of a  $t$ -bisimulation:  $\mathfrak{M}, \mathfrak{s} \sqsubseteq_t \mathfrak{M}', \mathfrak{s}'$  implies  $\mathfrak{M}, \mathfrak{s} \rightsquigarrow_t \mathfrak{M}', \mathfrak{s}'$ .

The basic theorem of the previous section is still valid under this new notion, but the proof requires some more elaboration.

**Theorem 2.** *If  $\mathfrak{M}, \mathfrak{s} \rightsquigarrow_t \mathfrak{M}', \mathfrak{s}'$ , then  $t \wedge v(\mathfrak{s}, X) = t \wedge v'(\mathfrak{s}', X)$  for every  $X \in L_{\square \diamond}^{\mathfrak{N}}$ .*

PROOF. The proof is left for the full version of the paper. ■

*Image-finite  $\mathcal{H}$ -models and weak  $t$ -bisimulations* One of the fundamental questions in the bisimulation-based analysis of modal languages, concerns the identification of cases in which the converse of Theorem 2 is true. Much obviously, it is not always true: the classical counterexample of two tree models, both with a finite branch for each natural number, one of which possesses an infinite branch, suffices (cf. [BdRV01, Chapter 2.2]). The simplest example of Hennessy-Milner classes of modal models (classes in which modal equivalence is itself a bisimulation relation) is the class of *image-finite* models, in which each state has only a finite number of successors. It is natural to consider a straightforward many-valued analog of this notion by considering  $\mathcal{H}$ -models in which for each state  $s$ , the set of successors of  $s$  is always finite and check whether in this case  $t$ -invariance implies  $t$ -bisimilarity. Formally, the notion of image-finite  $\mathcal{H}$ -models is defined as follows.

**Definition 7 (Image-finite  $\mathcal{H}$ -models).** An  $\mathcal{H}$ -model  $\mathfrak{M} = \langle S, g, v \rangle$  is called *image-finite* if for every  $s \in S$ , the set  $S_s = \{r \in S \mid g(s, r) \neq \perp\}$  is finite.

The following theorem states that for image-finite  $\mathcal{H}$ -models,  $t$ -invariance implies  $t$ -bisimilarity.

**Theorem 3.** Let  $\mathfrak{M} = \langle S, g, v \rangle$  and  $\mathfrak{M}' = \langle S', g', v' \rangle$  be image-finite  $\mathcal{H}$ -models for  $L_{\square\Diamond}^{\mathcal{H}}(\Phi)$ . Define the function  $Z$  from  $\mathcal{H} - \{\perp\}$  to  $2^{S \times S'}$  so that for every  $s \in S$ , every  $s' \in S'$  and every  $t \in \mathcal{H} - \{\perp\}$ ,  $\langle s, s' \rangle \in Z(t)$  iff modal truth is  $t$ -invariant for the transition between  $s$  and  $s'$ . Then  $Z$  is a weak bisimulation between  $\mathfrak{M}$  and  $\mathfrak{M}'$ .

PROOF. The proof is left for the full version of the paper. ■

## 4 Conclusions - Related Work

In this paper, we have contributed to the extensive literature on the importance and the fundamental nature of bisimulation. Our main aim has been to define a fine-grained notion of bisimulation for Heyting-valued modal languages and establish its basic facts. Our results have an interesting meaning for Knowledge Representation situations, when interpreted in the multiple-expert context. It remains to investigate appropriate extension of smallest and largest bisimulations in this context and address possible applications for Knowledge Engineering in complex epistemic situations.

## References

- [Acz88] P. Aczel. *Non-Well-Founded Sets*. CSLI Publications, 1988.
- [BD74] R. Balbes and Ph. Dwinger. *Distributive Lattices*. University of Missouri Press, 1974.
- [BdRV01] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Number 53 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2001.

- [DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.
- [dR93] M. de Rijke. *Extending Modal Logic*. PhD thesis, Institute for Logic, Language and Computation, University of Amsterdam, 1993.
- [EK05] P. Eleftheriou and C. D. Koutras. Frame constructions, truth invariance and validity preservation in many-valued modal logic. *Journal of Applied Non-Classical Logics*, 15(4):367–388, 2005.
- [Fit91] M. C. Fitting. Many-valued modal logics. *Fundamenta Informaticae*, 15:235–254, 1991.
- [Fit92] M. C. Fitting. Many-valued modal logics II. *Fundamenta Informaticae*, 17:55–73, 1992.
- [Ger99] J. D. Gerbrandy. *Bisimulations on Planet Kripke*. PhD thesis, Institute for Logic, Language and Computation, University of Amsterdam, 1999.
- [GG84] D. M. Gabbay and F. Guenther, editors. *Handbook of Philosophical Logic*. D. Reidel, Dordrecht, 1984.
- [HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32:137–162, 1985.
- [KdR97] N. Kurtonina and M. de Rijke. Bisimulations for temporal logic. *Journal of Logic, Language and Information*, 6:403–425, 1997.
- [KNP02] C. D. Koutras, Ch. Nomikos, and P. Peppas. Canonicity and completeness results for many-valued modal logics. *Journal of Applied Non-Classical Logics*, 12(1):7–41, 2002.
- [Kou03] C. D. Koutras. A catalog of weak many-valued modal axioms and their corresponding frame classes. *Journal of Applied Non-Classical Logics*, 13(1):47–72, 2003.
- [KP02] C. D. Koutras and P. Peppas. Weaker axioms, more ranges. *Fundamenta Informaticae*, 51(3):297–310, 2002.
- [MV03] M. Marx and Y. Venema. Local variations on a loose theme: Modal logic and decidability. In M. Y. Vardi and Sc. Weinstein, editors, *Finite Model Theory and its Applications*. Springer Verlag, 2003. To appear.
- [Ott99] M. Otto. Bisimulation-invariant Ptime and higher-dimensional mu-calculus. *Theoretical Computer Science*, 224:237–265, 1999.
- [Par91] D. Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.
- [RS70] H. Rasiowa and R. Sikorski. *The Mathematics of Metamathematics*. PWN - Polish Scientific Publishers, Warsaw, third edition, 1970.
- [San07] D. Sangiorgi. *On the origins of Bisimulation, Coinduction, and Fixed Points*. Technical Report UBLCS-2007-24, Department of Computer Science, University of Bologna, 2007 (available at: [http://www.cs.unibo.it/~sangio/DOC\\_public/history\\_bis\\_coind.pdf](http://www.cs.unibo.it/~sangio/DOC_public/history_bis_coind.pdf)).
- [Var97] M. Y. Vardi. Why is modal logic so robustly decidable? volume 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 561–572. AMS, 1997.
- [vB83] J. van Benthem. *Modal Logic and Classical Logic*. Bibliopolis, Naples, 1983.
- [vB84] J. van Benthem. Correspondence Theory, pages 167–247. Volume 2 of Gabbay and Guenther [GG84], 1984.

# Algorithmic Properties of Structures for Languages with Two Unary Functional Symbols

Ekaterina B. Fokina\*

Sobolev Institute of Mathematics  
Siberian Branch of the Russian Academy of Sciences  
4 Acad. Koptyug avenue  
630090 Novosibirsk Russia  
[e\\_fokina@math.nsc.ru](mailto:e_fokina@math.nsc.ru)

**Abstract.** When studying algorithmic properties of structures with interesting algebraic and model-theoretic properties, one often uses known structural properties of the structures. However, it is often the case that results on particular kinds of structures can be transferred to structures from many other interesting classes. One of the ways of such generalization involves coding of the original structure into a structure from the given class in a way that is effective enough to preserve interesting algorithmic properties. There are several constructions that allow us to reduce algorithmic questions for arbitrary structures to graphs. They also show that if we have a result for a graph, we also have it for a structure for any language containing at least one  $n$ -ary relational symbol, where  $n \geq 2$ . We prove that it is possible to generalize this approach and get the same results for structures for a language with two unary functional symbols. Thus, we get the results for structures for so-called rich languages.

## 1 Introduction

When studying algorithmic properties of structures with interesting algebraic and model-theoretic properties, one often uses known structural properties of the structures. To find and study such connections between computability-theoretic and algebraic properties of structures is one of the main questions of computable model theory. However, it is often the case that results on particular kinds of structures can be transferred to structures from many other interesting classes. One of the ways of such generalization involves coding of the original structure into a structure from the given class in a way that is effective enough to preserve interesting algorithmic properties. This approach is very well illustrated in [14] and [21]. Both papers contain a very nice survey on the known results about

---

\* The author was partially supported by the Russian Foundation for Basic Research (Grant 08-01-00336), the State Maintenance Program for the Leading Scientific Schools of the Russian Federation (Grant NSh-335.2008.1.) and Russian Science Support Foundation.

different computability-theoretic properties of many classes of structures and present several ways to transfer them to classes for various languages.

We introduce some basic definitions. We fix a computable Gödel numbering of a language  $\mathcal{L}$ . Let all structures have universes contained in  $\omega$ , which we think of as computable sets of constants.

**Definition 1.** *A structure  $\mathcal{A}$  of the language  $\mathcal{L}$  is computable if its domain is a computable subset of  $\omega$  and all basic operations and predicates are uniformly computable. We identify formulas with their Gödel numbers. Then computability of a structure is equivalent to the condition that the atomic diagram  $\mathcal{D}(\mathcal{A})$  of  $\mathcal{A}$  is computable, where  $\mathcal{D}(\mathcal{A})$  is the set of all atomic sentences and negations of atomic sentences with parameters in  $|\mathcal{A}|$  true in  $\mathcal{A}$ . A structure  $\mathcal{B}$  has a computable presentation if it is isomorphic to a computable structure  $\mathcal{A}$ . In this case we also say that  $\mathcal{B}$  is computably presentable. A structure  $\mathcal{A}$  is decidable if its complete diagram  $\mathcal{D}^c(\mathcal{A})$  is computable, where  $\mathcal{D}^c(\mathcal{A})$  is the set of all sentences with parameters in  $|\mathcal{A}|$  true in  $\mathcal{A}$ .*

**Definition 2.** *A theory  $T$  is decidable if it is a computable set of sentences. A complete theory  $T$  is  $\alpha$ -categorical if any two models of  $T$  of the power  $\alpha$  are isomorphic. A structure is  $\alpha$ -categorical if its theory is  $\alpha$ -categorical.*

There are different approaches to study algorithmic properties of computable structures. Here we list only some results illustrating possible ways to study computability-theoretic properties. More detailed survey can be found in [14] and [21].

One of the approaches is to study computable presentations of the structure up to  $\mathbf{d}$ -computable isomorphism.

**Definition 3.** *Given a degree  $\mathbf{d}$ , the  $\mathbf{d}$ -computable dimension of a computably presentable structure  $\mathcal{A}$  is the number of computable presentations of  $\mathcal{A}$  up to  $\mathbf{d}$ -computable isomorphism. If  $\mathcal{A}$  has  $\mathbf{d}$ -computable dimension 1 then it is  $\mathbf{d}$ -computably categorical.*

There are examples of all computable dimensions  $1 \leq n \leq \omega$ . Moreover, many of such examples were found in well-known classes of algebraic structures. See, for example, [10], [11], [12], [13], [15], [18], [24], [26].

A related question is to look at the computable dimension of a computably categorical structure when it is expanded by finitely many constants. It was shown that in this case the computable categoricity is not always preserved.

**Theorem 1 (Cholak, Goncharov, Khoussainov, and Shore in [5]).** *For each  $k > 0$  there exist a computably categorical structure  $\mathcal{A}$  and an  $a \in |\mathcal{A}|$  such that  $(\mathcal{A}, a)$  has computable dimension  $k$ .*

**Theorem 2 (Hirschfeldt, Khoussainov, and Shore in [20]).** *There are a computably categorical structure  $\mathcal{A}$  and an  $a \in |\mathcal{A}|$  such that  $(\mathcal{A}, a)$  has computable dimension  $\omega$ .*

**Definition 4.** Let  $\mathbf{d}$  be a degree. A structure  $\mathcal{A}$  with computable domain is  $\mathbf{d}$ -computable if its atomic diagram is  $\mathbf{d}$ -computable. The degree of  $\mathcal{A}$ , denoted by  $\deg(\mathcal{A})$ , is the least degree  $\mathbf{d}$  (which always exists) such that  $\mathcal{A}$  is  $\mathbf{d}$ -computable. An isomorphism from a structure  $\mathcal{B}$  to a  $\mathbf{d}$ -computable structure with computable domain is called a  $\mathbf{d}$ -computable presentation of  $\mathcal{B}$ . The degree of a presentation is the degree of the image of the presentation. If  $\mathcal{B}$  has a  $\mathbf{d}$ -computable presentation then it is  $\mathbf{d}$ -computably presentable.

**Definition 5.** The degree spectrum of a countable structure  $\mathcal{A}$  (denoted by  $DgSp(\mathcal{A})$ ) is the set of degrees of presentations of  $\mathcal{A}$ .

**Theorem 3 (Slaman in [27]; Wehner in [28]).** There is a structure  $\mathcal{A}$  that has presentations of every degree except  $\mathbf{0}$ . (In other words,  $DgSp(\mathcal{A}) = \mathcal{D} - \{\mathbf{0}\}$ , where  $\mathcal{D}$  is the set of all degrees.)

**Theorem 4 (Goncharov, Harizanov, Knight, McCoy, Miller, Solomon in [16]).** For every successor ordinal  $\alpha$  there exists a structure with presentations in exactly those degrees of sets  $X$ , that  $\Delta_\alpha^0(X)$  is not equal to  $\Delta_\alpha^0$ . In particular, for every finite  $n$  there exists a structure with presentations in exactly non-low <sub>$n$</sub>  degrees.

The other approach to study the computable presentation of a structure is to compare the images of additional relations in these presentations. Here we can look at degree spectra of relations on structures.

**Definition 6.** The degree spectrum of a relation  $U$  on the domain of a computable structure  $\mathcal{A}$  (denoted by  $DgSp_{\mathcal{A}}(U)$ ), is the set of degrees of the images of  $U$  in all computable presentations of  $\mathcal{A}$ .

There exist many interesting examples of computable structures and relations on their domains that have different degree spectra (see [19], [22], etc.).

The effective transformations from [14] and [21] show that many algorithmic properties of structures may be reduced to algorithmic properties of graphs. On the other hand, we can pass back from graphs to classes of all structures for languages containing at least one  $n$ -ary relational symbol, where  $n \geq 2$ . In this paper we show that such a transition is also possible to structures for languages with only two unary functions. The language, which contains either at least binary relational symbol or two unary functional symbols is called *rich*. Thus, the results of the present paper completes the picture for structures for all rich languages.

The third approach to study connections between definability and computability of classes of structures involves the notion of the index set. The well-known result of Nurtazin [25] shows that there exists a universal computable numbering of all computable structures for a fixed relational language. In other words, there exists a computable sequence  $\{\mathcal{A}_n\}_{n \in \omega}$  of computable structures, such that for every computable  $\mathcal{B}$  for this language we can effectively find  $\mathcal{A}_n$  which is isomorphic to  $\mathcal{B}$ . The *index set* of a structure  $\mathcal{B}$  for this language is the set  $I(\mathcal{B})$  of all indices of computable (isomorphic) copies of  $\mathcal{B}$  in this numbering.

For a class  $K$  of structures, closed under isomorphism, the *index set* is the set  $I(K)$  of all indices for computable members of  $K$ . Index sets were widely studied by many authors (see, for example, [1], [2], [3], [4], [6], [7] [17], etc.)

**Definition 7.** Let  $\Gamma$  be a complexity class (e.g.,  $\Sigma_3^0$ ).  $I(K)$  is  $m$ -complete  $\Gamma$  if  $I(K)$  is  $\Gamma$  and for any  $S \in \Gamma$ , there is a computable function  $f$  such that

$$n \in S \text{ iff } f(n) \in I(K).$$

This condition is equivalent to the condition that there is a uniformly computable sequence  $\{\mathcal{C}_n\}_{n \in \omega}$  for which

$$n \in S \text{ iff } \mathcal{C}_n \in K.$$

In [8] and [9] we studied the index sets of the following important classes structures: decidable structures, decidable structures with countably categorical theories and computable structures with decidable theories. We gave the complete characterization of degrees of complexity of the index sets for classes of structures for the language containing at least one  $n$ -ary relational symbol, where  $n \geq 2$ . This paper completes the proof for all rich languages.

## 2 Transformation from Graphs to $\mathcal{L}_0$ -Structures

Let  $\mathcal{L}_0 = \langle f, g \rangle$  be a language containing only two unary functional symbols. In this section we give an effective transformation of the class of graphs into the class of  $\mathcal{L}_0$ -structures. In the next two sections we'll show that it preserves all algorithmic and model-theoretic properties we are interested in.

Consider a graph  $\mathcal{G} = \langle |\mathcal{G}|, P \rangle$ . Using  $\mathcal{G}$ , we define a new  $\mathcal{L}_0$ -structure  $\mathcal{A}$ , in which  $\mathcal{G}$  will be definable in a very nice way. We build  $\mathcal{A}$  as follows.

The structure  $\mathcal{A}$  has its universe consisting of 3 pairwise disjoint infinite sets:

$$|\mathcal{A}| = U \cup V \cup W,$$

where  $U = \{u_i \mid i \in |\mathcal{G}|\}$ ,  $V = \{v_i \mid i \in |\mathcal{G}|\}$ ,  $W = \bigcup_i W_i = \{w_j^i \mid i \in \omega, j \in |\mathcal{G}|\}$ .

Now we need to define functions  $f$  and  $g$  on  $|\mathcal{A}|$ . We do it in the following way.

For all  $u_i \in U$  we let  $f(u_i) = u_i$ . For all  $v_i \in V$  we let  $f(v_i) = u_i$ . For all  $w_j^i \in W_i$  we let  $f(w_j^i) = u_i$ . That is, the function  $f$  is identical on the set  $U$ , it maps  $V$  1-to-1 onto  $U$ , and for every  $u_i \in U$  there are infinitely many elements  $w_j^i \in W$  that corresponds to  $u_i$  via  $f$ .

Now we define the function  $g$ . For all  $v_i \in V$  we let  $g(v_i) = v_i$ . For all  $u_i \in U$  we let  $g(u_i) = v_i$ . Values of  $g$  on elements from  $W$  depends on  $P(i, j)$ . If  $P(i, j)$  is true in  $\mathcal{G}$ , then we choose a unique index  $k$  such that in  $\mathcal{A}$  we have  $g(w_k^i) = v_j$ . If  $\neg P(i, j)$  is true in  $\mathcal{G}$ , then we choose a unique index  $k$  such that  $g(w_k^i) = v_j$ . For different  $j_s$  we choose different  $k_{j_s}$  corresponding to  $P(i, j_s)$  or  $\neg P(i, j_s)$ , and every  $w_k^i$  witnesses (via  $g(w_k^i)$ ) one of  $P(i, j)$  or  $\neg P(i, j)$  for some (unique)  $j$ . The function  $g$  defined as above is identical on the set  $V$  and maps  $U$  1-to-1 onto  $V$ . Moreover, it allows us to define  $P$  and  $\neg P$  in  $\mathcal{A}$  via existential formulas. The precise proof is given in the next section.

### 3 Algorithmic Properties of the Class of $\mathcal{L}_0$ -Structures

To get all the results on computable categoricity and computable dimensions of structures and degree spectra of structures and relations we use the result from [21]. In the paper some sufficient conditions were given for a transformation to preserve all the properties mentioned above. Here we remind definitions and the sufficient conditions from [21].

**Definition 8.** A countable structure  $\mathcal{A}$  is trivial if for some finite set  $S$  of elements of  $|\mathcal{A}|$ , every permutation of  $|\mathcal{A}|$  that keeps the elements of  $S$  fixed is an automorphism of  $\mathcal{A}$ .

The interesting case is nontrivial structures.

**Definition 9.** A relation  $U$  on a structure  $\mathcal{A}$  is invariant if for every automorphism  $f : \mathcal{A} \rightarrow \mathcal{A}$  we have  $f(U) = U$ .

**Definition 10.** A relation  $U$  on the universe of a structure  $\mathcal{A}$  is relatively intrinsically computable if for every presentation  $f : \mathcal{A} \rightarrow \mathcal{A}$ , the image  $f(U)$  is computable in  $\deg(\mathcal{A})$ .

Now let  $\mathcal{G}$  be a nontrivial, countable directed graph with edge relation  $E$  and let  $\mathcal{A}$  be a countable structure. The general idea of the sufficient conditions given below is to say that the transformation from graphs to structures for different language is rather effective. That is, the graph is definable in the structure in a nice way which allows us to preserve all algorithmic properties.

We give a more precise formalization of this idea. Let  $\mathcal{G}$  be a graph, and  $\mathcal{A}$  be a structure. Assume that there exist relatively intrinsically computable, invariant relations  $D(x)$  and  $R(x, y)$  on the universe of  $\mathcal{A}$  and a map  $G \mapsto A_G$  from the set of presentations of  $\mathcal{G}$  to the set of presentations of  $\mathcal{A}$  with the following properties.

- (P0) For every presentation  $G$  of  $\mathcal{G}$ , the structure  $A_G$  is  $\deg(G)$ -computable.
- (P1) For every presentation  $G$  of  $\mathcal{G}$  there exists a  $\deg(G)$ -computable map  $g_G : D(A_G) \xrightarrow[\text{onto}]{1-1} |G|$  such that  $R^{A_G}(x, y) \Leftrightarrow E^G(g_G(x), g_G(y))$  for every  $x, y \in D(A_G)$ .
- (P2) If  $h : D(\mathcal{A}) \xrightarrow[\text{onto}]{1-1} D(\mathcal{A})$  is such that  $R(x, y) \Leftrightarrow R(h(x), h(y))$  then  $h$  can be extended to an automorphism of  $\mathcal{A}$ .
- (P3) For every presentation  $G$  of  $\mathcal{G}$  there exists a  $\deg(G)$ -computable set of existential formulas  $\varphi_0(\bar{a}, \bar{b}_0, x), \varphi_1(\bar{a}, \bar{b}_1, x), \dots$  such that  $\bar{a}$  is a tuple of elements of  $|A_G|$ , each  $\bar{b}_i$  is a tuple of elements of  $D(A_G)$ , each  $x \in |A_G|$  satisfies some  $\varphi_i$ , and no two elements of  $|A_G|$  satisfy the same  $\varphi_i$ .

**Proposition 1.** Satisfaction of the conditions (P0)–(P3) guarantees that the following statements hold.

1.  $\text{DgSp}(\mathcal{A}) = \text{DgSp}(\mathcal{G})$ .

2. If  $\mathcal{G}$  is computably presentable then
- for any degree  $\mathbf{d}$ ,  $\mathcal{A}$  has the same  $\mathbf{d}$ -computable dimension as  $\mathcal{G}$ ;
  - if  $x \in |\mathcal{G}|$  then there exists an  $a \in D(\mathcal{A})$  such that  $(\mathcal{A}, a)$  has the same computable dimension as  $(\mathcal{G}, x)$ ;
  - if  $S \subseteq |\mathcal{G}|$  then there exists  $U \subseteq |D(\mathcal{A})|$ , such that  $DgSp_{\mathcal{A}}(U) = DgSp_{\mathcal{G}}(S)$  and if  $S$  is intrinsically c.e. then so is  $U$ .

The proof can be found in [21].

Therefore, what we need to do is to introduce invariant, relatively intrinsically c.e. relations  $D(x)$  and  $R(x, y)$  defining the universe of the graph and the edge relation correspondingly and check the conditions (P0)–(P3) for the transformation from Sect. 2.

First of all, it is easy to see that all the sets  $U, V$  and  $W$  are definable by quantifier-free formulas. For example,  $U = \{x | f(x) = x\}$ . Thus, we may use these sets in other formulas as abbreviation for their defining formulas. Under this convention, we set  $D(x) \Rightarrow U(x)$ . That is,  $D(\mathcal{A}) = U^{\mathcal{A}}$ .

We define  $R(x, y)$  and its negation via two existential formulas:

$$\langle x, y \rangle \in P \Leftrightarrow U(x) \& U(y) \& \exists z (W(z) \& (f(z) = x) \& (g(z) = y)),$$

$$\langle x, y \rangle \notin P \Leftrightarrow U(x) \& U(y) \& \exists z, t (W(z) \& V(t) \& (f(z) = x) \& (g(z) = t) \& (g(y) = t)).$$

It is easily seen from the definitions that  $D(x)$  and  $R(x, y)$  are invariant and relatively intrinsically c.e. Now we check the properties (P0)–(P3).

- (P0) Follows directly from the construction of the transformation.
- (P1) Let  $G$  be a presentation of  $\mathcal{G}$ . Existence of  $\deg(G)$ -computable map follows from the definition of  $f, g$  and from defining formulas for  $B^{A_G}$  and  $R^{A_G}$ .
- (P2) Let  $h$  be a 1-to-1 map of  $D(\mathcal{A})$  onto  $D(\mathcal{A})$  such that  $R(x, y) \Leftrightarrow R(h(x), h(y))$ . We extend  $h$  in the following way. For  $v \in V$  we find the corresponding  $u \in U$  such that  $g(u) = v$ . After that we let  $h(v) = g(h(u))$ . For  $w \in W$  we first check if  $g(w)$  is in  $U$  or in  $V$ . If  $g(w) = u \in U$  then we find the unique  $w' \in W$  connected with  $h(u)$ . Such  $w'$  exists by properties of  $h$ . We let  $h(w) = w'$ . The second case is similar.
- (P3) Let  $G$  be a presentation of  $\mathcal{G}$ . Every element  $v$  from  $V$  is the unique element satisfying the formula  $(g(v) = v) \& (g(u) = v) \& (f(u) = u)$  for corresponding  $u \in D(A_G) (= U^{A_G})$ . For  $w \in W$  the formula depends on the value of  $g(w)$ . If  $g(w) \in U$  then we write that there exists exactly one  $u \in U$  which is the value of  $g(w)$ . Similar for  $g(w) \in V$ .

Thus, we have proved the following theorems.

**Theorem 5.** *Let  $\mathcal{G}$  be a computably presentable graph. There exists a computably presentable  $\mathcal{L}_0$ -structure  $\mathcal{A}$  such that for any degree  $\mathbf{d}$ ,  $\mathcal{A}$  has the same  $\mathbf{d}$ -computable dimension as  $\mathcal{G}$ .*

**Corollary 1.** *For every  $1 \leq n \leq \omega$  there exist an  $\mathcal{L}_0$ -structure  $\mathcal{A}$  with computable dimension  $n$ .*

**Theorem 6.** Let  $\mathcal{G}$  be a computably presentable graph. There exists a computably presentable  $\mathcal{L}_0$ -structure  $\mathcal{A}$  such that for every  $x \in |\mathcal{G}|$  then there exists an  $a \in D(\mathcal{A})$  such that  $(\mathcal{A}, a)$  has the same computable dimension as  $(\mathcal{G}, x)$ .

**Corollary 2.** For every  $1 \leq n \leq \omega$  there exists a computably categorical  $\mathcal{L}_0$ -structure  $\mathcal{A}$  and an element  $a \in |\mathcal{A}|$ , such that the computable dimension of  $(\mathcal{A}, a)$  equals  $n$ .

**Theorem 7.** Let  $\mathcal{G}$  be a graph. There exists a structure  $\mathcal{A}$  for the language  $\mathcal{L}_0$  with only 2 unary functions such that  $DgSp(\mathcal{A}) = DgSp(\mathcal{G})$ .

**Corollary 3.** There exists a computable  $\mathcal{L}_0$ -structure with presentations in all degrees except  $\mathbf{0}$ .

**Theorem 8.** Let  $\mathcal{G}$  be a computably presentable graph. There exists a computably presentable  $\mathcal{L}_0$ -structure  $\mathcal{A}$  such that if  $S \subseteq |\mathcal{G}|$  then there exists a  $U \subseteq |D(\mathcal{A})|$  such that  $DgSp_{\mathcal{A}}(U) = DgSp_{\mathcal{G}}(S)$  and if  $S$  is intrinsically c.e. then so is  $U$ .

**Corollary 4.** For every computable ordinal  $\alpha$  there exists a computable  $\mathcal{L}_0$ -structure  $\mathcal{A}$  and a relation  $R$  on  $|\mathcal{A}|$ , such that  $R$  is intrinsically  $\Sigma^0_\alpha$  but not relatively intrinsically  $\Sigma^0_\alpha$ .

## 4 Results on Index Sets

Now we turn to the question about connections between computability-theoretic and structural properties of structures in classes of  $\mathcal{L}_0$ -structures. More precisely, instead of the language  $\mathcal{L}_0 = \langle f, g \rangle$  we consider a new relational language  $\mathcal{L} = \langle P_f, P_g \rangle$ . Here the predicate symbols  $P_f$  and  $P_g$  are binary. We will think of them as representing graphs of  $f$  and  $g$  correspondingly.

According to the Nurtazin's result, there exists a universal computable numbering  $\{\mathcal{A}_n\}_{n \in \omega}$  of all computable structures for the language  $\mathcal{L}$  consisting of 2 binary relational symbols. When talking about index sets of  $\mathcal{L}_0$ -structures with fixed algorithmic and model-theoretic properties, in fact, we consider index sets of  $\mathcal{L}$ -structures with the same properties and additional requirement on  $P_f, P_g$ . Namely, we require that these relations represent graphs of total functions in our structures.

Under these convention, using the results from [8] and [9] we prove the following theorem.

**Theorem 9.** 1. The index set  $CK$  of the class of decidable  $\mathcal{L}_0$ -structures is  $m$ -complete  $\Sigma^0_3$  set;  
 2. The index set  $CK_0$  of the class of decidable  $\mathcal{L}_0$ -structures with countably categorical theories is  $m$ -complete  $\Sigma^0_3 - \Sigma^0_3$  set (where  $\Sigma^0_3 - \Sigma^0_3$  is the class of differences of two  $\Sigma^0_3$  sets);  
 3. The index set  $DT$  of the class of  $\mathcal{L}_0$ -structures with decidable theories is  $m$ -complete  $\Sigma^{0,\emptyset(\omega)}_2$  set.

*Proof.* First of all, we need to proof that in any of these cases the index sets belong to the corresponding classes of complexity.

- Lemma 1.**
1.  $CK \in \Sigma_3^0$ .
  2.  $CK_0 \in \Sigma_3^0 - \Sigma_3^0$  (where  $\Sigma_3^0 - \Sigma_3^0$  is the class of differences of two  $\Sigma_3^0$  sets).
  3.  $DT \in \Sigma_2^{0,\emptyset^{(\omega)}}$ .

Proof of the lemma can be found in [8] and [9]. The only one additional condition for  $P_f$  and  $P_g$  is to represent graphs of total functions. This condition does not change the proofs.

To prove  $m$ -completeness of the index sets, we show that the transformation from Sect. 2 preserves decidability of structures as well as decidability and countable categoricity of their theories. Then from the results about index sets of graphs from [8] and [9] we get the proof of the theorem.

**Proposition 2.** Let  $\mathcal{G}$  be a graph and let  $\mathcal{A}$  be an  $\mathcal{L}_0$ -structure obtained from  $\mathcal{G}$  by the transformation from Sect. 2. Then

1.  $\mathcal{A}$  is computable  $\Leftrightarrow \mathcal{G}$  is computable;
2.  $\mathcal{A}$  is decidable  $\Leftrightarrow \mathcal{G}$  is decidable;
3.  $Th(\mathcal{A})$  is countably categorical  $\Leftrightarrow Th(\mathcal{G})$  is countably categorical;
4.  $Th(\mathcal{A})$  is decidable  $\Leftrightarrow Th(\mathcal{G})$  is decidable.

*Proof.* From the construction of the transformation and observations in Sect. 3 we get the first point.

Let  $\mathcal{G}$  be decidable. We show that in this case  $\mathcal{A}$  is also decidable. For this we prove that the complete diagram  $\mathcal{D}^c(\mathcal{A})$  is computably axiomatizable and complete, hence, decidable. We consider the following set of axioms.

1.  $P_f$  and  $P_g$  are graphs of total functions.
2. For every  $x, y$ , if  $P_f(x, y)$  then  $P_f(y, y)$ .
3. For every  $x, y$ , if  $P_g(x, y)$  then exactly one of the following holds: either  $\neg P_f(x, x) \& \exists z P_g(y, z) \& P_g(z, z)$  or  $P_f(x, x) \& P_g(y, y)$ .
4. For every  $x$ , if  $P_f(x, x)$  then there exists exactly one  $z$ , such that  $\neg P_f(z, z)$  and  $\neg P_g(z, z)$  and exactly one of 2 conditions holds:
  - (a)  $P_g(z, x)$  or
  - (b)  $\exists y(P_g(x, y) \& P_g(z, y))$ .
5. For every  $z$ , if  $\neg P_f(z, z) \& \neg P_g(z, z)$  then there exists a unique  $x$ , such that  $P_f(x, x) \& P_g(z, x)$  or  $P_g(x, x) \& P_g(z, x)$ .
6. For every sentence  $\psi$  from  $\mathcal{D}^c(\mathcal{G})$  we put into the set of axioms a sentence  $\psi'$ , which we define in the following way. We consider  $\psi$  to be in the prenex normal form. If  $\psi$  is an open formula, let  $\psi'$  be constructed from  $\psi$  by the substitution of positive and negative occurrences of  $P(x, y)$  for their definitions by existential formulas using  $P_f, P_g$ . If  $\psi = \exists x\psi_1$ , then we let  $\psi' = \exists x(P_f(x, x) \& \psi'_1)$ . If  $\psi = \forall x\psi_1$ , then we let  $\psi' = \forall x(P_f(x, x) \rightarrow \psi'_1)$ .

To prove that the theory is complete, we consider any two of its saturated models  $\mathcal{A}_1, \mathcal{A}_2$  of the cardinality  $\omega_1$ . We want to show that they are isomorphic. We define the predicate  $P$  on the set  $\{x | P_f(x, x)\}$  using its definition by existential formulas using  $P_f$  and  $P_g$ . The resulting graphs  $\mathcal{G}_1, \mathcal{G}_2$  are saturated models of  $\mathcal{D}^c(\mathcal{G})$ , hence, they are isomorphic. Let  $\varphi : \mathcal{G}_1 \rightarrow \mathcal{G}_2$  give this isomorphism. In a way, similar to the one from Sect. 3 we can extend the isomorphism to an isomorphism between  $\mathcal{A}_1$  and  $\mathcal{A}_2$ . Thus, the theory  $\mathcal{D}^c(\mathcal{A})$  is axiomatizable by the decidable set of axioms and complete, hence, it is decidable.

Similar reasoning gives the other two points.

This completes the proof of the theorem.

## References

1. Calvert W.: The isomorphism problem for classes of computable fields. *Archive for Mathematical Logic* **75** (2004) 327–336.
2. Calvert W.: The isomorphism problem for computable Abelian  $p$ -groups of bounded length. *Journal of Symbolic Logic* **70** (2005) 331–345.
3. Calvert W., Cummins D., Knight J. F., Miller S.: Comparing classes of finite structures. *Algebra and Logic* **43** (2004) 374–392.
4. Calvert W., Harizanov V., Knight J. F., Miller S.: Index sets of computable structures. *Algebra and Logic* **45** (2006) 306–325.
5. Cholak P., Goncharov S. S., Khoussainov B., Shore R. A.: Computably categorical structures and expansions by constants. *J. Symbolic Logic* **64** (1999) 13–37.
6. Csima B. F., Montalbán A., Shore R. A.: Boolean algebras, Tarski invariants, and index sets/ *Notre Dame J. Formal Logic* **47** (2006) 1–23.
7. Dobritsa V. P.: Complexity of the index set of a constructive model. *Algebra and Logic* **22** (1983) 269–276.
8. Fokina E. B.: Index Sets of Decidable Models. *Siberian Math. J.* **48** (2007) 939–948 (English translation).
9. Fokina E. B.: Index Sets of Computable Structures with Decidable Theories. *Computation and Logic in the Real World - Third Conference of Computability in Europe, CiE 2007, Siena, Italy, June 2007, Proceedings, Lecture Notes in Computer Science*, **4497** (2007) 290–296.
10. Goncharov S. S.: Problem of the number of non-self-equivalent constructivizations. *Algebra and Logic* **19** (1980) 401–414.
11. Goncharov S. S.: Groups with a finite number of constructivizations. *Soviet Math. Dokl.* **23** (1981) 58–61.
12. Goncharov S. S.: Limit equivalent constructivizations. *Mathematical Logic and the Theory of Algorithms*, vol. 2 of *Trudy Inst. Mat., Nauka, Sibirsk. Otdel.*, Novosibirsk, (1982) 4–12, in Russian.
13. Goncharov S. S.: Countable Boolean Algebras and Decidability. *Siberian School of Algebra and Logic*, Consultants Bureau, New York (1997).
14. Goncharov S. S.: Computability and Computable Models. Mathematical problems from applied logic. II. Logics for the XXIst century. Edited by Dov M. Gabbay, Sergey S. Goncharov and Michael Zakharyashev. International Mathematical Series (New York), Springer, New York (2007) 99–216.
15. Goncharov S. S., Dzgove V. D.: Autostability of models. *Algebra and Logic* **19** (1980) 28–37.

16. Goncharov, S., Harizanov V., Knight J.F., McCoy C., Miller R., Solomon R.: Enumerations in computable structure theory. *Annals of Pure and Applied Logic* **136** (2005), 219–246.
17. Goncharov S. S., Knight J. F.: Computable structure and non-structure theorems. *Algebra and Logic* **41** (2002) 351–373 (English translation).
18. Goncharov S. S., Molokov A. V., Romanovskii N. S.: Nilpotent groups of finite algorithmic dimension. *Siberian Math. J.* **30** (1989) 63–68.
19. Harizanov V.: The possible Turing degree of the nonzero member in a two element degree spectrum. *Ann. Pure Appl. Logic* **60** (1993) 1–30.
20. Hirschfeldt D. R., Khoussainov B., Shore R. A.: A computably categorical structure whose expansion by a constant has infinite computable dimension. *J. Symbolic Logic* **68** (2003) 1199–1241.
21. Hirschfeldt D. R., Khoussainov B., Shore R. A., Slinko A. M.: Degree Spectra and Computable Dimensions in Algebraic Structures *Annals of Pure and Applied Logic* **115** (2002) 71–113.
22. Khoussainov B., Shore R. A.: Computable isomorphisms, degree spectra of relations, and Scott families. *Ann. Pure Appl. Logic* **93** (1998) 153–193.
23. Khoussainov B., Shore R. A.: Solution of the Goncharov-Ash problem and the spectrum problem in the theory of computable models. *Doklady Math.* **61** (2000) 178–179, research announcement, Russian version in *Dokl. Akad. Nauk* **371** (2000) 30–31.
24. Lempp S., McCoy C., Miller R., Solomon R.: Computable categoricity of trees of finite height. *J. Symbolic Logic* **70** (2005) 151–215.
25. Nurtazin A. T.: Computable classes and algebraic criteria for autostability. Ph.D. Thesis, Institute of Mathematics and Mechanics, Alma-Ata (1974).
26. Remmel J. B.: Recursively categorical linear orderings. *Proc. Amer. Math. Soc.* **83** (1981) 387–391.
27. Slaman T. A.: Relative to any nonrecursive set. *Proc. Amer. Math. Soc.* **126** (1998) 2117–2122.
28. Wehner S.: Enumerations, countable structures, and Turing degrees. *Proc. Amer. Math. Soc.* **126** (1998) 2131–2139.

# Verification of Newman’s and Yokouchi’s Lemmas in PVS

André Luiz Galdino<sup>1,2</sup> and Mauricio Ayala-Rincón<sup>1\*</sup>

<sup>1</sup> Instituto de Ciências Exatas, Universidade de Brasília, Brazil

<sup>2</sup> Departamento de Matemática, Universidade Federal de Goiás

Campus de Catalão, Brazil

{galdino, ayala}@unb.br

**Abstract.** This paper shows how a formalization for Abstract Reduction Systems (ARSs) in which noetherianity was defined by the notion of well-foundedness over binary relations is used in order to prove results such as well-known Newman’s and Yokouchi’s Lemmas. The former is known as the *diamond lemma* and the latter states a property of commutation between ARSs. The *theory ars* was specified in the Prototype Verification System (PVS) for which to the best of our knowledge there are no available *theory* for dealing with rewriting techniques in general. In addition to proof techniques available in PVS, the verification of these lemmas implies an elaborated use of natural and noetherian induction.

**Key words:** Abstract Reduction Systems, Rewriting Systems, PVS.

## 1 Introduction

The Prototype Verification System (PVS), developed at the SRI and widely used by industrial and academic parties, consists of a specification language built on higher-order logic, which supports modularity by means of parameterized *theories*, with a rich type-system and a prover which uses the sequent-style. A PVS *theory*, *ars*, built over the PVS prelude libraries for sets and binary relations that is useful for the treatment of Abstract Reduction Systems (ARS) was reported in [4]. In the *theory ars* basic ARS notions such as reduction, derivation, normal form, confluence, local confluence, joinability, noetherianity, etc., were adequately specified in such a way that non elementary proof techniques such as noetherian induction are possible. In this paper we describe the usefulness of *ars* by describing proofs of Newman’s and Yokouchi’s lemmas. The former proof is a well-known classical application of noetherian induction and the latter is of interest because it is based on several applications of natural and noetherian induction. The inductive proof of the Newman’s Lemma given by Huet in [6] is a classical example of proofs in higher-order logic.

The novelty of this work is not to present mechanical proofs of ARS theorems in PVS that were done previously in other proof assistants. In fact, well-known

---

\* Both authors are partially supported by the Brazilian Research Council, CNPq.

formalizations of Newman's Lemma have been specified in several proof assistants, e.g., ACL2 [15], Coq [7], Isabelle [14], Boyer-Moore [16], Otter [3], among others. **ars** is presented as the basis for the formalization of an elaborated and robust PVS *theory* for full Term Rewriting Systems (TRSs) [5]. The motivation for doing this work is the fact that rewriting has been applied to the specification and synthesis of reconfigurable hardware [9] and that the correction of these specifications can be carried out by translating these rewriting specifications into the language of the PVS proof assistant as logic theories ([1] introduces a proved correct translation from ELAN rewriting specifications into PVS *theories*). And robust proof rewriting based methods are necessary to deal efficiently with the correctness of these *theories* that come from rewriting based specifications.

Section 2 presents analytical proofs of both lemmas. Sections 3 and 4 describe respectively the specification and verification of both lemmas in PVS. The theory **ars** together with proofs of Newman's, Yokouchi's, among other interesting lemmas is available at [www.mat.unb.br/~ayala/publications.html](http://www.mat.unb.br/~ayala/publications.html).

## 2 Background: analytical proofs

We suppose the reader is familiar with rewriting concepts and standard notations as presented in [2] or [17].

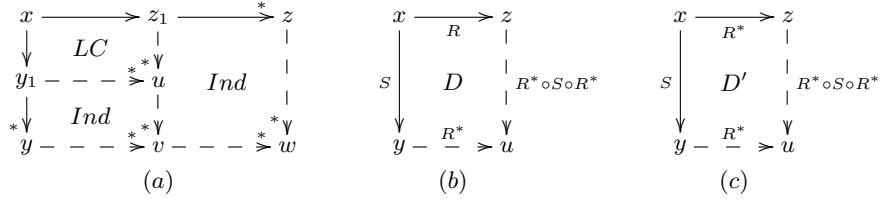
An abstract reduction relation is a binary relation  $R$  over a set  $T$ , denoted also as  $\langle R, T \rangle$ . The relation is identified as  $R$ ,  $\rightarrow_R$  or simply  $\rightarrow$ .  $R^+$  and  $R^*$  respectively denote the transitive and the reflexive transitive closure of  $R$ , denoted in arrow notation as  $\rightarrow^+$  and  $\rightarrow^*$ , respectively. In the elegant arrow notation, the inverse relation  $R^{-1}$ , its transitive and its reflexive transitive closures are respectively denoted as  $\leftarrow$ ,  ${}^+ \leftarrow$  and  ${}^* \leftarrow$ . The operator of composition is denoted as usual as  $\circ$ . An abstract reduction relation  $\rightarrow$  over  $T$  is said to be: *confluent* whenever  $({}^* \leftarrow \circ \rightarrow^*) \subseteq (\rightarrow^* \circ {}^* \leftarrow)$  and *locally confluent* whenever  $(\leftarrow \circ \rightarrow) \subseteq (\rightarrow^* \circ {}^* \leftarrow)$ .  $\rightarrow$  satisfies the *diamond property* whenever  $(\leftarrow \circ \rightarrow) \subseteq (\rightarrow \circ \leftarrow)$ . Two elements of  $T$ , say  $x, y$ , are said to be *joinable* whenever  $\exists u. x \rightarrow^* u {}^* \leftarrow y$ .  $\rightarrow$  is said to be *noetherian* whenever there is no infinite sequence of the form  $x_1 \rightarrow x_2 \rightarrow \dots$ .

**Lemma 1 (Newman's Lemma [11]).** *Let  $R$  be a noetherian relation defined on the set  $T$ . Then  $R$  is confluent if, and only if it is locally confluent.*

**proof (Sketch).** The  $\Rightarrow$ -direction follows immediately by definition.  $\Leftarrow$ -direction is proved by noetherian induction using the predicate

$$P(x) = \forall y, z. y {}^* \leftarrow x \rightarrow^* z \implies y \text{ and } z \text{ joinable}$$

Obviously  $R$  is confluent if  $P(x)$  holds for all  $x$ . Noetherian induction require us to show  $P(x)$  under the assumption  $P(t)$  for all  $t$  such that  $x \rightarrow^+ t$ . To prove  $P(x)$ , we analyze the divergence  $y {}^* \leftarrow x \rightarrow^* z$ . If  $x = y$  or  $x = z$ ,  $y$  and  $z$  are joinable immediately. Otherwise we have  $x \rightarrow y_1 \rightarrow^* y$  and  $x \rightarrow z_1 \rightarrow^* z$  as shown in the Figure 1(a), where as usual dashed arrows stand for *existence*. The existence of  $u$  follows by local confluence (*LC*) of  $R$ , the existence of  $v$  and  $w$  follows by induction hypothesis (*Ind*).  $\square$

**Fig. 1.** Proof of Newman's Lemma, Diagram  $D$  and Generalization of  $D$  as  $D'$ 

**Lemma 2 (Yokouchi's Lemma [18]).** *Let  $R$  and  $S$  be two relations defined on the same set  $T$ ,  $R$  being confluent and noetherian, and  $S$  having the diamond property. Suppose moreover that the diagram  $D$  as shown in the Figure 1(b) holds: Then the relation  $R^* \circ S \circ R^*$  has the diamond property.*

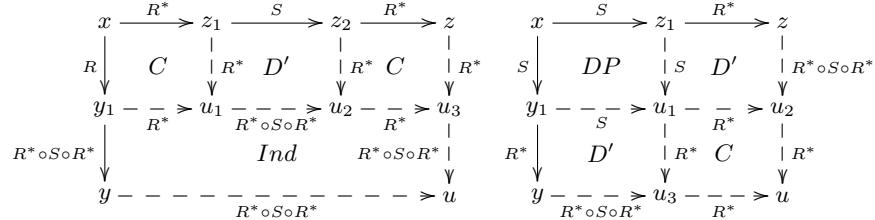
**proof (Sketch).** The proof starts by generalizing the diagram  $D$  of the lemma as the diagram  $D'$  in the Figure 1(c). This generalization is proved by noetherian induction using the predicate

$$P(x) := \forall y, z. xR^*z \wedge xSy \Rightarrow \exists u.(yR^*u \wedge zR^* \circ S \circ R^*u)$$

Then, to prove that  $R^* \circ S \circ R^*$  has the diamond property, one also proceeds by noetherian induction but this time using the predicate

$$P'(x) := \forall y, z. xR^* \circ S \circ R^*y \wedge xR^* \circ S \circ R^*z \Rightarrow \exists u.(yR^* \circ S \circ R^*u \wedge zR^* \circ S \circ R^*u)$$

One concludes, by induction in the length of the derivation of the first  $R^*$  in  $xR^* \circ S \circ R^*y$ . In other words, we distinguish between the cases  $xR \circ R^* \circ S \circ R^*y$  and  $xS \circ R^*y$  as is shown in the Figure 2, where  $C$  and  $DP$  stand for use of confluence of  $R$  and diamond property of  $S$  hypotheses, respectively.  $\square$

**Fig. 2.** Cases  $xR \circ R^* \circ S \circ R^*y$  and  $xS \circ R^*y$ 

### 3 Specification

The PVS theory `newman_yokouchi` presented in Table 1 specifies Newman's and Yokouchi's lemmas. This theory is parameterized as `newman_yokouchi[T]`, where (within the `newman_yokouchi` theory)  $T$  is treated as a fixed uninterpreted type.

**Table 1.** newman\_yokouchi PVS theory

---

```

newman_yokouchi[T : TYPE] : THEORY BEGIN
  IMPORTING results_confluence[T], noetherian[T]
  R, S: VAR PRED[[T,T]]

Newman_lemma: THEOREM
  noetherian?(R) => (confluent?(R) <=> local_confluent?(R))

Yokouchi_lemma_ax1: LEMMA
  (noetherian?(R) & confluent?(R) &
   (FORALL x,y,z: (S(x,y) & R(x,z)) =>
    (EXISTS (u:T): RTC(R)(y,u) & (RTC(R) o S o RTC(R))(z,u))))
  => (FORALL x,y,z: (S(x,y) & RTC(R)(x,z)) =>
       (EXISTS (w:T): RTC(R)(y,w) & (RTC(R) o S o RTC(R))(z,w)))

Yokouchi_lemma: THEOREM
  (noetherian?(R) & confluent?(R) & diamond_property?(S) &
   (FORALL x,y,z: (S(x,y) & R(x,z)) =>
    (EXISTS (u:T): RTC(R)(y,u) & (RTC(R) o S o RTC(R))(z,u))))
  => diamond_property?(RTC(R) o S o RTC(R))

END newman_yokouchi

```

---

$R$  and  $S$  denote reduction relations over  $T$  and  $x, y, z, w$  and  $u$  elements of  $T$ .  
 $\Rightarrow$ ,  $\Leftrightarrow$  and  $\&$  are abbreviations for IMPLIES, IFF and AND, respectively.

Newman's Lemma specification is straightforward and based on predicates over reduction relations. In the specification of the Yokouchi's Lemma  $RTC(R)$  denotes the reflexive transitive closure of the relation  $R$ , i.e.  $R^*$ . The composition of relations is denoted as  $\circ$ . The second lemma, *Yokouchi\_lemma\_ax1*, corresponds to the generalization  $D'$  of the diagram  $D$  presented in Figure 1.

The theories *results\_confluence*[ $T$ ] and *noetherian*[ $T$ ], also components of the whole *ars theory*, are imported by the *newman\_yokouchi theory*. The former contains results about confluence and the latter the definition of noetherianity formulated in terms of the notion of well-foundness as it will be explained.

Specifications of rewriting properties are exemplified by the following ones.

```

joinable?(R)(x,y): bool = EXISTS z:RTC(R)(x,z) & RTC(R)(y,z)
local_confluent?(R): bool = FORALL x,y,z: R(x,y) & R(x,z) =>
                                joinable?(R)(y,z)
confluent?(R): bool = FORALL x,y,z: RTC(R)(x,y) & RTC(R)(x,z) =>
                                joinable?(R)(y,z)
diamond_property?(R): bool = FORALL x,y,z: R(x,y) & R(x,z) =>
                                EXISTS r: R(y,r) & R(z,r)

```

In order to make easy the use of natural induction, closures of relations are built as unions of iterations of their compositions. For instance, the reflexive transitive closure operator  $RTC$  of a relation  $R$  is specified as the union of the iterations  $iterate(R,i)$ , for all  $i \geq 0$ , where  $iterate(R,i)$  specifies the relation

$\underbrace{R \circ R \cdots \circ R}_{i \text{ times}}$  available in the PVS `orders` library. The operators `iterate` and `IUnion`, available in the PVS prelude *theory* are specified as

```
iterate(R,i): RECURSIVE PRED[[T,T]] =
  IF i = 0 THEN =[T] ELSE iterate(R,i - 1) o R ENDIF
  MEASURE i
IUnion(A): set[T] = {x | EXISTS i: A(i)(x)}
```

and using them, RTC can be specified as

```
RTC(R): reflexive_transitive = IUnion(LAMBDA n: iterate(R,n))
```

Notice that the type of `RTC` is `reflexive_transitive`. This places in the basis of the system of types the intrinsic characteristics of the `RTC` construction creating the necessary proofs obligations to be proved (*PVS Type Correctness Conditions - TCCs*). This type is specified as follows.

```
reflexive_transitive?(R): bool = reflexive?(R) & transitive?(R)
reflexive_transitive: TYPE = (reflexive_transitive?)
```

Noetherianity is formalized in terms of well foundedness, and noetherian induction is then verified using the lemma `wf_induction`, which expresses the principle of *well-founded induction*. The notion of well-founded relations and this principle are available in the prelude *theory* [12].

```
noetherian?(R): bool = well_founded?(converse(R))
noetherian: TYPE = (noetherian?)

wf_induction: LEMMA
  (FORALL (p: pred[T]): (FORALL x: (FORALL y:
    y < x => p(y)) => p(x)) => (FORALL (x:T): p(x)))
```

The lemma `noetherian_induction` presented below corresponds to the principle of noetherian induction.

```
noetherian_induction: LEMMA (FORALL (R: noetherian, P: PRED[T]):
  (FORALL x, y: (TC(R)(x,y) => P(y)) => P(x)) => (FORALL x: P(x)))
```

This lemma uses the transitive closure operator `TC`, that is specified similarly to `RTC`. Its application depends on an adequate instantiation of the predicate `P`.

## 4 Verification

The verification of Newman's and Yokouchi's lemmas consists of 1857 lines (168247 bytes) of proofs. The formalizations of Newman's and Yokouchi's lemmas use 114 and 225 proof steps, respectively. Here the relevant fragment of the proof trees, focusing on the application of noetherian induction, are presented.

#### 4.1 Verification of Newman's Lemma

When the PVS prover is invoked the proof tree starts off with a root node (sequent) having no antecedent and as succedent the theorem to be proved.

```
Newman_lemma :
|-----
{1} FORALL (R: PRED[[T,T]]):
    noetherian?(R) => (confluent?(R) <=> local_confluent?(R))
```

The reduction relation R is correctly universally quantified, since it was declared as a variable in the *theory newman\_yokouchi* (see Table 1). After skolemization by applying the proof command (*skeep*), the conjunctive splitting command (*split*) is applied to the goal obtaining two subgoals. The first subgoal, *Newman\_lemma.1*, is to demonstrate that confluence implies local confluence, which is easily formalized. The second subgoal, *Newman\_lemma.2*, that is to demonstrate that local confluence implies confluence (under noetherianity hypothesis), is the truly interesting one.

For proving this subgoal, after disjunctive simplification with (*flatten*), one introduces the noetherian induction scheme *noetherian\_induction* and instantiates its predicate P as:

```
(LAMBDA (a:T): (FORALL (b,c:T): RTC(R)(a,b) & RTC(R)(a,c)
                      => joinable?(R)(b,c)))
```

Then, the subgoals *Newman\_lemma.2.1* and *2.2* presented below are obtained by applying the command (*split*), that splits the implication of the instantiated noetherian induction scheme. The first subgoal is easily verified by expanding the definition of the predicate *confluent?*, skolemization and adequate instantiation of the variables of the antecedent  $\{-1\}$ .

```
Newman_lemma.2.1 :
{-1} FORALL (x:T): FORALL (b,c:T):
    RTC(R)(x,b) & RTC(R)(x,c) => joinable?(R)(b,c)
[-2] local_confluent?(R) [-3] noetherian?(R)
|-----
[1] confluent?(R)

Newman_lemma.2.2 :
[-1] local_confluent?(R) [-2] noetherian?(R)
|-----
{1} FORALL (x:T): (FORALL (y:T): TC(R)(x,y) => (FORALL (b,c: T):
    RTC(R)(y,b) & RTC(R)(y,c) => joinable?(R)(b,c)))
=> (FORALL (b,c:T): RTC(R)(x,b) & RTC(R)(x,c)=> joinable?(R)(b,c))
```

To prove the latter subgoal, one needs to show  $P(x)$  under the assumption  $P(y)$  for all  $y$  such that  $x \rightarrow^+ y$ . After skolemization, expansion of the definition of RTC and hiding unnecessary formulas one obtains the following sequent.

```
Newman_lemma.2.2 :
[-1] FORALL (y:T): TC(R)(x,y) => (FORALL (b,c:T):
      RTC(R)(y,b) & RTC(R)(y,c) => joinable?(R)(b,c))
{-2} iterate(R,i)(x,b) {-3} iterate(R,j)(x,c)
[-4] local_confluent?(R) [-5] noetherian?(R)
|-----
[1] joinable?(R)(b, c)
```

To prove this goal, one analyzes the cases  $x = b$  or  $x = c$  or  $b \neq x \neq c$ . To contemplate these cases one uses the command (`case-replace "i = 0"`) which replaces `i` by 0 in the current subgoal and generates a second subgoal for the case  $x = b$ . Similarly, the case  $x = c$  is proved. The case  $x \neq b$  and  $x \neq c$ , i.e.,  $x \rightarrow x1 \rightarrow^* b$  and  $x \rightarrow x2 \rightarrow^* c$  corresponds to the following sequent obtained after some simplifications. Compare with the diagram of Figure 1(a) (replacing some variable symbols: `u`, `v` and `w`).

```
Newman_lemma.2.2.2.2.1.1 :
```

```
[-1] RTC(R)(x1,b) [-2] RTC(R)(x2,c)
[-3] FORALL (y:T): TC(R)(x,y) => (FORALL (b1,c1:T):
      RTC(R)(y,b1) & RTC(R)(y,c1) => joinable?(R)(b1,c1))
[-4] R(x,x1) [-5] R(x,x2) [-6] RTC(R)(x1,u) [-7] RTC(R)(x2,u)
[-8] noetherian?(R)
|-----
[1] j = 0 [2] i = 0 [3] joinable?(R)(b,c)
```

Firstly, make a copy of the formula -3 by using (`copy -3`).

The existence of `u` follows by expanding `local_confluent?` (instantiated with variables `x`, `x1` and `x2`), `joinable?`, by introducing `skolem` constants (`u`) and by applying disjunctive simplification `flatten`. Then one applies the lemma `R_subset_TC`, which states that a relation is contained in its transitive closure and one proves that  $x \rightarrow^+ x1$  and  $x \rightarrow^+ x2$ . Thus, the existence of `v` and `w` follows by induction hypothesis, that is by instantiating [-3] conveniently, and the lemma follows.

## 4.2 Verification of Yokouchi's Lemma

The verification starts with the sequent.

```
Yokouchi_lemma :
|-----
{1} FORALL (R, S: PRED[[T,T]]):
  (noetherian?(R) & confluent?(R) & diamond_property?(S) &
   (FORALL x,y,z: (S(x,y) & R(x,z)) =>
    (EXISTS (u:T): RTC(R)(y,u) & (RTC(R) o S o RTC(R))(z,u))))
=> diamond_property?(RTC(R) o S o RTC(R))
```

After skolemization and propositional flattening, one introduces the auxiliary lemma `Yokouchi_lemma_ax1` corresponding to the generalization  $D'$  in Figure 1. Then one obtains the new goal:

```

Yokouchi_lemma :
{-1} FORALL (R, S: PRED[[T,T]]):
(noetherian?(R) & confluent?(R) &
 (FORALL x,y,z: (S(x,y) & R(x,z)) =>
 (EXISTS (u:T): RTC(R)(y,u) & (RTC(R) o S o RTC(R))(z,u))))
=> (FORALL x,y,z: (S(x,y) & RTC(R)(x,z)) =>
 (EXISTS (w:T): RTC(R)(y,w) & (RTC(R) o S o RTC(R))(z,w)))
[-2] noetherian?(R) [-3] confluent?(R) [-4] diamond_property?(S)
[-5] FORALL x,y,z: (S(x,y) & R(x,z))
=> (EXISTS (u:T): RTC(R)(y,u) & (RTC(R) o S o RTC(R))(z,u))
|-----
[1] diamond_property?(RTC(R) o S o RTC(R))

```

Notice that the antecedents [-2], [-3] and [-5] correspond to the hypotheses of {-1}. Then, after a suitable instantiation of {-1}, propositional simplification and expansion of the definition `diamond_property?`, one introduces the noetherian induction scheme instantiating its predicate `P` as

```

LAMBDA (a:T):(FORALL (b,c:T):
 (RTC(R) o S o RTC(R))(a,c) & (RTC(R) o S o RTC(R))(a,b)
=> (EXISTS (d:T):
 (RTC(R) o S o RTC(R))(b,d) AND (RTC(R) o S o RTC(R))(c,d)))

```

Then, after splitting conjunctions, one obtains the following two subgoals:

```

Yokouchi_lemma.1 :
{-1} FORALL (x:T): FORALL (b,c:T):
 (RTC(R) o S o RTC(R))(x,c) & (RTC(R) o S o RTC(R))(x,b)
=> (EXISTS (d:T):
 (RTC(R) o S o RTC(R))(b,d) & (RTC(R) o S o RTC(R))(c,d))
...
[-7] (RTC(R) o S o RTC(R))(x,y) [-8] (RTC(R) o S o RTC(R))(x,z)
|-----
[1] EXISTS (r:T):
 (RTC(R) o S o RTC(R))(y,r) & (RTC(R) o S o RTC(R))(z,r)

```

and

```

Yokouchi_lemma.2 :
[-1] FORALL x,y,z: (S(x,y) & RTC(R)(x,z))=>
 (EXISTS (w:T): RTC(R)(y,w) & (RTC(R) o S o RTC(R))(z,w))
[-2] noetherian?(R) [-3] confluent?(R)
[-4] FORALL (x:T), (y:T), (z:T):
 S(x,y) & S(x,z) => (EXISTS (r:T): S(y,r) & S(z,r))
[-5] FORALL x,y,z: (S(x,y) & R(x,z)) =>
 (EXISTS (u:T): RTC(R)(y,u) & (RTC(R) o S o RTC(R))(z,u))
[-6] (RTC(R) o S o RTC(R))(x,y) [-7] (RTC(R) o S o RTC(R))(x,z)
|-----

```

```

{1}   FORALL (x:T): (FORALL (y:T): TC(R)(x,y) =>
  (FORALL (b,c:T): (RTC(R) o S o RTC(R))(y,c) &
   (RTC(R) o S o RTC(R))(y,b)
  => (EXISTS (d:T): (RTC(R) o S o RTC(R))(b,d) &
   (RTC(R) o S o RTC(R))(c,d)))
=> (FORALL (b,c:T):
   (RTC(R) o S o RTC(R))(x,c) & (RTC(R) o S o RTC(R))(x,b)
  => (EXISTS (d:T): (RTC(R) o S o RTC(R))(b,d) &
   (RTC(R) o S o RTC(R))(c,d)))

```

The subgoal `Yokouchi_lemma.1` is easily verified instantiating adequately the antecedent [-1] and asserting. The subgoal `Yokouchi_lemma.2` is proved following the scheme in Figure 2 as detailed below.

1. First step: introduce Skolem constants and consider the cases  $x = z_1$  and/or  $x = y_1$ .
2. Second step: invoke the lemma `iterate_RTC` which states that for all  $n$ ,  $\text{iterate}(R,n) \subseteq \text{RTC}(R)$ ; expand the definitions of composition of relations, `confluent?` and `joinable?`; hide irrelevant formulas; and then, apply disjunctive simplification.
3. Third step: conclude applying the confluence of  $R$ , the auxiliary lemma `Yokouchi_lemma_ax1` and induction hypothesis.

## 5 Conclusions and Future Work

This work illustrates that the `ars theory`, previously presented in [4], is in fact adequate for formalizing (well-known) non elementary results of ARSs. The formalizations of Newman's and Yokouchi's lemmas were described focusing on the key proof steps related with the applications of noetherian induction. Also it should be stressed here, that although this work does not advance the state of the art in the formalization of mathematics, since specifications of ARSs and even of TRSs are available since the development of the Rewriting Rule Laboratory (RRL) in the 1980's [8], it is of practical interest doubtless. In fact, the availability of rewriting proving technologies are essential in any modern proof assistant and to the best of our knowledge before the development of the `ars theory` neither rewriting *theories* nor rewriting libraries were available in PVS.

As current work we are developing `trs`, a more elaborated PVS *theory* for dealing with TRSs [5], that is of interest to verify the correction of concrete rewriting based specifications of computational objects such as reconfigurable hardware as mentioned in the introduction. By this extension rewriting strategies and new tactic-based proving techniques will be available in PVS in a natural manner. For this purpose, the type `term` built over a signature of function symbols is specified as an abstract data type [13] with the type of function symbols and the type of variables as its parameters. In the `trs theory` the type `term` is the actual parameter of the *theory* `ars[T]`. From this point, term positions are given as usual by finite sequences of naturals, and useful operations on terms

such as `subterm` at a given position and `replacement` of a subterm at a given position by using recursive declarations; `substitutions` are functions from variables into `term`. All `ars` definitions and results hold for the reduction relation induced over `term` by an specific TRS which is specified as a binary relation over `term`. The induced reduction relation is given by closing the rewriting one under substitutions and structure of terms (signature operations) as it is formalized in the standard rewriting literature [2, 17].

## References

1. M. Ayala-Rincón and T. M. Sant’Ana. SAEPTUM: Verification of ELAN Hardware Specifications using the Proof Assistant PVS. In *19th Symp. on Integrated Circuits and System Design - SBCCI06*, pages 125–130. ACM Press, 2006.
2. F. Baader and T. Nipkow. *Term Rewriting and All That*. CUP, 1998.
3. M. Bezem and T. Coquand. Neman’s Lemma - a Case Study in proof automation and geometric logic. *Bull. of the EATCS*, 79(86-100), 2003.
4. A.L. Galdino and M. Ayala-Rincón. A Theory for Abstract Rewriting Systems in PVS. In *33th Latin American Conference on Informatics - CLEI07*. Available: [www.mat.unb.br/~ayala/publications.html](http://www.mat.unb.br/~ayala/publications.html).
5. A.L. Galdino and M. Ayala-Rincón. A PVS Theory for Term Rewriting Systems. 2008. Available: [www.mat.unb.br/~ayala/publications.html](http://www.mat.unb.br/~ayala/publications.html).
6. G. Huet. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. *JACM*, 27(4):797–821, 1980.
7. G. Huet. Residual Theory in  $\lambda$ -calculus: A Formal development. *Jurnal of Functional Programming*, 4(3):371–394, 1994.
8. D. Kapur and H. Zhang. An overview of Rewrite Rule Laboratory (RRL). In N. Dershowitz, editor, *Proc. 3<sup>rd</sup> Int. Conf. on Rewriting techniques and Applications, LNCS*, 355, 1989.
9. C. Morra, J. Becker, M. Ayala-Rincón, and R. W. Hartenstein. FELIX: Using Rewriting-Logic for Generating Functionally Equivalent Implementations. In *15th Int. Conference on Field Programmable Logic and Applications - FPL 2005*, pages 25–30. IEEE CS, 2005.
10. C. Muñoz and M. Mayero. Real automation in the field. ICASE Interim Report 39 NASA/CR-2001-211271, NASA Langley Research Center, 2001.
11. M. H. A. Newman. On theories with a combinatorial definition of "equivalence". *Ann. of Math.*, 43(2):223–243, 1942.
12. S. Owre and N. Shankar. The PVS Prelude Library. Technical report, SRI-CSL-03-01, Computer Science Laboratory, SRI International, Menlo Park, CA, 2003.
13. S. Owre and N. Shankar. Abstract datatypes in PVS. Technical Report SRI-CSL-93-9R, Computer Science Laboratory, SRI International, Menlo Park, CA, December 1993. Extensively revised June 1997.
14. O. Rasmussen. The church-rosser theorem in isabelle: A proof porting experiment. Technical Report 364, University of Cambridge, Computer Lab., 1995.
15. J. L. Ruiz-Reina, J. A. Alonso, M. J. Hidalgo, and F.J. Martín-Mateos. Formalizing Rewriting in the ACL2 Theorem Prover. *LNCS*, 1930, 2001.
16. N. Shankar. A Mechanical Proof of the Church-Rosser theorem. *JACM*, 35:475–522, 1988.
17. C. J. van Rijsbergen, editor. *Term Rewriting Systems*. CUP, 2003.
18. H. Yokouchi. Church-Rosser Theorem for a Rewriting System on Categorical Combinators. *Theoretical Computer Science*, 65(3):271–290, 1989.

# Computation over Groups

Christine Gaßner\*

Institut für Mathematik und Informatik, Ernst-Moritz-Arndt-Universität,  
F.-L.-Jahn-Straße 15 a, 17487 Greifswald, Germany  
[gassnerc@uni-greifswald.de](mailto:gassnerc@uni-greifswald.de)

**Abstract.** We discuss the uniform model of computation over groups, in particular the  $P =?$  NP problems. We consider relativizations of the  $P =?$  NP question for groups, and we discuss in which setting the known method to extend and to expand structures in order to get  $P = NP$  for new structures over strings cannot be used.

## 1 Introduction

In [4] L. Blum, M. Shub, and S. Smale (BSS) introduced the uniform model of computation over the reals where the constants, the operations, and the relations in  $(\mathbb{R}; \mathbb{R}; \cdot, +, -, \geq)$  can be used in the instructions of programs. Ideas and methods of classical complexity theory (cf. [1, 2], for instance) were transferred to the BSS model (cf. [3, 4, 5]). The uniform model of computation over arbitrary algebraic structures considered in [6–10] was developed in analogy to the BSS model in order to investigate the computation over structures like groups. Here, we define this model for groups only. We want to discuss some special features of the computation over groups resulting from the existence of only one constant, which is neutral with respect to computations, and from properties of groups which are countable, which contain an element of infinite order or all elements of which have finite order. We define NP-complete problems, in particular those which are similar to the classical setting. We construct oracles  $\mathcal{O}$  in a uniform way for some classes of groups such that there holds  $P_{\mathcal{S}}^{\mathcal{O}} = NP_{\mathcal{S}}^{\mathcal{O}}$  or  $P_{\mathcal{S}}^{\mathcal{O}} \neq NP_{\mathcal{S}}^{\mathcal{O}}$  for several kinds of  $\mathcal{S}$ -machines over groups. Moreover, we explain why it is not possible to construct new relations over extensions of any group  $\mathcal{G}$  in the known way in order to get  $P_{\mathcal{S}} = NP_{\mathcal{S}}$  for new structures  $\mathcal{S}$  over strings if not permitting additional parameters.

For groups  $(G; e; \circ; =)$  we consider the computation over  $(G; e; \circ; =)$  without parameters and over  $(G; G; \circ; =)$  with parameters. The computation without additional parameters means that only the unit element  $e$  can be a machine constant. In the other case, we allow any element to be a machine constant. Any deterministic (BSS) machine restricted to a group  $(G; e; \circ; =)$ , (for instance, to  $(\mathbb{R}; 0; +; =)$ ) cannot compute new values in  $G \setminus \{e\}$  from an input of the form  $(e, \dots, e)$ . For this reason, neither the old form of machines in [4] nor the new form of machines in [3] is suitable for computation over groups. Therefore, in the

---

\* I thank R. Bialowons, M. Kläre, R. Schimming, and the referees for helpful hints.

definition of our model some features of the BSS model have to be modified so that the model becomes suitable for groups. Additionally to an infinite number of registers for the elements of the structure, each machine is provided with a finite number of index registers. The index registers can be used in order to simulate the usual *while* and *loop* instructions in processing all input values  $x_1, \dots, x_n$  for any  $n$ . In the model over the reals introduced in [4], two additional index registers are sufficient since auxiliary calculation, necessary to compute new indices, can be executed by means of the elements of the ring. In [3] the BSS machines assume a new form which is similar to the form of special Turing machines whose cells can contain real numbers. A blank symbol and index registers are not used. The length of the inputs  $(x_1, \dots, x_n)$  is determined by storing the unary code of  $n$  in additional registers on the left side of the point.

$$\dots, 0, 0, \underbrace{1, \dots, 1}_{n \times} . x_1, x_2, \dots, x_n, 0, 0, \dots$$

Hence, for this model, at least two constants are necessary.

## 2 The Model of Computation over Groups

For each group  $\mathcal{G} = (G; e; \circ; =)$  where  $|G| \geq 2$ , let  $\bar{\mathcal{G}} = (G; G; \circ; =)$ . For the structures  $\mathcal{S} \in \{\mathcal{G}, \bar{\mathcal{G}}\}$ , every  $\mathcal{S}$ -machine  $\mathcal{M}$  is provided with registers  $Z_1, Z_2, \dots$  for the elements of  $G$  and with a fixed number of registers  $I_1, I_2, \dots, I_{k_{\mathcal{M}}}$  for indices in  $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$ . For an input  $(x_1, \dots, x_n) \in G^\infty =_{\text{df}} \bigcup_{i=1}^{\infty} G^i$ , the sequence  $x_1, \dots, x_{n-1}, x_n, x_n, x_n, \dots$  is assigned to the registers  $Z_1, Z_2, \dots$  and the index registers get the content  $n$ . The labelled *computation*, *copy*, and *branching* instructions have the form  $Z_j := Z_{j_1} \circ Z_{j_2};$ ,  $Z_j := e$ ; (resp.  $Z_j := g$ ; for any machine constant  $g \in G$ ),  $Z_{I_j} := Z_{I_k};$ , and *if*  $Z_j = Z_k$  *then goto*  $l_1$  *else goto*  $l_2$ ;. The index registers are used in the copy instructions. For copying, we also allow  $I_j := 1;$ ,  $I_j := I_j + 1;$ , and *if*  $I_j = I_k$  *then goto*  $l_1$  *else goto*  $l_2$ ;. If the *output* instruction is reached, then  $(Z_1, \dots, Z_{I_1})$  is the output and the machine halts.

Note that the machines over another structure  $\mathcal{S}$  can use all operations and all relations of  $\mathcal{S}$  in the computation and branching instructions, respectively.

The *non-deterministic machines* are able to guess an arbitrary number of arbitrary elements  $y_1, \dots, y_m \in G$  in one step after the input and to assign the guesses to  $Z_{I_1+1}, \dots, Z_{I_1+m}$ . Note that we make no restriction of the domain for  $m$ . To simplify matters,  $m$  is independent of  $n$ . However, a machine can use at most  $t$  guesses within  $t$  steps. In any case, the size of an input  $(x_1, \dots, x_n)$  is, by definition, its arity  $n$ . The *digital* (or *binary*) *non-deterministic machines* cannot guess any elements. They only can execute additional instructions of the form *goto*  $l_1$  *or goto*  $l_2$ ;. Semi-decidability can be defined as usual. If  $e$  is the only constant, then we say that the *decidability* of a problem  $A \subseteq G^\infty$  results from the semi-decidability of  $A$  and of  $G^\infty \setminus A$ . Moreover, *oracle machines* can execute *if*  $(Z_1, \dots, Z_{I_1}) \in \mathcal{O}$  *then goto*  $l_1$  *else goto*  $l_2$ ; for some oracle  $\mathcal{O} \subseteq G^\infty$ .

Let  $M_{\mathcal{S}}(\mathcal{O})$  and  $M_{\mathcal{S}}^{[D]\mathbb{N}}(\mathcal{O})$  be the sets of deterministic and [digital] non-deterministic  $\mathcal{S}$ -machines, respectively, which can use the oracle  $\mathcal{O}$ .

Further, let  $P_S^\mathcal{O}$ ,  $DNP_S^\mathcal{O}$ , and  $NP_S^\mathcal{O}$  denote the usual complexity classes of decision problems  $A \subseteq G^\infty$  decided or non-deterministically recognized by an oracle machine in  $M_S(\mathcal{O})$ , in  $M_S^{DN}(\mathcal{O})$ , or in  $M_S^N(\mathcal{O})$  in polynomial time (where an input is in  $A$  iff this machine halts on this input: without guesses, for some guessed sequence of instructions, or for some guesses), and let  $P_S = P_S^\emptyset$ ,  $DNP_S = DNP_S^\emptyset$ , and  $NP_S = NP_S^\emptyset$ . Note, that, for groups  $\mathcal{G}$  containing at least two elements, we have  $P_{\mathcal{G}}^\mathcal{O} \subseteq DNP_{\mathcal{G}}^\mathcal{O} \subseteq NP_{\mathcal{G}}^\mathcal{O}$  and  $P_{\mathcal{G}}^\mathcal{O} \subseteq DNP_{\mathcal{G}}^\mathcal{O} \subseteq NP_{\mathcal{G}}^\mathcal{O}$ . For groups with a finite universe  $G = \{g_1, \dots, g_k\}$  where  $k \geq 2$  we get  $DNP_{\mathcal{G}}^\mathcal{O} = NP_{\mathcal{G}}^\mathcal{O}$  since a  $DNP_{\mathcal{G}}^\mathcal{O}$ -machine can select one of the instructions  $Z_j := g_1; \dots; Z_j := g_k$ ; non-deterministically.

### 3 A Finite Group with $P_{\mathcal{G}} \neq NP_{\mathcal{G}}$

For all infinite abelian groups  $\mathcal{G}$ , there holds  $P_{\mathcal{G}} \neq DNP_{\mathcal{G}}$  and  $P_{\bar{\mathcal{G}}} \neq DNP_{\bar{\mathcal{G}}}$  (cf. [13, 6, 14, 12]). Moreover, it is known that there are infinite abelian groups  $\mathcal{G}$  satisfying  $DNP_{\bar{\mathcal{G}}} = NP_{\bar{\mathcal{G}}}$  [11]. On the other hand there are infinite abelian groups  $\mathcal{G}$  satisfying  $DNP_{\mathcal{G}} \neq NP_{\mathcal{G}}$  and  $DNP_{\bar{\mathcal{G}}} \neq NP_{\bar{\mathcal{G}}}$ . For instance, there holds  $\{x \in \mathbb{Z} \mid (\exists y \in \mathbb{Z})(x = y + y)\} \in NP_{(\mathbb{Z}; 0; +; =)} \setminus DNP_{(\mathbb{Z}; \mathbb{Z}; +; =)}$ .

In order to demonstrate the difference between the computation with and without parameters let us consider the dihedral group  $\mathcal{D}_{2,4} = (D; e; \cdot; =)$  of order 8, the elements of which are the isometric transformations of a square onto itself. The group can be described by two generating elements  $r$  (a rotation of a square) and  $s$  (a reflection of a square) where  $r^4 = e$ ,  $s^2 = e$ , and  $sr = r^3s$  hold. We have  $D = \{e, r, r^2, r^3, s, rs, r^2s, r^3s\}$ . The center of the group which is, by definition, the set  $C = \{x \in D \mid (\forall y \in D)(x \cdot y = y \cdot x)\}$  consists of the elements  $e$  and  $r^2$ . The elements  $e, r^2, s, rs, r^2s, r^3s$  are of order 2, the elements  $r$  and  $r^3$  are of order 4.

**Proposition 1.** *For  $\mathcal{D}_{2,4} = (D; e; \cdot; =)$ , there holds  $P_{\mathcal{D}_{2,4}} \neq NP_{\mathcal{D}_{2,4}}$ .*

*Proof.* For  $\mathcal{D}_{2,4} = (D; e; \cdot; =)$ , we have  $D \setminus C \in NP_{\mathcal{D}_{2,4}}$  since for any  $x \in D \setminus C$ , an element  $y$  can be guessed for which  $x \cdot y \neq y \cdot x$  holds.

On the other hand, if  $\mathcal{M}$  is any deterministic  $\mathcal{D}_{2,4}$ -machine, then all inputs  $x \in D$  (which have the arity 1) of order 2 have the same computation path. Thus,  $\mathcal{M}$  accepts all inputs  $x \in D$  of order 2 or, in the other case, it rejects all inputs  $x \in D$  of order 2. Consequently, no deterministic  $\mathcal{D}_{2,4}$ -machine can decide the problem  $D \setminus C$ , since  $r^2 \in C$  and  $s \in D \setminus C$  are of the same order. Therefore,  $D \setminus C$  is not in  $P_{\mathcal{D}_{2,4}}$ .  $\square$

For  $\bar{\mathcal{D}}_{2,4} = (D; D; \cdot; =)$ , the simulation of any  $\bar{\mathcal{D}}_{2,4}$ -machine by a Turing machine, and vice versa, is possible. Since any element of  $D$  can be encoded by a tuple over  $\{0, 1\}$ , a Turing machine can simulate any computation over  $D$ , especially, it can simulate the instruction  $Z_j := d$ ; by using the code of  $d$ . On the other hand, any machine can simulate a Turing machine if it possesses two constants. If, in contrast to the  $\mathcal{D}_{2,4}$ -machines, any element of  $D$  is permitted to be a machine constant, we get compatible results to the classical setting.

**Proposition 2.**  $P_{\bar{\mathcal{D}}_{2,4}} = NP_{\bar{\mathcal{D}}_{2,4}}$  holds if and only if  $P = NP$  holds.

## 4 NP-Complete Problems

For any group  $\mathcal{G} = (G; e; \circ; =)$ , we can define a universal non-deterministic  $\bar{\mathcal{G}}$ -machine which is able to simulate each step of any non-deterministic  $\bar{\mathcal{G}}$ -machine  $\mathcal{M}$  on an input  $\mathbf{x}$  in polynomial time if it gets  $\mathbf{x}$  and a suitable code of  $\mathcal{M}$  as input.

Let  $G \cup \{a, b\}$  be an alphabet where  $a \neq b$  and  $\{a, b\} \subseteq G$  or  $a = 0$  and  $b = 1$ . In order to define such a universal problem, we first encode the programs of  $\bar{\mathcal{G}}$ -machines by strings in  $(G \cup \{a, b\})^*$  and then we transform these strings into suitable tuples. We distinguish between strings and tuples since the strings of the length  $k$  have to be transformed in  $k$ -tuples in order to store it in  $k$  registers.

**Definition 1.** For every non-empty string  $s = c_1 \cdots c_k \in (G^*)^{(=k)} \subset G^*$ , let  $[s]$  be the representation of  $s$  in the form of a tuple  $(c_1, \dots, c_k) \in G^k \subset G^\infty$ , that means that  $[c_1 \cdots c_k] = (c_1, \dots, c_k)$ .

**Definition 2.** Let  $Code_{a,b}^*$  be an injective mapping of the set of all deterministic and non-deterministic  $\bar{\mathcal{G}}$ -machines into  $(G \cup \{a, b\})^*$  such that, with the exception of the machine constants in  $G \setminus \{e\}$ , every character of the program is unambiguously translated into a string in  $\{a, b\}^*$  by this mapping and such that the codes contain the sub-string  $b^2$  as prefix only and the last character of the codes is  $a$ . The machine constants are encoded by themselves. For any  $\bar{\mathcal{G}}$ -machine  $\mathcal{M}$  and  $a, b \in G$ , let  $Code_{a,b}(\mathcal{M}) = [Code_{a,b}^*(\mathcal{M})]$ .

Note that  $(\mathbf{x}, [c_1 \cdots c_k])$  stands for  $(x_1, \dots, x_n, c_1, \dots, c_k)$ , and the like. The following problems can be recognized by a universal machine.

**Definition 3.** For  $\mathcal{G} = (G; e; \circ; =)$  with  $a, b \in G$  and  $a \neq b$  and for any  $\mathcal{O} \subseteq G^\infty$ , let the Universal  $NP_{\bar{\mathcal{G}}}^{\mathcal{O}}$ -Problem  $UNI_{\bar{\mathcal{G}}}(\mathcal{O})$  be the set

$$\{(\mathbf{x}, Code_{a,b}(\mathcal{M}), [b^t]) \mid \mathbf{x} \in G^\infty \& \mathcal{M} \in M_{\bar{\mathcal{G}}}^N(\mathcal{O}) \& \mathcal{M}(\mathbf{x}) \downarrow^t\}$$

where  $\mathcal{M}(\mathbf{x}) \downarrow^t$  means that  $\mathcal{M}$  accepts  $\mathbf{x}$  within  $t$  steps.

**Proposition 3.** For  $\mathcal{G}$ -machines where  $|G| \geq 2$ ,  $UNI_{\bar{\mathcal{G}}}(\emptyset)$  is  $NP_{\bar{\mathcal{G}}}$ -complete. Moreover,  $UNI_{\bar{\mathcal{G}}}(\mathcal{O})$  is  $NP_{\bar{\mathcal{G}}}^{\mathcal{O}}$ -complete.

**Definition 4.** For  $\mathcal{G} = (G; e; \circ; =)$  and any  $\mathcal{O} \subseteq G^\infty$ , let the Universal  $NP_{\bar{\mathcal{G}}}^{\mathcal{O}}$ -Problem  $UNI_{\bar{\mathcal{G}}}(\mathcal{O})$  be the set

$$\bigcup_{\substack{a,b \in G \\ a \neq b}} \{(\mathbf{x}, Code_{a,b}(\mathcal{M}), [b^t]) \mid \mathbf{x} \in G^\infty \& \mathcal{M} \in M_{\bar{\mathcal{G}}}^N(\mathcal{O}) \& \mathcal{M}(\mathbf{x}) \downarrow^t\}$$

$$\cup \underbrace{\{(e, e, e, \dots, e) \mid \mathcal{M} \in M_{\bar{\mathcal{G}}}^N(\mathcal{O}) \& \mathcal{M}([e^n]) \downarrow^t\}}_{c(n, C(\mathcal{M}), t) \times}$$

where  $c(n_1, n_2, n_3) = Cantor(Cantor(n_1, n_2), n_3)$  is the number of  $(n_1, n_2, n_3)$  given by the Cantor diagonal method and  $C(\mathcal{M})$  is the number whose binary representation is  $Code_{0,1}^*(\mathcal{M})$ .

If an input  $\mathbf{x}$  contains a component  $x_i \neq e$ , a reduction machine can compute  $(\mathbf{x}, \text{Code}_{x_i, e}(\mathcal{M}), \lceil e^t \rceil)$ . In the other case a reduction to the latter subset is possible.

**Proposition 4.** *For  $\mathcal{G}$ -machines,  $\text{UNI}_{\mathcal{G}}(\emptyset)$  is  $\text{NP}_{\mathcal{G}}$ -complete.  $\text{UNI}_{\mathcal{G}}(\mathcal{O})$  is  $\text{NP}_{\mathcal{G}}^{\mathcal{O}}$ -complete.*

## 5 Some Oracles $\mathcal{O}$ with $P_{\mathcal{S}}^{\mathcal{O}} = \text{NP}_{\mathcal{S}}^{\mathcal{O}}$

We transfer the ideas given in [1] and [5] and modify the definitions, appropriately. The tuples which can occur in the oracles  $\mathcal{O}_1^{(\mathcal{G})}$  and  $\mathcal{O}_2^{(\mathcal{G})}$  have the same form as the elements of a universal problem. Whereas the definition in [5] is recursive on the number of steps occurring in these tuples, we define the oracle recursively on the length (size) of tuples as in [1].

**Definition 5.** *Let  $\mathcal{O}_1 (= \mathcal{O}_1^{(\mathcal{G})})$  and  $\mathcal{O}_2 (= \mathcal{O}_2^{(\mathcal{G})})$  (For a given  $\mathcal{G}$ , we omit the index  $\mathcal{G}$ .) be universal oracles defined by*

$$\begin{aligned}\mathcal{O}_1 &= \bigcup_{k \in \mathbb{N}} V_k, & V_0 &= \emptyset, & V_k &= G^k \cap \text{UNI}_{\mathcal{G}}(\bigcup_{j < k} V_j), \\ \mathcal{O}_2 &= \bigcup_{k \in \mathbb{N}} W_k, & W_0 &= \emptyset, & W_k &= G^k \cap \text{UNI}_{\mathcal{G}}(\bigcup_{j < k} W_j).\end{aligned}$$

$\text{UNI}_{\mathcal{G}}(\mathcal{O}_1) \cap G^k \subseteq V_k$  and  $\text{UNI}_{\mathcal{G}}(\mathcal{O}_2) \cap G^k \subseteq W_k$  hold since we assume that the codes of the oracle machines do not depend on the used oracle. This implies  $\text{UNI}_{\mathcal{G}}(\mathcal{O}_1) = \mathcal{O}_1$  and  $\text{UNI}_{\mathcal{G}}(\mathcal{O}_2) = \mathcal{O}_2$ , respectively.

**Proposition 5.** *For any group  $\mathcal{G}$ , there holds  $P_{\mathcal{G}}^{\mathcal{O}_2} = \text{NP}_{\mathcal{G}}^{\mathcal{O}_2}$ . If  $|G| \geq 2$ , then there holds  $P_{\mathcal{G}}^{\mathcal{O}_1} = \text{NP}_{\mathcal{G}}^{\mathcal{O}_1}$ .*

## 6 Some oracles $\mathcal{Q}$ with $P_{\mathcal{S}}^{\mathcal{Q}} \neq \text{NP}_{\mathcal{S}}^{\mathcal{Q}}$

In order to get the inequality between the corresponding relativized complexity classes with respect to some oracles for groups  $\mathcal{G}$ , we want to define oracles  $\mathcal{Q}_i (= \mathcal{Q}_i^{(\mathcal{G})})$  recursively. The ideas particularly go back to [1] and [5]. As T. Baker, J. Gill, and R. Solovay and T. Emerson we use diagonalization techniques in order to construct these oracles and several problems  $L_i (= L_i^{(\mathcal{G})})$  such that  $L_1 \in \text{NP}_{\mathcal{G}}^{\mathcal{Q}_1} \setminus \text{DNP}_{\mathcal{G}}^{\mathcal{Q}_1}$ ,  $L_2 \in (\text{NP}_{\mathcal{G}}^{\mathcal{Q}_2} \cap \text{DNP}_{\mathcal{G}}^{\mathcal{Q}_2}) \setminus P_{\mathcal{G}}^{\mathcal{Q}_2}$ , and  $L_i \in \text{NP}_{\mathcal{G}}^{\mathcal{Q}_i} \setminus \text{DNP}_{\mathcal{G}}^{\mathcal{Q}_i}$  for  $i \in \{3, 4\}$  for several kinds of machines.

### 6.1 Constructions for Countable Sets of Machine Constants

The first definition works for all infinite groups. Since groups are structures of finite signature, we can take the positive integers in order to encode all programs of digital non-deterministic oracle  $(G; e; \circ; =)$ -machines using one given oracle. Consequently, the set of all couples of all polynomials and of these programs

whose form (including the oracle queries) is independent of the used oracle can be enumerated. Let  $i \in \mathbb{N}^+$  be the code of the  $i^{\text{th}}$  pair  $(p_i, P_i)$  which determines a class of digital non-deterministic oracle machines  $\{\mathcal{N}_i^{\mathcal{B}} \mid \mathcal{B} \subseteq G^\infty\}$  by the following.

- (a) For any input in  $G^n$ , the machine  $\mathcal{N}_i^{\mathcal{B}}$  executes  $p_i(n)$  steps of the program  $P_i$ .
- (b) If  $\mathcal{N}_i^{\mathcal{B}}$  queries an oracle, then  $\mathcal{N}_i^{\mathcal{B}}$  uses the oracle  $\mathcal{B}$ .
- (c)  $\mathcal{N}_i^{\mathcal{B}}$  simultaneously counts the number of steps which are determined by  $P_i$  by means of an additional index register.
- (d) For any input in  $G^n$  and for any guessed sequence of instructions, the machine  $\mathcal{N}_i^{\mathcal{B}}$  makes an output after at most  $p_i(n)$  steps of the execution of  $P_i$  if the output of  $P_i$  is reached in this time. If the output instruction of  $P_i$  is not reached in this time, then  $\mathcal{N}_i^{\mathcal{B}}$  does not halt for the guessed sequence of instructions and it does not compute new values after the first  $p_i(n)$  steps.

Then, for any digital non-deterministic  $\mathcal{G}$ -machine  $\mathcal{M}$ , which recognizes some problem in  $\text{DNP}_{\mathcal{G}}^{\mathcal{B}}$ , there is some  $i \in \mathbb{N}^+$  such that  $\mathcal{M}$  and  $\mathcal{N}_i^{\mathcal{B}}$  recognize the same problem.

**The Construction of  $\mathcal{Q}_1$ .** Let  $V_0 = \emptyset$  and  $m_0 = 0$ . We construct the set  $\mathcal{Q}_1$  in stages. For  $i \geq 1$ , let  $n_i$  some integer such that  $n_i > m_{i-1}$  and  $p_i(n_i) + n_i < 2^{n_i}$ . Moreover, let

$$W_i = \bigcup_{j < i} V_j,$$

$$\begin{aligned} V_i &= \{x \in G^{n_i} \mid \mathcal{N}_i^{W_i} \text{ does not accept } (e, \dots, e) \in G^{n_i} \text{ non-deterministically} \\ &\quad \& x \text{ is not queried by } \mathcal{N}_i^{W_i} \text{ on input } (e, \dots, e) \in G^{n_i}\}, \\ m_i &= 2^{n_i}. \end{aligned}$$

Finally, let  $\mathcal{Q}_1 = \bigcup_{i \in \mathbb{N}^+} W_i$  and  $L_1 = \{y \mid (\exists i \in \mathbb{N}^+) (y \in G^{n_i} \& V_i \neq \emptyset)\}$ .

Since the set  $L_1$  belongs to  $\text{NP}_{\mathcal{G}}^{\mathcal{Q}_1}$ , we get the following analogously to [1].

**Lemma 1.** *For any infinite group  $\mathcal{G}$ , there holds  $L_1 \in \text{NP}_{\mathcal{G}}^{\mathcal{Q}_1} \setminus \text{DNP}_{\mathcal{G}}^{\mathcal{Q}_1}$ .*

*Proof.* The property, that  $G$  is infinite, implies that  $V_i \neq \emptyset$  if  $\mathcal{N}_i^{W_i}$  does not accept  $(e, \dots, e) \in G^{n_i}$  non-deterministically. The computation of  $\mathcal{N}_i^{W_i}$  and  $\mathcal{N}_i^{\mathcal{Q}_1}$  is the same on input  $(e, \dots, e) \in G^{n_i}$  since the length of the tuples in the oracle queries are bounded by  $p_i(n_i) + n_i$  and the queries of  $\mathcal{N}_i^{W_i}$  and  $\mathcal{N}_i^{W_{i+1}}$ , with respect to  $V_i$ , are the same. Since, for any  $i \geq 1$ , the machine  $\mathcal{N}_i^{W_i}$  accepts  $(e, \dots, e) \in G^{n_i}$  iff  $L_1 \cap G^{n_i} = \emptyset$  and it does not accept  $(e, \dots, e) \in G^{n_i}$  iff  $L_1 \cap G^{n_i} = G^{n_i}$ , there holds  $L_1 \notin \text{DNP}_{\mathcal{G}}^{\mathcal{Q}_1}$ .  $\square$

**Proposition 6.** *For any infinite group  $\mathcal{G}$ , there is an oracle  $\mathcal{Q}$  such that  $\text{DNP}_{\mathcal{G}}^{\mathcal{Q}} \neq \text{NP}_{\mathcal{G}}^{\mathcal{Q}}$ .*

**Corollary 1.** *For any infinite group  $\mathcal{G}$ , there is an oracle  $\mathcal{Q}$  such that  $\text{P}_{\mathcal{G}}^{\mathcal{Q}} \neq \text{NP}_{\mathcal{G}}^{\mathcal{Q}}$ .*

For any group  $\mathcal{G}$  defined on a countable (finite or infinite) universe  $G$  where  $|G| \geq 2$ , the construction can be transferred to deterministic  $\bar{\mathcal{G}}$ -machines. Let us assume that the machines  $\tilde{\mathcal{N}}_i^{\mathcal{B}}$  are the deterministic oracle machines over  $\bar{\mathcal{G}}$  and that there is some  $g \in G \setminus \{e\}$ . Then, the sets  $\tilde{V}_i$ ,  $\tilde{W}_i$ ,  $\mathcal{Q}_2$ , and  $L_2$  can be defined in the same manner and  $L_2$  is in  $(NP_{\bar{\mathcal{G}}}^{\mathcal{Q}_2} \cap DNP_{\bar{\mathcal{G}}}^{\mathcal{Q}_2}) \setminus P_{\bar{\mathcal{G}}}^{\mathcal{Q}_2}$  since there are machines in  $M_{\bar{\mathcal{G}}}^N(Q_2)$  and in  $M_{\bar{\mathcal{G}}}^{DN}(Q_2)$  which can guess or select every combination of  $e$ 's and  $g$ 's non-deterministically in order to use it in a query. For a group of two elements, the construction is the same as in [1]. Here,  $p_i(n_i) < 2^{n_i}$  implies that  $\tilde{V}_i \neq \emptyset$  if  $\tilde{\mathcal{N}}_i^{\tilde{W}_i}$  does not accept  $(e, \dots, e) \in G^{n_i}$ .

**Lemma 2.** *For any group  $\mathcal{G}$  containing at least two elements, if  $G$  is countable, there holds  $L_2 \in DNP_{\bar{\mathcal{G}}}^{\mathcal{Q}_2} \setminus P_{\bar{\mathcal{G}}}^{\mathcal{Q}_2}$  and  $L_2 \in NP_{\bar{\mathcal{G}}}^{\mathcal{Q}_2}$ .*

**Proposition 7.** *If a group  $\mathcal{G}$  contains at least two elements and the universe is countable, then there is an oracle  $\mathcal{Q}$  such that  $P_{\bar{\mathcal{G}}}^{\mathcal{Q}} \neq DNP_{\bar{\mathcal{G}}}^{\mathcal{Q}}$  and  $P_{\bar{\mathcal{G}}}^{\mathcal{Q}} \neq NP_{\bar{\mathcal{G}}}^{\mathcal{Q}}$ .*

**Corollary 2.** *If a group  $\mathcal{G}$  contains at least two elements and the universe is countable, then there is an oracle  $\mathcal{Q}$  such that  $P_{\bar{\mathcal{G}}}^{\mathcal{Q}} \neq NP_{\bar{\mathcal{G}}}^{\mathcal{Q}}$ .*

## 6.2 Further Constructions for Infinite Sets of Machine Constants

Now we consider structures of the form  $\bar{\mathcal{G}} = (G; G; \circ; =)$  where the universe is infinite and does not need to be countable. We know that any oracle machine over  $\bar{\mathcal{G}}$  can be encoded by elements of  $G^\infty$  where, for any oracle, any query is encoded by the same code. Let  $\mathcal{U} \subseteq G^\infty$  be the set of codes for all couples of all polynomials and of all programs of digital non-deterministic oracle machines over  $\bar{\mathcal{G}}$  (independent of the used oracle). Then, any  $\mathbf{u} \in \mathcal{U}$  is the code for a pair  $(p_{\mathbf{u}}, P_{\mathbf{u}})$  which determines a class of digital non-deterministic oracle machines  $\{\mathcal{N}_{\mathbf{u}}^{\mathcal{B}} \mid \mathcal{B} \subseteq G^\infty\}$  by properties analogously to (a), (b), (c), and (d).

**The Construction of  $\mathcal{Q}_3$ .** Let us assume that  $G$  contains an element of infinite order,  $a$ . Let  $V_0 = \emptyset$ . We construct the set  $\mathcal{Q}_3$  in stages. For  $i \geq 1$ , let

$$\begin{aligned} K_i &= \{\mathbf{u} \in \mathcal{U} \mid (\forall j > i)(\forall \mathcal{B} \subseteq G^\infty) \\ &\quad (\mathcal{N}_{\mathbf{u}}^{\mathcal{B}} \text{ does not compute or use the value } a^j \text{ on input } \mathbf{u})\}, \\ W_i &= \bigcup_{k < i} V_k, \end{aligned}$$

$$V_i = \{(a^{i+1}, \mathbf{u}) \mid \mathbf{u} \in K_i \text{ \& } \mathcal{N}_{\mathbf{u}}^{W_i} \text{ does not accept } \mathbf{u} \text{ non-deterministically}\}.$$

Finally, let  $\mathcal{Q}_3 = \bigcup_{i \in \mathbb{N}^+} W_i$  and  $L_3 = \{\mathbf{y} \mid (\exists n \geq 2)((a^n, \mathbf{y}) \in \mathcal{Q}_3)\}$ .

**Lemma 3.** *For groups  $\mathcal{G}$  containing an element  $a$  for which  $a^i \neq a^j$  iff  $i \neq j$ ,  $L_3 \in NP_{\bar{\mathcal{G}}}^{\mathcal{Q}_3} \setminus DNP_{\bar{\mathcal{G}}}^{\mathcal{Q}_3}$ .*

*Proof.* Let us assume that there is some machine  $\mathcal{N}_{\mathbf{u}}^{\mathcal{Q}_3}$  with property (A).

(A)  $\mathcal{N}_{\mathbf{u}}^{\mathcal{Q}_3}$  recognizes  $L_3$ .

In any case we have  $\mathbf{u} \in L_3$  or  $\mathbf{u} \notin L_3$ . If  $\mathbf{u} \in L_3$ , then there is some  $i$  such that  $(a^{i+1}, \mathbf{u}) \in V_i$ . By the definition of  $K_i$  the machines  $\mathcal{N}_{\mathbf{u}}^{\mathcal{Q}_3}$  and  $\mathcal{N}_{\mathbf{u}}^{W_i}$  do not compute or use any value  $a^j$  on input  $\mathbf{u}$  if  $j > i$ . Hence,  $\mathcal{N}_{\mathbf{u}}^{\mathcal{Q}_3}$  works like  $\mathcal{N}_{\mathbf{u}}^{W_i}$  on  $\mathbf{u}$  and, consequently, it does not accept  $\mathbf{u}$  non-deterministically such that  $\mathbf{u} \notin L_3$  because of (A). If  $\mathbf{u} \notin L_3$ , then because of (A),  $\mathcal{N}_{\mathbf{u}}^{\mathcal{Q}_3}$  does not accept  $\mathbf{u}$ . However, by definition of  $K_1, K_2, \dots$  there is some  $i_0$  such that  $\mathbf{u} \in K_i$  for all  $i \geq i_0$ . Thus,  $(a^{i_0+1}, \mathbf{u}) \in \mathcal{Q}_3$  and hence  $\mathbf{u} \in L_3$ .  $\square$

**Proposition 8.** *For groups  $\mathcal{G}$  containing an element of infinite order, there is an oracle  $\mathcal{Q}$  such that  $\text{DNP}_{\mathcal{G}}^{\mathcal{Q}} \neq \text{NP}_{\mathcal{G}}^{\mathcal{Q}}$ .*

**Corollary 3.** *For groups  $\mathcal{G}$  containing an element of infinite order, there is an oracle  $\mathcal{Q}$  such that  $\text{P}_{\mathcal{G}}^{\mathcal{Q}} \neq \text{NP}_{\mathcal{G}}^{\mathcal{Q}}$ .*

Similar constructions are possible for any infinite group. For any infinite abelian group which does not contain an element of infinite order, there is an infinite sequence of elements such that any new element is independent of its predecessors. For these groups we can choose the following oracle.

**The Construction of  $\mathcal{Q}_4$ .** Let  $(G_i)_{i \in \mathbb{N}^+}$  be an infinite sequence of subgroups such that  $G_i \subset G_{i+1}$ . Let  $V_0 = \emptyset$ . We construct the set  $\mathcal{Q}_4$  in stages. For  $i \geq 1$ , let

$$\begin{aligned} K_i &= \{\mathbf{u} \in \mathcal{U} \mid (\forall j > i)(\forall \mathcal{B} \subseteq G^\infty) \\ &\quad (\mathcal{N}_{\mathbf{u}}^{\mathcal{B}} \text{ does not compute or use values in } G_j \setminus G_i \text{ on input } \mathbf{u})\}, \\ W_i &= \bigcup_{k < i} V_k, \end{aligned}$$

$$V_i = \{(g, \mathbf{u}) \mid g \in G_{i+1} \setminus G_i \text{ & } \mathbf{u} \in K_i \text{ & } \mathcal{N}_{\mathbf{u}}^{W_i} \text{ does not accept } \mathbf{u}\}.$$

$$\text{Let } \mathcal{Q}_4 = \bigcup_{i \in \mathbb{N}^+} W_i \text{ and } L_4 = \{\mathbf{y} \mid (\exists n \geq 2)(\exists g \in G_n)((g, \mathbf{y}) \in \mathcal{Q}_4)\}.$$

**Lemma 4.** *For any infinite abelian group  $\mathcal{G}$  which contains an infinite sequence of subgroups  $G_i \subset G_{i+1}$ ,  $L_4 \in \text{NP}_{\mathcal{G}}^{\mathcal{Q}_4} \setminus \text{DNP}_{\mathcal{G}}^{\mathcal{Q}_4}$ .*

**Proposition 9.** *For any infinite abelian group  $\mathcal{G}$  which does not contain an element of infinite order, there exists an oracle  $\mathcal{Q}$  such that  $\text{DNP}_{\mathcal{G}}^{\mathcal{Q}} \neq \text{NP}_{\mathcal{G}}^{\mathcal{Q}}$ .*

**Corollary 4.** *For any infinite abelian group  $\mathcal{G}$  which does not contain an element of infinite order, there exists an oracle  $\mathcal{Q}$  such that  $\text{P}_{\mathcal{G}}^{\mathcal{Q}} \neq \text{NP}_{\mathcal{G}}^{\mathcal{Q}}$ .*

### 6.3 Open Questions

In order to get  $\text{P}_{\mathcal{S}}^{\mathcal{Q}} \neq \text{NP}_{\mathcal{S}}^{\mathcal{Q}}$  for some structures  $\mathcal{S}$  in  $\{\mathcal{G}, \bar{\mathcal{G}}\}$ , we showed one of the inequalities  $\text{P}_{\mathcal{S}}^{\mathcal{Q}} \neq \text{DNP}_{\mathcal{S}}^{\mathcal{Q}}$  and  $\text{DNP}_{\mathcal{S}}^{\mathcal{Q}} \neq \text{NP}_{\mathcal{S}}^{\mathcal{Q}}$  by uniform constructions for some classes of groups. Of course, for special structures we even know  $\text{P}_{\mathcal{S}}^{\emptyset} \neq \text{DNP}_{\mathcal{S}}^{\emptyset}$  and  $\text{DNP}_{\mathcal{S}}^{\emptyset} \neq \text{NP}_{\mathcal{S}}^{\emptyset}$ . If the set of constants of any structure  $\mathcal{S}$  is not countable, then no uniform construction of oracles  $\mathcal{Q}$  is known which imply the inequalities  $\text{P}_{\mathcal{S}}^{\mathcal{Q}} \neq \text{DNP}_{\mathcal{S}}^{\mathcal{Q}}$ . Moreover we do not know any uniform construction of oracles  $\mathcal{Q}$  which imply the inequalities  $\text{P}_{\mathcal{S}}^{\mathcal{Q}} \neq \text{DNP}_{\mathcal{S}}^{\mathcal{Q}}$  for infinite structures containing only one constant or which imply the inequalities  $\text{P}_{\mathcal{S}}^{\mathcal{Q}} \neq \text{DNP}_{\mathcal{S}}^{\mathcal{Q}}$  for finite structures containing only one constant.

## 7 Expansions of Extensions with $P_S = NP_S$ ?

Let us explain the meaning of padding in the known constructions of structures  $\mathcal{S}$  over strings with  $P_S = NP_S$  and its bounds for groups. As in [9], for every group  $\mathcal{G} = (G; e; \circ; =)$ , where  $G$  contains two elements  $a$  and  $b$ , we can embed the structure  $\bar{\mathcal{G}}$  into a new structure of the form  $\bar{\mathcal{G}}^* = (G^*; G \cup \{\varepsilon\}; \circ, add, sub_l, sub_r, =)$  and afterward into a new structure of the form

$$\bar{\mathcal{G}}_{R_1}^* = (G^*; G \cup \{\varepsilon\}; \circ, add, sub_l, sub_r, R_1, =)$$

such that we get  $P_{\bar{\mathcal{G}}_{R_1}^*} = NP_{\bar{\mathcal{G}}_{R_1}^*}$ . Here,  $\varepsilon$  is the empty string,  $add$  is a binary operation for adding a character to a string, and  $sub_r$  and  $sub_l$  are unary operations for computing the last character and the remainder of a string, respectively. For the strings  $s \in G^*$ ,  $r \in G^* \setminus G$ , and  $c \in G$ , the operations are defined by  $add(s, c) = sc$ ,  $sub_l(sc) = s$ ,  $sub_r(sc) = c$ ,  $add(s, r) = \varepsilon$ ,  $sub_l(\varepsilon) = \varepsilon$ ,  $sub_r(\varepsilon) = \varepsilon$ ,  $s \circ r = \varepsilon$ , and  $r \circ s = \varepsilon$ . For  $\bar{\mathcal{G}}^*$ , the relation  $R_1$  can be derived from some universal oracle  $\mathcal{O}_1 = UNI_{\bar{\mathcal{G}}^*}(\mathcal{O}_1)$  by using, for example,  $a \in G$  for padding such that, for any  $r = \langle \mathbf{x} \rangle Code_{a,b}^*(\mathcal{M}) b^t$  (where, for  $\mathbf{x} \in (G^*)^\infty$ ,  $\langle \mathbf{x} \rangle$  is a suitable code for  $\mathbf{x}$  in  $G^*$ ), the string  $ra^{|r|}$  satisfies  $R_1$  if and only if

$$(\mathbf{x}, Code_{a,b}(\mathcal{M}), \lceil b^t \rceil) \in \mathcal{O}_1$$

holds. Then, any problem in  $NP_{\bar{\mathcal{G}}_{R_1}^*}$  can be reduced to the universal oracle  $\mathcal{O}_1$  and any oracle query can be replaced by a branching instruction defined by  $R_1$  by transforming each tuple of the form  $(\mathbf{x}, Code_{a,b}(\mathcal{M}), \lceil b^t \rceil)$  used in oracle queries into a single string. The definition of the additional relation  $R_1$  on padded codes of the elements of  $\mathcal{O}_1$  guarantees that any input, any computed values, and any guesses can be replaced by short strings without changing the path of computation, and that the short strings  $x_1, x_2, \dots$  in a query can be compressed into one single string in polynomial time. Since  $R_1$  has the properties (1) and (2), the strings of the form  $sw$  for which the test conditions  $R_1(sa^i)$  are satisfied for some  $i$ , can be replaced by strings  $s'w$  for which  $s'a^i$  satisfying  $R_1$ .

$$(\forall s \in G^*)(\forall i \in \mathbb{N})(\forall j \in \mathbb{N})(R_1(sa^i) \& R_1(sa^j) \rightarrow i = j), \quad (1)$$

$$(\forall s \in G^*)(\exists r \in G^*)(R_1(s) \rightarrow s = rba^{|r|+1}). \quad (2)$$

If  $G$  contains the neutral element  $e$  and a second element  $g$ , then we can also transform  $\mathcal{G}$  into a structure of the form  $(G^*; e, g, \varepsilon; \circ, add, sub_l, sub_r, R, =)$  in a similar way.

If we do not allow the additional parameter  $g \neq e$  as constant, the same definition of  $R$  for the construction of some  $\mathcal{G}_R^* = (G^*; e, \varepsilon; \circ, add, sub_l, sub_r, R, =)$  with  $P_{\mathcal{G}_R^*} = NP_{\mathcal{G}_R^*}$  is not possible. The reason is the following. Let us assume that the definition of an additional relation  $R_2$  is done on padded codes of the elements of some oracle  $\mathcal{O}_2$  and that the tuples of the form

$$(\mathbf{x}, Code_{a,b}(\mathcal{M}), \lceil b^t \rceil) \quad (3)$$

$$\lceil e^{c(n, C(\mathcal{M}), t)} \rceil \quad (4)$$

used in oracle queries could be transformed into single strings in polynomial time in order to check it by  $R_2$ . Then, any problem in  $\text{NP}_{G_{R_2}^*}$  could be reduced to  $\mathcal{O}_2$  containing tuples of the form (3) and (4) and any oracle query could be replaced by a branching instruction defined by  $R_2$ . In this case the padding of the codes of the tuples of the form (3) and (4) by adding an element  $a$  are necessary since the properties (1) and (2) are the base for shortening. But, if the components in an input are only  $e$ , then only  $e$  can be used for padding since any other element in  $G$  cannot be computed. That means that the elements of the form (4) could be padded by  $e$  only. Then, for any string  $s = e^k$  there are several  $i$  implying that  $R_2(se^i)$  is true and we do not know all  $i$  which satisfy this property. Consequently, the condition (1) cannot be true.

*Remark 1.* A natural extension of groups yielding structures  $\mathcal{S}$  with  $P_{\mathcal{S}} = \text{NP}_{\mathcal{S}}$  can be realized by embedding groups in structures over trees. The ideas are given in [10].

## References

1. BAKER, T., J. GILL, and R. SOLOVAY (1975). Relativizations of the  $P \stackrel{?}{=} NP$  question. *SIAM J. Comput.* 4, 431–442.
2. J. BALCÁZAR, J. DÍAZ, and J. GABARRÓ (1988/90). *Structural Complexity I* and *Structural Complexity II*. Springer-Verlag.
3. BLUM, L., F. CUCKER, M. SHUB, and S. SMALE (1998). *Complexity and Real Computation*. Springer-Verlag.
4. BLUM, L., M. SHUB, and S. SMALE (1989). On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the Amer. Math. Soc.* 21, 1–46.
5. EMERSON, T. (1994). Relativizations of the  $P \stackrel{?}{=} NP$  question over the reals (and other ordered rings). *Theoretical Computer Science* 133, 15–22.
6. GAßNER, C. (2001). The P-DNP problem for infinite abelian groups. *Journal of Complexity* 17, 574–583.
7. GAßNER, C. (2006). A structure with  $P = NP$ . *CiE 2006. Computer Science Report Series of the University of Wales Swansea* CSR 7, 85–94.
8. GAßNER, C. (2006). Expansions of structures with  $P = NP$ . *CiE 2006. Computer Science Report Series of the University of Wales Swansea* CSR 7, 95–104.
9. GAßNER, C. (2007). P = NP for Expansions Derived from Some Oracles. *CiE 2007. Technical report no. 487 June 2007*.
10. GAßNER, C. (2004). NP ⊂ DEC und P = NP für Expansionen von Erweiterungen von Strukturen endlicher Signatur mit Identitätsrelation. *Preprint-Reihe Mathematik, E.-M.-Arndt-Universität Greifswald*, Preprint 13/2004.
11. KOIRAN, P. (1994). Computing over the reals with addition and order. *Theoretical Computer Science* 133, 35–47.
12. MEER, K. (1992). A note on a  $P \neq NP$  result for a restricted class of real machines. *Journal of Complexity* 8, 451–453.
13. POIZAT, B. (1995), *Les Petits Cailloux*. Aléas.
14. PRUNESCU, M. (2002) A model-theoretical proof for  $P \neq NP$  over all infinite abelian groups. *J. Symb. Logic* 67, 235–238.

# Singularities of Holomorphic Functions in Subsystems of Second Order Arithmetic

Yoshihiro Horihata<sup>1</sup> and Keita Yokoyama<sup>2</sup>

<sup>1</sup> Mathematical Institute, Tohoku University, Sendai 980-8578, Japan.

[sa6m31@math.tohoku.ac.jp](mailto:sa6m31@math.tohoku.ac.jp) or [higurashi3873@yahoo.co.jp](mailto:higurashi3873@yahoo.co.jp)

<sup>2</sup> Department of Mathematics, Tokyo Institute of Technology,

2-12-1 Oh-okayama, Meguro-ku, Tokyo 152-8551, Japan.

[yokoyama@math.titech.ac.jp](mailto:yokoyama@math.titech.ac.jp) or [k\\_yoko.tautology@infoseek.jp](mailto:k_yoko.tautology@infoseek.jp)

**Abstract.** We study complex analysis in the context of weak subsystems of second order arithmetic. We are mainly concerned with integrability and singularities of holomorphic functions. Then, we develop a part of complex analysis concerned with Picard's little theorem. We show that Picard's little theorem is provable from  $\text{WKL}_0$  plus a version of the Riemann mapping theorem. Since a full version of the Riemann mapping theorem is provable in  $\text{ACA}_0$ , we can prove Picard's little theorem in  $\text{ACA}_0$ .

**Key words:** Reverse Mathematics, Picard's theorem, Riemann mapping theorem, second order arithmetic

## 1 Introduction

This paper is a contribution to the program of Reverse Mathematics. Reverse Mathematics was pioneered by Harvey Friedman and Stephan Simpson in the 1970s. Its goal is to develop large parts of ordinary mathematics in second order arithmetic and determine which axioms are exactly required to prove theorems. In this paper, we aim to develop some parts of complex analysis especially related to singularities.

We mainly consider subsystems  $\text{RCA}_0$ ,  $\text{WWKL}_0$ ,  $\text{WKL}_0$  and  $\text{ACA}_0$  in this paper.  $\text{RCA}_0$  is a system of recursive comprehension axioms. This is the weakest system we consider and we can prove basic theorems for analysis such as the intermediate value theorem, the mean value theorem and Taylor's theorem for holomorphic functions within  $\text{RCA}_0$ . Within  $\text{WKL}_0$ , we can prove the Heine/Borel theorem, and then, we can integrate a continuous function on a closed interval. If a continuous function is bounded, then, we can integrate it within a weaker system  $\text{WWKL}_0$ , in which we can show a weak version of the Heine/Borel compactness (see [7]). Within  $\text{ACA}_0$ , we can prove the Riemann mapping theorem. Actually, the Riemann mapping theorem is equivalent to  $\text{ACA}_0$  over  $\text{WKL}_0$  (see [6]).

Analysis in second order arithmetic is carried forward by many people (see, e.g. [3]). Complex analysis in second order arithmetic is carried forward in [5, 6]. We aim to expand these into studies for singularities or studies for Riemann surfaces.

## 2 Preliminaries

In  $\text{RCA}_0$ , we can define the real number system  $\mathbb{R}$ , the Euclidean space  $\mathbb{R}^n$  and continuous functions on  $\mathbb{R}^n$  in the usual way. We first define the complex number system and holomorphic functions.

**Definition 1 (the complex number system).** *The following definitions are made in  $\text{RCA}_0$ . We identify a complex number, an element of  $\mathbb{C}$ , with an element of  $\mathbb{R}^2$ , and we define  $+_{\mathbb{C}}$ ,  $\cdot_{\mathbb{C}}$  and  $|\cdot|_{\mathbb{C}}$  by:*

$$\begin{aligned}(x_1, y_1) +_{\mathbb{C}} (x_2, y_2) &= (x_1 + x_2, y_1 + y_2); \\ (x_1, y_1) \cdot_{\mathbb{C}} (x_2, y_2) &= (x_1 x_2 - y_1 y_2, x_1 y_2 + x_2 y_1); \\ |(x, y)|_{\mathbb{C}} &= \|(x, y)\|_{\mathbb{R}^2} = \sqrt{x^2 + y^2}.\end{aligned}$$

We write  $(0, 1) = i$  and  $(x, y) = x + iy = z$ , where  $x, y \in \mathbb{R}$  and  $z \in \mathbb{C}$ . We usually leave out the subscript  $\mathbb{C}$ . A continuous (partial) function from  $\mathbb{C}$  to  $\mathbb{C}$  is a continuous (partial) function from  $\mathbb{R}^2$  to  $\mathbb{R}^2$ .

**Definition 2 (holomorphic functions).** *The following definition is made in  $\text{RCA}_0$ . Let  $D$  be an open subset of  $\mathbb{C}$ , and let  $f, f'$  be continuous functions from  $D$  to  $\mathbb{C}$ . Then a pair  $(f, f')$  is said to be holomorphic if*

$$\forall z \in D \quad \lim_{w \rightarrow z} \frac{f(w) - f(z)}{w - z} = f'(z).$$

*Informally, we write  $f$  for a holomorphic function  $(f, f')$ .*

Within  $\text{RCA}_0$ , we can show Taylor's theorem, i.e., a holomorphic function is an analytic function. Thus, we can prove basic properties for holomorphic functions in  $\text{RCA}_0$ .

Next, we define line integrals. Let  $a, b$  be elements of  $\mathbb{C}$  and let  $r$  be a positive real number. Then we define

$$\begin{aligned}[a, b] &:= \{a + (b - a)x \in \mathbb{C} \mid 0 \leq x \leq 1\}, \\ \partial B(a; r) &:= \{z \in \mathbb{C} \mid |z - a| = r\}.\end{aligned}$$

**Definition 3 (line integral).** *Let  $D$  be an open or closed subset of  $\mathbb{C}$ , and let  $f$  be a continuous function from  $D$  to  $\mathbb{C}$ . Then the following definitions are made in  $\text{RCA}_0$ .*

1. *Let  $\gamma$  be a continuous function from  $[0, 1]$  to  $D$ . Then, we define  $\int_{\gamma} f(z) dz$ , the line integral of  $f$  along  $\gamma$ , as*

$$\int_{\gamma} f(z) dz = \lim_{|\Delta| \rightarrow 0} S_{\gamma}^{\Delta}(f)$$

*if this limit exists. Here,  $\Delta$  is a partition of  $[0, 1]$ , i.e.  $\Delta = \{0 = x_0 \leq \xi_1 \leq x_1 \leq \dots \leq \xi_n \leq x_n = 1\}$ ,  $S_{\gamma}^{\Delta}(f) = \sum_{k=1}^n f(\gamma(\xi_k))(\gamma(x_k) - \gamma(x_{k-1}))$  and  $|\Delta| = \max\{x_k - x_{k-1} \mid 1 \leq k \leq n\}$ .*

2. If  $[a, b] \subseteq D$ , we define  $\gamma(t) = a + (b - a)t$  and define  $\int_{[a,b]} f(z) dz$  as

$$\int_{[a,b]} f(z) dz = \int_{\gamma} f(z) dz.$$

3. If  $\partial B(a; r) \subseteq D$ , we define  $\gamma(t) = a + r \exp(2\pi i t)$  and define  $\int_{\partial B(a;r)} f(z) dz$  as

$$\int_{\partial B(a;r)} f(z) dz = \int_{\gamma} f(z) dz.$$

Let  $f$  be a continuous function from  $D \subseteq \mathbb{C}$  to  $\mathbb{C}$ , and let  $[a, b] \subseteq D$ . A *modulus of integrability* along  $[a, b]$  for  $f$  is a function  $h_{[a,b]}$  from  $\mathbb{N}$  to  $\mathbb{N}$  such that for all  $n \in \mathbb{N}$  and for all partitions  $\Delta_1, \Delta_2$  of  $[0, 1] \subseteq \mathbb{R}$ , if  $|\Delta_1|, |\Delta_2| < 2^{-h_{[a,b]}(n)}$  then  $|S_{[a,b]}^{\Delta_1}(f) - S_{[a,b]}^{\Delta_2}(f)| < 2^{-n+1}$ . We say that  $f$  is *effectively integrable* on  $D$  when for every  $[a, b] \subseteq D$ , we can find a modulus of integrability along  $[a, b]$ .

**Theorem 1.** Let  $D \subset \mathbb{C}$ . Then, the following assertions are equivalent over  $\text{RCA}_0$ .

1.  $\text{WKL}_0$ .
2. Every continuous function on  $D$  is effectively integrable.

For the proof, see [3, Theorem IV.2.7].

Next, we define subsystem  $\text{WWKL}_0$  which is introduced in [7].

**Definition 4.**  $\text{WWKL}_0$  is the system which consists of  $\text{RCA}_0$  plus  $\text{WWKL}$ , where the  $\text{WWKL}$  is weak-weak König's lemma;

if an infinite tree  $T \subseteq 2^{<\mathbb{N}}$  has no path, then  $\lim_{n \rightarrow \infty} |\{\sigma \in T \mid \text{lh}(\sigma) = n\}|/2^n = 0$ .

The Heine/Borel theorem is not provable in  $\text{WWKL}_0$ , but a weak version of the Heine/Borel theorem is provable.

**Theorem 2.** The following assertions are equivalent over  $\text{RCA}_0$ .

1.  $\text{WWKL}_0$ .
2. Weak Heine/Borel covering theorem; if  $\bigcup_{n \in \mathbb{N}} B(a_n; r_n)$  is a covering of  $[0, 1]$ , then

$$\begin{aligned} & \exists \langle (b_{ij}, c_{ij})_{j \leq l_i} \mid i \in \mathbb{N}, l_i \in \mathbb{N} \rangle \text{s.t.} \\ & [0, 1] \subseteq \bigcup_{n < i} B(a_n; r_n) \cup \bigcup_{j \leq l_i} (b_{ij}, c_{ij}) \wedge \lim_{i \rightarrow \infty} \sum_{j < l_i} |c_{ij} - b_{ij}| = 0. \end{aligned}$$

For the proof, see [7].

By the above theorem, we only need  $\text{WWKL}_0$  to integrate a bounded function.

**Theorem 3.** Let  $D \subset \mathbb{C}$ . Then, the following assertions are equivalent over  $\text{RCA}_0$ .

1.  $\text{WWKL}_0$ .
2. Every bounded continuous function on  $D$  is effectively integrable.

For the proof, see [4, Theorem 3].

### 3 Complex analysis in weak second order arithmetic

We review some recent studies on complex analysis in weak second order arithmetic.

We first state two basic theorems for holomorphic functions contained in [5].

**Definition 5 (analytic function).** *The following definition is made in  $\text{RCA}_0$ . Let  $D$  be an open subset of  $\mathbb{C}$ . An analytic function on  $D$  is defined to be a triple  $(f, \{a_n, r_n\}_{n \in \mathbb{N}}, \{\alpha_{nk}\}_{n \in \mathbb{N}, k \in \mathbb{N}})$ , where  $f$  is a continuous function from  $D$  to  $\mathbb{C}$ ,  $a_n, \alpha_{nk} \in \mathbb{C}$  and  $r_n \in \mathbb{R}^+$ , satisfying the following conditions:*

1.  $\bigcup_{n \in \mathbb{N}} B(a_n; r_n) = D$ ;
2.  $\forall n \in \mathbb{N} \forall z \in B(a_n; r_n) f(z) = \sum_{k \in \mathbb{N}} \alpha_{nk} z^k$ .

**Theorem 4 (Taylor's theorem).** *The following is provable in  $\text{RCA}_0$ . Holomorphic functions are analytic, i.e. given a holomorphic function  $(f, f')$  on an open subset  $D \subseteq \mathbb{C}$ , we can effectively find an analytic function  $(f, \{a_n, r_n\}_{n \in \mathbb{N}}, \{\alpha_{nk}\}_{n \in \mathbb{N}, k \in \mathbb{N}})$  on  $D$ . In particular, the derivative of a holomorphic function is holomorphic.*

**Theorem 5 (Cauchy's integral theorem).** *The following assertions are equivalent over  $\text{RCA}_0$ .*

1.  $\text{WKL}_0$ .
2. *Cauchy's integral theorem for triangles: if  $f$  is a holomorphic function on an open subset  $D \subseteq \mathbb{C}$ , then for all  $\triangle abc \subseteq D$ ,  $\int_{\partial \triangle abc} f(z) dz$  exists and*

$$\int_{\partial \triangle abc} f(z) dz = 0.$$

To prove Taylor's theorem, we need to use Cauchy's integral theorem ‘locally’, i.e., we need to find a good neighborhood for Cauchy's theorem at each point. Note that a holomorphic function is locally effectively integrable in  $\text{RCA}_0$ . As we stated, we can show many theorems for holomorphic functions such as maximal value principle or mean value principle in  $\text{RCA}_0$ . By Theorem 4, we can apply Cauchy's integral theorem to a holomorphic function  $f$  if  $f$  is effectively integrable within  $\text{RCA}_0$ . However, Cauchy's integral theorem as itself is not provable in  $\text{RCA}_0$ .

Based on the line integrability in  $\text{WKL}_0$  or  $\text{WWKL}_0$  (Theorems 1 and 3) and the previous two theorems, we develop some more complex analysis in second order arithmetic. Proofs of the following theorems can be found in [1]. We first study Laurent expansion.

**Theorem 6 (Laurent expansion).** *The following is provable in  $\text{RCA}_0$ . Let  $f$  be an effectively integrable holomorphic function on  $D = \{z \mid 0 \leq R_1 < |z - a| < R_2\}$ . Then, for all  $z \in D$ ,*

$$f(z) = \sum_{n=1}^{\infty} \frac{a_{-n}}{(z - a)^n} + \sum_{n=0}^{\infty} a_n (z - a)^n$$

where  $R_1 < r < R_2$  and

$$\begin{aligned} a_{-n} &= \frac{1}{2\pi i} \int_{\partial B(a;r)} f(\zeta)(\zeta - a)^{n-1} d\zeta, \\ a_n &= \frac{1}{2\pi i} \int_{\partial B(a;r)} \frac{f(\zeta)}{(\zeta - a)^{n+1}} d\zeta. \end{aligned}$$

Using effective integrability, we can show this theorem as usual. However, as for Cauchy's integral theorem, we cannot omit the assumption that  $f$  is effectively integrable. Actually, the following theorem holds.

**Theorem 7.** *The following assertions are equivalent over  $\text{RCA}_0$ .*

1.  $\text{WKL}_0$ .
2. *If  $f$  is a holomorphic function on  $D = \{z \mid 0 \leq R_1 < |z - a| < R_2\}$ , then, there exists  $\{a_n\}_{n \in \mathbb{Z}}$  such that  $f(z) = \sum_{n \in \mathbb{Z}} a_n(z - a)^n$  for all  $z \in D$ .*

We next study isolated singularities.

**Definition 6 (isolated essential singularity).** *The following definition is made in  $\text{RCA}_0$ . Let  $f$  be a holomorphic function on  $D = \{z \mid 0 < |z - a| < R\}$ . Then  $a$  is said to be an isolated essential singularity if there exists  $\{a_n\}_{n \in \mathbb{Z}}$  such that  $f(z) = \sum_{n \in \mathbb{Z}} a_n(z - a)^n$  for all  $z \in D$  and  $\forall m \in \mathbb{N} \exists k \geq m a_{-k} \neq 0$ .*

We can prove the following two theorems within  $\text{WWKL}_0$ .

**Theorem 8 (Riemann's theorem on removable singularities).** *The following is provable in  $\text{WWKL}_0$ . Let  $f$  be a holomorphic function on  $D = \{z \mid 0 < |z - a| < r\}$ . If there exists  $r' > 0$  such that  $r' < r$  and  $f$  is bounded on  $\{z \mid 0 < |z - a| < r'\}$ , then there exists a holomorphic function  $\tilde{f}$  on  $D \cup \{a\}$  such that  $\tilde{f}(z) = f(z)$  for all  $z \in D$ .*

The following theorem is a special case of Picard's theorem.

**Theorem 9 (Casorati/Weierstraß theorem).** *The following is provable in  $\text{WWKL}_0$ . Let  $f$  be a holomorphic function on  $D = \{z \mid 0 < |z - a| < r\}$  and  $a$  be an isolated essential singularity. Then,  $f(D)$  is dense in  $\mathbb{C}$ .*

To prove these theorems, we need to apply Cauchy's theorem for bounded holomorphic functions. This can be done in  $\text{WWKL}_0$  by Theorem 3. Thus, we can imitate usual proofs within  $\text{WWKL}_0$ . We can also prove the following two theorems and many other theorems in  $\text{WWKL}_0$  by the same reasoning. These theorems play key roles in the proof of Picard's theorem.

**Theorem 10 (Liouville theorem).** *The following is provable in  $\text{WWKL}_0$ . If  $f$  is a bounded entire function, then it is a constant function.*

**Theorem 11 (Schwarz reflection principle).** *The following is provable in  $\text{WWKL}_0$ . Let  $D \subseteq \mathbb{C}^+ = \{x + iy \mid y > 0\}$  be an open set and let  $L = (a, b) \subseteq \mathbb{R}$  be an open interval such that  $L = \partial D \cap \mathbb{R}$ . Let  $f$  be a continuous function on  $D \cup L$  such that  $f$  is holomorphic on  $D$  and  $f(z) \in \mathbb{R}$  for all  $z \in L$ . Then there exists a holomorphic function  $\tilde{f}$  on  $\tilde{D} = D \cup \{z \in \mathbb{C} \mid \bar{z} \in D \cup L\}$  such that  $\tilde{f}(z) = f(z)$  for all  $z \in D \cup L$ .*

Finally, we recall the Riemann mapping theorem contained in [6].

**Theorem 12 (Riemann mapping theorem).** *The following assertions are equivalent over  $\text{RCA}_0$ .*

1.  $\text{ACA}_0$ .
2. *If  $D \subseteq \mathbb{C}$  is a simply connected open set  $D \neq \mathbb{C}$ , then there exists a conformal map  $f : D \rightarrow B(0; 1)$ .*

## 4 Picard's little theorem

To study singularities, we need covering spaces.

**Definition 7 (covering space).** *Let  $X, D \subseteq \mathbb{C}$  be open sets, and let  $\pi$  be a continuous surjective function from  $X$  to  $D$ . Let  $\{U_{ij}\}_{i \in I, j \in J}$  and  $\{V_i\}_{i \in I}$  be sequences of open sets, and let  $\pi_{ij}$  be homeomorphic functions from  $U_{ij}$  to  $V_i$ . Then,  $\pi$  is said to be a covering map and a hextuple  $(X, D, \pi, U_{ij}, V_i, \pi_{ij})$  is said to be a covering space of  $D$  if they satisfy the following:*

$$\begin{aligned} &\text{each of } U_{ij} \text{ and } V_i \text{ is simply connected;} \\ &D = \bigcup_{i \in I} V_i; \\ &\forall i \in I \quad \pi^{-1}(V_i) = \bigcup_{j \in J} U_{ij}, \\ &\forall i \in I \quad \forall j \in J \quad \pi|_{U_{ij}} = \pi_{ij}. \end{aligned}$$

The following lemma plays a key role of the proof of Picard's little theorem.

**Lemma 1 (lifting).** *The following assertions are equivalent over  $\text{RCA}_0$ .*

1.  $\text{WKL}_0$ .
2. *Let  $D_0, D \subseteq \mathbb{C}$  be open sets, and let  $(X, D, \pi, U_{ij}, V_i, \pi_{ij})$  be a covering space of  $D$ . If  $D_0$  is simply connected and  $f$  is a continuous function from  $D_0$  to  $D$ , then, there exists a continuous function  $\hat{f}$  from  $D_0$  to  $X$  such that  $\pi \circ \hat{f} = f$ .*

*Proof.* We first show  $1 \rightarrow 2$  in case  $D_0 = [0, 1]$ . We reason within  $\text{WKL}_0$ . Let  $D_0 = [0, 1]$  and  $a_{ij} = j/2^{i+1}$  for all  $j \leq 2^{i+1}$ . By  $\text{WKL}_0$ , we can define  $W_i$  as follows:

$$W_i = \bigcup_{\exists k \overline{f(B(a_{ij}; 2^{-i}))} \subseteq V_k} B(a_{ij}; 2^{-i}).$$

Then  $W_i \subseteq W_{i+1}$  for all  $i \in \mathbb{N}$  and  $[0, 1] \subseteq \bigcup_{i \in \mathbb{N}} W_i$ . By the Heine/Borel theorem, there exists  $N \in \mathbb{N}$  such that  $[0, 1] \subseteq W_N$ . Take a sequence  $\langle k_j \mid j \leq 2^{N+1} \rangle$  such that  $f(a_{Nj}) \in V_{k_j}$  for all  $j \leq 2^{N+1}$ . Then, by the definition of the covering space, we can take a sequence  $\langle l_m \mid m \leq 2^{N+1} \rangle$  such that

$$\pi_{k_m, l_m}^{-1}(f(a_{N, m+1})) \in U_{k_{m+1}, l_{m+1}}$$

for all  $m < 2^{N+1}$ . Thus we can define a lifting  $\hat{f}$  as follows:

$$\hat{f}(x) = \pi_{k_j, l_j}^{-1} \circ f(x) \text{ if } x \in [a_{N,j}, a_{N,j+1}].$$

Similarly, we can construct a lifting in case  $D_0 = [0, 1] \times [0, 1]$ . The general case can be proved easily by the previous two cases.

Next, we show  $\neg 1 \rightarrow \neg 2$ . We assume  $\neg \text{WKL}_0$ . Let  $D_0 = \{(x, y) \subseteq \mathbb{R}^2 \mid -1 \leq x \leq 1, -1 \leq y \leq 1\}$ ,  $D = \partial D_0$ , and  $X = \mathbb{R}$ . Then we can take  $\pi : X \rightarrow D$  as a covering map by winding  $X$  on  $D$ . By  $\neg \text{WKL}_0$ , there exists a continuous function  $f : D_0 \rightarrow D$  such that  $f|_D = id_D$  (by Shioji and Tanaka[2]). However, there is no lifting of  $f$ .

The complete proof of the previous lemma is in [1].

Now we study Picard's little theorem. For this, we consider a version of the Riemann mapping theorem:

- (\*) *Let  $D$  be an interior of a simple closed curve on the Riemann sphere. Then,  $D$  is conformally equivalent to the unit open ball  $B(0; 1)$  and a conformal map  $h : D \rightarrow B(0; 1)$  can be expanded into a homeomorphism  $\bar{h} : \bar{D} \rightarrow \bar{B}(0, 1)$ .*

We prove Picard's little theorem, which asserts that an entire function is a constant if it omits two points, within  $\text{WKL}_0 + (*)$ .

We reason within  $\text{WKL}_0$ . Let  $f$  be an entire function which omits two points. Without loss of generality, we assume that  $f : \mathbb{C} \rightarrow \mathbb{C} \setminus \{0, 1\}$ . By (\*) and the Schwarz reflection principle, we can construct a holomorphic covering map  $\pi : B(0; 1) \rightarrow \mathbb{C} \setminus \{0, 1\}$  in the usual way. Then, by Lemma 1, there exists a holomorphic function  $\hat{f} : \mathbb{C} \rightarrow B(0; 1)$  such that  $\pi \circ \hat{f} = f$ . By the Liouville theorem,  $\hat{f}$  is constant. Thus,  $f$  is also constant and this completes the proof of Picard's little theorem.

Since a full version of Riemann mapping theorem is provable in  $\text{ACA}_0$ , we can prove Picard's theorem within  $\text{ACA}_0$ .

**Theorem 13 (Picard's theorem).** *The following is provable in  $\text{ACA}_0$ . Let  $f$  be a holomorphic function from  $\mathbb{C}$  to  $\mathbb{C}$ . If the range of  $f$  omits two points, then,  $f$  is a constant function.*

We conjecture that (\*) is provable within  $\text{WKL}_0$ , which implies that  $\text{WKL}_0$  proves Picard's theorem. We have not managed the reversal of Picard's little theorem.

## References

1. Y. Horihata. Reverse mathematics of complex analysis and general topology (in Japanese), Feb. 2008. Master's Thesis, Tohoku University, February 2008.
2. N. Shioji and K. Tanaka. Fixed point theory in weak second-order arithmetic. *Annals of Pure and Applied Logic*, 47:167–188, 1990.
3. S. G. Simpson. *Subsystems of Second Order Arithmetic*. Springer-Verlag, 1999.
4. K. Yokoyama. Reverse mathematics for Fourier expansion. In the local proceedings of the fourth conference on computability in Europe 2008.
5. K. Yokoyama. Complex analysis in subsystems of second order arithmetic. *Arch. Math. Logic*, 46:15–35, 2007.

6. K. Yokoyama. Non-standard analysis in  $\text{ACA}_0$  and Riemann mapping theorem. *Math. Logic Quart.*, 53(2):132–146, 2007.
7. X. Yu and S. G. Simpson. Measure theory and weak König’s lemma. *Arch. Math. Logic*, 30:171–180, 1990.

# Modelling Linear Cellular Automata with the Minimum Stage Corresponding to CCSG based on LFSR \*

Yoon-Hee Hwang<sup>1</sup>, Sung-Jin Cho<sup>2</sup>, Un-Sook Choi<sup>3</sup>, and Han-Doo Kim<sup>4</sup>

<sup>1</sup> Department of Information Security, Graduate School  
Pukyong National University

Busan 608-737, Korea, [yhhwang@pknu.ac.kr](mailto:yhhwang@pknu.ac.kr)

<sup>2</sup> Division of Mathematical Sciences, Pukyong National University  
Busan 608-737, Korea, [sjcho@pknu.ac.kr](mailto:sjcho@pknu.ac.kr)

<sup>3</sup> Department of Multimedia Engineering, Tongmyong University  
Busan 626-847, Korea, [choies@tu.ac.kr](mailto:choies@tu.ac.kr)

<sup>4</sup> Institute of Mathematical Sciences and School of Computer Aided Science  
Inje University, Gimhae 621-749, Korea, [mathkhd@inje.ac.kr](mailto:mathkhd@inje.ac.kr)

**Abstract.** Cellular Automata(CA) with simple, regular, modular and cascadable structures, which the VLSI design community prefer, was made an alternative proposal of LFSRs. Pseudorandom sequences were produced by generators which accompany several LFSRs joined by non-linear functions or irregular clocking techniques. Clock-Controlled Shrinking Generators(CCSGs) are a class of clock-controlled sequence generators. Clock-controlled LFSRs have become important building blocks for keystream generators in stream cipher applications, because they are known to produce sequences of long period and high linear complexity. In this paper, we propose a method of modelling linear CA with the minimum stage corresponding to CCSGs based on LFSR using the Cho et al.'s synthesis algorithm.

## 1 Introduction

The VLSI era has ushered in a new phase of activities into the research of linear machines, and specially the local neighborhood CA structures. The VLSI design community prefer simple, regular, modular and cascadable structures with local interconnections. The CA provides a wonderful solution in all these respects [1]. CA have the characters of simplicity of basic components, locality of CA interactions, massive parallelism of information processing, and exhibit complex global properties. These ensure that CA have higher speed and more potential applications than LFSR. The locality of signal path of CA contributes more higher speed than LFSR. So in the form of VLSI implementation, CA have more speed advantages than LFSR [2].

---

\* This work was supported by grant No. (R01-2006-000-10260-0) from the Basic Research Program of the Korea Science and Engineering Foundation.

Pseudorandom sequences were produced by generators which accompany several LFSRs joined by nonlinear functions or irregular clocking techniques. The theory for CA based pseudorandom number generator is well developed [1] and  $n$ -stage linear CA can be designed to generate sequences with desirable properties: maximum period  $2^n - 1$ , uniform distribution of  $n$ -tuples and balanced distribution of 1 and 0 ([3],[4]).

Pseudorandom sequence generators intend to be used in a stream cipher. Especially, the Shrinking Generator(SG) proposed by Coppersmith et al. [5] is a popular form of pseudorandom sequence generators that employ the irregular clocking. It has one or more LFSRs whose clocking is controlled by the output sequence of one. Such a sequence is called a *clock-controlled sequence* [6]. The SG generally uses two sources of pseudorandom sequences to create the third source of pseudorandom sequence, having better cryptographic quality(long period, high linear complexity, good statistical properties, etc.) than the original sources.

Clock-controlled LFSRs have become important building blocks for keystream generators in stream cipher applications, because they are known to produce sequences of long period and high linear complexity ([7], [8]).

In [9], they showed that CCSGs can be described in terms of linear CA configurations by using mirror image and the Cattell and Muzio synthesis algorithm [10]. Since the CA obtained by the Sabater et al.'s method has the maximum stage, the method has a waste of space. Also the sequence obtained by CA is not secure because the rule of this CA is symmetrical.

In this paper, we propose a new method of modelling linear CA with the minimum stage corresponding to CCSGs based on LFSR using the Cho et al.'s synthesis algorithm to overcome these weak points [11].

## 2 90/150 CA Preliminaries

CA are considered to be a good model of complex systems in which an infinite one-dimensional array of finite state machines(cells) updates itself in a synchronous manner according to a uniform local rule. In the long history of the study of CA, generally speaking, the number of internal states of each cell is finite and the local state transition rule is defined in a such way that the state of each cell depends on the previous states of itself and its neighboring cells [12]. CA consists of a number of cells. In a 3-neighborhood dependency, the next state  $q_i(t+1)$  of a cell is assumed to be dependent only on itself and on its two neighbors (left and right), and is denoted as

$$q_i(t+1) = f(q_{i-1}(t), q_i(t), q_{i+1}(t))$$

where  $q_i(t)$  represents the state of the  $i$ -th cell at the  $t$ -th instant of time and  $f$  is the next-state function. The cells evolve in discrete time steps according to some deterministic rule that depends only on logical neighborhood. The CA structure

investigated by Wolfram [13] can be viewed as a discrete lattice of sites (cells), where each cell can assume either the value 0 or 1. In effect, each cell consists of a storage element (D flip-flop) and a combinatorial logic implementing the next-state function.

The next-state function of a cell is called its *rule*. If the rule of a CA cell involves only XOR logic, then it is called a *linear rule*. A CA with all the cells having linear rules is called a *linear CA*. If all the CA cells obey the same rule, then the CA is said to be a *uniform CA*; otherwise, it is a *hybrid CA*. A CA is said to be a *Null Boundary CA*(NBCA) if the left(right) neighbor of the leftmost(rightmost) terminal cell is connected to logic 0-state.

If the next-state function of a cell is expressed in the form of a truth table, then the decimal equivalent of the output is conventionally called the rule number for the cell [13].

**Table 1.** Rule 90 and rule 150

Neighborhood state	111	110	101	100	011	010	001	000	
Next state	0	1	0	1	1	0	1	0	rule 90
Next state	1	0	0	1	0	1	1	0	rule 150

In Table 1, the top row gives all eight possible states of the three neighboring cells (the left neighbor of the  $i$ -th cell, the  $i$ -th cell itself, and its right neighbor) at the time instant  $t$ . The second and third rows give the corresponding states of the  $i$ -th cell at the time instant  $t + 1$  for two CA rules. The corresponding combinatorial logic for the above rules can be specified as

$$\begin{aligned} \text{rule 90 : } q_i^{t+1} &= q_{i-1}^t \oplus q_{i+1}^t \\ \text{rule 150 : } q_i^{t+1} &= q_{i-1}^t \oplus q_i^t \oplus q_{i+1}^t \end{aligned}$$

In this paper, CA are NBCA with rule 90 and 150. A natural form for the specification is an  $n$ -tuple  $\langle d_1, d_2, \dots, d_n \rangle$ , called the *rule vector*, where

$$d_i = \begin{cases} 0, & \text{if cell } i \text{ uses rule 90} \\ 1, & \text{if cell } i \text{ uses rule 150} \end{cases}$$

Now we define the *state*  $q^t$  of a CA at time  $t$  to be the  $n$ -tuple

$$q^t = (q_1^t, q_2^t, \dots, q_n^t)^T$$

where  $A^T$  is the transpose of the matrix  $A$ . Hence

$$q^{t+1} = f(q^t) (= T_n q^t)$$

where the product is a matrix-vector multiplication over  $GF(2)$ . We call this matrix  $T_n$  the CA *state-transition matrix*. If  $\mathbf{C}$  is a CA whose rule vector is  $\langle d_1, d_2, \dots, d_n \rangle$ , then the state-transition matrix  $T_n$  of  $\mathbf{C}$  is a tridiagonal matrix

$$T_n = \begin{pmatrix} d_1 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1 & d_2 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & d_3 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 1 & d_{n-1} & 1 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 & d_n \end{pmatrix}$$

The characteristic polynomial  $\Delta_n$  of  $\mathbf{C}$  is defined by

$$\Delta_n = |T_n - xI|$$

where  $x$  is an indeterminate,  $I$  is the  $n \times n$  identity matrix and  $T_n$  is the state-transition matrix of  $\mathbf{C}$ .

For any  $n$ -stage 90/150 CA whose state-transition matrix is  $T_n$ , the minimal polynomial for  $T_n$  is the same as the characteristic polynomial for  $T_n$  [15].

A polynomial is said to be a *CA-polynomial* if it is the characteristic polynomial of some CA [10]. All irreducible polynomials are CA-polynomials([10], [11]).

In [10], authors proposed a method for the synthesis of one-dimensional 90/150 Linear Hybrid Group Cellular Automata(LHGCA) for irreducible CA-polynomial.

In [11], Cho et al. proposed a new method for the synthesis of one-dimensional 90/150 LHGCA for any CA-polynomial. In this case CA-polynomial need not irreducible. This algorithm is efficient and suitable for all practical applications. Table 2 shows an algorithm for finding the 90/150 CA for the given CA-polynomial. In this paper we propose a new method of modelling linear CA with the minimum stage corresponding to CCSGs based on LFSR using this algorithm.

### 3 Clock-Controlled Shrinking Generator

Two LFSRs are used, both clocked regularly. If the output of the first LFSR is 1, the output of the second LFSR becomes the output of the generator. If the output of the first LFSR is 0, however, the output of the second is discarded. This mechanism suffers from timing attacks on the second generator, since the speed of the output is variable in a manner that depends on the second generator's state. This can be alleviated by buffering the output.

CCSGs are a class of clock-controlled sequence generators [14]. They have applications to cryptography, error correcting codes and digital signature. A CCSG consists of two LFSRs **A**(control register) and **B**(generating register). The **A** is clocked normally, but the **B** is clocked by one plus the integer value

**Table 2.** Cho et al.'s Synthesis Algorithm

Algorithm Cho et al.'s Synthesis Algorithm	
Input :	CA-polynomial $f(x)$
Output :	90/150 group/nongroup CA
Step 1 :	Make the matrix $B$ which is the $n \times n$ matrix obtained by reducing the $n$ polynomials $x^{i-1} + x^{2i-1} + x^{2i} \pmod{f(x)}$ ( $i = 1, 2, \dots, n$ ).
Step 2 :	Solve the equation $Bv = (0, \dots, 0, 1)^T$ .
Step 3 :	Construct a Krylov matrix $H = K(C^T, v)$ by the seed vector $v$ which is a solution of the equation in Step 2.
Step 4 :	Compute the LU factorization $H = LU$ .
Step 5 :	Compute CA for $f(x)$ by the matrix $U$ .

represented in selected  $w$  fixed stages of the  $\mathbf{A}$ . The output bits of the system are produced by shrinking the output of  $\mathbf{B}$  under the control of  $\mathbf{A}$  as the following. At any time  $t$  the output of  $\mathbf{B}$  is taken if the current output of  $\mathbf{A}$  is 1, otherwise it is discarded. Suppose as the following Table 3.

**Table 3.** LFSRs  $\mathbf{A}$  and  $\mathbf{B}$ 

LFSR	stage	characteristic polynomial	initial state
$\mathbf{A}$	$m$	$R(x)$	$A_0$
$\mathbf{B}$	$n$	$S(x)$	$B_0$

$F$  is a function that acts on the state of  $\mathbf{A}$  at a given time  $t$  to determine the number of times which  $\mathbf{B}$  is clocked such that

$$F(A_t) = 1 + 2^0 A_{i_0}(t) + 2^1 A_{i_1}(t) + \dots + 2^{w-1} A_{i_{w-1}}(t)$$

for  $w < m$ , and distinct integers  $i_0, i_1, \dots, i_{w-1} \in \{0, 1, \dots, m-1\}$ ,  $A_t$  is the state at the time instant  $t$ . If no stages are selected (i.e.  $w=0$ ), define  $F(A_t) = 1$ .

In this way, the output sequence of a CCSG is obtained from a double decimation. First, the sequence  $\{b_i\}$  of  $\mathbf{B}$  is decimated by  $F(A_t)$  giving rise to the sequence  $\{b'_i\}$ . Next, if the output of  $\mathbf{A}$  is 1,  $b'_i$  becomes the output of the generator, otherwise  $b'_i$  is discarded.

**Example 3.1** Let  $\mathbf{A}$  be the 4-stage LFSR with the characteristic polynomial  $R(x) = x^4 + x + 1$  and the initial state  $(0, 0, 0, 1)$ . The sequence  $\{a_i\}$  generated by  $\mathbf{A}$  is

$$\{a_i\} = \{0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, \dots\}$$

with period  $2^4 - 1 = 15$ . And let  $\mathbf{B}$  be the 5-stage LFSR with the characteristic polynomial  $S(x) = x^5 + x^2 + 1$  and the initial state  $(0, 0, 0, 0, 1)$ . The sequence

$\{b_i\}$  generated by  $\mathbf{B}$  is

$$\{b_i\} = \{0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, \dots\}$$

with period  $2^5 - 1 = 31$ . If  $w = 1$ , then

$$F(A_t) = 1 + 2^0 A_{i_0}(t)$$

Thus  $\{X_{t_i}\}$  is produced by  $F(A_t)$  as the following:

$$\{X_{t_i}\} = \{1, 1, 1, 2, 1, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 2, \dots\}$$

In [14], they defined the cumulative function  $G_A$  of  $\mathbf{A}$  to be

$$G_A(X_{t_i}) = 2^{m-1}(2^w + 1) - 1$$

Then  $G_A(X_{t_i}) = 2^{4-1}(2^1 + 1) - 1 = 23$ . That is,  $1 + 1 + 1 + 2 + 1 + 1 + 2 + 2 + 1 + 2 + 1 + 2 + 2 + 2 + 2 = 23$ . In brief, after clocking  $\mathbf{A}$   $2^4 - 1 (= 15)$  times,  $\mathbf{B}$  is clocked 23 times.

According to the following,

$$\begin{cases} b'_0 := b_0 \\ b'_{i+1} := b_j, \quad j = \sum_{k=0}^i X_{t_i} \end{cases}$$

the underlined bits  $\underline{0}$  or  $\underline{1}$  of  $\{b_i\}$  are outputted in order to produce the sequence  $\{b'_i\}$ .

$\{b_i\}$	$0, 0, \underline{0}, 0, 1, \underline{0}, \underline{0}, \underline{1}, 0, \underline{1}, 1, \underline{0}, \underline{0}, 1, \underline{1}, \underline{1}, 1, \underline{1}, \dots$
$\{X_{t_i}\}$	$1, 1, 1, 2, 1, 1, 2, 2, 1, 2, 1, 2, 2, 2, 2, 1, 1, 1, 2, \dots$
$\{b'_i\}$	$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, \dots$

Then the output sequence  $\{z_i\}$  of the CCSG is given by shrinking  $\{b'_i\}$  with  $\{a_i\}$

$\{a_i\}$	$0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, \dots$
$\{b'_i\}$	$0, 0, 0, \underline{0}, 0, 0, \underline{1}, \underline{1}, 0, \underline{0}, 1, \underline{1}, \underline{0}, \underline{1}, 0, 1, 1, \underline{1}, 1, 0, \underline{1}, \dots$
$\{z_i\}$	$0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, \dots$

The underlined bits  $\underline{0}$  or  $\underline{1}$  of  $b'_i$  are outputted.

## 4 90/150 CA-based CCSG

In this section, we analyze the period of sequences generated by CCSG based on LFSR.

**Definition 4.1** Let  $\{a'_i\}$  be the sequence obtained by concatenations of  $\{C_i\}$ 's.

$$\begin{aligned} C_0 &:= 1 \\ C_i &:= \begin{cases} 1, & X_{t_i} = 1, \\ \underbrace{(0, \dots, 0)}_k, & X_{t_i} = k, (k \geq 2). \end{cases} \end{aligned}$$

**Example 4.2**  $\{b'_i\}$  in Example 3.1 can be obtained by shrinking  $\{b_i\}$  with  $\{a'_i\}$ . That is,  $\{X_{t_i}\}$  in Example 3.1 can be represented by  $\{a'_i\}$ .

$$\{a'_i\} = \{1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, \dots\}$$

If  $a'_i$  is 1,  $b_i$  becomes the output of the generator, otherwise the output of  $b_i$  is discarded. This is just  $b'_i$ . The decimated sequence  $\{b'_i\}$  is given by

$\{a'_i\}$	1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, \dots
$\{b_i\}$	0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, \dots
$\{b'_i\}$	0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, \dots

**Theorem 4.3** Let **A** (resp. **B**) be an  $m$  (resp.  $n$ )-stage LFSR whose characteristic polynomial is primitive. And let  $\{a'_i\}$  be the sequence obtained by concatenations of  $\{C_i\}$ 's in Definition 4.1. The period of  $\{a'_i\}$  is  $2^{m-1}(2^w + 1) - 1$  for a given  $w$ .

*Proof.* Because  $G_A(X_{t_i}) = 2^{m-1}(2^w + 1) - 1$  for a given  $w$ , the period of  $\{a'_i\}$  is  $2^{m-1}(2^w + 1) - 1$  by Definition 4.1.

**Theorem 4.4** Let **A** (resp. **B**) be an  $m$  (resp.  $n$ )-stage LFSR whose characteristic polynomial is primitive. And let  $\{b'_i\}$  be a sequence given by shrinking  $\{b_i\}$  with  $\{a'_i\}$ . The period of  $\{b'_i\}$  is

$$\frac{(2^m - 1)\text{lcm}(G_A(X_{t_i}), 2^n - 1)}{G_A(X_{t_i})}$$

*Proof.* The period of the output sequence  $\{b_i\}$  of **B** is  $2^n - 1$ .  $\{a'_i\}$  repeats  $G_A(X_{t_i})$  period sequences  $\frac{\text{lcm}(G_A(X_{t_i}), 2^n - 1)}{G_A(X_{t_i})}$  times and  $(2^m - 1)$  1's occurs in a full period of  $\{a'_i\}$ . Thus the period of  $\{b'_i\}$  is

$$\frac{(2^m - 1)\text{lcm}(G_A(X_{t_i}), 2^n - 1)}{G_A(X_{t_i})}$$

**Theorem 4.5** Let **A** (resp. **B**) be an  $m$  (resp.  $n$ )-stage LFSR whose characteristic polynomial is primitive. And let  $\{z_i\}$  be a sequence given by shrinking  $\{b'_i\}$  with  $\{a_i\}$ . The period of  $\{z_i\}$  is

$$\frac{2^{m-1} \operatorname{lcm}(G_A(X_{t_i}), 2^n - 1)}{G_A(X_{t_i})}$$

*Proof.* The period of the output sequence  $\{b'_i\}$  is  $(2^m - 1) \frac{\operatorname{lcm}(G_A(X_{t_i}), 2^n - 1)}{G_A(X_{t_i})} (= h)$ .  $\{a_i\}$  repeats  $\frac{\operatorname{lcm}(G_A(X_{t_i}), 2^n - 1)}{G_A(X_{t_i})}$  period sequences  $\frac{\operatorname{lcm}(2^m - 1, h)}{2^m - 1}$  times and  $(2^{m-1})$  1's occurs in a full period of  $\{a_i\}$ . Thus the period of  $\{z_i\}$  is

$$\begin{aligned} & \frac{2^{m-1} \operatorname{lcm}(2^m - 1, \frac{\operatorname{lcm}(G_A(X_{t_i}), 2^n - 1)}{G_A(X_{t_i})})}{2^m - 1} \\ &= 2^{m-1} \operatorname{lcm}(G_A(X_{t_i}), 2^n - 1) / G_A(X_{t_i}) \end{aligned}$$

**Remark** If  $2^n - 1$  and  $G_A(X_{t_i})$  are relatively prime,  $\operatorname{lcm}(G_A(X_{t_i}), 2^n - 1) / G_A(X_{t_i}) = 2^n - 1$ . Therefore in this case, the period of the output sequence by CCGS is  $2^{m-1}(2^n - 1)$ . Thus the characteristic polynomial of such output sequence is of the form  $F(x) = (n \text{ stage primitive polynomial})^N$ ,  $2^{m-2} < N \leq 2^{m-1}$ .

In [9], they proposed the algorithm that converts a given CCGS into a CA-based linear model using mirror image and the Cattell and Muzio synthesis algorithm. Therefore  $N = 2^{m-1}$  is the maximum stage. Also this CA-based linear model is symmetrical and has a waste of space.

## 5 Modelling Linear Cellular Automata with the minimum stage

In this section, we propose a method that converts a given CCGS into a CA-based linear model by using Cho et al.'s Synthesis Algorithm [11].

According to the previous results, the following algorithm that converts a given CCGS into a CA-based linear model is introduced in Table 5.

The following example shows modelling of linear CA with the minimum stage corresponding to CCGS based on LFSR using the above algorithm.

**Example 5.1** Let **A** be the 4-stage LFSR with a primitive polynomial of degree 3, and **B** be the 5-stage LFSR with a primitive polynomial of degree 5. Let  $w = 1$ . Then CCGS with **A** and **B** has the characteristic polynomial  $F(x) = (5\text{-stage primitive polynomial})^{(2^{4-2}-1)} = (x^5 + x^2 + 1)^3$ . By the proposed algorithm, we can compute a CA with  $T_{15} = < 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1 >$ . If the algorithm in [9] is used, they must compute a CA with  $T_{20} = < 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1 >$  corresponding to  $F(x) = (x^5 + x^2 + 1)^4$ .

**Table 5.** Algorithm for modelling 90/150 CA

<b>Algorithm</b>	ModellingOf90/150CA
<b>Input</b>	: A CCSG characterized by: The stages $m$ of LFSR( <b>A</b> ) and $n$ of LFSR( <b>B</b> ), $w$ ( $2^n - 1$ and $G_A(X_{t_i}) = 2^{m-1}(2^w + 1) - 1$ are relatively prime.)
<b>Output</b>	: Linear CA with the minimum stage corresponding to CCSG based on LFSR
<b>Step 1</b>	: Compute the characteristic polynomial $F(x)$ for the given CCSG, where $F(x) = (n \text{ stage primitive polynomial})^N$ , $N = 2^{m-2} + 1$ .
<b>Step 2</b>	: Compute CA by Algorithm “Cho et al.’s Synthesis Algorithm”.

## 6 Conclusion

In this paper we proposed a new method of modelling linear CA with the minimum stage corresponding to CCSGs based on LFSR to overcome the weak points of the Sabater et al.’s method.

## References

1. P.P. Chaudhuri, D.R. Chowdhury, S. Nandy and S. Chattopadhyay, *Additive Cellular Automata Theory and Applications 1*, IEEE Computer Society Press, California, (1997)
2. Z. Chuanwu and L. Libin, “VLSI characteristic of cellular automata as LFSR,” *Communications and Information Technology, 2005. ISCIT 2005, IEEE International Symposium*. **2** (2005) 1031-1034
3. A.J. Menezes, P.C. van Oorschot and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, (1997)
4. P.H. Bardell, W.H. McAnney and J. Savir, *Built-In Test for VLSI: Pseudorandom Techniques*, A WILEY-INTERSCIENCE PUBLICATION, (1987)
5. D. Coppersmith, H. Krawczyk and Y. Mansour, “The shrinking generator,” *Lecture Notes in Computer Science*, **773** (1994) 22-39
6. G. Gong, “Theory and applications of  $q$ -ary interleaved sequences,” *IEEE Transaction on Information Theory*, **41-2** (1995) 400-411
7. J.D. Golic, “Toward fast correlation attacks on irregularly clocked shift registers,” *Advances in Cryptology - EUROCRYPT’95, Lecture Notes in Computer Science*, **921** (1995) 248-262
8. D. Gollmann and W.G. Chambers, “Clock controlled shift registers: a review,” *IEEE J. Sel. Ar. Commun.*, **7-4** (1989) 525-533
9. A.F. Sabater and D.G. Martinez, “Modelling nonlinear sequence generators in terms of linear cellular automata,” *Applied Mathematical Modelling*, **31** (2007) 226-235

10. K. M. Cattell and Jon C. Muzio, "Synthesis of One-Dimensional Linear Hybrid Cellular Automata," *IEEE Trans. Comput-Aided Des. Integr. Circuits Syst.*, **15-3** (1996) 325-335
11. S.J. Cho, U.S. Choi, H.D. Kim, Y.H. Hwang, J.G. Kim and S.H. Heo, "New synthesis of one-dimensional 90/150 linear hybrid group cellular automata," *IEEE Trans. Comput-Aided Des. Integr. Circuits Syst.*, **26-9** (2007) 1720-1724
12. H. Umeo, T. Yanagihara and M. Kanazawa, "State-Efficient Firing Squad Synchronization Protocols for Communication-Restricted Cellular Automata," *Lecture Notes in Computer Science* , **4173** (2006) 169-181
13. S. Wolfram, "Statistical mechanics of cellular automata," *Rev. Mod. Phys.* **55** (1983) 601-644
14. A. Kanso, "Clock-controlled shrinking generators," *Lecture Notes in Computer Science* , **2727** (2003) 443-451
15. M. Serra, T. Slater. J.C. Muzio and D.M. Miller, "The analysis of one dimensional linear cellular automata and their aliasing properties," *IEEE Trans. Comput-Aided Des. Integr. Circuits Syst.*, **9** (1990) 767-778

# Multitape Ordinal Machines and Primitive Recursion

Bernhard Irrgang and Benjamin Seyfferth

University of Bonn, Mathematical Institute  
Beringstraße 1, D-53115 Bonn, Germany  
`irrgang@math.uni-bonn.de, benjamin.seyfferth@gmx.de`

**Abstract.** We introduce a multitape version of the ordinal TURING machines which are defined in [4] as TURING machines computing on tapes of transfinite length in transfinite time. These machines are used to compute the primitive recursive ordinal functions which have a classical theory developed by JENSEN and KARP [3]. Making use of that theory we are able to 1. identify the  $\Delta_1(L_\alpha)$ -definable subsets of  $\alpha$  as computable (if  $\alpha$  is closed with respect to primitive recursive functions) and 2. characterize admissible ordinals by the means of transfinite computations. Similar results linking  $\alpha$ -recursion theory and ordinal computability are contained in [6].

**Key words:** Ordinal computability, multitape TURING machine, primitive recursive ordinal function, primitive recursive set function, admissible ordinal.

## 1 Introduction

Ordinal computability studies machine models generalized to perform computations on ordinals. Such machines have been used to compute GÖDEL's hierarchy of constructible sets  $L$  ([4], [5]) and can provide a computational approach to  $\alpha$ -recursion theory ([6]). The  $\alpha$ -TURING machine is a standard TURING program computing on tapes of length a limit ordinal  $\alpha$  using a lim inf-rule to determine the machine configuration at limit times. The machine is said to *terminate* if it runs for less than  $\alpha$  many steps, otherwise it *diverges*.

In [3] JENSEN and KARP established a connection between the primitive recursive *ordinal* functions ( $\text{Primo}_\alpha$ , a generalization of the usual primitive recursive functions on natural numbers) and the primitive recursive *set* functions ( $\text{Prims}$ ) that are used in the theory of GÖDEL's constructible universe ([2]). The multitape  $\alpha$ -TURING machines introduced in this paper are a straightforward multitape version of the  $\alpha$ -TURING machine and are well suited to handle the calculus of  $\text{Primo}_\alpha$  functions, i.e. every  $\text{Primo}_\alpha$  function is also multitape  $\alpha$ -computable (Theorem 1).

Using a result from [3] that the  $\text{Primo}_\alpha$  functions are exactly the  $\text{Prims}$  functions that map ordinals to ordinals we prove Theorem 2: If  $\alpha$  is an ordinal closed under  $\text{Primo}_\alpha$  functions then the  $\Delta_1(L_\alpha[B])$  definable subsets of  $\alpha$  are exactly

the ones that are multitape  $\alpha$ -computable in  $B$ . Furthermore we are able to give a characterization of *admissible ordinals* by the means of multitape  $\alpha$ -TURING machines (Theorem 3): A limit ordinal is admissible iff there is no multitape  $\alpha$ -computable function mapping some  $\beta < \alpha$  cofinally into  $\alpha$ . Admissible ordinals, which play an important role in generalizing recursion theory, are classically defined by the means of definability over a constructible level  $L_\alpha$  or axiomatized through the axioms of Kripke-Platek set theory. We hence provide an alternative approach to admissibility from a computational perspective.

Similar theorems are already contained in [6] but obtained in a different way: A (single-tape)  $\alpha$ -computable truth function for  $L_\alpha$  is employed to transfer definability over  $L_\alpha$  to the computational context. The computability of this truth function however requires  $\alpha$  to be sufficiently closed with respect to ordinal arithmetic. Seeing this in a talk the second author gave in a seminar on ordinal computability in Bonn in November 2007, the first author suggested to make use of the well developed theory of primitive recursive ordinal and set functions and applied them in the proofs of Theorems 2 and 3. We kindly thank PETER KOEPKE for his suggestions and support of this work.

## 2 Multitape $\alpha$ -TURING Machines

A program  $P$  for a standard TURING-machine with  $k$  tapes (each with an independent read-write head) can be seen as a finite subset of  $\{0, 1\}^k \times \omega \times \{0, 1\}^k \times \omega \times \{-1, +1\}^k$ . An element  $(a, s, a', s', d) \in P$  codes the following instruction: If the  $k$ -many read write heads read the symbols corresponding to the entries of the vector  $a \in \{0, 1\}^k$  and the machine is in state  $s \in \omega$  then have the heads write the entries of  $a' \in \{0, 1\}^k$  to the respective tapes, change the machine state to  $s' \in \omega$  and move the read-write heads according to the vector  $d \in \{-1, +1\}^k$ . Similarly to previous studies ([4, 6]) this can be used as a basis for the following notion of a transfinite computation according to  $P$ :

At successor times the program  $P$  is used as in the standard TURING machine case, with the single exception that when dealing with tapes of transfinite length a convention has to be found what should happen when a read-write-head is being moved left from a cell indexed by a limit ordinal. In this situation we want the head to be reset to the beginning of the tape (cell number ‘0’).

For limit times the TURING-program cannot determine the tape content, head positions and program state so we have to define them in a sensible way. Following the lines of [4] we use inferior limits: We want each single cell of every tape to contain the liminf of its previous values, the machine state to be the liminf of the previous machine states and every tape’s read/write-head to be located on the cell indexed by the liminf over the positions it previously assumed in the limit machine state, i.e. the least cell that was read cofinally often while the machine was in the same state as at the limit time.

More formally:

**Definition 1.** Let  $\alpha$  be a limit ordinal or  $\alpha = \text{Ord}$ . Let  $P \subseteq \{0, 1\}^k \times \omega \times \{0, 1\}^k \times \omega \times \{-1, 1\}^k$  be finite and let  $T_0 = (T_{0,0}, T_{1,0}, \dots, T_{(k-1),0}) \in ({^\alpha}\{0, 1\})^k$

be the initial tape content of the  $k$ -many tapes of length  $\alpha$ . A triple

$$(T_\theta, H_\theta, S_\theta)_{\theta \leq \Theta}$$

is a  $k$ -tape  $\alpha$ -TURING computation by  $P$  on input  $T_0$  if the following conditions hold:

- $\Theta \leq \alpha$ ;
- $S_\theta \in \omega$  for  $\theta \leq \Theta$  ;
- $H_\theta = (H_{0,\theta}, H_{1,\theta}, \dots, H_{(k-1),\theta})$   
where  $H_{i,\theta} \in \alpha$  for  $0 \leq i < k$  and for  $\theta \leq \Theta$  ;
- $T_\theta = (T_{0,\theta}, T_{1,\theta}, \dots, T_{(k-1),\theta})$   
where  $T_{i,\theta} : \alpha \rightarrow \{0, 1\}$  for  $0 \leq i < k$  and for  $\theta \leq \Theta$  ;
- $(T_\theta, H_\theta, S_\theta)_{\theta \leq \Theta}$  is defined recursively in  $P$  and the initial tape contents  $T_{i,0}$  in the following way:

**Termination:** Let  $\theta \leq \Theta < \alpha$  and let  $(T_{\theta'}, H_{\theta'}, S_{\theta'})_{\theta' \leq \theta}$  be already defined. If there is no  $(a, s, a', s', d) \in P$  where  $(T_{i,\theta}(H_{i,\theta}))_{i < k} = a$  and  $S_\theta = s$  then the computation terminates, i.e.  $\theta = \Theta$ .

**Successor step:** Let  $\theta \leq \Theta$ , let  $(T_{\theta'}, H_{\theta'}, S_{\theta'})_{\theta' \leq \theta}$  be already defined and let there be a  $c = (a, s, a', s', d) \in P$  where  $(T_{i,\theta}(H_{i,\theta}))_{i < k} = a$  and  $S_\theta = s$ . Choose  $c$  minimally with respect to some fixed well-order on  $P$ . As usual we want the configuration  $(T_{\theta+1}, H_{\theta+1}, S_{\theta+1})$  to be derived from  $(T_\theta, H_\theta, S_\theta)$  according to the instruction  $c$ .

Let  $a' = (a'_0, a'_1, \dots, a'_{k-1})$ . For all  $i < k$  we require:

$$\begin{aligned} T_{i,\theta+1}(\xi) &= \begin{cases} a_i & , \text{ if } \xi = H_{i,\theta} \\ T_{i,\theta}(\xi) & , \text{ else} \end{cases} \\ H_{i,\theta+1} &= \begin{cases} H_{i,\theta} + 1 & , \text{ if } d = +1 \\ H_{i,\theta} - 1 & , \text{ if } d = -1 \text{ and } H_{i,\theta} \text{ is a successor ordinal} \\ 0 & , \text{ if } d = -1 \text{ and } H_{i,\theta} \text{ is a limit ordinal} \end{cases} \\ S_{i,\theta+1} &= s'. \end{aligned}$$

**Limit step:** Now let  $\theta \leq \Theta$  be a limit ordinal and let  $(T_{\theta'}, H_{\theta'}, S_{\theta'})_{\theta' < \theta}$  be already defined. For  $i < k$  and  $\xi < \alpha$  set

$$\begin{aligned} S_\theta &= \liminf_{\theta' < \theta} S_{\theta'} \\ H_{i,\theta} &= \liminf_{\theta' < \theta, S_{\theta'} = S_{\theta'}} H_{i,\theta'} \\ T_{i,\theta}(\xi) &= \liminf_{\theta' < \theta} T_{i,\theta'}(\xi). \end{aligned}$$

Note that the machine configuration at limit times is always defined whenever the configurations at all previous stages are defined. If  $\theta = \Theta = \alpha$  we say that the computation diverges.

The primitive recursive ordinal functions we want to compute are  $n$ -ary functions mapping ordinals  $< \alpha$  to  $\alpha$ . So we define:

**Definition 2.** Let  $B \subseteq \alpha$  and  $n \in \omega$ . A function  $f : \alpha^n \rightarrow \alpha$  is called  $k\text{-}\alpha$ -computable in  $B$  if there is a  $k$ -tape TURING program  $P$  and a finite sequence of ordinal parameters  $\pi = (\pi_1, \pi_2, \dots, \pi_m) \in \alpha^m$  s.t.  $k \geq n+m+2$  and for all  $\xi = (\xi_1, \xi_2, \dots, \xi_n) \in \alpha^n$  the  $k$ -tape  $\alpha$ -TURING computation by  $P$  with input  $T_0$   $(T_\theta, H_\theta, S_\theta)_{\theta \leq \Theta_\xi}$  is of the form

- $\Theta_\xi < \alpha$ , i.e. the computation terminates;
- for every  $i < n$  there is a read-only tape containing  $\chi_{\{\xi_i\}}$ ;
- for every  $j < m$  there is a read-only tape containing  $\chi_{\{\pi_j\}}$ ;
- there is a read-only tape containing  $\chi_B$ ;
- all other tapes are initially empty;
- there is a tape that at time  $\Theta_\xi$  contains  $\chi_{\{f(\xi)\}}$ .

If  $B = \emptyset$  we call  $f$   $k\text{-}\alpha$ -computable.

A function  $g : \alpha^n \rightarrow \alpha$  is called multitape  $\alpha$ -computable if there is a  $k$  such that  $g$  is  $k\text{-}\alpha$ -computable.

The question arises which closure properties an ordinal  $\alpha$  must have so that the notions of  $\alpha$ -computable in  $B$  defined in [6] and multitape  $\alpha$ -computable in  $B$  coincide. Clearly this is true for admissible ordinals, but much weaker closure properties will certainly suffice.

**Lemma 1.** Let  $f : \alpha^n \rightarrow \alpha$  be  $k\text{-}\alpha$ -computable by the program  $P$  in parameters  $\pi$ . Let  $(T_\theta, H_\theta, S_\theta)_{\theta \leq \Theta_\xi}$  be the  $k\text{-}\alpha$ -computation by  $P$  for  $f(\xi)$ . Then the function  $\text{Time}_f : \alpha^n \rightarrow \alpha$  which maps  $\xi \mapsto \Theta_\xi$  is  $(k+1)\text{-}\alpha$ -computable.

*Proof.* We extend  $P$  to a  $(k+1)$ -tape TURING program which still computes  $f$  but where every instruction in  $P$  additionally moves the  $(k+1)$ -st tape's head to the right. The computation terminates after  $\Theta_\xi$  many steps, i.e. there is no command for the final configuration. Add a new instruction for this configuration that writes a '1' at the current head position of the  $(k+1)$ -st tape. Since the  $k+1$ -st tape's head now points to a '1' instead of a '0' the computation terminates with this new instruction. It follows that  $\text{Time}_f$  is  $(k+1)\text{-}\alpha$ -computable.

### 3 Primitive Recursive Functions

The familiar primitive recursive functions on natural numbers are those recursive functions generated by a weak recursion scheme that allows recursive definitions using the supremum over the previous function values. A generalization of this concept to functions operating on the universe of sets has for instance been used in the study of the constructible hierarchy ([2]). In [3] JENSEN and KARP defined the notion of primitive recursiveness for functions mapping ordinals to ordinals and developed their theory. Following [3] we define:

**Definition 3.** The class  $\text{Prim}_O(B)$  of primitive recursive ordinal functions in  $B \subseteq \text{Ord}$  is defined as the minimal set containing all the functions of type (1) to (5) and closed under the schemes for substitution (a) and (b) and recursion ( $R$ ).

- (1)  $f(\xi) = \chi_B(\xi)$
- (2)  $\text{pr}_{n,i}(\xi) = \xi_i$ , for all  $n \in \omega$ ,  $\xi = (\xi_1, \dots, \xi_n)$  and  $1 \leq i \leq n$ .
- (3)  $f(\xi) = 0$
- (4)  $f(\xi) = \xi + 1$
- (5)  $c(\xi, \zeta, \gamma, \delta) = \begin{cases} \xi, & \text{if } \gamma < \delta \\ \zeta, & \text{else} \end{cases}$
- (a)  $f(\xi, \zeta) = g(\xi, h(\xi), \zeta)$
- (b)  $f(\xi, \zeta) = g(h(\xi), \zeta)$
- (R)  $f(\zeta, \xi) = g(\sup\{f(\eta, \xi) \mid \eta < \zeta\}, \zeta, \xi)$

We write  $\text{Prim}_O$  instead of  $\text{Prim}_O(\emptyset)$ .

If  $B \subseteq \alpha$  and  $\alpha$  is an ordinal that is closed under  $\text{Prim}_O(B)$  functions then we call a function  $f : \alpha^n \rightarrow \alpha$   $\text{Prim}_O(B)$  iff it is the restriction of a  $\text{Prim}_O(B)$  function.

Definition 3 is a special case of the following definition of primitive recursive set functions also taken from [3]:

**Definition 4.** Let  $X$  be a one-place set function. The class  $\text{Prims}(X)$  of primitive recursive set functions in  $X$  is defined as the minimal set containing all the functions of type (1) to (5) and closed under the schemes for substitution (a) and (b) and recursion (R).

- (1)  $F(x) = X(x)$
- (2)  $Pr_{n,i}(\mathbf{x}) = x_i$ , for all  $n \in \omega$ ,  $\mathbf{x} = (x_1, \dots, x_n) \in \alpha^n$  and  $1 \leq i \leq n$ .
- (3)  $F(x) = 0$
- (4)  $F(x, y) = x \cup \{y\}$
- (5)  $C(x, y, u, v) = \begin{cases} x, & \text{if } u \in v \\ y, & \text{else} \end{cases}$
- (a)  $F(\mathbf{x}, \mathbf{y}) = G(\mathbf{x}, H(\mathbf{x}), \mathbf{y})$
- (b)  $F(\mathbf{x}, \mathbf{y}) = G(H(\mathbf{x}), \mathbf{y})$
- (R)  $F(y, \mathbf{x}) = G(\sup\{F(z, \mathbf{x}) \mid z < y\}, y, \mathbf{x})$

We write  $\text{Prims}$  instead of  $\text{Prims}(\emptyset)$ .

If  $B$  is a set and  $X$  is the unary function  $X(x) = x \cap B$  then we may write  $\text{Prims}(B)$  for  $\text{Prims}(X)$ .

If  $B \subseteq \alpha$  and  $\alpha$  is an ordinal that is closed under  $\text{Prim}_O(B)$  functions then we call a function  $f : \alpha^n \rightarrow \alpha$   $\text{Prims}(B)$  iff it is the restriction of a  $\text{Prims}(B)$  function.

An  $n$ -ary relation  $R \subseteq V^n$  is  $\text{Prims}(X)$  if there is a  $\text{Prims}(X)$  function  $F_R : V^n \rightarrow V$  s.t.  $F_R(\mathbf{x}) = 0$  iff  $\mathbf{x} \in R$ .

**Theorem 1.** If  $\alpha$  is closed under  $\text{Prim}_O$  functions, then every  $\text{Prim}_O(B)$  function  $f : \alpha^n \rightarrow \alpha$  is multitape  $\alpha$ -computable in  $B$ . Furthermore there is a  $\text{Prim}_O$  function  $M_f$  that majorizes  $\text{Time}_f$ , i.e.  $\forall \xi \in \alpha^n M(\xi) > \text{Time}_f(\xi)$ .

*Proof.* Functions (1) to (4) are easily seen to be multitape  $\alpha$ -computable with majorizing functions the maximum input ordinal plus 1.

(5) We define a program to compute  $C$ . Move the heads of the tapes containing  $\gamma$  and  $\delta$  to the right to decide whether  $\gamma < \delta$ . According to the outcome move the output tape's head together with the head of the tape containing  $\xi$  or  $\zeta$  to the right until a '1' is read. Copy the '1' to the output tape and stop. This program runs at most  $M_C(\gamma, \delta, \xi, \zeta) = \max\{\gamma, \delta\} + \max\{\xi, \zeta\} + 1$ -many steps. Since  $\alpha$  is closed under  $\text{Primo}$  functions and ordinal addition is  $\text{Primo}$  the program terminates and  $C$  is multitape  $\alpha$ -computable.

(a) Let  $g$  be  $k$ - $\alpha$ -computable by a program  $P_g$  with parameters  $\pi_g$  and majorizing function  $M_g$ ,  $h$   $l$ - $\alpha$ -computable by  $P_h$  with parameters  $\pi_h$  and majorized by function  $M_h$ . We define a  $(k+l)$ -tape TURING program that computes  $f$  using  $\pi_g \hat{\wedge} \pi_h$  as parameters. The program first runs  $P_h$  to compute  $h(\xi)$ . Note that any program computing  $f$  in parameters  $\pi_g \hat{\wedge} \pi_h$  uses tapes containing the components of  $\xi, \zeta, \pi_h, \pi_g$  and one tape containing the characteristic function of  $B$ ; these can be used as input tapes for  $P_h$  and later on for  $P_g$  respectively. After resetting all heads to position zero the program continues with running  $P_g$  with the output tape of  $P_h$ , now containing  $h(\xi)$ , as additional input tape. Resetting of heads takes only finitely many machine steps thanks to the well-foundedness of ordinals (to recognize head position '0' we may assume additional parameter tapes containing  $\chi_{\{0\}}$ ). So the new Program will run in less than  $\alpha$  many steps since addition of ordinals is  $\text{Primo}$  and  $\alpha$  is closed under  $\text{Primo}$  functions. We can set  $M_f = (M_g \cdot 2) + (M_h \cdot 2)$ .

(b) Similarly to (a).

(R) Let  $g$  be  $k$ - $\alpha$ -computable by  $P_g$  in parameters  $\pi_g$  and majorized by  $M_g$ . We describe a  $(k+2)$ -tape TURING program that computes  $f$  with parameters  $\pi_g$ . The algorithm runs through  $\zeta$ -many *stages*. In stage  $\eta < \zeta$  the new  $(k+1)$ -st tape contains the current value of  $\sup\{f(\eta', \xi) \mid \eta' < \eta\}$ .  $P_g$  is called once to compute  $f(\eta, \xi) = g(\sup\{f(\eta', \xi) \mid \eta' < \eta\}, \eta, \xi)$ . If necessary the value of  $\sup\{f(\eta', \xi) \mid \eta' \leq \eta\}$  has to be updated. The stage concludes by erasing all the work tapes used by  $P_g$  and resetting all heads to zero. We have to ensure that for all  $\eta < \zeta$  the following conditions hold:

- (i) At the time of the call of  $P_g$  to compute  $f(\eta, \xi)$  the  $(k+1)$ -st tape contains in fact  $\sup\{f(\eta', \xi) \mid \eta' \leq \eta\}$ .
- (ii) The number of machine steps the program uses before entering stage  $\eta$  is less than  $\alpha$ .

For (i) use the  $(k+2)$ -nd tape to save the value of  $f(\eta, \xi) = \gamma$  not as  $\chi_{\{\gamma\}}$  but as  $\chi_{\gamma+1}$ . At the beginning of every stage use tape  $(k+2)$  to write a '1' to the  $\sup\{f(\eta', \xi) \mid \eta' \leq \eta\}$ -th cell of tape  $(k+1)$  and use this as input to  $P_g$ . (i) holds inductively at successor stages. So let  $\eta$  be a limit ordinal. By the liminf-rule tape  $(k+2)$  contains in fact  $\chi_{\sup\{f(\eta', \xi) \mid \eta' \leq \eta\}}$ . So (i) holds.

Stage  $\eta$  consists of the following operations:

- Find the first 0 on tape  $(k+2)$  and write a 1 to the respective cell of tape  $(k+1)$  (note that the the first '0' occurs in cell number  $\sup\{f(\eta', \xi) \mid \eta' < \eta\} + 1$ ):

As seen above the number of machine steps  $\beta_{\text{find}} = \sup\{f(\eta', \xi) \mid \eta' < \eta\} + 2$  is less than  $\alpha$ .

- Reset tape  $(k+1)$ 's head: This needs at most  $\beta_{\text{reset}(k+1)} = \sup\{f(\eta', \xi) \mid \eta' < \eta\}$  many steps.
- Run  $P_g$  to compute  $f(\eta, \xi) = g(\sup\{f(\eta', \xi) \mid \eta' < \eta\}, \eta, \xi)$ , the number of steps  $\beta_g = \text{Time}_g(\sup\{f(\eta', \xi) \mid \eta' < \eta\}, \eta, \xi)$  is  $(k+1)$ - $\alpha$ -computable therefore  $< \alpha$ .
- Reset the head of  $P_g$ 's output tape and update tape  $(k+2)$  if necessary: This can be decided and done in at most  $\beta_{\text{update}} = f(\eta, \xi) \cdot 2 < \alpha$  many steps.
- Erase the work tapes used by  $P_g$ . WLOG  $P_g$  maintains a ‘timer’ tape as in Lemma 1 which we can use to determine up to which cell the tapes have to be erased. Since only cells up to index  $\beta_g$  may have been used by  $P_g$  (all heads were at position 0 when  $P_g$  was called) this takes again at most  $\beta_g \cdot 2$  many steps ( $\cdot 2$  since we have to reset the heads before erasing the tapes).
- Reset all heads. This takes at most  $\max\{\beta_{\text{find}}, \beta_g\}$  many steps.

We define a majorizing function  $M_f$  for  $\text{Time}_f$  by  $\text{Prim}_O$  recursion:

$$\begin{aligned} M_f(\eta) = & \sup_{\eta' < \eta} M_f(\eta') \\ & + (\sup\{f(\eta', \xi) \mid \eta' \leq \eta\} + 2) \cdot 2 \\ & + M_g(\sup\{f(\eta', \xi) \mid \eta' \leq \eta\}, \eta) \\ & + f(\eta, \xi) \cdot 2 \\ & + M_g(\sup\{f(\eta', \xi) \mid \eta' \leq \eta\}, \eta) \cdot 2 \\ & + \max\{\sup\{f(\eta', \xi) \mid \eta' \leq \eta\}, M_g(\sup\{f(\eta', \xi) \mid \eta' \leq \eta\}, \eta, \xi)\} \end{aligned}$$

It follows from inductive analysis of the algorithm above that  $\text{Time}_f < M_f$ . Note that as supremum of the first  $\eta$  many values of a  $\text{Prim}_O$  function  $\sup_{\eta' < \eta} M_f(\eta')$  is  $\text{Prim}_O$  and therefore  $< \alpha$ . So (ii) holds.

## 4 Applications

The following theorems are similar to Theorem 7 and 9 in [6] which provide a connection between  $\alpha$ -recursion theory and ordinal computability. The proofs found in [6] explicitly give a truth function for bounded formulas in  $L_\alpha$  which is  $\alpha$ -computable if  $\alpha$  is sufficiently closed with respect to ordinal arithmetic. Instead we use facts from the classical theory of  $\text{Prims}$  functions to obtain these results.

**Theorem 2.** *If  $\alpha$  is an ordinal closed under  $\text{Prim}_O$  functions then  $A \subseteq \alpha$  is  $\Delta_1(L_\alpha[B])$  iff  $A$  is multitape  $\alpha$ -computable in  $B$ .*

*Proof.* ‘ $\Leftarrow$ ’ follows from the recursion theorem as in [6].

‘ $\Rightarrow$ ’ Let  $A \subseteq \alpha$  be  $\Delta_1(L_\alpha[B])$  by

$$\begin{aligned} \gamma \in A \leftrightarrow L_\alpha[B] \models \exists x \phi[x, \gamma, \mathbf{p}] \\ \gamma \notin A \leftrightarrow L_\alpha[B] \models \exists x \psi[x, \gamma, \mathbf{q}]. \end{aligned}$$

where  $\phi, \psi$  are  $\Sigma_0^B$  formulas. So we have:

$$\begin{aligned}\gamma \in A &\leftrightarrow L_\alpha[B] \models \exists x \phi[x, \gamma, \mathbf{p}] \\ &\leftrightarrow \exists x \in L_\alpha[B] \phi[x, \gamma, \mathbf{p}]\end{aligned}$$

In [3], Lemma 3.2, it is shown that there is a one-one  $\text{Prims}_S$  function  $N$  mapping the ordinals onto the constructible sets s.t.  $N \upharpoonright \alpha : \alpha \xrightarrow{\text{bij}} L_\alpha$ . It is easily seen that there is also a one-one  $\text{Prims}(B)$  function  $N'$  mapping the ordinals onto  $L[B]$  s.t.  $F = N' \upharpoonright \alpha : \alpha \xrightarrow{\text{bij}} L_\alpha[B]$ . So we can write:

$$\leftrightarrow \exists \xi \in \alpha \phi[F(\xi), \gamma, \mathbf{F}(\boldsymbol{\pi})]$$

Like in [2], Lemma I.2.4, we see that every  $\Sigma_0^B$  relation is  $\text{Prims}(B)$ . Hence there is a  $\text{Prims}(B)$  function  $G$  s.t.  $G(z) = 0$  iff  $\phi[z]$ :

$$\begin{aligned}\leftrightarrow \exists \xi \in \alpha G(F(\xi), \gamma, \mathbf{F}(\boldsymbol{\pi})) &= 0 \\ \leftrightarrow \exists \xi \in \alpha g(\xi, \gamma, \boldsymbol{\pi}) &= 0.\end{aligned}$$

Where

$$g(\xi, \gamma, \boldsymbol{\pi}) = \begin{cases} 0 & , \text{ if } G(F(\xi), \gamma, \mathbf{F}(\boldsymbol{\pi})) = 0 \\ 1 & , \text{ else.} \end{cases}$$

Since  $G$  is  $\text{Prims}(B)$  so is  $g$  by (a) and (5).  $g$  mapping ordinals to ordinals,  $B \subset \alpha$ , and we see like in Theorem 3.5 in [3] that  $g$  is  $\text{Prim}_O(B)$ . Similarly we obtain a function  $h$  for  $\psi$ . Now we can describe the following algorithm which computes the characteristic function of  $A$ :

```
WHILE ( $g(\xi, \gamma, \boldsymbol{\pi}) = 1$  AND  $h(\xi, \gamma, \boldsymbol{\eta}) = 1$ ) DO  $\xi + +$ ;  
IF  $g(\xi, \gamma, \boldsymbol{\pi}) = 0$  THEN STOP = 0;  
IF  $h(\xi, \gamma, \boldsymbol{\pi}) = 0$  THEN STOP = 1;
```

We analyse the algorithm into its stages  $\xi < \alpha$ , each consisting mainly of one computation for  $g$  and one for  $h$  plus some erasing of work tapes and resetting of heads. We have to make sure that this algorithm behaves nicely at limit stages, i.e. actually reaches every limit stage  $\delta < \alpha$ . Again it will be necessary to store the counter  $\xi$  as  $\chi_{\xi+1}$  and to decode this into  $\chi_\xi$  at the beginning of every stage. Similar to the proof of Theorem 1 we see that the algorithm up to stage  $\xi$  uses a number of steps majorized by a  $\text{Prim}_O$  function  $M(\xi)$ . Again also  $\sup_{\zeta < \delta} M(\zeta)$  for  $\delta < \alpha$  is  $\text{Prim}_O$  so the algorithm reaches every stage.

We can now give a characterization of admissible ordinals solely based on ordinal computations.

**Theorem 3.** *A limit ordinal  $\alpha$  is admissible iff there is no multitape  $\alpha$ -computable function mapping some  $\beta < \alpha$  cofinally into  $\alpha$ .*

*Proof.*  $\alpha$  is admissible iff there is no  $\Sigma_1(L_\alpha)$ -definable total function that maps some  $\beta < \alpha$  cofinally into  $\alpha$  (cf. [1], Lemma II.7.2).

If  $\alpha$  is already closed under  $\text{Primo}_\alpha$  functions then any multitape  $\alpha$ -computable function  $f$  is  $\Delta_1(L_\alpha)$  (and therefore  $\Sigma_1(L_\alpha)$ ) by the recursion theorem as in [6] (cf. Theorem 2). Conversely let  $\alpha$  be not admissible and let  $f : \beta \xrightarrow{\text{cof}} \alpha$  be a  $\Sigma_1(L_\alpha)$ -definable total function on  $\beta$ . So we have:

$$\begin{aligned} f(\gamma) = \delta &\leftrightarrow L_\alpha \models \exists x \phi[x, \gamma, \delta, \mathbf{p}] \\ &\leftrightarrow \exists x \in L_\alpha \phi[x, \gamma, \delta, \mathbf{p}] \end{aligned}$$

Where  $\phi$  is a  $\Sigma_0$  function. With  $F : \alpha \xrightarrow{\text{bij}} L_\alpha$   $\text{Prims}$  from Lemma 3.2 in [3]:

$$\leftrightarrow \exists \xi \in \alpha \phi[F(\xi), \gamma, \delta, \mathbf{F}(\boldsymbol{\pi})]$$

Since  $\phi$  is  $\Sigma_0$  it is  $\text{Prims}$  ([2], Lemma I.2.4):

$$\begin{aligned} &\leftrightarrow \exists \xi \in \alpha G(F(\xi), \gamma, \delta, \mathbf{F}(\boldsymbol{\pi})) = 0 \\ &\leftrightarrow \exists \xi \in \alpha g(\xi, \gamma, \delta, \boldsymbol{\pi}) = 0. \end{aligned}$$

Where

$$g(\xi, \gamma, \delta, \boldsymbol{\pi}) = \begin{cases} 0 & , \text{ if } G(F(\xi), \gamma, \delta, \mathbf{F}(\boldsymbol{\pi})) = 0 \\ 1 & , \text{ else.} \end{cases}$$

Since  $G$  is  $\text{Prims}$  so is  $g$  by (a) and (5).  $g$  is mapping ordinals to ordinals and is therefore  $\text{Primo}_\alpha$  by Theorem 3.5 in [3]. The desired algorithm goes through less than  $\alpha$  many stages to compute  $f(\gamma)$  given  $\gamma$  as input. In every stage  $\eta < \alpha$  the algorithm computes the values of  $g(\xi, \gamma, \delta, \boldsymbol{\pi})$  for all  $\xi, \delta < \eta$ .  $g$  is  $\text{Primo}_\alpha$  so  $\text{Time}_g$  is multitape  $\alpha$ -computable majorized by  $M_g$  which in turn is  $\text{Primo}_\alpha$ . So  $\sup_{\xi, \delta < \eta} M(\xi, \gamma, \delta, \boldsymbol{\pi})$  is less than  $\alpha$  and every stage  $\eta$  is reached by the algorithm. At some stage one computation for  $g$  will return 0 and the value  $\delta = f(\gamma)$  is found. So  $f$  is multitape  $\alpha$ -computable as required.

In the case that  $\alpha$  is not closed under  $\text{Primo}_\alpha$  functions we will define  $\beta < \alpha$  and a multitape  $\alpha$ -computable total function  $f : \beta \xrightarrow{\text{cof}} \alpha$ .

Any limit ordinal is closed under (1)-(4) so assume in the following closure under (1)-(4). If  $\alpha$  is not closed under (5) then also for one instance of (5)  $f(\zeta_0, \zeta_1, \zeta_2, \zeta_3)$  the canonical algorithm will not terminate in less than  $\alpha$  many steps. Analysing the algorithm in the proof of Theorem 1 we can extract two ordinals  $\gamma, \delta < \alpha$  with  $\gamma + \delta \geq \alpha$  (set  $\gamma = \max\{\zeta_2, \zeta_3\}$  and  $\delta = \max\{\zeta_0, \zeta_1\}$ ). So there is a  $\beta \leq \delta$  such that the multitape  $\alpha$ -computable function  $f : \beta \longrightarrow \alpha$ ,  $\xi \mapsto \gamma + \xi$  is cofinal in  $\alpha$ .

If  $\alpha$  is closed under two functions  $f$  and  $g$  so it is also closed under  $f(\boldsymbol{\xi}, \boldsymbol{\zeta}) = g(\boldsymbol{\xi}, h(\boldsymbol{\xi}), \boldsymbol{\zeta})$  and under  $f'(\boldsymbol{\xi}, \boldsymbol{\zeta}) = g(h(\boldsymbol{\xi}), \boldsymbol{\zeta})$ . So if  $\alpha$  is closed under (1)-(5) but not closed under  $\text{Primo}_\alpha$  functions it has to be not closed under (R).

Assume  $\alpha$  closed under (1)-(5),(a),(b) but not closed under (R). Since the  $\text{Primo}_\alpha$  functions are defined by recursion, there are  $\text{Primo}_\alpha$  functions  $f, g$  s.t.

$f(\zeta, \xi) = g(\sup\{f(\eta, \xi) \mid \eta < \zeta\}, \zeta, \xi)$  is an instance of (R) and  $\alpha$  closed under  $g$  but not closed under  $f$ . Choose  $\beta$  minimally s.t.  $f(\beta, \xi) \notin \alpha$ .  $f(\beta, \xi) = g(\sup_{\gamma < \beta} f(\gamma, \xi), \beta, \xi)$  and since  $\alpha$  is closed under  $g$  we have that  $\sup_{\gamma < \beta} f(\gamma, \xi) = \alpha$ . Now  $f \upharpoonright \beta : \beta \longrightarrow \alpha$  is cofinal in  $\alpha$ .

## References

1. KEITH DEVLIN. *Constructibility*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1984.
2. KEITH DEVLIN. *Aspects of Constructibility*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, Heidelberg, 1973.
3. RONALD B. JENSEN AND CAROL KARP. *Primitive recursive set functions*. In: Axiomatic Set Theory, Proceedings of Symposia in Pure Mathematics, Volume XIII, Part I. American Mathematical Society, Providence, Rhode Island, 1971.
4. PETER KOEPKE. *Turing computations on ordinals*. The Bulletin of Symbolic Logic 11 (2005), 377–397.
5. PETER KOEPKE AND MARTIN KOERWIEN. *Ordinal computations*. Mathematical Structures in Computer Science (2006), 867–884.
6. PETER KOEPKE AND BENJAMIN SEYFFERTH. *Ordinal Machines and Admissible Recursion Theory*. Submitted to: Annals of Pure and Applied Logic, CiE 2007 special volume, to be published 2008.

# Prescribed Learning of Indexed Families\*

Sanjay Jain<sup>1</sup>, Frank Stephan<sup>2</sup>, and Nan Ye<sup>3</sup>

<sup>1</sup> Department of Computer Science,  
National University of Singapore, Singapore 117590, Republic of Singapore.  
Email: [sanjay@comp.nus.edu.sg](mailto:sanjay@comp.nus.edu.sg).

<sup>2</sup> Department of Computer Science and Department of Mathematics,  
National University of Singapore, Singapore 117543, Republic of Singapore.  
Email: [fstephan@comp.nus.edu.sg](mailto:fstephan@comp.nus.edu.sg).

<sup>3</sup> Department of Computer Science and Department of Mathematics,  
National University of Singapore, Singapore 117543, Republic of Singapore.  
Email: [u0407028@nus.edu.sg](mailto:u0407028@nus.edu.sg).

**Abstract.** This work extends studies of Angluin, Lange and Zeugmann on how learnability of a language class depends on the hypothesis space used by the learner. While previous studies mainly focused on the case where the learner chooses a particular hypothesis space, the goal of this work is to investigate the case where the learner has to cope with all possible hypothesis spaces. In that sense, the present work combines the approach of Angluin, Lange and Zeugmann with the question of how a learner can be synthesized. The investigation for the case of uniformly r.e. classes has been done by Jain, Stephan and Ye [6]. This paper investigates the case for indexed families and gives a special attention to the notions of conservative and non U-shaped learning.

## 1 Introduction

The goal of inductive inference [1, 2, 4] is to model the process of learning rigorously. Following many real-world scenarios, the learner observes more and more data which in the limit uniquely determines the concept to be learnt. The learner is supposed to determine the target concept from the data it observes. Following the model of linguistics, the concept to be learnt is always considered to be an (often infinite) set of finite items which can be coded as natural numbers. The language to be learnt is chosen from a concept class  $\{L_0, L_1, L_2, \dots\}$  and the learner is using an explicit hypothesis space  $\{H_0, H_1, H_2, \dots\}$ . This hypothesis space may be either the same as  $\{L_0, L_1, L_2, \dots\}$  (exact learning [1]) or chosen by the learner (class-preserving and class-comprising learning [8, 13, 14]) or imposed on the learner (prescribed and uniform learning [6]). Angluin [1] considered the important case that the concept class and hypothesis class are both given by an indexed family, that is, the class is uniformly recursive. She

---

\* The work for this paper is supported in part by NUS grants number R252-000-212-112 and R252-000-308-112. A full version is available as Technical Report TRB9/07, School of Computing, National University of Singapore, 2007.

has given a characterization when such a class is explanatorily learnable and introduced also important variants like consistent and conservative learning.

The goal of the present work is to study prescribed and uniform learning and to contrast the results obtained for them to the well-studied cases of exact, class-preserving and class-comprising learning. The idea that the learner has to accept a given choice of the hypothesis class is not completely new; besides the case of exact learning (for which the results would be equivalent to the (not considered case of) class-preserving prescribed learning), it has also been considered under the framework of synthesis of learners. But the models like those considered by Zilles [15, 16] differ from the scenario in the present work. Jain, Stephan and Ye [6] have studied the more general case of uniformly r.e. concept and hypothesis spaces in a separate paper. The main difference to the setting in the r.e. case is that there it is more reasonable to consider class-preserving-uniformly and class-preserving-prescribed learning instead of uniform and prescribed learning. Furthermore, the relation between non U-shaped learning and conservative learning depends crucially on the indexed family nature of the hypothesis space.

The study of prescribed and uniform learning is done for the criteria of finite learning (Section 2), conservative learning (Section 3), non U-shaped learning (Section 4) and the various notions of monotonic learning (Section 5).

In the following, we will provide more details, but have to introduce some formal notations first. Let  $\mathbb{N}$  be the set of natural numbers and  $\langle \cdot, \cdot \rangle$  is a fixed pairing function: a recursive bijective mapping from  $\mathbb{N}^2$  to  $\mathbb{N}$ . Furthermore,  $|S|$  denotes the cardinality of set  $S$ .  $\varphi_0, \varphi_1, \varphi_2, \dots$  denotes a fixed acceptable numbering of the partial recursive functions from  $\mathbb{N}$  to  $\mathbb{N}$ . In some cases, we use  $\varphi_i$  as a function of two arguments. In such cases one implicitly assumes a pairing function being used to code the inputs: thus,  $\varphi_i(x, y)$  means  $\varphi_i(\langle x, y \rangle)$ . The set  $W_e$  is the domain of  $\varphi_e$ . The set  $\mathbb{K} = \{e : e \in W_e\}$  is the diagonal halting problem which is used as a standard example of an r.e. but nonrecursive set. Let  $\mathbb{K}_t$  denote the set of elements enumerated into  $\mathbb{K}$  within  $t$  steps, via some standard enumeration procedure. We assume without loss of generality that  $\mathbb{K}_0 = \emptyset$ .

**Definition 1.** A learner is a mapping from  $(\mathbb{N} \cup \{\#\})^*$  to  $\mathbb{N} \cup \{?\}$ . Let  $M$  be a given learner,  $\{L_0, L_1, L_2, \dots\}$  be a language class and  $\{H_0, H_1, H_2, \dots\}$  be a hypothesis space. In this paper,  $M$  is a partial-recursive function and  $\{L_0, L_1, L_2, \dots\}, \{H_0, H_1, H_2, \dots\}$  are indexed families of subsets of the natural numbers, that is, the mappings  $e, x \mapsto L_e(x)$  and  $e, x \mapsto H_e(x)$  are recursive functions from  $\mathbb{N} \times \mathbb{N}$  to  $\{0, 1\}$ . Let  $\sigma, \tau, \rho$  range over  $(\mathbb{N} \cup \{\#\})^*$ . Furthermore, let  $\sigma \subseteq \tau$  denote that  $\tau$  is an extension of  $\sigma$  as a string.  $T$  is a text if  $T$  is a total function which maps  $\mathbb{N}$  to  $\mathbb{N} \cup \{\#\}$  and  $T$  is a text for  $L_a$  iff the numbers occurring in  $T$  are exactly those in  $L_a$ .

A learner *converges* [4] on  $T$  to  $b$  iff there is an  $n$  with  $M(T[m]) = b$  for all  $m \geq n$ ; here  $T[m]$  is the finite string consisting of the first  $m$  members of  $T$ .

The learner  $M$  is *total* if  $M(\sigma)$  is defined for all finite strings  $\sigma$  in  $(\mathbb{N} \cup \{\#\})^*$ . For the learning criteria in this paper, a given learner can always be effectively converted to a total leaner which learns the same classes. Hence learners are assumed to be total here onwards.

The learner  $M$  is *finite* [4] if for every text  $T$  there is one index  $e$  such that for all  $n$ , either  $M(T[n]) = ?$  or  $M(T[n]) = e$ .

The learner  $M$  is *confident* [10] if  $M$  is total and converges on every text  $T$  to a hypothesis.

The learner  $M$  is *conservative* [1] if for all  $\sigma, \tau$  with  $H_{M(\sigma)} \neq H_{M(\sigma\tau)}$  there is an  $x$  occurring in  $\sigma\tau$  such that  $x \notin H_{M(\sigma)}$ .  $M$  is *non U-shaped* [3] if there are no  $a$  and  $\sigma, \tau, \rho \in (L_a \cup \{\#\})^*$  such that  $H_{M(\sigma)} = H_{M(\sigma\tau\rho)} = L_a$  and  $H_{M(\sigma\tau)} \neq L_a$ . In other words,  $M$  never changes from a correct to an incorrect and then back to a correct hypothesis.  $M$  is *decisive* [3] if there are no  $\sigma, \tau, \rho$  such that  $H_{M(\sigma\tau\rho)} = H_{M(\sigma)}$  and  $H_{M(\sigma\tau)} \neq H_{M(\sigma)}$ . In other words,  $M$  never returns to a once abandoned hypothesis (even semantically).

The learner  $M$  is *monotonic* [5] if for every  $L_a$  and for every  $\sigma, \tau \in (L_a \cup \{\#\})^*$  the inclusion  $L_a \cap H_{M(\sigma)} \subseteq L_a \cap H_{M(\sigma\tau)}$  holds.  $M$  is *strong-monotonic* [5] if for all  $\sigma, \tau \in (\mathbb{N} \cup \{\#\})^*$  the inclusion  $H_{M(\sigma)} \subseteq H_{M(\sigma\tau)}$  holds.

Here note that  $?$  is not considered as a conjecture and thus the constraints in conditions like conservative, monotonic, strong-monotonic, non U-shaped and decisive refer only to inputs where  $M$  makes a conjecture and does not output  $?$ : so, more formally, a learner would be strong-monotonic iff for all  $\sigma, \tau$ ,  $M(\sigma) \neq ?$  and  $M(\sigma\tau) \neq ?$  implies  $H_{M(\sigma)} \subseteq H_{M(\sigma\tau)}$ . Similarly for the other criteria.

Finite learning is quite restrictive since the learner has to make up its mind without having viewed all of the available infinite information. Learning in the limit (or just “learning”) is more powerful since the learner can revise its hypothesis a finite but arbitrary number of times.

In this paper we will only be concerned about learning indexed families and using hypothesis spaces which are also indexed families. Angluin, Kapur, Lange and Zeugmann [1, 7–9, 12–14] studied how learnability of the family to be learned depends on the hypothesis space  $\{H_0, H_1, H_2, \dots\}$  used by the learner. To formalize this, they introduced the notions of exact, class-preserving and class-comprising learning. In addition to this we consider notions like uniform and prescribed learning [6]. Here  $I$  ranges over properties of the learner as defined in Definition 1, so  $I$  stands for “conservative”, “finite”, “monotonic” and so on.

**Definition 2.** In the following, let  $\{L_0, L_1, L_2, \dots\}$  and  $\{H_0, H_1, H_2, \dots\}$  be indexed families.

$\{L_0, L_1, L_2, \dots\}$  is explanatorily learnable [4] with hypothesis space  $\{H_0, H_1, H_2, \dots\}$  iff there is a learner  $M$  which converges on every text of a language  $L_a$  to a hypothesis  $b$  such that  $H_b = L_a$ .

For a property  $I$  from Definition 1,  $\{L_0, L_1, L_2, \dots\}$  is  $I$  learnable with hypothesis space  $\{H_0, H_1, H_2, \dots\}$  iff there is a learner  $M$  which explanatorily learns  $\{L_0, L_1, L_2, \dots\}$  using hypothesis space  $\{H_0, H_1, H_2, \dots\}$  and furthermore satisfies the requirement  $I$ .

$\{L_0, L_1, L_2, \dots\}$  is class-comprisingly  $I$  learnable iff it is  $I$  learnable with some hypothesis space  $\{H_0, H_1, H_2, \dots\}$ ; note that learnability automatically implies  $\{L_0, L_1, L_2, \dots\} \subseteq \{H_0, H_1, H_2, \dots\}$ .

$\{L_0, L_1, L_2, \dots\}$  is class-preservingly  $I$  learnable iff it is  $I$  learnable with some hypothesis space  $\{H_0, H_1, H_2, \dots\}$  satisfying  $\{H_0, H_1, H_2, \dots\} = \{L_0, L_1, L_2, \dots\}$ .

$\{L_0, L_1, L_2, \dots\}$  is exactly  $I$  learnable iff it is  $I$  learnable with  $\{L_0, L_1, L_2, \dots\}$  itself taken as the hypothesis space.

$\{L_0, L_1, L_2, \dots\}$  is prescribed  $I$  learnable iff it is  $I$  learnable with respect to every hypothesis space  $\{H_0, H_1, H_2, \dots\}$  such that  $\{L_0, L_1, L_2, \dots\} \subseteq \{H_0, H_1, H_2, \dots\}$ .

$\{L_0, L_1, L_2, \dots\}$  is uniformly  $I$  learnable iff there is a recursive enumeration of partial-recursive functions  $M_0, M_1, M_2, \dots$  such that whenever  $\varphi_e$  is a decision-procedure  $b, x \mapsto H_b(x)$  for an indexed family  $\{H_0, H_1, H_2, \dots\} \supseteq \{L_0, L_1, L_2, \dots\}$  then  $M_e$  is an  $I$  learner for  $\{L_0, L_1, L_2, \dots\}$  using this hypothesis space  $\{H_0, H_1, H_2, \dots\}$ .

$\{L_0, L_1, L_2, \dots\}$  is class-preserving-uniformly  $I$  learnable iff there is a recursive enumeration of partial-recursive functions  $M_0, M_1, M_2, \dots$  such that whenever  $\varphi_e$  is a decision-procedure  $b, x \mapsto H_b(x)$  for an indexed family  $\{H_0, H_1, H_2, \dots\} = \{L_0, L_1, L_2, \dots\}$  then  $M_e$  is an  $I$  learner for  $\{L_0, L_1, L_2, \dots\}$  using this hypothesis space  $\{H_0, H_1, H_2, \dots\}$ .

**Remark 3.** For explanatory learning, class comprising learning and exact learning are the same as shown in [13]. This can be easily extended to the equivalence between uniform learning and exact learning, by noting that given a language class  $\{L_0, L_1, L_2, \dots\}$ , then for any hypothesis space  $\{H_0, H_1, H_2, \dots\} \supseteq \{L_0, L_1, L_2, \dots\}$ , one can find in the limit a  $b$  such that  $H_a = L_a$  for each  $a$ .

Exact finite learning and class comprising finite learning are the same [13]. For strong-monotonic, monotonic and conservative learning, there is a proper hierarchy for learning from exact, class preserving and class comprising hypothesis spaces [13]. For every criterion  $I$ , the following implications hold:

- Every uniformly  $I$  learnable family is also class-preserving-uniformly  $I$  learnable and prescribed  $I$  learnable.
- Every class-preserving-uniformly  $I$  learnable family and every prescribed  $I$  learnable family is also exactly  $I$  learnable.
- Every exactly  $I$  learnable family is also class-preservingly  $I$  learnable.
- Every class-preservingly  $I$  learnable family is also class-comprisibly  $I$  learnable.

It depends on the actual choice of  $I$  what other implications hold (besides the transitive ones).

For example, for confident learning, the class containing all  $\{x\}$  where  $|W_x| < \infty$  and  $\{x, y\}$  where  $x \neq y$  is not class-preservingly but class-comprisibly confidently learnable. Although confident learnability becomes more general in the class-comprising case, one can show that it coincides for all other criteria from Definition 2. Suppose that  $N$  is an exact confident learner for the class  $\{L_0, L_1, L_2, \dots\}$  and  $e$  is given such that  $\varphi_e$  is total and the hypothesis space  $\{H_0, H_1, H_2, \dots\}$  satisfies  $H_d = \{x : \varphi_e(\langle d, x \rangle) = 1\}$  for all  $d$  and  $\{L_0, L_1, L_2, \dots\} \subseteq \{H_0, H_1, H_2, \dots\}$ .

$H_1, H_2, \dots\}$ . Then  $M_e$  simulates the learner  $N$  as follows: if  $N$  on text  $T$  converges to  $a$  then  $M_e$  on text  $T$  converges to the least  $b$  such that  $H_b = L_a$ .

From the definition of uniform learning, we can easily obtain the following useful lemma.

**Lemma 4.** *Let  $\mathcal{L}$  be a uniformly  $I$  learnable indexed family. If  $\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2, \dots$  is a recursive enumeration of indexed family hypothesis spaces for  $\mathcal{L}$ , then there exists a recursive enumeration of learners  $M_0, M_1, M_2, \dots$  such that  $M_n$   $I$  learns  $\mathcal{L}$  with respect to  $\mathcal{H}_n$ .*

We will often make use of the following simple set in our proofs.

**Definition 5.** Define  $S = \cup_{n=0,1,2,\dots} J_n$ , where  $J_n$  contains for each  $e < n$  the first element, if any, of  $W_e$  enumerated from  $I_n = \{2^n - 1, 2^n, 2^n + 1, \dots, 2^{n+1} - 2\}$ . Then:  $S$  is recursively enumerable;  $S$  intersects with every infinite recursively enumerable set; for every  $n$  there is an  $m$  in  $I_n$  which is not in  $S$ . In other words,  $S$  is a simple set [11]. Let  $S_t$  be the set of elements enumerated into  $S$  within  $t$  steps via some standard procedure. Here we take  $S_0 = \emptyset$ .

In the following sections, without loss of generality we assume for  $i, j < |\{L_0, L_1, L_2, \dots\}|$  that  $L_i = L_j$  implies  $i = j$ .

## 2 Finite Learning

Finite learning or one-shot learning requires the learner to make a correct guess using only finite amount of information. So it is not a surprise that this criterion turns out to be very restrictive for prescribed and uniform learning, as shown below. The following theorem gives some characterization results, and separates various notions of finite learning. It can be shown that class comprising finitely learnable classes are also exactly finitely learnable [13].

**Theorem 6.** *Let  $\{L_0, L_1, L_2, \dots\}$  be exactly finitely learnable.*

- (a)  $\{L_0, L_1, L_2, \dots\}$  is not uniformly finitely learnable.
- (b)  $\{L_0, L_1, L_2, \dots\}$  is class-preserving-uniformly finitely learnable.
- (c)  $\{L_0, L_1, L_2, \dots\}$  is prescribed finitely learnable iff the class is finite and for all  $i, j < |\{L_0, L_1, L_2, \dots\}|$ , either  $i = j$  or  $L_i \not\subseteq L_j$ .

Hence, the class  $\{\{0\}, \{1\}, \{2\}, \dots\}$  is exactly finitely learnable but not prescribed finitely learnable; furthermore, a class is prescribed finitely learnable iff it is finitely learnable and finite.

## 3 Conservative Learning

Conservative learning is non-trivial: there is an infinite indexed family which is uniformly conservatively learnable. This is shown in the next example.

**Example 7.** *Let  $L_a = \mathbb{N} - \{a\}$  for all  $a \in \mathbb{N}$ . Then  $\{L_0, L_1, L_2, \dots\}$  is uniformly conservatively learnable.*

The class used in above example consists of co-finite sets only. The next result shows that this is necessary for uniform conservative learning.

**Theorem 8.** *If  $\{L_0, L_1, L_2, \dots\}$  is uniformly conservatively learnable then every set  $L_a$  is cofinite. Moreover, there is a recursive function  $r$  bounding the non-elements of  $L_a$  for all  $a < |\{L_0, L_1, L_2, \dots\}|$ .*

**Proof.** Let  $S$  be as in Definition 5. Furthermore, let  $G_a = L_a$  if  $a < |\{L_0, L_1, L_2, \dots\}|$  and  $G_a = \mathbb{N}$  otherwise.

We define a sequence of hypothesis spaces  $\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2, \dots$ , where for each  $n \in \mathbb{N}$  the space  $\mathcal{H}_n = \{H_0^n, H_1^n, H_2^n, \dots\}$  is defined as follows:

$$H_{\langle i,j \rangle}^n = \begin{cases} G_i, & \text{if } j \notin S \text{ and } j > n; \\ G_i \cup \{t+1, t+2, t+3, \dots\}, & \text{if } j \in S_{t+1} - S_t \text{ and } j > n; \\ \mathbb{N}, & j \leq n. \end{cases}$$

$\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2, \dots$  is a recursive enumeration of indexed families. Since  $\{L_0, L_1, L_2, \dots\}$  is uniformly conservatively learnable, there exists a recursive enumeration of learners  $M_0, M_1, M_2, \dots$  such that for all  $n$ ,  $M_n$  conservatively learns  $\{L_0, L_1, L_2, \dots\}$  with respect to  $\mathcal{H}_n$ .

For all  $a < |\{L_0, L_1, L_2, \dots\}|$  and  $n \in \mathbb{N}$ , let  $e = \langle v(a, n), w(a, n) \rangle$  be the first number found (in some dovetailing search) such that  $M_n$  outputs  $e$  on the canonical text  $T_a$  of  $L_a$  and one of the following conditions hold:

- (a)  $w(a, n) \in S$  and  $L_a \subseteq H_{\langle v(a, n), w(a, n) \rangle}^n$  (note that this can be verified by finding a  $t$  with  $w(a, n) \in S_t$  and checking  $L_a(x) \leq H_{\langle v(a, n), w(a, n) \rangle}^n(x)$  for all  $x \leq t$ );
- (b)  $v(a, n) = a$ ;
- (c)  $w(a, n) \leq n$ .

First note that for every  $a < |\{L_0, L_1, L_2, \dots\}|$  there is an  $n$  such that either  $w(a, n) \leq n$  or  $w(a, n) \in S$ : Otherwise the set  $\{w(a, n) : n \in \mathbb{N}\}$  would be an infinite r.e. set disjoint to  $S$  which does not exist as  $S$  is simple. Hence there is a recursive function  $u$  which searches for this  $n$ ; that is,  $w(a, u(a)) \leq u(a) \vee w(a, u(a)) \in S$  for all  $a < |\{L_0, L_1, L_2, \dots\}|$ .

It is easy to see that there is a further recursive function  $r$  such that either  $r(a) = 0 \wedge w(a, u(a)) \leq u(a)$  or  $r(a) > 0 \wedge w(a, u(a)) \in S_{r(a)}$ . Note that  $L_a \subseteq H_{\langle v(a, u(a)), w(a, u(a)) \rangle}^{u(a)}$  and  $\{r(a), r(a)+1, r(a)+2, \dots\} \subseteq H_{\langle v(a, u(a)), w(a, u(a)) \rangle}^{u(a)}$ . Since each  $M_n$  is conservative, it follows that  $L_a = H_{\langle v(a, u(a)), w(a, u(a)) \rangle}^{u(a)}$  and thus  $\mathbb{N} - L_a$  contains only elements below  $r(a)$  for all  $a < |\{L_0, L_1, L_2, \dots\}|$ .  $\square$

For prescribed conservative learning, we have a less stringent necessary condition as compared to uniform conservative learning.

**Theorem 9.** *If  $\{L_0, L_1, L_2, \dots\}$  is prescribed conservatively learnable then almost every set in  $\{L_0, L_1, L_2, \dots\}$  is cofinite. Moreover, there is a recursive function  $r$  bounding the non-elements of the cofinite  $L_a$ .*

**Proof.** Let  $S$  and  $I_n$  be as in Definition 5. Define a hypothesis space  $\{H_0, H_1, H_2, \dots\}$  as follows. For  $m \in \mathbb{N}$ , define  $H_m$  as follows: suppose  $n$  is such that  $m \in I_n$ ; then let

$$H_m = \begin{cases} L_n, & \text{if } m \notin S; \\ L_n \cup \{m+t, m+t+1, \dots\}, & \text{if } m \in S_{t+1} - S_t \text{ for some } t \in \mathbb{N}. \end{cases}$$

$\{H_0, H_1, H_2, \dots\}$  is an indexed family which is a superclass of  $\{L_0, L_1, L_2, \dots\}$ . Let  $M$  be a learner for  $\{L_0, L_1, L_2, \dots\}$  with respect to  $\{H_0, H_1, H_2, \dots\}$ . Let  $m_n$  be the first index output by  $M$  on the canonical text  $T_n$  for  $L_n$  such that  $m_n \in I_n$  ( $m_n$  may not always be defined). Then the set  $\{m_n : n \in \mathbb{N} \text{ and } m_n \text{ exists}\}$  is recursively enumerable. Let  $e$  be an index for this set. If there are infinitely many coinfinite sets in  $\{L_0, L_1, L_2, \dots\}$ , then there is an  $a > e$  such that  $L_a$  is coinfinite. Since  $L_a$  is coinfinite,  $m_a$  exists (as only indices in  $I_n$  can be indices for coinfinite  $L_n$ ). Furthermore,  $m_a \in S$  because  $m_a$  is the only element in  $W_e \cap I_a$ . But this implies  $H_{m_a} \supset L_a$ . Hence  $M$  cannot be conservative. The function  $r$  can be found by techniques similar to those in the proof of Theorem 8.  $\square$

The above is not a characterization as the class of all cofinite sets is not learnable in the limit. The next example shows that there is also a learnable class of cofinite sets which is not prescribed conservatively learnable; this class is still class-comprising conservative learnable.

**Example 10.** Let  $\{L_0, L_1, L_2, \dots\}$  contain all sets of the form  $\mathbb{N} - \{a\}$  and all sets of the form  $\mathbb{N} - \{a, b\}$  where  $a < b$ ,  $a \in \mathbb{K} - \mathbb{K}_b$ . Then  $\{L_0, L_1, L_2, \dots\}$  is class-preservingly conservatively learnable but not prescribed conservatively learnable.

The following theorem shows that class-preserving-uniformly conservative learnability and prescribed conservative learnability are not comparable.

**Theorem 11.** (a) There exists  $\{L_0, L_1, L_2, \dots\}$  which is class-preserving-uniformly conservatively learnable, but not prescribed conservatively learnable.

(b) There exists  $\{L_0, L_1, L_2, \dots\}$  which is prescribed conservatively learnable but not class-preserving-uniformly conservatively learnable.

## 4 Non U-Shaped Learning

Every conservative learner is clearly non U-shaped. Furthermore, one can modify a conservative learner to be decisive by only changing to a new hypothesis if it is consistent with the input.

The following theorem thus shows that non U-shaped learning is equivalent to conservative learning in the case of exact, class-preserving and class-preserving-uniform learning.

**Theorem 12.** Assume that  $\{L_0, L_1, L_2, \dots\}$  is class-preserving-uniformly non U-shaped learnable. Then  $\{L_0, L_1, L_2, \dots\}$  is already class-preserving-uniformly conservatively learnable by the same learner. The same applies for exact and class-preserving learning.

**Proof.** We show that if  $M$  non U-shaped learns  $\{L_0, L_1, L_2, \dots\}$  with respect to a class-preserving hypothesis space  $\{H_0, H_1, H_2, \dots\}$ , then  $M$  conservatively learns  $\{L_0, L_1, L_2, \dots\}$  with respect to  $\{H_0, H_1, H_2, \dots\}$ . Assume  $M$  is not conservative, then there exist  $\tau, \sigma$  such that  $H_{M(\tau)} \neq H_{M(\tau\sigma)}$ , but  $\text{content}(\tau\sigma) \subseteq H_{M(\tau)}$ . Since  $\{H_0, H_1, H_2, \dots\}$  is class-preserving, there exists an  $n$  such that  $L_n = H_{M(\tau)}$ . Let  $T$  be a text for  $L_n$ , then  $\tau\sigma T$  is a text for  $L_n$ . However,  $M$  is not non U-shaped on  $\tau\sigma T$ , as it first outputs a correct hypothesis  $H_{M(\tau)} = L_n$  and then abandons it. The same argument applies for an exact hypothesis space.  $\square$

However, for class-comprising learning, non U-shaped learning is more powerful than conservative learning, as shown by the following theorem.

**Theorem 13.** *Assume that  $\{L_0, L_1, L_2, \dots\}$  contains all sets  $\{x, x+1, x+2, \dots\}$  and all finite sets  $D$  such that there is an  $s$  with  $\min(D) \in \mathbb{K}_{s+1} - \mathbb{K}_s$  and  $0 < |D| < s$ . Then  $\{L_0, L_1, L_2, \dots\}$  has a non U-shaped class-comprising learner but not a conservative class-comprising learner.*

The following theorem gives a sufficient condition for uniform non U-shaped learnability. Furthermore, this condition helps us to separate uniform non U-shaped learnability from prescribed conservative learnability.

**Theorem 14.** *If the class  $\{L_0, L_1, L_2, \dots\}$  is exactly finitely learnable then  $\{L_0, L_1, L_2, \dots\}$  is uniformly non U-shaped learnable. In particular, there are classes which are uniformly non U-shaped learnable but not prescribed conservatively learnable.*

**Proof.** Let  $M$  be an exact finite learner for  $\{L_0, L_1, L_2, \dots\}$ , we define a recursive enumeration of non U-shaped learners  $M_0, M_1, M_2, \dots$  which uniformly learn  $\{L_0, L_1, L_2, \dots\}$ . For each  $i \in \mathbb{N}$ ,  $M_i(T[t])$  is defined as follows: if  $M(T[t]) = ?$ , then output  $?$ ; if  $M(T[t]) = e$ , then for each  $j \leq t$ , define  $r_j = \min\{x : x > t \text{ or } L_e(x) \neq \varphi_i(j, x)\}$ . Output the minimal  $j$  which maximizes  $r_j$ . It can be easily verified that the above learners witness that  $\{L_0, L_1, L_2, \dots\}$  is uniformly non U-shaped learnable.

Take any exactly finitely learnable language collection with infinitely many coinfinite languages, then it is uniformly non U-shaped learnable but not prescribed conservatively learnable. An example is the language collection  $\{L_0, L_1, L_2, \dots\}$  where  $L_n = \{0, 2, 4, 6, \dots\} \cup \{2n + 1\}$ .  $\square$

With the following result we can see that non U-shaped learning and decisive learning are equivalent for prescribed learning and uniform learning.

**Theorem 15.** *If  $\{L_0, L_1, L_2, \dots\}$  is prescribed non U-shaped learnable then  $\{L_0, L_1, L_2, \dots\}$  is also prescribed decisively learnable. If  $\{L_0, L_1, L_2, \dots\}$  is uniformly non U-shaped learnable then  $\{L_0, L_1, L_2, \dots\}$  is also uniformly decisively learnable.*

**Proof.** It suffices to show that if  $M$  non U-shaped learns  $\{L_0, L_1, L_2, \dots\}$  with respect to a given hypothesis space  $\{H_0, H_1, H_2, \dots\}$ , then we can effectively build another learner  $M'$  which decisively learns  $\{L_0, L_1, L_2, \dots\}$  with respect to  $\{H_0, H_1, H_2, \dots\}$ . The desired  $M'$  can be defined as follows. Given a text  $T$ , let  $M'(T[0]) = M(T[0])$ . For  $t > 0$ ,  $M'(T[t]) = M(T[t])$  if for all  $t' < t$ , for some  $x \leq t$ ,  $H_{M(T[t])}(x) \neq H_{M'(T[t'])}(x)$ ; and  $M'(T[t]) = M'(T[t-1])$  otherwise. It can be easily verified that  $M'$  is decisively learning  $\{L_0, L_1, L_2, \dots\}$  using  $\{H_0, H_1, H_2, \dots\}$ .  $\square$

As in the case of conservative learning, class-preserving-uniform non U-shaped learnability and prescribed non U-shaped learnability are not comparable as well.

**Theorem 16.** (a) *There exists an  $\{L_0, L_1, L_2, \dots\}$  which is class-preserving-uniformly non U-shaped learnable but not prescribed non U-shaped learnable.*

(b) *There exists an  $\{L_0, L_1, L_2, \dots\}$  which is prescribed non U-shaped learnable but not class-preserving-uniformly non U-shaped learnable.*

## 5 Monotonic Learning

The prescribed and uniform versions of strong-monotonic and monotonic learning are very restrictive.

**Theorem 17.** (a)  *$\{L_0, L_1, L_2, \dots\}$  is prescribed strong-monotonically learnable iff  $\{L_0, L_1, L_2, \dots\}$  is finite.*

(b)  *$\{L_0, L_1, L_2, \dots\}$  cannot be uniformly strong-monotonically learnable.*

As in the case of uniform conservative learning, there also exists an infinite class which is uniformly monotonically learnable.

**Example 18.** Let  $L_a = \{a\}$ . Then  $\{L_0, L_1, L_2, \dots\}$  is an infinite class which is uniformly monotonically learnable.

The following result shows that in fact it is necessary for a class to contain only finite sets in order to be uniformly monotonically learnable.

**Theorem 19.** *If  $\{L_0, L_1, L_2, \dots\}$  is uniformly monotonically learnable, then  $\{L_0, L_1, L_2, \dots\}$  contains only finite sets.*

For prescribed monotonic learning, finitely many sets in the language class can violate the above necessary condition for uniform monotonic learning.

**Theorem 20.** *If  $\{L_0, L_1, L_2, \dots\}$  is prescribed monotonically learnable, then  $\{L_0, L_1, L_2, \dots\}$  contains only finitely many infinite sets.*

**Theorem 21.** (a) *There exists a class  $\{L_0, L_1, L_2, \dots\}$  which is class-preserving-uniformly strong-monotonically learnable but not prescribed monotonically learnable.*

(b) *There exists a class  $\{L_0, L_1, L_2, \dots\}$  which is prescribed monotonically learnable but not class-preserving-uniformly monotonically learnable.*

- (c) *Every prescribed strong-monotonically learnable class is also class-preserving-uniformly strong-monotonically learnable.*

**Acknowledgements:** We thank the anonymous referees for helpful comments.

## References

1. Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.
2. Lenore Blum and Manuel Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.
3. Ganesh Baliga, John Case, Wolfgang Merkle, Frank Stephan and Rolf Wiehagen. When unlearning helps. *Information and Computation*, to appear.
4. E. Mark Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
5. Klaus-Peter Jantke. Monotonic and Non-monotonic Inductive Inference. *New Generation Computing*, 8:349–360, 1991.
6. Sanjay Jain, Frank Stephan and Nan Ye. Prescribed Learning of R.E. Classes. In M. Hutter, R. Servedio and E. Takimoto, editors, *Algorithmic Learning Theory, 18th International Conference, ALT' 07*, Sendai, Japan, October 2007, pages 64–78. Lecture Notes in Artificial Intelligence 4754. Springer Verlag, 2007.
7. Steffen Lange. *Algorithmic Learning of Recursive Languages*. Habilitationsschrift, der Fakultät für Mathematik und Informatik der Universität Leipzig eingereichte, Mensch und Buch Verlag, Berlin, 2000.
8. Steffen Lange and Thomas Zeugmann. Language learning in dependence on the space of hypotheses. *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, Santa Cruz, California, United States, pages 127–136, 1993.
9. Steffen Lange, Thomas Zeugmann and Shyam Kapur. Monotonic and dual monotonic language learning. *Theoretical Computer Science*, 155:365–410, 1996.
10. Daniel Osherson, Michael Stob and Scott Weinstein. *Systems That Learn, An Introduction to Learning Theory for Cognitive and Computer Scientists*. Bradford — The MIT Press, Cambridge, Massachusetts, 1986.
11. Emil Post. Recursively enumerable sets of positive integers and their decision problems, *Bulletin of the American Mathematical Society*, 50:284–316, 1944.
12. Thomas Zeugmann. *Algorithmisches Lernen von Funktionen und Sprachen*. Habilitationsschrift, Technische Hochschule Darmstadt, 1993.
13. Thomas Zeugmann and Steffen Lange. A guided tour across the boundaries of learning recursive languages. *Algorithmic Learning for Knowledge-Based Systems*, final report on research project Gosler, edited by Klaus P. Jantke and Steffen Lange, *Springer Lecture Notes in Artificial Intelligence* 961:193–262, 1995.
14. Thomas Zeugmann, Steffen Lange and Shyam Kapur. Characterizations of monotonic and dual monotonic language learning, *Information and Computation*, 120:155–173, 1995.
15. Sandra Zilles. Separation of uniform learning classes. *Theoretical Computer Science*, 313:229–265, 2004.
16. Sandra Zilles. Increasing the power of uniform inductive learners. *Journal of Computer and System Sciences*, 70:510–538, 2005.

# Lower Bounds for Syntactically Multilinear Algebraic Branching Programs

Maurice J. Jansen

Centre for Theory in Natural Science  
Department of Computer Science, University of Aarhus  
IT-Parken, Aabogade 34, DK-8200 Aarhus N, Denmark.  
`mjjansen@daimi.au.dk`

**Abstract.** It is shown that any weakly-skew circuit can be converted into a skew circuit with constant factor overhead, while preserving either syntactic or semantic multilinearity. This leads to considering syntactically multilinear algebraic branching programs (ABPs), which are defined by a natural read-once property. A  $2^{n/4}$  size lower bound is proven for ordered syntactically multilinear ABPs computing an explicitly constructed multilinear polynomial in  $2n$  variables. Without the ordering restriction a lower bound of level  $\Omega(n^{3/2}/\log n)$  is observed, by considering a generalization of a hypercube covering problem by Galvin [1].

**Key words:** Computational complexity, arithmetical circuits, lower bounds, multilinear polynomials, algebraic branching programs.

## 1 Introduction

It is not known whether polynomial size arithmetical circuits ( $\text{VP}$ ) are computationally more powerful than polynomial size arithmetical formulas ( $\text{VP}_e$ ). For the former, we have a surprising construction by Valiant, Skyum, Berkowitz and Rackoff, which shows that  $\text{VP} = \text{VNC}_2$  [2]. For the latter, we know by a result of Brent that  $\text{VP}_e = \text{VNC}_1$  [3]. Recently, Raz made a breakthrough by showing that polynomial size *multilinear* circuits are strictly more powerful than polynomial size *multilinear* formulas. Raz proved that

**Theorem 1** ([4]).  $\text{s-mlin-VNC}_1 \neq \text{s-mlin-VNC}_2$ .

Here “s-mlin-” denotes *syntactic* multilinearity. Technically, multilinearity comes in two flavors: *syntactic* and *semantic* (See Section 2). For formulas these two notions are equivalent, but this is not known to be true for circuits.

Intermediate between circuits and formulas we have so-called *skew* and *weakly-skew circuits* (See [5]). Letting  $\text{VP}_s$  and  $\text{VP}_{ws}$  stand for the classes of  $p$ -families of polynomials that have skew circuits or weakly-skew circuits of polynomial size, respectively, Malod an Portier prove that  $\text{VP}_s = \text{VP}_{ws}$ . The situation can be summarized as follows:

**Theorem 2** ([5, 2]).  $\text{VNC}_1 \subseteq \text{VP}_s = \text{VP}_{ws} \subseteq \text{VNC}_2 = \text{VP}$ .

A priori it is not clear whether the equality  $\text{VP}_s = \text{VP}_{ws}$  holds up when passing to the multilinear variants of these classes, as the proofs in [5] appeal to the completeness of the determinant or trace of iterated matrix multiplication for the class  $\text{VP}_{ws}$ . For the determinant, currently no polynomial size multilinear circuits are known. Furthermore, multilinearity is not necessarily preserved under Valiant projections.

### 1.1 Results

It will be demonstrated one can convert any weakly-skew circuit into a skew-circuit with constant factor overhead, while maintaining either syntactic or semantic multilinearity. Further, it will be observed that the conversion without multilinearity restrictions can be done with constant factor overhead as well<sup>1</sup>. Note, that the conversions indicated in [5] suffers an at least cubic blow-up in size, as an appeal is made to either polynomial size skew circuits for the determinant [6], or for the trace of iterated matrix multiplication. As a consequence one obtains that

**Theorem 3.**

$$\text{s-mlin-VNC}_1 \subseteq \text{s-mlin-VP}_s = \text{s-mlin-VP}_{ws} \subseteq \text{s-mlin-VNC}_2 = \text{s-mlin-VP}.$$

In the above, the rightmost equality was proven in [7]. Looking at Theorem 3, the question is raised whether perhaps the techniques used to prove Theorem 1 can be strengthened to show that  $\text{s-mlin-VP}_s \neq \text{s-mlin-VNC}_2$ , or that we can prove  $\text{s-mlin-VNC}_1 \neq \text{s-mlin-VP}_s$ , by showing, in the terminology of [7, 8], some *full rank polynomial* has polynomial size skew circuits.

As a skew circuit can be transformed into an *algebraic branching program* (ABP, See [9]) with relatively little overhead, we turn to algebraic branching programs to investigate the above questions. If the initial skew circuit is syntactically multilinear, this results in an ABP  $B$  which is syntactically multilinear in the following natural sense: on any directed path in  $B$ , any variable can appear at most once. This can be thought of as the algebraic analog of a Boolean read-once branching program. In the latter model we know of tight exponential lower bounds [10]. Also exponential lower bounds are known for ABPs in the non-commutative case [9]. Bryant introduced so-called *ordered binary decision diagrams* (OBDDs), for which he proved exponential lower bounds [11]. These are read-once Boolean branching programs in which variables are restricted to appear in the same order on all directed paths. This restriction can naturally also be considered for ABPs, leading to the following result:

**Theorem 4.** *Let  $X$  be a set of  $2n$  variables. For any field  $F$ , there exists an extension field  $G$  of  $F$  and explicitly constructed multilinear polynomial  $f \in G[X]$ , such that any ordered algebraic branching program over  $X$  and  $G$  computing  $f$  has at least  $2^{n/4}$  nodes.*

---

<sup>1</sup> This observation has simultaneously been made by Kaltofen and Koiran in an unpublished paper, which was unknown to the author at the time of this research. They do not consider the preservation of multilinearity.

Finally, the problem of proving lower bounds for unrestricted ABPs and (unordered) multilinear ABPs is explored. For any fixed constant  $0 < \alpha < 1$ , it will be shown that any unrestricted ABP requires size  $\Omega(n^{3/2}\alpha\sqrt{1-\alpha})$  to compute the elementary symmetric polynomial of degree  $\lfloor\alpha n\rfloor$  in  $n$  variables. Next a relation between proving lower bounds for multilinear ABPs and the generalization of a hypercube covering problem by Galvin will be established [1]. By straightforward counting this yields a lower bound for multilinear ABPs of level  $\Omega(\frac{n^{3/2}}{\log n})$ , for computing any full rank polynomial. Potentially however, this technique yields up to quadratic lower bounds, provided linear lower bounds can be proven for certain generalizations of Galvin's Problem.

## 2 Preliminaries

For non-negative integer  $n$ ,  $[n]$  denotes the set  $\{1, 2, \dots, n\}$ . Let  $F$  be a field and  $X = \{x_1, x_2, \dots, x_n\}$  be a set of variables. An arithmetical circuit  $\Phi$  over  $F$  and  $X$  is a directed acyclic graph with nodes of in-degree zero or two. Nodes with in-degree zero are called *inputs* and are labeled by variables or field elements. Nodes with in-degree two are called *gates* and have labels  $\in \{\times, +\}$ . For each gate  $g$  in  $\Phi$ , one has associated the *polynomial computed by  $g$* , denoted by  $\Phi_g$ , which is defined in the obvious manner. Also let  $\Phi_g$  stand for the subcircuit rooted at gate  $g$ . It will be made clear from the context which meaning is intended. Denote by  $X_g$  the set of variables used by the subcircuit  $\Phi_g$ . The size of  $\Phi$ , denoted by  $|\Phi|$ , is taken to be the number gates. If the underlying graph of  $\Phi$  is a tree,  $\Phi$  is called a *formula*. For a polynomial  $f$ ,  $C(f)$  and  $L(f)$  denote the smallest size of a circuit or formula, respectively, computing  $f$ .

An arithmetical circuit  $\Phi$  is called *weakly-skew* if at every multiplication gate  $g$  with inputs  $g_1$  and  $g_2$ , one of  $\Phi_{g_1}$  and  $\Phi_{g_2}$  is disjoint from the rest of  $\Phi$ .  $\Phi$  is called *skew* if for each multiplication gate at least one  $g_1$  and  $g_2$  is an input gate (See [5]). For a polynomial  $f$ ,  $C_{ws}(f)$  and  $C_s(f)$  denote the smallest size of a weakly-skew or skew circuit computing  $f$ , respectively.

A polynomial  $f$  is called *multilinear* if for any monomial of  $f$ , every variable has degree at most one. A circuit  $\Phi$  is *semantically multilinear* if every polynomial computed at any gate of  $\Phi$  is multilinear.  $\Phi$  is called *syntactically multilinear* if for each multiplication gate  $g$  with inputs  $g_1$  and  $g_2$ ,  $X_{g_1}$  and  $X_{g_2}$  are disjoint. For a polynomial  $f$ ,  $C_{syn}(f)$  and  $C_{sem}(f)$  denote syntactic and semantic multilinear circuit size, respectively. Similarly,  $L_{syn}(f)$  and  $L_{sem}(f)$  denote syntactic and semantic multilinear formula size. These definitions will be combined in the obvious manner. For example,  $C_{syn,ws}(f)$  denotes the smallest size of a syntactically multilinear weakly-skew circuit computing  $f$ . Standard notation for arithmetical circuit classes will be followed (See e.g. [12]). For any class  $\mathcal{C} \in \{VNC_1, VNC_2, VP_s, VP_{ws}, VP\}$ , let  $syn\text{-mlin-}\mathcal{C}$  and  $mlin\text{-}\mathcal{C}$  stand for the class of  $p$ -families of polynomials that have  $\mathcal{C}$ -circuits which additionally are required to be syntactically or semantically multilinear, respectively.

**Definition 1 (See [9]).** *An algebraic branching program (ABP) over a field  $F$  and a set of variables  $X$  is a 4-tuple  $B = (G, w, s, t)$ , where  $G = (V, E)$  is a*

*directed acyclic graph for which  $V$  can be partitioned into levels  $L_0, L_1, \dots, L_d$ , where  $L_0 = \{s\}$  and  $L_d = \{t\}$ . Vertices  $s$  and  $t$  are called the source and sink of  $B$ , respectively. Edges may only go between consecutive levels  $L_i$  and  $L_{i+1}$ . The weight function  $w : E \rightarrow F[X]$  assigns homogeneous linear forms to the edges of  $G$ . For a path  $p$  in  $G$ , we extend the weight function by  $w(p) = \prod_{e \in p} w(e)$ . For  $i, j \in V$ , let  $P_{i,j}$  be the collection of all path in  $G$  from  $i$  to  $j$ . The program  $B$  computes the polynomial  $\sum_{p \in P_{s,t}} w(p)$ . The size of  $B$  is taken to be  $|V|$ .*

For a linear form  $L(x) = \sum_{i=1}^n c_i x_i$ , define  $\text{coef}[L, x_i] = c_i$ . An ABP  $B$  is called *syntactically multilinear* if for any directed path  $p$  in  $B$ , for all  $i$ , there is at most one edge  $e$  on  $p$  with  $\text{coef}[w(e), x_i] \neq 0$ .  $B(f)$  denotes the smallest size of an ABP computing  $f$ , and  $B_{syn}(f)$  denotes such size with the additional restriction of syntactic multilinearity.

### 3 Circuit Transformations

It is convenient to work with the following data structure: a *skew schedule* is a directed acyclic graph  $G$  with weights on the edges  $\in F \cup X$ , where the out-degree of a vertex is either zero, one or two, and where for any vertex  $v$  with distinct edges  $e_1 = (v, w)$  and  $e_2 = (v, u)$ , the labels of  $e_1$  and  $e_2$  equal 1. For a directed acyclic graph  $G$  with node  $s \in V[G]$ , a path  $p$  in  $G$  is called a *maximal path with starting point  $s$* , if the first vertex of  $p$  is  $s$  and the last vertex of  $p$  has no outgoing edges.

**Lemma 1.** *For any polynomial  $f$ ,  $C_{syn,s}(f) \leq 5C_{syn,ws}(f)$ .*

*Proof.* First process  $\Phi$  so that any addition gate has its input coming in from different gates by inserting dummy addition gates. This at most doubles the size. Let  $e'$  be the new size. Let  $g_1, g_2, \dots, g_{e'}$  be a topological sort of the gates of  $\Phi$ , where wlog. we assume  $\Phi_{g_{e'}} = f$ , and that  $g_{e'}$  is the only gate with out-degree zero. Let  $g_{-m+1}, g_{-m+2}, \dots, g_0$  be the set of inputs of  $\Phi$ . Sequentially for stages  $k = 1, 2, \dots, e'$ , we construct a skew schedule  $G_k$  from  $G_{k-1}$ . To initialize, let  $G_0$  consists of  $m$  distinct directed edges. For each input  $g$  of  $\Phi$ , we select a unique edge among  $E[G_0]$  and put the label of  $g$  on it. Let  $B$  be the set of vertices in  $G_0$  with out-degree zero. The set  $B$  will remain as a subset of vertices in each  $G_k$ . We will never change the in-degree of vertices in  $B$ . At the beginning of stage  $k$ , the skew schedule  $G_{k-1}$  will satisfy:

1. Each node  $g = g_{k'}$  with  $k' < k$  will correspond one-to-one with a vertex  $v_g \in V[G_{k-1}] \setminus B$ .
2. Let  $\mathcal{G}_{k-1}$  be the set of nodes  $g_{k'}$  with  $k' < k$ , that are used by gates  $g_{k''}$  for some  $k'' \geq k$ . For any  $g \in \mathcal{G}_{k-1}$ ,  $\sum_{p \in \mathcal{P}} \prod_{e \in p} w(e) = \Phi_g$ , where  $\mathcal{P}$  is the collection of all maximal paths with starting point  $v_g$  in  $G_{k-1}$ ,
3. On any directed path in  $G_{k-1}$  no variable appears more than once,
4. For any node  $g = g_{k'}$  with  $k' < k$ , the set of nodes not in  $B$  reachable in  $G_{k-1}$  from  $v_g$ , is precisely  $\{v_{g'} : g' \in \Phi_g\}$ .

At the beginning of stage  $k = 1$ , we have that  $\mathcal{G}_0$  is the set of all input gates. For each input gate  $g$ ,  $v_g$  is defined to be the starting vertex of the unique edge we have selected for it. Properties (1)-(4) can now be verified to hold. At stage  $k$  we do the following:

**Case I:**  $g_k = +(g_i, g_j)$ . We have that  $g_i, g_j \in \mathcal{G}_{k-1}$ . We construct  $G_k$  from  $G_{k-1}$  by adding one new vertex  $w$  with edges of weight 1 from  $w$  to  $v_{g_i}$  and  $v_{g_j}$ . No parallel edges are created since  $i \neq j$ . Let us verify the needed properties. Property (3) is clear. It is also clear that if we let  $\mathcal{P}$  be the collection of all maximal paths starting in  $w$  that  $\sum_{p \in \mathcal{P}} \prod_{e \in p} w(e) = \Phi_{g_k}$ . If we are at the last iteration, i.e.  $k = e'$ , then this is all we are required to verify. Otherwise,  $g_k$  will be used later on, i.e.  $g_k \in \mathcal{G}_k$ . Observe that  $\mathcal{G}_k = \mathcal{G}_{k-1} \cup \{g_k\} - S$ , where  $S \subseteq \{g_i, g_j\}$ . We define  $v_{g_k} = w$ . By our above observation for the vertex  $w$ , and the fact that we do not modify connectivity for the other vertices, Property (2) holds. Property (4) is clear.

**Case II:**  $g_k = \times(g_i, g_j)$ . Wlog. assume  $\Phi_{g_j}$  is disjoint from the rest of  $\Phi$ . We have that  $g_i, g_j \in \mathcal{G}_{k-1}$ . For  $s \in \{i, j\}$ , let  $W_s$  be the set of vertices in  $G_{k-1}$  reachable from  $v_{g_s}$ .  $W_i$  and  $W_j$  are disjoint. Namely, suppose  $w \in W_i \cap W_j$ . If  $w \notin B$  then Property (4) implies there exists a shared node in  $\Phi_{g_i}$  and  $\Phi_{g_j}$ , which is a contradiction. In case  $w \in B$ , then since we do not add edges into  $w$ , we have a vertex  $w' = v_{g'}$  for some input gate  $g'$  with  $w' \in W_i \cap W_j$ . Hence we again have a contradiction.

Let  $E \subseteq W_j$  be the set of vertices reachable from  $v_{g_j}$  with out-degree zero. We define  $G_k$  to be the graph  $G_{k-1}$  modified by adding an edge  $(v, v_{g_i})$  of weight 1 for each  $v \in E$ . We add<sup>2</sup> a new vertex  $w$  and edge  $(w, v_{g_j})$  with weight 1, and let  $v_{g_k} = w$ . Since  $W_i \cap W_j = \emptyset$ , no vertex from  $E$  is reachable from  $v_{g_i}$ . Hence  $G_k$  is an acyclic graph. Observe  $G_k$  is a skew schedule. We will now verify Properties (1)-(4).

Let  $\mathcal{P}$  be the set of maximal paths starting in  $v_{g_k}$  in  $G_k$ . Let  $\mathcal{P}_s$  be the set of maximal paths in  $G_{k-1}$  starting in  $v_{g_s}$ , for  $s \in \{i, j\}$ . For  $p \in \mathcal{P}_i$  and  $q \in \mathcal{P}_j$ , let  $q \# p$  denote the path in  $G_k$  that is  $(v_{g_k}, v_{g_j})$ , followed by  $q$ , followed by the edge with weight 1 into  $v_{g_i}$ , followed by  $p$ . We have that  $\mathcal{P} = \{q \# p : q \in \mathcal{P}_j, p \in \mathcal{P}_i\}$ . This means  $\sum_{r \in \mathcal{P}} \prod_{e \in r} w(e) = \sum_{p \in \mathcal{P}_i} \prod_{e \in p} w(e) \cdot \sum_{q \in \mathcal{P}_j} \prod_{e \in q} w(e) = \Phi_{g_i} \cdot \Phi_{g_j} = \Phi_{g_k}$ . In case this was the last iteration, i.e.  $k = e'$ , this is all we need together with Property (3) to be verified below. Otherwise, since  $g_k$  will be used later again,  $g_k \in \mathcal{G}_k$ . Observe that  $\mathcal{G}_k = \mathcal{G}_{k-1} - S \cup \{g_k\}$ , where  $S$  is the set of nodes in  $\Phi_{g_j}$ .

By what we observed above, Property (2) holds for  $g_k$ . For  $g \neq g_k$  in  $\mathcal{G}_k$ , the only way Property (2) can be disturbed is if some vertex  $w \in E$  is reachable from  $v_g$  in  $G_{k-1}$ . This means some  $w' \notin B$  is reachable from both  $v_g$  and  $v_{g_j}$  in  $G_{k-1}$ , but then  $\Phi_g$  and  $\Phi_{g_j}$  share a vertex by ‘previous’ Property (4). Since  $\Phi$  is weakly-skew,  $g$  must be a node in  $\Phi_{g_j}$ , but that is a contradiction since  $g \in \mathcal{G}_k$ .

To check Property (3), we note that the only edges with variables are of the form  $(v, b)$  with  $b \in B$  and  $v = v_g$ , for some input  $g \in \Phi$ . Property (3) can be violated only, if in  $G_{k-1}$  we have for such  $(v, b)$  that some vertex in  $E$  can be

---

<sup>2</sup> This is not strictly necessary, but we do so to simplify the proof.

reached from  $b$ , and that in  $G_{k-1}$  there exists a path starting in  $v_{g_i}$  going over an edge  $(v', b')$  with  $b' \in B$  and  $v' = v_{g'}$ , for some input gate  $g'$ , that has the same variable label as  $(v, b)$ . This means that  $g' \in \Phi_{g_i}$ . Similar as above, by Property (4), it must be that  $g \in \Phi_{v_{g_j}}$ . By syntactic multilinearity, we conclude the labels of  $(v, b)$  and  $(v', b')$  must be different. Property (4) clearly holds.

This completes the description the graphs  $G_1, G_2, \dots, G_{e'}$ .  $G_{e'}$  is a skew schedule of size at most  $2m + e' \leq 5e'$ . We can evaluate it node by node in a bottom-up fashion. This yields a syntactically multilinear skew circuit computing  $f$  of size at most  $5e' \leq 10e$  gates. To optimize the constant to be 5 instead of 10, we observe adding dummy addition gates is not required.  $\square$

**Lemma 2.** *For any polynomial  $f$ ,  $C_{sem,s}(f) \leq 5C_{sem,ws}(f)$ .*

*Proof.* Modify the multiplication case in the proof of Lemma 1 as follows. For each variable  $x_i$  appearing in the polynomial  $\Phi_{g_i}$ , replace any edge weight  $x_i$  in the induced subgraph  $G_{k-1}[W_j]$  by zero. This does not alter the polynomial represented at vertex  $v_{g_j}$ , since it cannot contain variable  $x_i$ . Polynomials represented at other vertices in  $G_k[W_j]$  can have changed, but cannot be used at later stages. The substitution has forced all these polynomials to be multilinear.  $\square$

Removing Property (3) in the proof of Lemma 1 immediately yields a proof that for any polynomial  $f$ ,  $C_s(f) \leq 5C_{ws}(f)$ . We put together the basic facts about the measures that are being considered.

**Lemma 3.** *For a homogeneous polynomial of degree  $d$ ,*

1.  $C(f) \leq C_{ws}(f) \leq L(f)$  and  $C_{syn}(f) \leq C_{syn,ws}(f) \leq L_{syn}(f)$ .
2.  $C_{ws}(f) \leq C_s(f) \leq 5C_{ws}(f)$  and  $C_{syn,ws}(f) \leq C_{syn,s}(f) \leq 5C_{syn,ws}(f)$ .
3.  $B(f) \leq d \cdot (4C_s(f) + 2)$  and  $B_{syn}(f) \leq d \cdot (4C_{syn,s}(f) + 2)$ .
4.  $\alpha\sqrt{C_s(f)/n} \leq B(f)$  and  $\alpha\sqrt{C_{syn,s}(f)/n} \leq B_{syn}(f)$ , for some constant  $\alpha > 0$ .

## 4 Ordered ABPs

Let  $B = (G, w, s, t)$  be an ABP over a field  $F$  and set of variables  $X = \{x_1, x_2, \dots, x_n\}$ . Say a directed path  $p$  from  $s$  to  $t$  respects a permutation  $\pi : [n] \rightarrow [n]$ , if whenever an edge  $e_1$  appears before an edge  $e_2$  on  $p$  and  $\text{coeff}[w(e_1), x_{\pi(i)}] \neq 0$  and  $\text{coeff}[w(e_2), x_{\pi(j)}] \neq 0$ , one has that  $i < j$ .  $B$  is called *ordered*, if there exists a permutation  $\pi$  that is respected by all directed  $s, t$ -paths. For a polynomial  $f$ , we denote ordered ABP size by  $B_{ord}(f)$ . We first observe that lower bounds for non-commutative ABPs of Nisan [9] transfer to the ordered model. This follows by evaluating the ordered ABP over non-commutative variables. For example, one obtains that

**Theorem 5.** *Any ordered algebraic branching program  $B = (G, w, s, t)$  computing the permanent or determinant of an  $n \times n$  matrix of variables has size at least  $2^n$ .*

The above bound for the permanent and determinant is of level  $2^{\Omega(\sqrt{N})}$ , where  $N$  is the number of variables. In [9] a bound of  $2^{\Omega(N)}$  is proven for the non-commutative model, but this is for a polynomial that is not multilinear. In order to obtain a bound of level  $2^{\Omega(N)}$ , we now turn to the aforementioned full rank polynomials [4, 7, 8].

Let  $X = \{x_1, x_2, \dots, x_{2n}\}$ ,  $Y = \{y_1, y_2, \dots, y_n\}$  and  $Z = \{z_1, z_2, \dots, z_n\}$  be sets of variables. Following [4, 7, 8], for a multilinear polynomial  $g \in F[Y, Z]$ , we define the  $2^n \times 2^n$  partial derivatives matrix  $M_g$ , by taking  $M_g(m_1, m_2) =$  coefficient of monomial  $m_1 m_2$  in  $g$ , where  $m_1$  and  $m_2$  range over all multilinear monic monomials in  $Y$  and  $Z$  variables, respectively. A partition of  $X$  into  $Y$  and  $Z$  is any bijection  $A : X \rightarrow Y \cup Z$ . For  $f \in F[X]$  denote by  $f^A$  the polynomial obtained from  $f$  by substitution of  $x_i$  by  $A(x_i)$ , for all  $i \in [2n]$ .  $f$  is of full rank if for every partition  $A$ ,  $\text{rank } M_{f^A} = 2^n$ . For multilinear  $g \in F[Y, Z]$ , let  $Y_g$  and  $Z_g$  be  $Y$  and  $Z$  variables appearing in  $g$ .

**Proposition 1 ([8]).** *Let  $g, g_1, g_2 \in F[Y, Z]$  be multilinear polynomials. Then*

1.  $\text{rank } M_g \leq 2^{\min(|Y_g|, |Z_g|)}$ ,
2.  $\text{rank } M_{g_1+g_2} \leq \text{rank } M_{g_1} + \text{rank } M_{g_2}$ , and
3.  $\text{rank } M_{g_1 \cdot g_2} = \text{rank } M_{g_1} \cdot \text{rank } M_{g_2}$ , provided  $Y_{g_1} \cap Y_{g_2} = \emptyset$  and  $Z_{g_1} \cap Z_{g_2} = \emptyset$ .

We obtain the following lower bound:

**Theorem 6.** *Let  $X$  be a set of  $2n$  variables, and let  $F$  be a field. For any full rank homogeneous polynomial  $f$  of degree  $n$  over  $X$  and  $F$ ,  $B_{\text{ord}}(f) \geq 2^{n/4}$ .*

A proof of above theorem will appear in the full version of the paper. To indicate the idea, suppose that  $B$  respects the permutation  $\pi : [2n] \rightarrow [2n]$ . In this case consider any partition  $A$  that assign all  $n$   $y$ -variables to  $\{x_{\pi(1)}, x_{\pi(2)}, \dots, x_{\pi(n)}\}$  and all  $n$   $z$ -variables to  $\{x_{\pi(n+1)}, x_{\pi(n+2)}, \dots, x_{\pi(2n)}\}$ . Applying this partition to  $B$ , we obtain an ABP computing  $f^A$ . Partition  $A$  has been selected to frustrate “progress” of the rank measure  $\text{rank } M_{g^A}$ , for gates  $g$  of  $B$ . Regardless of that, the final output  $f^A$  must still have  $\text{rank } M_{f^A} = 2^n$ . Using Proposition 1, it is then possible to argue this implies the middle level  $L_{n/2}$  must of size at least  $2^{n/4}$ .

We need to modify the construction of a full rank polynomial from [7] to yield a *homogeneous* full rank polynomial. This can be done, provided we work over a suitable extension field of the underlying field  $F$ .

Let  $\mathcal{W} = \{\omega_{i,l,j}\}_{i,l,j \in [2n]}$  be sets of variables. For each interval  $[i, j] \subseteq 2n$  of even length, we define a polynomial  $f_{i,j} \in F[X, \mathcal{W}]$  inductively as follows: if  $|[i, j]| = 0$ , then define  $f_{i,j} = 1$ . If  $|[i, j]| > 0$ , define  $f_{i,j} = (x_i + x_j)f_{i+1,j-1} + \sum_l \omega_{i,l,j} f_{i,l} f_{l+1,j}$ , where we sum over all  $l$  such that  $|[i, l]|$  is even. Finally,  $f$  is defined to be  $f_{1,2n}$ . It follows inductively that  $f_{i,j}$  is multilinear and homogeneous of degree  $|(i, j)|/2$  in the  $X$ -variables. It can be verified that  $f$  is indeed a full rank polynomial when considered as a polynomial in  $G[X]$ , where  $G = F(\mathcal{W})$  is the field of rational functions over the field  $F$  and variables  $\mathcal{W}$ . Theorem 4 then follows from Theorem 6. More details will appear in the full version.

## 5 Unrestricted and Multilinear ABPs

Consider the following observation by Kristoffer Hansen: if for an ABP  $B$  the number of edges between any two consecutive levels  $L_d$  and  $L_{d+1}$  is at most  $K < n$ , then the polynomial  $f(x_1, x_2, \dots, x_n)$  computed by  $B$  vanishes on a linear space of dimension at least  $n - K$ . Provided this is a contradiction for  $f$ , one concludes that  $\max(|L_d|, |L_{d+1}|) \geq \sqrt{K}$ . For example, for the elementary symmetric polynomial of degree  $d$  in  $n$  variables defined by  $\sum_{S \subset [n], |S|=d} \prod_{i \in S} x_i$ , using Theorem 1.1 in [13], this yields the following theorem:

**Theorem 7.** *Let  $\alpha$  be a constant with  $0 < \alpha < 1$ , and assume  $\alpha n$  is integer. Over fields of characteristic zero, for the elementary symmetric polynomial  $S_n^{\alpha n}$  of degree  $\alpha n$  in  $n$  variables, it holds that  $B(S_n^{\alpha n}) = \Omega(n^{3/2} \alpha \sqrt{1 - \alpha})$ .*

Above argument is limited to yield  $O(n^{3/2})$  lower bounds, for polynomials of degree  $\Theta(n)$ . In order to overcome this, we turn to a hypercube covering problem.

Consider  $H_{2n} = \{-1, 1\}^{2n}$  and  $B_{2n} = \{0, 1\}^{2n}$  over the real numbers. Define inner product  $x \cdot y = \sum_{i=1}^{2n} x_i y_i$ . Let  $\mathbf{1}$  denote the vector  $1^{2n}$ . Let  $H_{2n}^e = \{x \in H_{2n} : x \cdot \mathbf{1} = 0\}$ . Of interest are minimal size coverings of  $H_{2n}^e$  by hyperplanes, where coefficients for the defining equations are taken from particular subsets of  $B_{2n}$ . More precisely, for  $W \subset [2n]$ , define  $B_{2n}^W = \{b \in B_{2n} : \text{wt}(b) \in W\}$ , where  $\text{wt}(b) = b \cdot \mathbf{1}$ . Define

$$Q(n, W, d) = \min\{|E| : E \subseteq B_{2n}^W, (\forall x \in H_{2n}^e), (\exists e \in E), |x \cdot e| \leq d\}.$$

Finding the value of  $Q(n, W, d)$  becomes interesting only for certain weight sets  $W$ . For example, if  $2n \in W$ , then  $E = \{\mathbf{1}\}$  trivially covers all of  $H_{2n}^e$ . The *discrepancy* parameter  $d$  should also be small w.r.t.  $\min(W)$ , e.g. trivially  $Q(n, W, \min(W)) = 1$ . Also, in case  $W$  does not contain an even number and  $d = 0$ , we have that  $Q(n, W, d)$  is ill-defined. In all other cases  $Q(n, W, d)$  is well-defined. Namely, say  $2\ell \in W$ . By taking  $E$  to be all  $2n$  cyclic shifts of  $1^{2\ell} 0^{2n-2\ell}$ , we have that for each  $x \in H_{2n}^e$ , there must exist some  $e \in E$ ,  $x \cdot e = 0$ . Similarly, this set  $E$  works, in case  $W$  contains no even numbers, but  $d \geq 1$ . The crucial question is whether for cases that avoid trivialities, one can do significantly better than  $|E|$  being linear in  $n$ .

The special case of finding  $m(k) := Q(2k, \{2k\}, 0)$  is a problem posed originally by Galvin (See [14]). For an upper bound, note one requires only half of all  $4k$  cyclic shifts, i.e.  $m(k) \leq 2k$ . For odd  $k$ , Frankl and Rödl proved the linear lower bound  $m(k) > \epsilon k$ , for fixed  $\epsilon > 0$  [1]. The proof relies on a strong result in extremal set theory they proved, which resolved a \\$250 problem of Erdős. Later the bound was improved to  $m(k) \geq 2k$ , for odd  $k$  [15].

Consider  $Q(n, [\epsilon_0 n, (1 + \epsilon_1)n], 2 \log n)$ , for fixed  $0 < \epsilon_0 < \epsilon_1 < 1$ . From Theorem 8 below it will follow, that proving an  $\tilde{\Omega}(n)$  lower bounds on this quantity would yield an  $\tilde{\Omega}(n^2)$  multilinear ABP lower bound. In light of the result by Frankl and Rödl such a linear lower bound appears plausible. Also note the linear lower bound by Alon et al. for covering the entire hypercube, in

case the defining equations have coefficients in  $\{-1, 1\}$  instead of in  $\{0, 1\}$  [14]. They define for  $n \equiv d \pmod{2}$ ,  $K(n, d) = \min\{|E| : E \subseteq H_n, (\forall x \in H_n), (\exists e \in E), |x \cdot e| \leq d\}$ , and prove  $K(n, d) = \lceil n/(d+1) \rceil$ .

**Theorem 8.** *Let  $X$  be a set of  $2n$  variables, and let  $F$  be a field. For any full rank homogeneous polynomial  $f$  of degree  $n$  over  $X$  and  $F$ ,  $B_{syn}(f) = \Omega\left(\sum_{r=1}^{n-1} \min(n, Q(n, [r, n+r], 2 \log n))\right)$ .*

*Proof.* Let  $B = (G, w, s, t)$  be a multilinear ABP computing  $f$ . Let  $L_0, L_1, \dots, L_n$  be the levels of  $B$ . For  $v, w \in V[G]$ , let  $f_{v,w}$  denote the polynomial computed by the subprogram of  $B$  with source  $v$  and sink  $w$ . Let  $X_{v,w}$  denote the set of all variables appearing on directed paths from  $v$  to  $w$ . Consider  $r$  such that  $0 < r < n$ . Write  $f = f_{s,t} = \sum_{v \in L_r} f_{s,v} f_{v,t}$ .

By syntactic multilinearity, we have that  $|X_{s,v}| \geq r$  and  $|X_{v,t}| \geq n - r$ . The latter implies  $|X_{s,v}| \leq n + r$ , again by syntactic multilinearity. Let  $\chi(X_{s,v}) \in B_{2n}^{[r,n+r]}$  denote the characteristic vector of  $X_{s,v}$ .

Suppose that  $|L_r| < Q(n, [r, n+r], 2 \log n)$ . Then there exists  $\gamma \in H_{2n}^e$  such that for every  $b \in \{\chi(X_{s,v}) : v \in L_r\}$ ,  $|\gamma \cdot b| > 2 \log n$ . Let  $A : X \rightarrow Y \cup Z$  be any partition that assigns a  $Y$  variable to  $x_i$ , if  $\gamma_i = 1$ , and a  $Z$  variable otherwise, for all  $i \in [2n]$ .

Let  $B'$  be the branching program obtained from  $B$  by substituting according to  $A$ . For nodes  $v$  and  $w$ , we let  $Y_{v,w}$  and  $Z_{v,w}$  denote the sets of  $y$  and  $z$  variables, respectively, appearing on paths from  $v$  to  $w$  in  $B'$ . Then for any  $v \in L_r$ ,  $\min(Y_{s,v}, Z_{s,v}) \leq (Y_{s,v} + Z_{s,v})/2 - \log n \leq |X_{s,v}|/2 - \log n$ . Hence, by Item 1 of Proposition 1, we have that  $\text{rank } M_{f_{s,v}^A} \leq 2^{|X_{s,v}|/2 - \log n}$ . By syntactic multilinearity, none of the variables appearing on paths from  $s$  to  $v$  can appear on paths from  $v$  to  $t$ . So  $|X_{v,t}| \leq 2n - |X_{s,v}|$ . By Item 1 of Proposition 1 we get  $\text{rank } M_{f_{v,t}^A} \leq 2^{n-|X_{s,v}|/2}$ . Using multiplicativity (Proposition 1, Item 3), we conclude  $\text{rank } M_{f_{s,v}^A f_{v,t}^A} \leq 2^{n-\log n}$ , and thus using subadditivity (Proposition 1, Item 2) and since  $f = \sum_{v \in L_r} f_{s,v} f_{v,t}$  that  $\text{rank } M_f^A \leq |L_r| 2^{n-\log n}$ . Since  $f$  is of full rank,  $\text{rank } M_f^A = 2^n$ . We conclude that  $|L_r| \geq n$ .  $\square$

Counting yields that  $Q(n, [r, n+r], d) = \Omega((d+1)^{-1} \sqrt{\frac{r(n-r)}{2n}})$ . Applying Theorem 8, and summing for  $r \in [\epsilon_0 n, \epsilon_1 n]$ , for fixed  $0 < \epsilon_0 < \epsilon_1 < 1$  yields

**Theorem 9.** *Let  $X$  be a set of  $2n$  variables. For any field  $F$ , there exists an extension field  $G$  of  $F$  and explicitly constructed multilinear polynomial  $f \in G[X]$ , such that any multilinear algebraic branching program over  $X$  and  $G$  computing  $f$  has  $\Omega(\frac{n\sqrt{n}}{\log n})$  nodes.*

## 6 Conclusions

It should be noted the ABP-model can be quite powerful. The prime example being that, given two  $n \times n$  matrices  $X$  and  $Y$  of variables, one can compute  $f = \sum_{i,j \in [n]} (XY)_{ij}$  with a syntactically multilinear ABP with  $O(\sqrt{N})$  many

nodes, where  $N = 2n^2$  is the number of input variables. This is an example of a polynomial, for which the multilinear ABP-model is at least “quadratically more efficient” than the syntactic multilinear circuit model. In the latter model, the currently best known lower bound for an explicit function is of level  $\Omega(\frac{n^{4/3}}{\log^2 n})$  [8]. Theorem 8 supplies a lower bound strategy, which yielded an  $\Omega(\frac{n^{3/2}}{\log n})$  lower bound for multilinear ABPs. By resolving a certain generalization of Galvin’s Problem this method can yield an  $\tilde{\Omega}(n^2)$  lower bound for syntactically multilinear ABPs.

*Acknowledgments* I thank Peter Bro Miltersen, Kristoffer Hansen and Oded Lachish for helpful discussions.

## References

1. P. Frankl and V. Rödl. Forbidden intersections. *Trans. Amer. Math. Soc.*, 300(1):259–286, 1987.
2. L. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput.*, 12:641–644, 1983.
3. R. Brent. The parallel evaluation of general arithmetic expressions. *J. Assn. Comp. Mach.*, 21:201–206, 1974.
4. R. Raz. Separation of multilinear circuit and formula size. *Theory of Computing*, 2(6):121–135, 2006.
5. Guillaume Malod and Natacha Portier. Characterizing Valiant’s algebraic complexity classes. In *MFCS*, pages 704–716, 2006.
6. S. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Inf. Proc. Lett.*, 18:147–150, 1984.
7. R. Raz and A. Yehudayoff. Balancing syntactically multilinear arithmetical circuits. *Journal of Computational Complexity* (to appear).
8. R. Raz, A. Shpilka, and A. Yehudayoff. A lower bound for the size of syntactically multilinear arithmetic circuits. In *Proc. 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 438–448, 2007.
9. N. Nisan. Lower bounds for non-commutative computation: extended abstract. In *Proc. 23rd Annual ACM Symposium on the Theory of Computing*, pages 410–418, 1991.
10. A. Andreev, J. Baskov, E. Clementi, and R. Rolim. Small pseudo-random sets yield hard functions: new tight lower bounds for branching programs. In *Proc. 26th Annual International Conference on Automata, Languages, and Programming*, volume 1644 of *Lect. Notes in Comp. Sci.*, pages 179–189, 1999.
11. R.E. Bryant. On the complexity of vlsi implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Trans. Computers*, 40(2):205–213, 1991.
12. P. Bürgisser, M. Clausen, and M.A. Shokrollahi. *Algebraic Complexity Theory*. Springer Verlag, 1997.
13. A. Shpilka. Affine projections of symmetric polynomials. In *Proc. 16th Annual IEEE Conference on Computational Complexity*, pages 160–171, 2001.
14. N. Alon, E. Bergmann, D. Coppersmith, and A. Odlyzko. Balancing sets of vectors, 1988.
15. H. Enomoto, P. Frankl, N. Ito, and K. Nomura. Codes with given distances. *Graphs and Combinatorics*, 3:25–38, 1987.

# The Use of Information Affinity in Possibilistic Decision Tree Learning and Evaluation

Ilyes Jenhani<sup>1</sup>, Salem Benferhat<sup>2</sup>, and Zied Elouedi<sup>1</sup>

<sup>1</sup> LARODEC, Institut Supérieur de Gestion de Tunis, Tunisia.

<sup>2</sup> CRIL, Université d'Artois, Lens, France.

[ilyes.j@lycos.com](mailto:ilyes.j@lycos.com), [benferhat@cril.univ-artois.fr](mailto:benferhat@cril.univ-artois.fr), [zied.elouedi@gmx.fr](mailto:zied.elouedi@gmx.fr)

**Abstract.** This paper investigates the issue of building decision trees from data with imprecise class values where imprecision is encoded in the form of possibility distributions. The Information Affinity similarity measure is introduced into the well-known gain ratio criterion in order to assess the homogeneity of a set of possibility distributions representing instances's classes belonging to a given training partition. For the experimental study, we proposed an information affinity based performance criterion which we have used in order to show the performance of the approach on well-known benchmarks.

## 1 Introduction

Machine learning and data mining researches have rapidly emerged in the last decade. Especially, classification is considered as one of the most successful branches of Artificial Intelligence and it is playing a more and more important role in real-world applications.

Classification tasks are ensured by several approaches such as: discriminant analysis, artificial neural networks, k-nearest neighbors, Bayesian networks, decision trees. etc. The latter, namely, decision trees, is considered as one of the most popular classification techniques. They are able to represent knowledge in a flexible and easy form which justifies their use in decision support systems, intrusion detection systems, medical diagnosis, etc.

For many real-world problems and particularly for classification problems, imprecision is often inherent in modeling these applications and should be considered when building classifiers. For example, for some instances, an expert or a sensor may be unable to give the exact class value: an expert in ballistics in the scientific police who is unable to provide the exact type of a gun used in a crime, a mechanic who is unable to provide the exact fault of an engine, a doctor who cannot specify the exact disease of a patient, etc.

Hence, in these different examples, the expert can provide imprecise classifications expressed in the form of a ranking on the possible classes. Obviously, rejecting these pieces of information in a learning process is not a good practice. A suitable theory dealing with such situations is possibility theory which is a non-classical theory of uncertainty proposed by [12] and developed by [4].

In this paper, we propose a new decision tree approach that allows the induction of decision trees from imprecisely labeled instances, i.e., whose class labels are given in the form of possibility distributions. We introduced the concept of similarity into the attribute selection step of the proposed approach.

It is important to mention that existing possibilistic decision trees do not deal with uncertainty in classes, except, the work we have proposed in [7] using the concept of non-specificity in building possibilistic decision trees. A work proposed by Borgelt and al. [2] deals with crisp (standard) training sets: the authors encode the frequency distributions as possibility distributions (an interpretation which is based on the context model of possibility theory [2]) in order to define a possibilistic attribute selection measure. The possibilistic decision tree approach proposed by Hüllermeier [5] uses a possibilistic branching within the lazy decision tree technique. Again, this work does not deal with any uncertainty in the classes of the training objects. A work by Ben Amor et al. [1] have dealt with the classification of objects having possibilistic uncertain attribute values within the decision tree technique.

This paper is organized as follows. Section 2 gives necessary background on possibility theory. Section 3 describes some basics of decision trees. Then, in Section 4, we present our approach, so-called Aff-PDT. Section 5 presents and analyzes experimental results carried out on modified versions of commonly used data sets from the U.C.I. repository [9]. Finally, Section 6 concludes the paper.

## 2 Possibility Theory

### **Possibility distribution**

Given a universe of discourse  $\Omega = \{\omega_1, \omega_2, \dots, \omega_n\}$ , a fundamental concept of possibility theory is the *possibility distribution* denoted by  $\pi$ .  $\pi$  corresponds to a function which associates to each element  $\omega_i$  from the universe of discourse  $\Omega$  a value from a possibilistic scale  $[0,1]$ . This value is called a *possibility degree*: it encodes our knowledge on the real world.

By convention,  $\pi(\omega_i) = 1$  means that it is fully possible that  $\omega_i$  is the real world,  $\pi(\omega_i) = 0$  means that  $\omega_i$  cannot be the real world (is impossible). Flexibility is modeled by allowing to give a possibility degree from  $[0,1]$ . In possibility theory, extreme cases of knowledge are given by:

- *Complete knowledge*:  $\exists \omega_i, \pi(\omega_i) = 1$  and  $\forall \omega_j \neq \omega_i, \pi(\omega_j) = 0$ .
- *Total ignorance*:  $\forall \omega_i \in \Omega, \pi(\omega_i) = 1$  (all values in  $\Omega$  are possible).

### **Normalization**

A possibility distribution  $\pi$  is said to be *normalized* if there exists at least one state  $\omega_i \in \Omega$  which is totally possible. In the case of sub-normalized  $\pi$ ,

$$Inc(\pi) = 1 - \max_{\omega \in \Omega} \{\pi(\omega)\} \quad (1)$$

is called the *inconsistency degree* of  $\pi$ . It is clear that, for normalized  $\pi$ ,  $\max_{\omega \in \Omega} \{\pi(\omega)\} = 1$ , hence  $Inc(\pi)=0$ . The measure *Inc* is very useful in assessing the degree of conflict between two distributions  $\pi_1$  and  $\pi_2$  which is given by  $Inc(\pi_1 \wedge \pi_2)$ . We take the  $\wedge$  as the minimum operator. Obviously, when  $\pi_1 \wedge \pi_2$

gives a sub-normalized possibility distribution, it indicates that there is a conflict between  $\pi_1$  and  $\pi_2$  ( $Inc(\pi_1 \wedge \pi_2) \in [0, 1]$ ).

#### **Information Affinity: a possibilistic similarity measure**

After a deep study of existing possibilistic similarity measures, we have proposed in a recent work [6], a new similarity index satisfying interesting properties.

The information affinity index, denoted by  $InfoAff$  takes into account a classical informative distance, namely, the Manhattan distance along with the well known inconsistency measure.  $InfoAff$  is applicable to any pair of normalized possibility distributions.

**Definition 1** Let  $\pi_1$  and  $\pi_2$  be two possibility distributions on the same universe of discourse  $\Omega$ . We define a measure  $InfoAff(\pi_1, \pi_2)$  as follows:

$$InfoAff(\pi_1, \pi_2) = 1 - \frac{d(\pi_1, \pi_2) + Inc(\pi_1 \wedge \pi_2)}{2} \quad (2)$$

where  $d(\pi_1, \pi_2) = \frac{1}{n} \sum_{i=1}^n |\pi_1(\omega_i) - \pi_2(\omega_i)|$  represents the Manhattan distance between  $\pi_1$  and  $\pi_2$  and  $Inc(\pi_1 \wedge \pi_2)$  tells us about the degree of conflict between the two distributions (see Equation (1)).

Two possibility distributions  $\pi_1$  and  $\pi_2$  are said to have a strong affinity (resp. weak affinity) if  $InfoAff(\pi_1, \pi_2) = 1$  (resp.  $InfoAff(\pi_1, \pi_2) = 0$ ).

For sake of simplicity, in the rest of the paper, a possibility distribution  $\pi$  on a finite set  $\Omega = \{\omega_1, \omega_2, \dots, \omega_n\}$  will be denoted by  $\pi[\pi(\omega_1), \pi(\omega_2), \dots, \pi(\omega_n)]$ .

### 3 Decision trees

Decision trees, also called classification trees, are graphical models with a tree-like structure: they are composed of three basic elements: decision nodes corresponding to attributes, edges or branches which correspond to the different possible attribute values. The third component consists of leaves including objects that typically belong to the same class or that are very similar.

Several algorithms for building decision trees have been developed, e.g., **ID3** [10] and its successor **C4.5** "the state-of-the-art" algorithm developed by Quinlan [11]. These algorithms have many components to be defined:

a) **Attribute selection measure** generally based on information theory, serves as a criterion in choosing among a list of candidate attributes at each decision node, the attribute that generates partitions where objects are distributed less randomly, with the aim of constructing the smallest tree among those consistent with the data. The well-known measure used in the **C4.5** algorithm of Quinlan [11] is the gain ratio. Given an attribute  $A_k$ , the information gain relative to  $A_k$  is defined as follows:

$$Gain(T, A_k) = E(T) - E_{A_k}(T) \quad (3)$$

where

$$E(T) = - \sum_{i=1}^n \frac{n(C_i, T)}{|T|} \log_2 \frac{n(C_i, T)}{|T|} \quad (4)$$

and

$$E_{A_k}(T) = \sum_{v \in D(A_k)} \frac{|T_v^{A_k}|}{|T|} E(T_v^{A_k}) \quad (5)$$

$n(C_i, T)$  denotes the number of objects in the training set  $T$  belonging to the class  $C_i$ ,  $D(A_k)$  denotes the finite domain of the attribute  $A_k$  and  $|T_v^{A_k}|$  denotes the cardinality of the set of objects for which the attribute  $A_k$  has the value  $v$ . Note that  $\frac{n(C_i, T)}{|T|}$  corresponds to the probability of the class  $C_i$  in  $T$ . Thus,  $E(T)$  corresponds to the *Shannon entropy* of the set  $T$ . The gain ratio is given by:

$$Gr(T, A_k) = \frac{Gain(T, A_k)}{SplitInfo(T, A_k)} \quad (6)$$

where  $SplitInfo(T, A_k)$  represents the potential information generated by dividing  $T$  into  $n$  subsets. It is given by:

$$SplitInfo(T, A_k) = - \sum_{v \in D(A_k)} \frac{|T_v^{A_k}|}{|T|} \log_2 \frac{|T_v^{A_k}|}{|T|} \quad (7)$$

*b) Partitioning strategy* consisting in partitioning the training set according to all possible attribute values (for symbolic attributes) which leads to the generation of one partition for each possible value of the selected attribute. For continuous attributes, a discretization step is needed.

*c) Stopping criteria* stopping the partitioning process. Generally, we stop the partitioning if all the remaining objects belong to only one class, then the node is declared as a leaf labeled with this class value. We, also, stop growing the tree if there is no further attribute to test. In this case, we take the majority class as the leaf's label.

## 4 Affinity based possibilistic decision trees

An affinity based possibilistic decision tree (Aff-PDT) has the same representation of a standard decision tree, i.e., it is composed of *decision nodes* for testing attributes, *branches* specifying attribute values and *leaves* dealing with classes of the training set.

### 4.1 Imperfection in classification problems

As models of the real world, databases, or more specifically, training sets are often permeated with forms of imperfections, including imprecision and uncertainty. The topic of imperfect databases is gaining more and more attention the last years [8] since commercial database management systems are not able to deal with such kind of information.

Examples of imperfect class values include the exact type of an attack in an intrusion detection system, the exact cancer class of a patient in cancer diagnosis

applications, the exact location or type of a detected aerial engine in military applications, etc. These imperfections might result from using unreliable information sources, such as faulty reading instruments, or input forms that have been filled out incorrectly (intentionally or inadvertently).

In order to deal with such kind of imperfection, in this work, we used a convenient mathematical model, namely, possibility theory [4, 12]. More formally, a possibility degree will be assigned to each possible class value indicating the possibility that the instance belongs to a given class [3]. These possibility degrees can be obtained from direct expert's elicitation, i.e., each expert is asked to quantify by a real number between 0 and 1 the possibility that a training instance belongs to each one of the different classes of the problem.

## 4.2 Building procedure

In the possibilistic setting, instances classes in the training set will be represented by possibility distributions over the different classes of the problem instead of exact classes. Hence, one must find a way to assess homogeneity of a given training partition. The idea consists in measuring the entropy of each partition weighted by the mean similarity degree of the possibility distributions in the corresponding partition. Let us define the basic components for the Aff-PDT approach:

### a) Attribute selection measure

Given a training set  $T$  (the initial partition) containing  $n$  instances and given the set of attributes, let us denote by  $\pi_i$  the possibility distribution labeling the class of the instance  $i$  in  $T$ .

In standard decision trees, homogeneity of a partition is determined by the entropy of that partition. However, in our context,  $\pi_i$ 's are most of the time very different, so it has no sense to directly compute their frequencies in order to determine the entropy of  $T$  (the entropy will be equal to 1). Moreover, one cannot simply view each  $\pi_i$  as a new class. First, because the number of classes will be exponential. Second, there are similar distributions that should be considered as globally expressing same or similar pieces of information. For instance, we cannot simply consider the distributions [1, 0.2] and [1, 0.21] as two different exclusive classes, but we will consider them as similar.

Hence, we need a finite set of Meta-Classes  $MC_{j=1..m}$ . Each  $MC_j$  corresponds to a meta-class which gathers together all possibility distributions similar to a predefined wrapper possibility distribution (say  $WD_j$ ). More precisely, wrapper possibility distributions are binary possibility distributions (i.e.,  $\forall \omega \in \Omega, \pi(\omega) \in \{0, 1\}$ ) representing special cases of complete knowledge, partial ignorance and total ignorance representing the set of reference distributions.

After specifying the set of Meta-Classes  $MC$ , we will assign to each possibility distribution  $\pi_i$  labeling an instance  $i$  a meta-class  $MC_j$  such that:  $MC_j = \arg \max_{j=1}^m \{InfoAff(\pi_i, WD_j)\}$  where  $m$  is the total number of meta-classes and  $InfoAff$  corresponds to the information Affinity index (Equation (2)). Note that, as in standard decision trees, ties are broken arbitrarily.

After mapping the different  $\pi_i$ 's to their corresponding  $MC_j$ 's, it becomes possible to assess the discriminative power of each attribute in partitioning a set into homogeneous subsets by extending the well-known gain ratio criterion [11]. First, we define the Affinity-Entropy Gain ( $AGain$ ) of an attribute  $A_k$  by:

$$AGain(T, A_k) = AE(T) - AE_{A_k}(T) \quad (8)$$

where

$$AE(T) = - \sum_{j=1}^m (AvgAff(MC_j)) * \left( \frac{|MC_j|}{|T|} \log_2 \frac{|MC_j|}{|T|} \right) \quad (9)$$

and

$$AE_{A_k}(T) = \sum_{v \in D(A_k)} \frac{|T_v^{A_k}|}{|T|} SE(T_v^{A_k}) \quad (10)$$

where  $|MC_j|$  in Equation (9) denotes the number of objects in the training set  $T$  belonging to the meta-class  $MC_j$ . Obviously, to compensate for the information loss resulting from grouping resemblant  $\pi_i$ 's into their corresponding  $MC_j$ 's, we have introduced the  $AvgAff(MC_j)$  factor which corresponds to the average similarity between the original possibility distributions  $\pi_{p=1..n}$  assigned to  $MC_j$ :

$$AvgAff(MC_j) = \frac{\sum_{p=1}^{n-1} \sum_{q=p+1}^n InfoAff(\pi_p, \pi_q)}{\frac{n*(n-1)}{2}} \quad (11)$$

**Proposition 1** *When dealing with crisp training sets, i.e., with precise classes ( $MC_j \equiv C_j$ ), we will always have  $AvgAff(MC_j) = 1$  and  $|MC_j|$  will correspond to number of instances labeled by the same class  $C_j$ , thus we recover the standard C4.5 approach.*

Then, the Affinity-gain ratio is expressed in the same way as the classical gain ratio using  $SplitInfo$  (Equation (7)):

$$AGr(T, A_k) = \frac{AGain(T, A_k)}{SplitInfo(T, A_k)} \quad (12)$$

Obviously, the attribute maximizing  $AGr$  will be assigned to the decision node at hand.

**Example 1** *Let us use a modified version of the golf data set [9] to illustrate the notion of wrapper distributions. Let  $T$  be the training set composed of fourteen instances  $i_{1..14}$ . A possibility distribution was given for each possible class of each instance of  $T$ . The set of wrapper distributions relative to this example is  $WD = \{[1, 0], [0, 1], [1, 1]\}$ . Consequently,  $MC = \{MC_1, MC_2, MC_3\}$  such that  $MC_1 = \{i_4, i_9, i_{10}, i_{11}, i_{12}, i_{13}\}$ ,  $MC_2 = \{i_1, i_2, i_6, i_8, i_{14}\}$  and  $MC_3 = \{i_3, i_5, i_7\}$ . The Affinity-Entropy of the set  $T$  is computed (using Equation (9)) as follows :  $AE(T) = -0.956 * (\frac{6}{14} * \log_2 \frac{6}{14}) - 0.95 * (\frac{5}{14} * \log_2 \frac{5}{14}) - 0.966 * (\frac{3}{14} * \log_2 \frac{3}{14}) = 1.464$ .*

**Table 1.** Imprecisely labeled Training Set

	<b>Outlook</b>	<b>Temp</b>	<b>Humidity</b>	<b>Wind</b>	$C_1$	$C_2$
$i_1$	sunny	hot	high	weak	0.2	1
$i_2$	sunny	hot	high	strong	0.4	1
$i_3$	overcast	hot	high	weak	1	0.7
$i_4$	rainy	mild	high	weak	1	0
$i_5$	rainy	cool	normal	weak	1	0.8
$i_6$	rainy	cool	normal	strong	0.4	1
$i_7$	overcast	cool	normal	strong	1	0.9
$i_8$	sunny	mild	high	weak	0.3	1
$i_9$	sunny	cool	normal	weak	1	0.3
$i_{10}$	rainy	mild	normal	weak	1	0
$i_{11}$	sunny	mild	normal	strong	1	0.2
$i_{12}$	overcast	mild	high	strong	1	0
$i_{13}$	overcast	hot	normal	weak	1	0.3
$i_{14}$	rainy	mild	high	strong	0	1

**b) Partitioning strategy**

Since we only deal with nominal attributes, the partitioning strategy will be the same as with standard decision trees.

**c) Stopping criteria**

We will stop growing the tree if:

1. There is no further attribute to test.
2.  $AGain \leq 0$ , i.e., no information is gained.
3.  $|T_p|=0$ , i.e., the generated partition does not contain any instance.

**d) Structure of leaves**

Leaves of our induced Aff-PDT trees will be labeled by possibility distributions on the different classes rather than crisp classes. In fact, when the above stopping criterion 1 or 2 is satisfied for a training partition  $T_p$  containing  $n$  possibility distributions, we will declare a leaf labeled by the representative possibility distribution of that set ( $\pi_{Rep}$ ), that is, the possibility distribution which corresponds to the closest distribution to all the remaining distributions in the set  $T_p$ :

$$\pi_{Rep}(T_p) = \arg \max_{i=1}^n \left\{ \frac{\sum_{j \neq i} InfoAff(\pi_i, \pi_j)}{(n-1)} \right\}$$

Hence, when considering the special case of a leaf with only certain possibility distributions, if we take the fully possible class of ( $\pi_{Rep}(T_p)$ ) as a final decision, we join the solution of majority class adopted by standard decision trees.

Finally, when stopping criterion 3 is satisfied, we declare an empty leaf labeled by a randomly chosen wrapper possibility distribution from  $WD$ .

**4.3 Classification procedure**

Once the Aff-PDT is constructed, we can classify any new object given values of its attributes. We start with the root of the constructed tree and follow the path corresponding to the observed value of the attribute in the interior node of the tree. This process is continued until a leaf is encountered. As mentioned

above, each leaf of our decision tree will be labeled by a possibility distribution over the different class values. Hence, to make a decision about the class of a given object, the decision maker can take the fully possible class label (i.e. the class having a possibility degree equal to 1).

## 5 Experimental results

Our experimental studies are divided in two parts. First, we evaluate our Aff-PDT approach. Second, we compare our results with those of the C4.5 algorithm if we ignored uncertainty. Note that we do not intend to compare Aff-PDT with C4.5 since this latter do not deal with uncertainty: the aim of the comparison is to show whether ignoring uncertainty in training data is a good practice or not.

The experimental study is based on several data sets selected from the U.C.I repository [9]: (1) Wisconsin Breast cancer (699 instances, 8 attributes, 2 classes), (2) Voting (497, 16, 2), (3) Balance scale (625, 4, 3), (4) Solar Flare (1389, 10, 3), (5) Nursery (12960, 8, 5). We have modified these data sets by transforming the original crisp classes by possibility distributions over the different classes. We have used levels of uncertainty ( $L\%$ ) when generating these possibilistic training sets: for each training instance from the  $L\%$  randomly chosen instances, we have assigned a possibility degree equal to 1 to the original class and a random possibility degree to the remainders in an uniform way. To each one of the remaining  $(100 - L)\%$  instances, we have assigned a completely sure possibility distribution corresponding to the original crisp instance's class.

In order to determine the accuracy of the induced trees, we have used two criteria, the first is relative to the percentage of correct classification ( $PCC = \frac{\text{number of well classified instances}}{\text{total number of classified instances}} \times 100$ ) and the second corresponds to a similarity based criterion ( $PCC\_Aff$ ) which we have proposed in [6] as a new criterion that is more appropriate to the possibilistic context:

$$PCC\_Aff = \frac{\sum_{j=1}^n InfoAff(\pi^{res}, \pi^j)}{\text{total\_nbr\_classified\_inst}} \times 100 \quad (13)$$

Let us recall that the output of a possibilistic decision tree is given in the form of a possibility distribution ( $\pi^{res}$ ). Thus, the standard  $PCC$  is computed by choosing for each instance to classify the class having the highest possibility degree (equal to 1). If more than one class is obtained, then one of them is chosen randomly. Finally, this class label is compared with the true class label.

**Table 2.** Aff-PDT ( $PCC\_Aff$  and standard deviation)

$L\%$	0%	30%	50%
W.B.cancer	93.37(1.3)	91.2(1.7)	88.71(2.1)
Voting	96.76 (1.7)	95.35(1.9)	94.11(1.9)
Solar Flare	87.26(2.2)	84.90(1.8)	83.77(1.6)
Balance	83.54 (1.5)	73.83 (1.2)	72.88(1.2)
Nursery	98.74 (0.8)	96.96(1.1)	95.95(1.4)

Table 2 reports the different obtained results after varying the training sets' level of uncertainty  $L\%$  for each database.  $PCC_{Aff}$  values of the induced Aff-PDT trees are complemented by standard deviations after the use of a 10-fold cross validation testing process. Note that high values of the  $PCC_{Aff}$  criterion do not only imply that the induced trees are accurate but also imply that the possibility distributions provided by the induced Aff-PDT trees are of high quality and faithful to the original possibility distributions. From Table 2, we can see that  $PCC_{Aff}$  values decrease when  $L\%$  increases. This can be explained by the fact that the higher the level of uncertainty ( $L\%$ ), the less informative the training set becomes (consequently, the harder the learning becomes), and therefore the less accurate the predictions are.

Now, let us see what happens when ignoring imprecisely labeled training instances when building decision trees. To respond to this question, we have conducted our experimentations as follows: for each training set and for each uncertainty level  $L\%$ , we have induced an Aff-PDT tree. On the other hand, a C4.5 tree was induced from the corresponding training set, i.e., the standard training set from which we have discarded the  $L\%$  instances to which we have assigned imprecise class labels since the C4.5 algorithm cannot deal with such instances. Then, both approaches are evaluated on the same testing sets: standard testing sets for C4.5 trees have been used and their corresponding testing sets (with completely sure possibility distributions on the original class labels) for Aff-PDT trees: this corresponds to one iteration of the 10-fold cross validation process used for the evaluation of the approach. Table 3 reports the different obtained results after varying the training sets' level of uncertainty  $L\%$  for each database.  $MPCC$  denotes the mean  $PCC$  (complemented by standard deviation) of the induced decision trees for the 10-fold cross validation process.

**Table 3.** C4.5 and Aff-PDT (MPCC and standard deviation)

Database	Method	$L = 0\%$	$L = 30\%$	$L = 50\%$
W.B.cancer	C4.5	94.54(1.1)	91.05(2.5)	90.11(3.2)
	Aff-PDT	94.54(1.1)	91.63(2.3)	90.73(2.6)
Voting	C4.5	94.56(3.2)	90.15(3.8)	87.27(4.6)
	Aff-PDT	94.56(3.2)	91.62(3.5)	88.52(4.0)
Solar flare	C4.5	81.96(3.3)	77.03(3.7)	74.37(3.9)
	Aff-PDT	81.96(3.3)	80.57(3.7)	78.48(3.9)
Balance	C4.5	78.48(4.2)	74.78(5.3)	70.38(5.7)
	Aff-PDT	78.48(4.2)	77.06(4.9)	74.82(5.4)
Nursery	C4.5	98.78(0.8)	94.45(1.6)	92.81(2.6)
	Aff-PDT	98.78(0.8)	97.11(1.2)	94.37(2.2)

Table 3 shows that the Aff-PDT approach gives interesting results when compared with the C4.5 algorithm. Again, we can see that classification accuracies of both approaches decrease when the level of uncertainty increases (for the same explanation provided above for Table 2). In spite of this decrease in accuracy, we can see that the classification rate of Aff-PDT is always (even slightly) greater

than the one of C4.5. Note that the aim of this comparison is not to directly compare the two approaches. In fact, the C4.5 is used only in certain environments: it is trained from reduced training sets (imprecisely labeled instances are omitted) while the Aff-PDT approach deals with both certain and uncertain environments: it is trained from complete training sets (including both precisely and imprecisely labeled instances). Besides, Table 3 confirms Proposition 1. In fact, our approach recovers the C4.5 one when dealing with crisp instances (with precise labels, i.e.,  $L\% = 0$ ).

From the results given in this table, we can conclude that, generally, rejecting training instances, classes of which are imprecisely defined, is not a good practice and reduces the accuracy of the induced classifier. This issue can be avoided and well handled by the use of the proposed Aff-PDT approach which can exploit the information contained in imprecise labels.

## 6 Conclusion

This paper proposes a generalization of the C4.5 approach to the imprecise setting. The new approach has the advantage of allowing the induction of decision trees from training instances having possibilistic class labels. The proposed Aff-PDT approach blends information affinity with entropy in order to asses the homogeneity of a given training partition. Experiments have shown that rejecting training instances, classes of which are imprecisely defined, is not a good practice and reduces the accuracy of the induced classifier. We plan to add an automatic clustering phase for the specification of the wrapper distributions which could enhance the performance of the approach.

## References

1. N. Ben Amor, S. Benferhat, Z. Elouedi: Qualitative classification and evaluation in possibilistic decision trees, (*FUZZ-IEEE'04*), Hungary, 2004, 653-657.
2. C. Borgelt, J. Gebhardt, R. Kruse: Concepts for Probabilistic and Possibilistic Induction of Decision Trees on Real World Data. (*EUFIT'96*), 1996, 1556-1560.
3. T. Denoeux and L. M. Zouhal. Handling possibilistic labels in pattern classification using evidential reasoning. *Fuzzy Sets and Systems*, 122(3), 2001, 47-62.
4. D. Dubois and H. Prade: Possibility theory: An approach to computerized processing of uncertainty, *Plenum Press*, New York, 1988.
5. E. Hüllermeier. Possibilistic Induction in decision tree learning. *ECML'02*, Helsinki, Finland, 2002, 173-184.
6. I. Jenhani, N. Ben Amor, Z. Elouedi, S. Benferhat and K. Mellouli: Information Affinity: a new similarity measure for possibilistic uncertain information, *EC-SQARU'07*, Hammamet, Tunisia, 2007, 840-852.
7. I. Jenhani, N. Ben Amor, Z. Elouedi: Decision Trees as Possibilistic Classifiers, *International Journal of Approximate Reasoning*, Elsevier, 2007, to appear.
8. A. Motro: Sources of Uncertainty, Imprecision and Inconsistency in Information Systems. *Uncertainty Management in Information Systems: From Needs to Solutions*, 1996, 9-34.
9. P. M. Murphy, D. W. Aha: UCI repository of machine learning databases, 1996.
10. J. R. Quinlan: Induction of decision trees, *Machine Learning*, 1, 1986, 81-106.
11. J. R. Quinlan: C4.5: Programs for machine learning, Morgan Kaufmann, 1993.
12. L. A. Zadeh: Fuzzy sets as a basis for a theory of possibility, *Fuzzy Sets and Systems*, 1, 1978, 3-28.

# Ordering Finite Labeled Trees

Herman Ruge Jervell

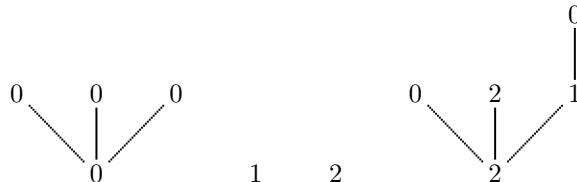
Department of Computer Science, Pb 1080  
University of Oslo  
N-0317 Oslo, Norway  
[herman@ifi.uio.no](mailto:herman@ifi.uio.no)

**Abstract.** Previously — in CiE 2005 [2005] — we have given an ordering of finite trees and showed that it is a well ordering reaching up to the small Veblen ordinal. Here we extend this ordering to finite labeled trees — similar to Takeutis ordinal diagrams [1975] — and show that this extended ordering is also a well ordering.

**Key words:** ordinals, ordinal notation, finite labeled trees

## 1

We have given a wellordered set of labels  $\Lambda$  and consider finite trees with labels at the nodes. In the trees the branches are ordered — from left to right. Below there are four examples of finite labeled trees



Here the smallest label is 0, then 1, 2 .... The first tree is the ordinal  $\epsilon_0$ . The second tree — consisting of only one node with label 1 — is the smallest tree larger than all trees with just 0 as label. It corresponds to the small Veblen ordinal. The third tree corresponds to the Howard ordinal. The fourth tree is larger than the three others and is just an example of the more general type of labeled trees considered.

## 2

In CiE 2005 [2005] we gave an ordering of finite trees. The ordering was given by

$$\boxed{\mathbf{A} < \mathbf{B} \Leftrightarrow \mathbf{A} \leq \langle \mathbf{B} \rangle \vee (\langle \mathbf{A} \rangle < \mathbf{B} \wedge \langle \mathbf{A} \rangle < \langle \mathbf{B} \rangle)}$$

where

$\mathbf{A} \leq \langle \mathbf{B} \rangle$  : There is an immediate subtree  $\mathbf{B}_i$  of  $\mathbf{B}$  such that either  $\mathbf{A} < \mathbf{B}_i$  or  $\mathbf{A} = \mathbf{B}_i$

$\langle \mathbf{A} \rangle < \mathbf{B}$  : For all immediate subtrees  $\mathbf{A}_j$  of  $\mathbf{A}$  we have  $\mathbf{A}_j < \mathbf{B}$

$\langle \mathbf{A} \rangle < \langle \mathbf{B} \rangle$  : The inverse lexicographical ordering of the immediate subtrees — we first check which sequence have smallest length, and if they have equal length we look at the rightmost immediate subtree where they differ

We then prove by induction over subtrees

- The relation is transitive
- The relation is total:  $A < B \vee A = B \vee B < A$  where  $A = B$  means the ordinary equality between trees and the three cases are mutually exclusive
- The relation is decidable

We used  $\Pi_1^1 - CA$  to prove that the relation is a well order. There is a 1-1 correspondence between the finite trees and the ordinals less than the small Veblen ordinal.

### 3

In the generalization to finite labeled trees we shall

- Consider more orderings — an ordering  $<_n$  for each label  $n$  and an ordering  $<_\infty$
- Have a proof of the well ordering going beyond  $\Pi_1^1 - CA$
- Consider a new notion — visibility in a labeled tree

First the new notion — visibility. For each label  $n$  we define  $n$ -visibility. Say that we are in a labeled tree  $S$  and at node  $\nu$ . Then from node  $\nu$  a node  $\mu$  is  $n$ -visible if

- Node  $\mu$  has label  $\geq n$
- All nodes strictly between  $\nu$  and  $\mu$  have labels  $> n$

So from node  $\nu$  we can  $n$ -see all nodes which can be reached through nodes with labels  $\geq n$  and up to the first nodes with label  $n$ . A node with label  $\leq n$  will block the  $n$ -visibility for all nodes above it. We use the visibility to define the  $n$ -subtrees of a tree  $S$

$\langle S \rangle_n$  = the sequence of all  $n$ -subtrees  $n$ -visible from the root of  $S$

Now we are ready to define the orderings

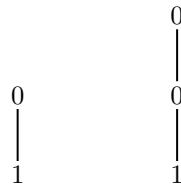
$$\begin{aligned} S <_j T &\Leftrightarrow S \leq_j \langle T \rangle_j \vee (\langle S \rangle_j <_j T \wedge S <_{j+} T) \\ S <_\infty T &\Leftrightarrow \text{lexicographical ordering} \end{aligned}$$

Here we use abbreviations like for the finite trees

- $\leq_j$  means either ordinary  $=$  or  $<_j$
- $S \leq_j \langle T \rangle_j$  : There exists a  $j$ -subtree  $T_0$  of  $T$  with  $S \leq_j T_0$
- $\langle S \rangle_j < T$  : For all  $j$ -subtrees  $S_0$  of  $S$ ,  $S_0 <_j T$
- $j+$  is the smallest label in  $S$  and  $T$  larger than  $j$  if it exists, else it is  $\infty$
- The lexicographical ordering is such that we compare in priority
  - The labels at the root of  $S$  and the root of  $T$
  - For the same label  $i$ : The lengths of  $\langle S \rangle_i$  and  $\langle T \rangle_i$
  - The rightmost place where the two sequences differ in the  $>_i$ -ordering

As for the finite trees we get that the orderings are transitive, total and decidable. This is proved by induction over the subtrees and using the finite set of labels occurring in the trees. The following differs from our theory of finite trees

- We consider many orderings
- In the theory of finite trees to each ordinal there corresponded a unique finite tree. This is not so any more. The two finite labeled trees below is equal in the ordering



Both trees have the same 1-immediate subtrees — they have none, and to get equality this is what we compare.

## 4

To make the proof of the well ordering more perspicuous we go through the following cases

- Only label 0
- Finite number of labels
- A well ordered set of labels

Trees with only label 0 is the same as finite trees without labels. Our proof here is not the same as in [2005]. The proof is a variant of the usual minimal bad argument. We remind the reader

**Bad tree:** A tree is bad if there is an infinitely descending sequence starting with the tree

**Minimal tree:** A tree is minimal if no immediate subtree is bad

Now consider the case where the only label is 0. We must prove that there are no bad trees. Assume we have a bad tree  $S_0$ . If  $S_0$  is not minimal, then by going to an immediate subtree we get a smaller tree which is bad. And then we can continue this going to immediate subtrees until we get a bad tree where all its immediate subtrees are not bad. We call this minimal bad tree for

$$S_0^0$$

Consider now the bad sequence starting with it. We shall construct a sequence of minimal bad trees. Say we have constructed the minimal bad trees

$$S_0^0 >_0 S_1^0 >_0 \cdots >_0 S_n^0$$

The sequence can be continued with bad trees

$$S_0^0 >_0 S_1^0 >_0 \cdots >_0 S_n^0 >_0 T_{n+1} >_0 T_{n+2} >_0 \cdots$$

Observe now that for any immediate subtree  $U_{n+1}$  of  $T_{n+1}$  we have  $T_{n+1} >_0 U_{n+1}$  and by transitivity  $S_n^0 >_0 U_{n+1}$ . Hence by continuing taking immediate subtrees of  $T_{n+1}$  we get a minimal bad tree  $S_{n+1}^0$  with

$$S_0^0 >_0 S_1^0 >_0 \cdots >_0 S_n^0 >_0 S_{n+1}^0$$

and we can continue the construction to get an infinite sequence of minimal bad trees

$$S_0^0 >_0 S_1^0 >_0 \cdots >_0 S_n^0 >_0 S_{n+1}^0 >_0 S_{n+2}^0 >_0 \cdots$$

We next observe that from the definition of the ordering

$$S <_j T \Leftrightarrow S \leq_j \langle T \rangle_j \vee (\langle S \rangle_j <_j T \wedge S <_{j+} T)$$

we cannot use the first condition. If  $\langle S_m^0 \rangle \geq_0 S_{m+1}^0$  then  $S_m^0$  would have an immediate subtree which is bad — contradicting the minimality of  $S_m^0$ . We must use the second condition and get

$$S_0^0 >_\infty S_1^0 >_\infty \cdots >_0 S_n^0 >_\infty S_{n+1}^0 >_\infty S_{n+2}^0 >_\infty \cdots$$

Now look at the lexicographical ordering. Here our trees have only labels 0. From some stage off all lengths of sequences of immediate subtrees must be the same, and we get an immediate subtree which starts an infinitely  $>_0$ -descending sequence and a bad immediate subtree — contradicting that we had a sequence of minimal trees.

## 5

Now we come to the case where we have a finite number of labels. We must generalize the notions of bad and of minimal

**$n$ -bad:** There is an infinite  $>_n$ -descending sequence

**$n$ -minimal:** No  $m$ -immediate subtree is  $m$ -bad for any  $m \leq n$

Now assume we have a 0-bad tree. As in the previous section we construct a 0-minimal 0-bad sequence

$$S_0^0 >_0 S_1^0 >_0 S_2^0 >_0 \dots$$

We get as above

$$S_0^0 >_1 S_1^0 >_1 S_2^0 >_1 \dots$$

This is a 1-bad sequence. We must construct a sequence which is also 1-minimal. Observe that going to the 1-immediate subtrees does not destroy the 0-minimality. By going to the 1-immediate subtrees we may get rid of some 0-immediate subtrees but we do not create any new 0-immediate subtrees. Therefore we get a 1-minimal 1-bad sequence

$$S_0^1 >_1 S_1^1 >_1 S_2^1 >_1 \dots$$

And then we continue this line for line until we have the sequence

$$S_0^\infty >_\infty S_1^\infty >_\infty S_2^\infty >_\infty \dots$$

which is  $k$ -minimal for all  $k$ . Then from some stage in the sequence we have the same label  $n$  at the root and the same length of the  $n$ -immediate subtrees. But then we get an  $n$ -immediate subtree which is  $n$ -bad — contradicting the  $n$ -minimality of all trees in the sequence.

## 6

In the first construction of minimal bad sequence we used  $\Pi_1^1$ -CA. We had to decide whether an immediate subtree is bad or not. This is more problematic for the case where we have more labels. The construction of a 0-minimal 0-bad sequence of trees use  $\Pi_1^1$ -CA as before. But we had to do more in the construction of a 1-minimal 1-bad sequence. So say we have the 0-minimal 0-bad sequence

$$S_0^0 >_0 S_1^0 >_0 S_2^0 >_0 \dots$$

And we get from the 0-minimality

$$S_0^0 >_1 S_1^0 >_1 S_2^0 >_1 \dots$$

Now we had to show that this sequence can also be made 1-minimal. But then we had to decide whether an immediate subtree is starting an infinite sequence of  $>_1$  descending 0-minimal trees. This goes beyond  $H_1^1$ -CA. Further down in the construction we do the same — we had to decide whether an immediate subtree is starting an infinite sequence of  $>_{n+1}$  descending  $n$ -minimal trees.

## 7

We now consider the general case where the labels are taken from a well ordered set. As before we assume we have a 0-bad tree, and then construct trees

$$S_m^n \text{ for each label } n \text{ and each number } m$$

Each row is a sequence of  $n$ -minimal  $n$ -bad trees

$$S_0^n >_n S_1^n >_n S_2^n >_n \dots$$

The construction goes row for row. We must say how the construction goes at limit rows. Remember that in the construction row  $n$  and  $n+1$  may be quite similar. *The first place  $m$  where they differ  $S_m^{n+1}$  is a subtree of  $S_m^n$ .* We then prove that in the columns the trees are mostly as the tree above — in each column there are only a finite number of changes. For assume not. Call the leftmost column  $m$  with an infinite number of changes for the critical column. Then from some row  $n$  there are no changes to the left of the critical column  $m$ . That means that from row  $n$  all changes in the critical column comes from going from a tree to a subtree. But this can only be done a finite number of times — and the critical column was not critical. In each column there are only a finite number of changes. And we have the obvious construction for limit rows — just take the limit along each column.

Now look at a limit row

$$T_0^\lambda >_\lambda T_1^\lambda >_\lambda T_2^\lambda >_\lambda \dots$$

Here we have

- The row gives a  $\lambda$ -bad sequence
- All elements are  $n$ -minimal for each  $n < \lambda$ .

Now we can use the construction above to also get a  $\lambda$ -bad sequence which is  $\lambda$ -minimal. And then we continue as before. The contradiction comes at row  $\infty$ .

**Theorem 1.** *If the labels  $\Lambda$  is well ordered, then  $<_0$  is a well order.*

**8**

We shall now prove that all the orderings  $<_n$  are well orderings. Assume we have proved it for all labels  $< n$ . But then

- No tree is  $m$ -bad for  $m < n$
- All trees are  $m$ -minimal for  $m < n$ .

Assume now we have an  $n$ -bad tree. So we get an  $n$ -bad sequence

$$T_0^n >_n T_1^n >_n T_2^n >_n \dots$$

This sequence is  $m$ -minimal for all  $m < n$  since all trees are. But then we go through the construction as above and get an  $n$ -bad  $n$ -minimal sequence

$$S_0^n >_n S_1^n >_n S_2^n >_n \dots$$

and we can continue the construction as above to get the last row

$$S_0^\infty >_\infty S_1^\infty >_\infty S_2^\infty >_\infty \dots$$

which is  $\infty$ -bad and  $m$ -minimal for all labels  $m$ . And then we have a contradiction as above and can conclude

**Theorem 2.** *All orderings  $<_k$  and  $<_\infty$  are well orderings.*

**9**

We now consider trees with labels up to some finite number  $N$ . Then consider the treeclasses

- $\mathbb{T}_N$  — trees with label  $N$  at the root
- $\mathbb{T}_{N-1}$  — trees with label  $N - 1$  or  $N$  at the root
- $\mathbb{T}_{N-2}$  — trees with label  $N - 2$ ,  $N - 1$  or  $N$  at the root
- $\dots$
- $\mathbb{S}_0$  — trees with only label 0
- $\mathbb{S}_1$  — trees with only label 0 or 1
- $\mathbb{S}_2$  — trees with only label 0, 1 or 2
- $\dots$

We then have

**Theorem 3.** *The following are isomorphic orderings*

- $\mathbb{T}_N$  and  $<_N$  —  $\mathbb{S}_0$  and  $<_0$
- $\mathbb{T}_{N-1}$  and  $<_{N-1}$  —  $\mathbb{S}_1$  and  $<_0$
- $\mathbb{T}_{N-2}$  and  $<_{N-2}$  —  $\mathbb{S}_2$  and  $<_0$
- $\dots$

And we have

**Theorem 4.** *The one node trees*

- 1 is the supremum of  $\mathbb{S}_0$  under  $<_0$
- 2 is the supremum of  $\mathbb{S}_1$  under  $<_0$
- 3 is the supremum of  $\mathbb{S}_2$  under  $<_0$
- ...

## 10

Let us now compare our finite labeled trees with the ordinal diagrams of Takeuti. It is easiest to compare them with the variant of ordinal diagrams of finite order made by Levitz [1970]. We use the following variant of the ordinal diagrams  $O(n)$

- In  $O(n)$  we consider labeled finite trees which are unordered
- We have given a pair of two numbers as the labels
  - The first number is from 0 to  $n$  and is called the order
  - The second number is a natural number called the degree
- In our labeled trees we defined sequences  $\langle T \rangle_k$  for each order  $k$
- In the ordinal diagrams  $O(n)$  we define similarly multisets  $[T]_k$  for each order  $k$
- The orderings  $<_k$  are defined similarly for the labeled finite trees and the ordinal diagrams using the gap condition
- The orderings  $<_\infty$  are defined lexicographically but this comes out differently for our ordered labeled trees and the unordered ordinal diagrams
- In the ordinal diagrams the elements of the multisets  $[T]_k$  all have the same order at the root but they may have different degrees
- Then for the multisets we first compare the elements of largest degree and so on

Levitz have the connection between the finite ordinal diagrams and iterated inductive definitions

**Theorem 5 (Levitz).** *The ordinal diagram  $O(n)$  gives the ordinals connected with  $ID_{n-1}$ . In particular the ordinal diagrams with order 0 and 1 give the ordinal below the Howard ordinal.*

On the other hand we give connections between the unordered ordinal diagrams and our ordered finite labeled trees. We can use the degrees to embed the ordered trees into the unordered trees and use the possibility of having large branchings to embed ordered trees with large degrees. We get

**Theorem 6.** *The finite labeled trees  $\mathbb{S}_n$  corresponds to the ordinal diagrams  $O(n)$ . In particular the labeled tree with the single node 2 under  $<_0$  corresponds to the Howard ordinal.*

Helmut Pfeiffer [1972] has generalized Levitz connections between ordinal diagrams and Schüttes notation system to arbitrary wellordered set of orders. As above we also get connections to finite labeled trees where the labels come from a wellordered set.

## References

- [1970] Levitz, Hilbert: On the relationship between Takeuti's ordinal diagrams  $O(n)$  and Schütte's system of ordinal notations  $\Sigma(n)$ . Intuitionism and Proof Theory. (Eds Myhill, Vesley, Kino). North-Holland
- [1972] Pfeiffer, Helmut: Vergleich zweier Bezeichnungssysteme für Ordinalzahlen. Arch. math. Logik 15, 1972.
- [1975] Takeuti, Gaisi: Proof theory. North-Holland
- [2005] Jervell, Herman Ruge: Finite trees as ordinals. CiE 2005, Springer Lecture Notes.

# Towards Reverse Proofs-as-Programs

Reinhard Kahle

CENTRIA and Departamento de Matemática, Universidade Nova de Lisboa,  
2829-516 Caparica, Portugal  
[kahle@mat.uc.pt](mailto:kahle@mat.uc.pt)

**Abstract.** In this programmatic paper we renew the well-known question “What is a proof?”. Challenged by computer based theorem provers, we argue that a (good) proof is not the formalized one. Looking for the notion of equality of proofs, we propose to investigate proofs in relation to MOSCHOVAKIS’s theory of algorithms. This should be carried out by a “reverse engineering” of program extraction functions underlying the *proofs-as-programs* paradigm.

**Key words:** Proof; equality of proofs; algorithm; proofs-as-programs

## 1 What is a proof?

The birth of *science* is often attributed to the Ancient Greeks since they introduced the concept of *proof* in mathematical arguments. While the success of mathematics since the ancient world is an everlasting story, it is surprising that the central notion of proof in its own only recently got in the focus of research. Although one might find a lot of controversial “proofs” in the history of mathematics, the question of what is a proof had a direct impact on mathematics only in the last 150 years.

A first example seems to be HILBERT’s proof of his *finite basis theorem* which caused a sensation due to its non-constructive character. In a first reaction, GORDAN labeled it as *theology* rather than *mathematics*. From a modern perspective, it was not the notion of proof which was changed by HILBERT, but rather the (traditional) meaning of a particular mathematical concept: the existential quantifier. In fact, most of the controversies about certain proofs may actually be regarded as controversies about the concepts which are involved in these proofs. However, historically HILBERT’s non-constructive proof was important since it gave rise to a profound discussion of the foundations of mathematics, which eventually led to the introduction of *proof theory* by HILBERT himself as a new discipline within mathematical logic. This discussion was accompanied by the rigid *formalization* of mathematics in the work of BOOLE, PEANO, FREGE, WHITEHEAD and RUSSELL (to mention the most important ones). As a result we nowadays have at hand a quite clear defined notion of *formalized proof* in mathematics. But these formalized proofs have a specific aim in the foundations of mathematics, and, as a matter of fact, a working mathematician nearly never formalizes a proof in such a rigid way.

Formalized proofs play, however, the central role in computer-aided theorem proving. But, at present, computer generated proofs do not meet the standards we have for (good) mathematical proofs. They also do not help (yet) to decide the question of equality of proofs. Based on the close relation of (constructive) proofs and algorithms, we propose to investigate (constructive) proofs as *inverse images* of algorithms under a program extraction function. Using an existing notion of equality for algorithms, as given, for instance, in MOSCHOVAKIS's theory of algorithms [26], one can expect to obtain a corresponding notion of equality for (constructive) proofs which should meet certain requirements we expect for such an equality.

## 2 The computer challenge

The use of computers in mathematical theorem proving is an issue which is becoming more and more important. In this field one can distinguish two different directions: *Proof search* and *Proof check*. Proof search suffers from the well-known complexity problems and is not very promising for general mathematical problems.<sup>1</sup> While it might be very hard to find a proof, to *check* a proof for correctness is, in general, of lower complexity, although it can be rather technical and long. Thus, proof check seems to be a perfect application for computers.

In a recent book [36] (see also our review [18]), edited by WIEDIJK, seventeen theorem provers are presented how they *prove* the irrationality of  $\sqrt{2}$ . This book gives an interesting insight into the state of the art of theorem proving, showing that in particular the *proof representation* is still far from being satisfactory for a human reader. In the introduction the editor presents a six line proof of a theorem, taken from the textbook of HARDY and WRIGHT [15, p. 39f]:

“The traditional proof ascribed to Pythagoras runs as follows. If  $\sqrt{2}$  is rational, then the equation

$$a^2 = 2b^2$$

is soluble in integers  $a, b$  with  $(a, b) = 1$ . Hence  $a^2$  is even, and therefore  $a$  is even. If  $a = 2c$ , then  $4c^2 = 2b^2$ ,  $2c^2 = b^2$ , and  $b$  is also even, contrary to the hypothesis that  $(a, b) = 1$ .”

This proof should be understandable for everybody with basic mathematical knowledge. The interesting thing is that the editor seems to see the overall objective of proofs in *proof check* when he writes after this proof [36, p. 3]: “Ideally, a computer should be able to take this text as input and check it for its correctness.”

We think that this perspective is misdirecting—at least with respect to mathematical proofs. Of course, the correctness of a proof is essential; but, in most cases, it is not the primary objective of the proof. Only if a proof is the very first

---

<sup>1</sup> But, of course, proof search has its merits for specialized problems which are properly modeled, in particular, with respect to applications.

of a theorem, its correctness is the first question. Of course, we do not like to see faulty proofs later on, but when a theorem is accepted in the mathematical community as true, most of its proofs have the role to *convince* the reader of the correctness of the theorem. Here, the *representation* of a proof is essential. Over centuries, mathematicians developed a rather sophisticated, however implicit, standard to represent proofs which allow to convince other mathematicians. It is just a matter of fact, that theorem provers still do not have the ability to represent proofs in a way that they can compete with usual textbook proofs. To guarantee correctness, they have to take into account too many details, details which a mathematician does not like to see exposed in the proof. This was formulated by SCOTT [33, p. ix] as follows: “[F]or verification (...) *checkable proofs* have to be generated and archived. Computers are so fast now that hundreds of pages of steps of simplifications can be recorded even for simple problems. Hence, we are faced with the questions, ‘What really is a proof?’ and ‘How much detail is needed?’”.

The two questions of SCOTT can be described as the *computer challenge*.

Let us mention here one particular shortcoming of proofs given by computers. They seem to be inadequate for an exchange between different proof systems. Thus, we would like to challenge the theorem prover community to provide *interfaces* which allow to transfer proofs performed in one theorem prover to another. Of course, many systems differ significantly in the underlying logic, the internal representation of datatypes, automatic components, user defined extensions, etc. However, a proof—in the sense we look for—of the irrationality of  $\sqrt{2}$  should not depend on any of these particularities. Thus, if there is something like an abstract proof of this theorem, the theorem provers should be able to give such one, and they should be able to exchange it among each other. We conjecture that a very lot of the disturbing details, which most likely depend on the specific implementation of a theorem prover, would be filtered out in the proof representation suitable for computer interaction.<sup>2</sup>

At present, we consider that computers are still not able to give us a satisfactory representation of proofs, as we would like to have for mathematical theorems.<sup>3</sup> The critique can be even extended to *formalized* proofs in general.<sup>4</sup> We will not discuss this issue at length here, but let us mention only two indications for it: First, mathematical textbooks as well as research papers still avoid formalized proofs as much as possible; they serve, as everybody knows, rarely for didactical reasons or to convince the reader of the correctness of an argument. Secondly, one should have a look how we examine proofs: We would hardly be

---

<sup>2</sup> AVIGAD [3] proposes to use *methods* (or *tactics*) as available in theorem provers as *Isabelle* for such a purpose. This is definitely a step in the right direction, but we think, that the methods have still not the sufficient level of abstraction. In particular, they depend too much on the particulars of the implementation.

<sup>3</sup> We are aware, that this opinion is not shared by everybody, in particular, several advocates of computer-aided theorem provers. Let us clarify, that here we do not refer to the question of the *correctness* of proofs, but only to the question of (human understandable) representation of proofs.

<sup>4</sup> Cf., for example, [3, p. 129].

satisfied if a student would give us the raw text of a formalized proof. But, our principle argument against formalized proofs is that it fails to provide a satisfactory notion of *equality of proofs*, which goes beyond the literal equality.<sup>5</sup>

### 3 Equality of proofs

It seems to be a deficiency of *proof theory* to not be able, up to today, to provide a satisfying criterion for equality of proofs.

The only proposal which is generally discussed is PRAWITZ's *normalization conjecture*, which proposes to identify two proofs with the same normal form, [30, 31]. A good discussion of the proposal can be found in DOŠEN [12]<sup>6</sup>. It also includes the “rather neglected” [12, p. 477] *generalization proposal* of LAMBEK, which is stated in terms of category theory. Both are of interest in logic and category theory; however, they failed to apply to mathematical proofs. This is admitted by DOŠEN when he writes [12, p. 500]: “Faced with two concrete proofs in mathematics—for example, two proofs of the Theorem of Pythagoras, or something more involved—it could seem pretty hopeless to try to decide whether they are identical just armed with the Normalization Conjecture or the Generality Conjecture.” Then, however, he expresses some hopes: “But this hopelessness might just be the hopelessness of formalization. We are overwhelmed not by complicated principles, but by sheer quantity.”

But, the normalization conjecture suffers from a conceptional problem: although it might be an interesting technical *logical* approach, it definitely is not the notion of equality of proof we can accept in mathematics. By the opposite, the introduction of a lemma is often considered as the specific achievement in a new proof of a theorem.<sup>7</sup>

About the general claims that category theory and/or proof nets are able to solve the problem of equality of proofs, cf., for instance, [11, 35], we like to remark that these approaches are mainly concerned with the *logical* structure of a proof, but not with mathematical arguments. This is often even admitted, or at least posed as a challenge, as in the citation of DOŠEN given above, or by STRASSBURGER [35, §4]: “Let me finish with mentioning some of the questions that are still waiting for an answer: [...] Are these identifications useful from the point of view of mathematics (i.e., can we use them for identifying real mathematical proofs)?”

To sharpen the question of equality a fundamental distinction should be made first: whether we consider proofs of *one theorem*, or proofs of *different theorems*.

---

<sup>5</sup> Of course, there are proposals around for other notions of equality of proofs, first of all, PRAWITZ's normalization conjecture. However, we do not consider them as satisfactory; see the discussion below.

<sup>6</sup> This article also contains comprehensive references to the normalization conjecture and to the generalization proposal which we don't repeat here.

<sup>7</sup> MOSCHOVAKIS (personal communication) suggested that one should identify the normalform of a proof with *its denotation* rather with *its meaning*. For our question, however, it is the meaning we would like to capture.

For two different theorems (most likely with some structural similarities) we might have proofs which are considered “essentially the same”, in the sense that they are substitutional variants of each other or that they follow the same pattern etc. This case is, for instance, investigated by use *skeletons* of (formal) proofs, i.e., proof trees in which the formulas at the leafs are replaced by variables, cf. e.g. [19, 5]. With certain unification methods, BAAZ [4] showed how different proofs can be identified as instances of the same skeleton, and, *in this respect* they can be considered as equal. However, there are more informal notions of equality of proofs in different areas. It is a standard experience that mathematicians, as soon as they realize *any* equality, tend to abstract from the concrete cases and to build a theory which encapsulates the common part of the different instances. This is also the case for proofs: BAAZ<sup>8</sup> pointed to the example of *universal algebra* which grows out of the experience that the *proof* of the homomorphism theorem is “always the same” in the different algebras. This might be a very good example to show that mathematicians indeed already have a tool to “define” equality of proofs, just by introducing an abstract theory which identifies the “different” instances.<sup>9</sup>

The question when two proofs of one theorem are equal seems to be more subtle. Very often we can easily decide that two proofs are *different*, just by identifying some clear differences. But to give a theoretical account for criteria of equality is much more complicated. One might separate two levels here: the *structure* of a proof, and the *demonstration of single steps* within a large proof. The problem is that there is no clear way to separate these two aspects. One attempt could be to relate them to the differentiation made by theorem provers between *tactics* (methods) and the actual proof terms. The tactic should represent the general structure of a proof, while single steps can (often) be given to an automatic problem solver unit. As the separation might be, we now have even two questions: what does it mean that the structure of two proofs is equal? And, what does it mean that two demonstrations of a single proof step are equal?

In the following we will compare the question of equality of proofs with the question of equality for algorithms.

## 4 What is an algorithm?

It exists a field very closely related to proofs with analogous questions: there is no commonly accepted abstract notion of an *algorithm*. In the next section we will discuss how the field of algorithms can be directly linked to the field of proofs via the *proofs-as-programs* paradigm. Before, we like to review shortly an

---

<sup>8</sup> Personal communication.

<sup>9</sup> It is an interesting observation that this form of *unifying* different theories implies that there are common patterns in the *proofs*, not only in the *theorems*: since the abstract theorem should have *one* proof in the abstract theory, it will be possible to give the “same” proof as instantiations of this abstract one for all instances, at least *ex post*.

answer to the question *what an algorithm is* given by MOSCHOVAKIS, [21, 22, 24–26].

In [26, §3], entitled *The Insufficiency of Machine Models*, MOSCHOVAKIS argues convincingly on the basis of the *mergesort* algorithm, given by a recursive specification, that machine models are not good for defining algorithms. Assuming that algorithms are machines, he emphasizes two problems [26, p. 924f]:

“Now the *first* obvious problem is that there are many compilation procedures, and so we don’t get one, but many machines with competing claims ‘to be’ the mergesort algorithm. Moreover, there are essential differences among these machines, .... On an intuitive level, these machines are, of course, equivalent, but it is hard to see how to make this notion of equivalence precise, ....; and if we could make the relevant equivalence relation precise, then one could argue that *the mergesort algorithm is the appropriate equivalence class* (which is much wider than machine isomorphism), and not any particular member of it. “One might try to get out of this dilemma by choosing some one, ‘natural’, ‘most general’ machine which implements the mergesort, .... It is not clear how that could be done in a systematic way for all recursive algorithms, but in any case, it would not suffice: because we want to know that the conclusion of Prop. 3.1 [some formal properties of mergesort] holds for all ‘implementations of the mergesort’, not just the most general one, ....”

“The *second* problem is that the details of particular implementations are irrelevant for the elementary proof of Prop. 3.1, ....”

It is striking that these problems are the same we encounter for mathematical proofs, if we would like to identify them with formal proofs: there are many formal systems around, as *Hilbert-style calculi*, *sequent calculi*, *natural deduction*, *dialogues*, etc., and neither we can distinguish one as the ‘natural’ or ‘most general’ one nor we have any reasonable idea what could be the equivalence relation over different calculi. The second problem relates to the fact that formal proofs—in particular the ones given by theorem provers—lavish us with irrelevant details.

In contrast to machines, MOSCHOVAKIS proposes to define algorithms in terms of *recursors* which are associated with recursion equations. He stresses the non-syntactic character of recursors saying “...I have avoided the word ‘definition’ in their name since it suggests syntactical objects, which algorithms are not” and adding the footnote: “Recursors are related to systems of recursive equations in the same way that differential operators are related to differential equations.” [26, p. 925, footnote 11]. Even if he is quite cautious with claiming to have found a definite answer to the question what an algorithm is,<sup>10</sup> to our knowledge, this is the first and only proposal which provides a—non-trivial and non-useless—approach to equality of algorithms, [26, §8.2], (which, concededly, still needs further elaboration).

---

<sup>10</sup> He writes: “I would not claim, however, that mine is the only approach, or the best approach — or, perhaps, even an adequate approach: my chief goal is to convince the reader that the problem of founding the theory of algorithm is important, and that it is ripe for solution.” [26, p. 920].

In view of the similarity of the situation with respect to proofs and algorithms, it seems to be promising to investigate whether, and if so, to which extend, MOSCHOVAKIS's abstract notion of algorithm can give rise to an abstract notion of proof. In the following section we discuss in which way such a relation could be established.

## 5 Reverse Proofs-as-programs

The *proofs-as-programs* paradigm can be taken directly from the *Curry-Howard-isomorphism*. Writing (constructive) proofs as  $\lambda$  terms, they are in principle already functional programs.<sup>11</sup>

Based on this theoretic relation, the practical implementation of *proofs-as-programs* is already quite advanced. As one of the first implementations, we just mention BATES and CONSTABLE's system PRL, [6]. Nowadays, every good proof system implements a proofs-as-programs component.<sup>12</sup> In this process, the main emphasize is put on the question how *constructive content*, needed for the programs, can be extracted from proofs using *classical logic*. For our purpose the particular logical framework is not important; all we need is that the programs are generated from a proof by use of a given *program extraction function*.

Our main suggestion is to attack the questions of *what is a proof* and how equality of proofs can be defined by use of *reverse engineering* of the extraction function:

*Given MOSCHOVAKIS's notion of algorithm, we have to study the inverse images of such algorithms under a program extraction function.*

The hope is to find (one-to-one) correspondences between concepts about algorithms and about proofs. From the “non-reverse” proofs-as-program perspective, some interesting examples concerning the principle of mathematical induction can already be found in [20]. A very concrete instance of a particular concept we are interested is, is given in [6, p. 115]: “Another possibility in problems of this sort is that some property of  $A(n, p)$  can be generalized to add an extra parameter, say  $A(m, n, p)$ , and then we can use induction on  $m$ . This technique is called *generalization* in Polya [29]; Dijkstra [10], Gries [14], and Reynolds [32] call it *weakening* in the context of programming.”

If one focus on MOSCHOVAKIS's notions introduced for algorithms, at best, this should lead in a natural way to a concept which corresponds to *recursors* on the side of the proofs. Also, one can hope to get an analog of the equality relation given for recursors.

In its first step, this analysis applies to proofs only which give rise to algorithms, first of all, constructive proofs of  $\Pi_2^0$  statements. However, in any

<sup>11</sup> There exists a lot of literature about the Curry-Howard-isomorphism which we will not include here. As a standard reference, we can refer to [34].

<sup>12</sup> We restrict ourselves to refer to the book [27], which contains a lot of additional references.

reasonable account to equality of proof, however a notion of equality for such constructive proofs will look like, this notion has to be part of the notion of equality of proofs in general. Therefore, we do not see this approach as restrictive, but, on the contrary, as a step which provides, firstly a necessary condition for a general notion of equality, and secondly as a heuristic guideline which probably can give rise to an extension to general proofs.

In any case, there are already several questions which have to be checked: neither it is clear whether the inverse image of a given algorithm is unique—one might even wonder if there always is one—, nor whether a given extraction function is canonical or whether the same proof can give rise to different algorithms.<sup>13</sup>

A negative answer to the first of these questions would mean that we have two proofs which are considered as different, but the extracted algorithms are equal. In such a case, it would be interesting to see on which parts the difference of these proofs is based.

A negative answer to the second question would mean that the extraction function itself has to be analyzed more carefully, because it would *add* computationally relevant components to a proof.

All in all, our proposal of reverse engineering on program extraction should not only provide insights in the notion of proof, but also contribute to a better understanding of program extraction, algorithms, and verification proofs.

*Note added in proof.* A referee drew our attention to the paper *Problems in the logic of provability* of BEKLEMISHEV and VISSER [7] of which we were not aware while writing this text. In fact, their consideration about an *Informal concept of proof* in section 2, coincide to a large extend with our view. BEKLEMISHEV and VISSER propose to relate the notion of proof with the notion of algorithm as given in the theory of *Abstract State Machines* by GUREVICH with references to [8, 9]. Of course, our proposal should work, in principle, with any theory of algorithms which provide a notion of equality. We just proposed MOSCHOVAKIS’s theory because we are more familiar with it. A comparison of both, GUREVICH’s and MOSCHOVAKIS’s, theories would be desirable in any case, but also with respect to its implications for a notion of equality of proofs. About the abstract theory of provability (or also the more specific logic of proofs of ARTËMOV, [2]) which is proposed as an account in this context by BEKLEMISHEV and VISSER we are rather sceptical, since it seems to be formulated on a “too abstract” level. However, the fruitfulness of it is still subject to research.

## 6 The broader view: Sense and Denotation as Proof and Truth

It is an old question why the two equations  $a = a$  and  $a = b$  should have a different epistemological status, if  $a$  and  $b$  refer to (denote) the same object.

---

<sup>13</sup> MOSCHOVAKIS raised the question of the relation between (constructive) proofs and algorithms already in [25, §3.4].

This question, still asked by POINCARÉ [28], gave reason to FREGE’s seminal invention of the distinction between *sense* and *denotation*, [13]. This distinction is discussed in length in philosophical logic, and there are several attempts to formalize *sense* within *intensional logic*, cf. e.g., [1]. Up to today, these approaches, however, did not succeed to provide a satisfactory analysis.

An alternative approach was suggested by MOSCHOVAKIS who analyses sense and denotation as *algorithm* and *value*, [23]:

“In contemporary computing terms, we have defined an *algorithm* which computes the truth value of  $\chi$ . . . This algorithm is the *referential intension* of just **intension** of  $\chi$ ,

$int(\chi)$  = the algorithm which computes the truth value of  $\chi$ ,

and we propose to model the sense of  $\chi$  by its referential intension.”

Of course, for the underlying notion of algorithm, MOSCHOVAKIS suggests to use the notion discussed above.

At other places we considered to relate intentional phenomena to *proofs* while the extensional counterpart is related to *truth*. Thus our reading of a sense of a formula  $\phi$  would be a proof of  $\phi$ ; in this context, obviously, a formula can have different senses if it has different proofs. This approach depends on the underlying axiom system (or better: the underlying axiom system is implicitly part of the sense of formula). But, with this reading the epistemological difference of, for instance,  $2 = 2$  and  $2 = \sqrt{4}$  is obviously on the level of senses: the first one has a trivial proof—it is an axiom (in any reasonable axiom system)—; the second one requires some deduction steps.

Some first applications of our proposal are given in [17] with respect to necessity, and in [16] with respect to belief revision. However, these approaches are still informal, and a better theory of proofs is needed, a theory which we are looking for in this paper.

## References

1. C. Anthony Anderson. General intensional logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic*, volume II, pages 355–385. Kluwer, 1984.
2. S. Artemov. Logic of proofs. *Annals of Pure and Applied Logic*, 67:29–59, 1994.
3. J. Avigad. Mathematical method and proof. *Synthese*, 153:105–149, 2006.
4. M. Baaz. Note on the generalization of calculations. *Theoretical Computer Science*, 224(1–2):3–11, 1999.
5. M. Baaz and P. Pudlák. Kreisel’s conjecture for  $l\exists_1$ . In P. Clote and J. Krajíček, editors, *Arithmetic Proof Theory and Computational Complexity*, pages 30–49. Oxford University Press, 1993.
6. J. L. Bates and R. L. Constable. Proofs as programs. *ACM Transactions on Programming Languages and Systems*, 7(1):113–136, 1985.
7. L. Beklemishev and A. Visser. Problems in the logic of provability. In D. M. Gabbay, S. S. Goncharov, and M. Zakharyaschev, editors, *Mathematical Problems from Applied Logic*, volume I, pages 77–136. Springer, 2006.

8. A. Blass and Y. Gurevich. Algorithms vs. machines. *Bulletin of the European Association for Theoretical Computer Science*, 77:96–118, 2002.
9. A. Blass and Y. Gurevich. Algorithms: A quest for absolute definitions. *Bulletin of the European Association for Theoretical Computer Science*, 81:195–225, 2003.
10. E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.
11. K. Došen and Z. Petrić. *Proof-Theoretical Coherence*. KCL Publications, 2004.
12. K. Došen. Identity of proofs based on normalization and generality. *The Bulletin of Symbolic Logic*, 9(4):477–503, 2003.
13. G. Frege. Über Sinn und Bedeutung. *Zeitschrift für Philosophie und philosophische Kritik*, (NF 100):25–50, 1892.
14. D. Gries. *The Science of Programming*. Springer, 1982.
15. G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford, 4th edition, 1960.
16. R. Kahle. Structured belief bases. *Logic and Logical Philosophy*, 10:49–62, 2002.
17. R. Kahle. A proof-theoretic view of necessity. *Synthese*, 148(3):659–673, 2006.
18. R. Kahle. Review of: Freek Wiedijk (editor), *The Seventeen Provers of the World*. Springer, 2006, [36]. *Studia Logica*, 87:369–374, 2007.
19. J. Krajíček and P. Pudlák. The number of proof lines and the size of proofs in first order logic. *Archive for Mathematical Logic*, 27:69–84, 1988.
20. Z. Manna and R. J. Waldinger. Toward automatic program synthesis. *Communications of the ACM*, 14:151–165, 1971.
21. Y. Moschovakis. The formal language of recursion. *The Journal of Symbolic Logic*, 54:1216–1252, 1989.
22. Y. Moschovakis. A mathematical modeling of pure, recursive algorithms. In A. R. Meyer and M. A. Taitslin, editors, *Logic at Botik '89*, pages 208–229. Springer, 1989.
23. Y. Moschovakis. Sense and denotation as algorithm and value. In J. Oikkonen and J. Väänänen, editors, *Logic Colloquium '90*, pages 210–249. Springer, 1994.
24. Y. Moschovakis. A game-theoretic, concurrent and fair model of the typed lambda-calculus, with full recursion. In M. Nielsen and W. Thomas, editors, *CSL '97*, pages 341–359. Springer, 1998.
25. Y. Moschovakis. On founding the theory of algorithms. In H. D. Dales and G. Oliveri, editors, *Truth in mathematics*, pages 71–104. Clarendon Press, 1998.
26. Y. Moschovakis. What is an algorithm? In B. Engquist and W. Schmid, editors, *Mathematics unlimited — 2001 and beyond*, pages 919–936. Springer, 2001.
27. I. H. Poernomo, J. N. Crossley, and M. Wirsing. *Adapting Proofs-as-Programs*. Springer, 2005.
28. H. Poincaré. *la Science et l'Hypothèse*. Flammarion, 1902.
29. G. Polya. *How To Solve It*. Princeton University Press, 1945.
30. D. Prawitz. *Natural Deduction*. Almqvist and Wiksell, 1965.
31. D. Prawitz. Ideas and results in proof theory. In J. E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 235–307. North-Holland, 1971.
32. J. C. Reynolds. *The Craft of Programming*. Prentice-Hall, 1981.
33. D. Scott. Foreword. In F. Wiedijk, editor, *The Seventeen Provers of the World*, pages vii–xii. Springer, 2006.
34. M. H. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard isomorphism*. Elsevier, 2006.
35. L. Straßburger. What is a logic, and what is a proof? In Jean-Yves Beziau, editor, *Logica Universalis*, pages 135–145. Birkhäuser, 2005.
36. F. Wiedijk, editor. *The Seventeen Provers of the World*. Springer, 2006.

# Plotkin Definability Theorem for Atomic-Coherent Information Systems

Basil A. Karádais

Mathematisches Institut, Ludwig-Maximilians-Universität  
Theresienstraße 39, 80333 München, Germany  
[karadais@math.lmu.de](mailto:karadais@math.lmu.de)

**Abstract.** By a *Plotkin definability theorem* we mean here a statement of the following sort: a PCF-like language can be extended to a language so that a functional will be definable by a term if and only if it is computable [1–4]. We prove this result directly for a class of *Scott information systems* [5] that feature “atomicity” and “coherence”.

**Key words:** Atomic-Coherent Information Systems, Partial Computable Functionals, PCF Definability.

## 1 Partial Continuous Functionals

In late seventies, Gordon Plotkin [1] proved PCF-definability of computable functions on flat domains. What we prove in section 2 directly is an analogous result in the slightly but interestingly different setting of “atomic-coherent information systems”. These were introduced in [6] to serve as a better yet interpretation of data types than general information systems: they induce domains which are nonflat, in a way that allows constructors to be injective and to have disjoint ranges. Moreover, and interestingly enough, it can be shown that these domains are exactly the “coherent” ones (see Appendix), a notion that also goes back to Plotkin and the late seventies [7].

In this section we list notions and facts that form the raw material for the main result, drawing mainly from [6, §2.2]; for further preliminary material see Appendix. In the second section we proceed to our main theorem, namely, a direct proof of Plotkin’s definability result for atomic-coherent information systems; contrary to what one might expect, we will see that the proof goes through in a substantially more intricate way than in the flat case.

**Basic notions and facts.** The information systems we will use are simple in that all discussion of consistence and entailment of information can be conducted on the primary level of atomic pieces of information, just by using binary relations. An *atomic-coherent information system*—from now on *acis*<sup>1</sup>—is a triple  $\alpha = (T_\alpha, \diamond_\alpha, \triangleright_\alpha)$ , where  $T_\alpha$  is a nonempty countable set of *atomic pieces of*

---

<sup>1</sup> We will be using the term as a proper english noun, hence allowing for the forms *acises* for the plural and *acis's* for the possessive.

*information* or just *atoms* (or *tokens*),  $\diamond_\alpha$  is the *consistency*, a reflexive and symmetric binary relation on  $T_\alpha$  and  $\triangleright_\alpha$  is the *entailment*, a reflexive and transitive binary relation on  $T_\alpha$ , so that *consistency propagates through entailment*, that is,

$$\forall_{a,b,c \in T_\alpha} . a \diamond_\alpha b \wedge b \triangleright_\alpha c \rightarrow a \diamond_\alpha c .$$

We will drop the subscripts in places we can afford to. For  $U, V \subseteq T$  adopt the following shorthands:  $U \diamond a := \forall_{b \in U} b \diamond a$ ,  $U \diamond V := \forall_{a \in V} U \diamond a$ ,  $a \triangleright U := \forall_{b \in U} a \triangleright b$ ,  $U \triangleright a := \exists_{b \in U} b \triangleright a$ ,  $U \triangleright V := \forall_{a \in V} U \triangleright a$ . Call  $U \subseteq^f T_\alpha$  a (*formal*) *neighborhood* (or *finite approximation* or *consistent set*), and write  $U \in \text{Con}_\alpha$ , if  $a \diamond_\alpha b$  for all  $a, b \in U$ . Call  $u \subseteq T_\alpha$  an *ideal (set)* (or *element* or *point*), and write  $u \in \text{Ide}_\alpha$ , if  $a \diamond_\alpha b$  for all  $a, b \in u$ , and, whenever  $a \triangleright_\alpha b$  for some  $a \in T_\alpha$ , then  $b \in u$ . In the following we will try to keep the notation fixed: we will write  $a, b, \dots$  for atoms,  $U, V, \dots$  for consistent sets,  $X, Y, \dots$  for arbitrary finite sets of atoms, and  $u, v, \dots$  for infinite sets of atoms, mainly ideals.

**Lemma 1.** *Let  $\alpha = (T, \diamond, \triangleright)$  be an acis. For  $a, b \in T$ ,  $U, V, W \in \text{Con}$ ,  $u, v \in \text{Ide}$ , the following hold.*

1.  $a \triangleright b \rightarrow a \diamond b$ ,
2.  $a_1 \triangleright b_1 \wedge a_2 \triangleright b_2 \wedge a_1 \diamond a_2 \rightarrow b_1 \diamond b_2$ ,
3.  $U_1 \triangleright V_1 \wedge U_2 \triangleright V_2 \wedge U_1 \diamond U_2 \rightarrow V_1 \diamond V_2$ ,
4.  $U \diamond V \wedge V \triangleright W \rightarrow U \diamond W$ ,
5.  $u \triangleright v \leftrightarrow v \subseteq u$ .

Denote the *empty ideal*  $\emptyset \in \text{Ide}$  by  $\perp$ . For a finite set of atoms  $X \subseteq^f T$ , define its (*deductive*) *closure* by  $\overline{X} := \{a \in T \mid X \triangleright a\}$  and the *cone (of ideals)* above it by  $\nabla X := \{u \in \text{Ide} \mid X \subseteq u\}$ . Denote by  $\text{Con}$  the class of all closures of formal neighborhoods and by  $\text{Kgl}$  the class of all cones in the acis.

**Lemma 2.** *Let  $X, Y \subseteq T$  and  $U, V \in \text{Con}$ .*

1.  $X \in \text{Con} \leftrightarrow \overline{X} \in \text{Ide}$ ,
2.  $\overline{\overline{X}} = \overline{X \cup Y}$ ,
3.  $X \diamond Y \leftrightarrow \overline{X} \diamond \overline{Y}$ ,
4.  $X \triangleright Y \leftrightarrow \overline{X} \supseteq \overline{Y}$ ,
5.  $U \diamond V \rightarrow \nabla U \cap \nabla V = \nabla(U \cup V)$ .

Let  $\alpha$  and  $\beta$  be two acises. Define their *function space*  $\alpha \rightarrow \beta$  by  $T_{\alpha \rightarrow \beta} := \text{Con}_\alpha \times T_\beta$ ,  $(U, a) \diamond_{\alpha \rightarrow \beta} (V, b)$  when  $U \diamond_\alpha V \rightarrow a \diamond_\beta b$ , and  $(U, a) \triangleright_{\alpha \rightarrow \beta} (V, b)$  when  $V \triangleright_\alpha U \wedge a \triangleright_\beta b$ . Define their *cartesian product*  $\alpha \times \beta$ , by  $T_{\alpha \times \beta} := T_\alpha \uplus T_\beta$ ,  $a \diamond_{\alpha \times \beta} b$  when either  $a, b$  belong to different acises or are consistent in the same one, and  $a \triangleright_{\alpha \times \beta} b$  when  $a$  entails  $b$  in either  $\alpha$  or  $\beta$ ; in this way we are able to approximate any  $(u, v) \in \text{Ide}_{\alpha \times \beta}$  separately in each component.

Both the function space and the cartesian product of two acises can be proven to be again acises. Moreover, one can show that the pairs  $(u, v) \in \text{Ide}_\alpha \times \text{Ide}_\beta$  are in a bijective correspondence with the ideals  $u \cup v \in \text{Ide}_{\alpha \times \beta}$ , as well as that

the *Scott continuous* mappings from  $\text{Ide}_\alpha$  to  $\text{Ide}_\beta$  (see Appendix) are exactly the ideals of  $\alpha \rightarrow \beta$  [6].

In the following we will be largely concerned with “application of ideals”. In general, define (*set*) *application*  $\cdot : \mathcal{P}(T_{\alpha \rightarrow \beta}) \times \mathcal{P}(T_\alpha) \rightarrow \mathcal{P}(T_\beta)$ , by

$$\{(X_i, b_i)\}_{i \in I} \cdot Y :=_\beta \{b_i \mid Y \triangleright_\alpha X_i\}.$$

**Lemma 3.** *For the application operation the following hold.*

1. *If  $U \in \text{Con}_{\alpha \rightarrow \beta}$  and  $V \in \text{Con}_\alpha$ , then  $U \cdot V \in \text{Con}_\beta$ . Furthermore, if  $U \diamondsuit_{\alpha \rightarrow \beta} U'$  and  $V \diamondsuit_\alpha V'$  then  $U \cdot V \diamondsuit_\beta U' \cdot V'$ .*
2. *For all  $V \in \text{Con}_\alpha$ , it is  $U \triangleright_{\alpha \rightarrow \beta} U'$  if and only if  $U \cdot V \triangleright_\beta U' \cdot V$ .*
3. *For all  $U \in \text{Con}_{\alpha \rightarrow \beta}$ , if  $V \triangleright_\alpha V'$  then  $U \cdot V \triangleright_\beta U \cdot V'$ .*
4. *It is  $\overline{U \cdot V} := \overline{U} \cdot \overline{V}$ .*
5. *For  $X \subseteq^f T_{\alpha \rightarrow \beta}$ ,  $Y \subseteq^f T_\alpha$  and  $Z \subseteq^f T_\beta$ , where  $\alpha$  and  $\beta$  are fixed, the relation  $Z = X \cdot Y$  is  $\Sigma_1^0$ -definable.*

**Arithmetical and boolean acises.** We define the acis of lazy natural numbers as follows. Let  $*$  be a *preatom*, intuitively meaning least information. Natural numbers given as usual by the constructors  $0$  and  $S$  give rise to an acis  $\mathbb{N}$  as follows:  $a \in T_{\mathbb{N}}$  if  $a = *$ , or  $a = 0$ , or  $a = Sa'$  for some  $a' \in T_{\mathbb{N}}$ ;  $a \diamondsuit_{\mathbb{N}} b$  if  $a = *$ , or  $b = *$ , or  $a = b = 0$ , or  $a = Sa' \wedge b = Sb'$  for  $a' \diamondsuit_{\mathbb{N}} b'$ ;  $a \triangleright_{\mathbb{N}} b$  if  $b = *$ , or  $a = b = 0$ , or  $a = Sa' \wedge b = Sb'$  for  $a' \triangleright_{\mathbb{N}} b'$ . Similarly, the boolean numbers  $\text{tt}$  and  $\text{ff}$  give rise to an acis  $\mathbb{B}$  as follows:  $b \in T_{\mathbb{B}}$  if  $b = *$ , or  $b = \text{tt}$ , or  $b = \text{ff}$ ;  $b_1 \diamondsuit_{\mathbb{B}} b_2$  if  $b_1 = *$ , or  $b_2 = *$ , or  $b_1 = b_2 = \text{tt}$ , or  $b_1 = b_2 = \text{ff}$ ;  $b_1 \triangleright_{\mathbb{B}} b_2$  if  $b_2 = *$ , or  $b_1 = b_2 = \text{tt}$ , or  $b_1 = b_2 = \text{ff}$ . Note that the least info atom  $*$  is not considered to be a proper atom and that we will count it out whenever we talk about atoms, unless otherwise stated; on the level of ideals, least info is expressed by  $\perp$ .

Acises which are function spaces whose target is  $\mathbb{N}$  will be called *arithmetical acises*. In particular, we think of ideals of  $\mathbb{N}$  as *partial numbers* and ideals of an arithmetical function space as *partial continuous functionals*. Write  $\infty$  for the ideal  $\{\dots, S^n*, \dots, S*\}$  and  $n$  for the ideal  $\{S^n0\}$ . Furthermore, we take the risk of overloading the symbols  $\text{tt}$  and  $\text{ff}$  to mean the corresponding ideals of  $\mathbb{B}$  as well. We refer to function spaces consisting of both arithmetical and boolean acises as *arithmetical-boolean acises*.

Define the *total ideals*  $G_\alpha \subseteq \text{Ide}_\alpha$ , in an arithmetical acis  $\alpha$ , inductively as follows:  $u \in G_{\mathbb{N}}$  if  $u = 0$ , or  $u = Sv$  for some  $v \in G_{\mathbb{N}}$ ;  $u \in G_{\alpha \rightarrow \beta}$  if  $u(v) \in G_\beta$  for all  $v \in G_\alpha$ . In what follows, the total numbers  $0, 1, \dots$  will mainly serve as indices.

## 2 Computable Functionals

Call a partial continuous functional *computable* if it is  $\Sigma_1^0$ -definable as a set of atoms. It is direct to check (cf. Appendix) that evaluation and currying functionals are computable, that composition, application and cartesian products of computable functionals are computable, that projections are computable, and, consequently, that the functional  $\llbracket M \rrbracket$  is computable for any term  $M$ .

**Recursion on parallel conditionals and existentials.** We pave the road for the definability statement by introducing the standard fixed point and parallel functionals. The proofs of the corresponding statements found in [3] are easy to adapt to our nonflat setting.

Let  $\alpha$  be an arbitrary acis and  $f : \alpha \rightarrow \alpha$  a continuous mapping. An ideal set  $u \in \text{Id}_{\alpha}$  is said to be the *least fixed point* of  $f$  if  $f(u) = u$  and  $f(v) = v \rightarrow u \subseteq v$  for all  $v \in \text{Id}_{\alpha}$ . One can prove that the mapping  $f$  has a least fixed point given by the equation  $\Upsilon(f) = \bigcup_{n \in G_{\mathbb{N}}} f^n(\perp)$ , and that the *fixed point functional*  $\Upsilon : (\alpha \rightarrow \alpha) \rightarrow \alpha$  is continuous and computable for any  $\alpha$ .

From now on we restrict our attention to arithmetical-boolean acises. Define the *parallel conditional functional*  $\text{pcond} : \mathbb{B} \rightarrow \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$  by

$$\text{pcond}(p, u, v) := \begin{cases} u & p = \text{tt} \\ v & p = \text{ff} \\ u \cap v & p = \perp \end{cases} .$$

One can prove that the parallel conditional functional is continuous and computable.

Define the *existential functional*  $\text{exist} : (\mathbb{N} \rightarrow \mathbb{B}) \rightarrow \mathbb{B}$  by

$$\text{exist}(u) = \begin{cases} \text{ff} & u(\perp) = \text{ff} \\ \text{tt} & \exists_{n \in G_{\mathbb{N}}} u(n) = \text{tt} \\ \perp & \text{otherwise} \end{cases} .$$

For the first clause of the definition, notice that whenever  $u(\perp) = \text{ff}$ , it should be  $v(\perp) = \text{ff}$  for all ideals  $v \triangleright_{\mathbb{N} \rightarrow \mathbb{B}} u$ . Again, one can prove that the existential functional is continuous and computable.

Call a partial continuous functional  $u \in \text{Id}_{\alpha_1 \rightarrow \dots \rightarrow \alpha_p \rightarrow \mathbb{N}}$  *recursive in*  $\text{pcond}$  and  $\text{exist}$  if it can be defined explicitly for all arguments  $v_1, \dots, v_p$  by an equation

$$u(v_1, \dots, v_p) = t(v_1, \dots, v_p)$$

where  $t$  is a simply-typed lambda term (cf. Appendix) built up from variables  $v_1, \dots, v_p$ , algebra constructors, fixed point functionals, parallel conditional functionals and existential functionals.

A functional we will need in what follows is the *disjunction functional*  $\text{or} : \mathbb{B} \rightarrow \mathbb{B} \rightarrow \mathbb{B}$  defined by

$$\text{or}(p, q) := \text{pcond}(p, \text{tt}, q)$$

which unfolds to

$$\text{or}(p, q) := \begin{cases} \text{tt} & p = \text{tt} \vee q = \text{tt} \\ \text{ff} & p = q = \text{ff} \\ \perp & \text{otherwise} \end{cases} .$$

This is a continuous and computable functional, since it is defined by  $\text{pcond}$ . Its  $p$ -ary generalization we will denote by  $\text{OR}_{i=1}^p$ .

**Enumeration and inconsistency functionals.** We begin with two lemmas.

**Lemma 4.** *The following hold.*

1. *For all  $a, b \in T_{\mathbb{N}}$ , we have the following conditional dichotomy property:*  $a \diamond_{\mathbb{N}} b \rightarrow (a \triangleright_{\mathbb{N}} b \vee b \triangleright_{\mathbb{N}} a)$ .
2. *For an arithmetical acis  $\alpha$ , if  $\{(U_l, b_l)\}_{l \leq n} \in \text{Con}_{\alpha \rightarrow \mathbb{N}}$  and  $u \in \text{Id}_{\alpha}$ , then  $\overline{\{(U_l, b_l)\}_{l \leq n}}(u) \in \text{Con}_{\mathbb{N}}$ .*

We introduce a notion of “relative height” between partial numbers. Let  $u, v \in \text{Id}_{\mathbb{N}}$  and  $n \in G_{\mathbb{N}}$ . Say that  $u$  is above  $n$ , and write  $u \succeq n$ , if it contains information built from at least  $n$  constructors, that is, if either  $u \triangleright_{\mathbb{N}} S^{n-1}0$  or  $u \triangleright_{\mathbb{N}} S^n*$ . Say further that  $u$  is above  $v$ , and write  $u \succeq v$ , if it is above any index below  $v$ , that is, if  $v \succeq n \rightarrow u \succeq n$  for all  $n \in G_{\mathbb{N}}$ . Note that aboveness is not antisymmetric, since  $S^n0 \succeq S^{n+1}\perp$  and  $S^{n+1}\perp \succeq S^n0$  for all  $n$ . It is also obvious that aboveness between ideals is a total preorder with single maximum element  $\infty$  and single minimum element  $\perp$ , as well as that, for total ideals, it reduces to the standard  $\geq$  relation.

This straightforward notion, which is merely based on the simple tree structure of the lazy natural numbers, is nevertheless the necessary and sufficient step beyond the techniques used in the flat case. The calculus it provides is summarized in the following.

**Lemma 5.** *Concerning aboveness  $\succeq \subseteq \text{Id}_{\mathbb{N}} \times \text{Id}_{\mathbb{N}}$  the following hold.*

1. *If  $u \triangleright_{\mathbb{N}} v$  then  $u \succeq v$ .*
2. *If  $v \succeq w$  and  $w \succeq v$  then  $v = w$  for all  $v, w \subseteq u$ ,  $u \in \text{Id}_{\mathbb{N}}$ .*
3. *If  $u \diamond_{\mathbb{N}} v$  and  $u \succeq v$  then  $u \triangleright_{\mathbb{N}} v$ .*
4. *If  $u \succeq v$  and  $u \not\diamond_{\mathbb{N}} v$  then  $v \in G_{\mathbb{N}}$ .*

For every arithmetical-boolean acis we fix an enumeration of consistent sets  $\{U_n\}_{n \in G_{\mathbb{N}}}$  so that  $U_0 = \emptyset$  holds, and the following are primitive recursive relations:  $U_n \diamond_{\alpha} U_m$ ,  $U_n \triangleright_{\alpha} U_m$ ,  $U_n^{\alpha \rightarrow \beta} \cdot U_m^{\alpha} = U_k^{\beta}$  and  $U_n \cup U_m = U_k$ , with  $k = 0$  if  $U_n \not\diamond_{\alpha} U_m$ . In the following we will lift the notational convention of the first section and write  $b$  for *singleton neighborhoods*, whenever we want to stress that they behave as *atoms*.

We will use *inconsistency functionals*  $\text{incns}_{\alpha} : \alpha \rightarrow G_{\mathbb{N}} \rightarrow \mathbb{B}$ , defined by

$$\text{incns}_{\alpha}(u, n) := \begin{cases} \text{tt} & u \not\diamond_{\alpha} U_n \\ \text{ff} & u \triangleright_{\alpha} U_n \\ \perp & \text{otherwise} \end{cases} .$$

having application in mind: let  $\beta$  be an arbitrary acis and  $(U_n, b_n)$  an atom of a partial continuous functional  $v$  of type  $\alpha \rightarrow \beta$ ; the only case where the atom contributes information concerning the value of  $v(u)$  is for  $\text{incns}_{\alpha}(u, n)$  to be  $\text{ff}$ , so  $b_n$  will belong to the value.<sup>2</sup>

<sup>2</sup> To make case distinctions in subsequent proofs less branching, we adopt the following convention. Let  $\alpha$  and  $\beta$  be arbitrary arithmetical-boolean acises. An ideal mapping  $f : \alpha \rightarrow \mathbb{N} \rightarrow \beta$  that satisfies  $f(u, x, v) = \perp$  for  $x \notin G_{\mathbb{N}}$ , will be informally typed by  $\alpha \rightarrow G_{\mathbb{N}} \rightarrow \beta$ , and be treated only on its arguments where  $x$  is total.

In order to define the functionals  $\text{incns}_\alpha$ , we will further need to define an appropriate *enumeration functional*  $\text{en}_\alpha : G_{\mathbb{N}} \rightarrow \mathbb{N} \rightarrow \alpha$  to be able to enumerate all finitely generated extensions of  $\overline{U_m}$ , for  $U_m \in \text{Con}_\alpha$ .

**Proposition 6.** *Let  $\alpha$  be an acis over  $\mathbb{N}$  and  $\mathbb{B}$ .*

1. *There exists a functional  $\text{en}_\alpha : G_{\mathbb{N}} \rightarrow \mathbb{N} \rightarrow \alpha$ , with the following properties.*

$$\begin{aligned}\text{en}_\alpha(m, x) &= \overline{U_m}, \text{ when } x \notin G_{\mathbb{N}}, \\ \text{en}_\alpha(m, n) &= \overline{U_n}, \text{ when } U_n \triangleright_\alpha U_m,\end{aligned}$$

*which is recursive in  $\text{pcond}$  and  $\text{exist}$ .*

2. *The inconsistency functional  $\text{incns}_\alpha$  is recursive in  $\text{pcond}$  and  $\text{exist}$ .*

*Proof.* By induction on  $\alpha$ .

For the enumeration functionals. Let  $\alpha = \alpha_1 \rightarrow \cdots \rightarrow \alpha_p \rightarrow \mathbb{N}$ ,  $U_m \in \text{Con}_{\alpha_1 \rightarrow \cdots \rightarrow \alpha_p \rightarrow \mathbb{N}}$  and  $j, k$ , and  $h$  be primitive recursive functions such that

$$U_m = \{(U_{j(m,1,l)}, \dots, U_{j(m,p,l)}, b_{k(m,l)})\}_{l < h(m)},$$

with

$$l > l' \rightarrow \overline{b_{k(m,l)}} \succeq \overline{b_{k(m,l')}}. \quad (1)$$

In this representation we have  $U_{j(m,i,l)} \in \text{Con}_{\alpha_i}$  and  $b_{k(m,l)} \in \text{Con}_{\mathbb{N}}$ , for all  $l < h(m)$  and all  $1 \leq i \leq p$ , while  $h(m)$  denotes the number of elements of  $U_m$ .<sup>3</sup> Consider the collection  $\{U_{m,l}\}_l$  of progressive approximations to  $U_m$ ; in particular, for  $l \in G_{\mathbb{N}}$ , define

$$\begin{aligned}U_{x,l} &:= \emptyset \text{ if } x \notin G_{\mathbb{N}}, \\ U_{m,0} &:= \emptyset, \\ U_{m,l+1} &:= U_{m,l} \cup \{(U_{j(m,1,l)}, \dots, U_{j(m,p,l)}, b_{k(m,l)})\}.\end{aligned}$$

Observe that  $U_{m,h(m)} = U_m$ .

For  $\mathbf{u} \in \text{Id}_{\alpha_1 \rightarrow \cdots \rightarrow \alpha_p}$ , let  $q_{\mathbf{u},m,l}$  express whether the application  $\overline{U_{m,l+1}}(\mathbf{u})$  does not contribute information to the value of  $\overline{U_{m,l}}(\mathbf{u})$ , that is,

$$q_{\mathbf{u},m,l} := \bigvee_{i=1}^p \text{incns}_{\alpha_i}(u_i, j(m, i, l)) = \begin{cases} \text{tt} & \exists_{i=1}^p u_i \not\propto_{\alpha_i} U_{j(m,i,l)} \\ \text{ff} & \forall_{i=1}^p u_i \triangleright_{\alpha_i} U_{j(m,i,l)} \\ \perp & \text{otherwise} \end{cases}$$

---

<sup>3</sup> Observe that, since  $j$ ,  $k$ , and  $h$  are primitive recursive, the respective generated ideals are total, and so all values  $\bar{j}(m, i, l)$ ,  $\bar{k}(m, l)$ , and  $\bar{h}(m)$  will be total for total arguments. Based on this observation, we write  $j(m, i, l)$  for  $\bar{j}(m, i, l)$ , and so on, with the convention that  $m$ ,  $i$ , and  $l$ , as well as  $n$  (for  $\bar{m}$ ,  $\bar{i}$ ,  $\bar{l}$ , and  $\bar{n}$  respectively), will denote total ideals in  $\mathbb{N}$ ; for not necessarily total ones, together with  $u$  and  $v$ , we reserve  $x$  and  $y$ .

Observe that, in the last case, it is  $\exists_{i=1}^p u_i \not\triangleright_{\alpha_i} U_{j(m,i,l)}$ , but we still have  $\forall_{i=1}^p u_i \diamondsuit_{\alpha_i} U_{j(m,i,l)}$ . Similarly, let

$$q_{y,m,l} := \text{incns}_{\mathbb{N}}(y, k(m, l)) = \begin{cases} \text{tt} & y \not\triangleright_{\mathbb{N}} b_{k(m,l)} \\ \text{ff} & y \triangleright_{\mathbb{N}} b_{k(m,l)} \\ \perp & \text{otherwise} \end{cases}.$$

Observe again that in the last case, by the dichotomy property, we have  $\overline{b_{k(m,l)}} \triangleright_{\mathbb{N}} y$ . Define now an auxiliary functional  $\Psi : \alpha_1 \rightarrow \dots \rightarrow \alpha_p \rightarrow \mathbb{N} \rightarrow \mathbb{N} \rightarrow G_{\mathbb{N}} \rightarrow \mathbb{N}$  by

$$\begin{aligned} \Psi_{\mathbf{u},x}(y, l) &:= y \text{ if } x \notin G_{\mathbb{N}}, \\ \Psi_{\mathbf{u},m}(y, 0) &:= y, \\ \Psi_{\mathbf{u},m}(y, l+1) &:= \text{pcond}\left(q_{\mathbf{u},m,l}, \Psi_{\mathbf{u},m}(y, l), \right. \\ &\quad \left.\overline{b_{k(m,l)}} \cup \text{pcond}\left(q_{\Psi_{\mathbf{u},m}(y,l),m,l}, \perp, \Psi_{\mathbf{u},m}(y, l)\right)\right). \end{aligned}$$

Using Lemma 5 and Lemma 4 respectively, one can prove the following.

**Lemma 7.** *It is  $\Psi_{\mathbf{u},x}(\perp, l) = \overline{U_{x,l}}(\mathbf{u})$ .*

**Lemma 8.** *If  $\forall_{l < h(m)} \cdot q_{\mathbf{u},m,l} \neq \text{tt} \rightarrow q_{y,m,l} = q_{\mathbf{u},m,l}$ , then  $\forall_{l < h(m)} \Psi_{\mathbf{u},m}(y, l) = y$ .*

Let

$$\begin{aligned} \Phi(\mathbf{u}, x, y) &:= \Psi_{\mathbf{u},x}(y, \overline{h}(x)), \\ \text{en}(m, x)(\mathbf{u}) &:= \Phi(\mathbf{u}, m, \Phi(\mathbf{u}, x, \perp)). \end{aligned}$$

One can now easily show how the desired properties of **en** hold. The first one follows from Lemma 7. For the second one, suppose that  $U_n \triangleright_{\alpha_1 \rightarrow \dots \rightarrow \alpha_p \rightarrow \mathbb{N}} U_m$ , for  $n \in G_{\mathbb{N}}$ ; we get  $\text{en}(m, n)(\mathbf{u}) = \Psi_{\mathbf{u},m}(\overline{U_n}(\mathbf{u}), h(m))$  by Lemma 7, with  $\overline{U_n}(\mathbf{u}) \triangleright_{\mathbb{N}} \overline{U_m}(\mathbf{u})$  by Lemma 3; let  $l < h(m)$ ; if  $q_{\mathbf{u},m,l} = \text{ff}$ , then  $u_i \triangleright_{\alpha_i} U_{j(m,i,l)}$  for all  $i$ , so  $\overline{U_{m,l}}(\mathbf{u}) = \overline{b_{k(m,l)}}$ , which gives  $\overline{U_m}(\mathbf{u}) \triangleright_{\mathbb{N}} b_{k(m,l)}$ ; by hypothesis and transitivity, we have  $\overline{U_n}(\mathbf{u}) \triangleright_{\mathbb{N}} b_{k(m,l)}$ , which yields  $q_{\overline{U_n}(\mathbf{u}),m,l} = \text{ff}$ ; if, on the other hand,  $q_{\mathbf{u},m,l} = \perp$ , then, by hypothesis, it is  $u_i \diamondsuit_{\alpha_i} U_{j(m,i,l)}$  for all  $i$  and

$$\bigvee_{i=1}^p U_{j(m,i,l)} \triangleright_{\alpha_i} U_{j(n,i,l')} \wedge b_{k(n,l')} \triangleright_{\mathbb{N}} b_{k(m,l)}$$

for some  $l' < h(n)$ ; by propagation of consistency through entailment and the definition of consistency in a function space, we get

$$\overline{U_n}(\mathbf{u}) \diamondsuit_{\mathbb{N}} b_{k(n,l')} \wedge b_{k(n,l')} \triangleright_{\mathbb{N}} b_{k(m,l)},$$

which, by propagation again, gives  $\overline{U_n}(\mathbf{u}) \diamondsuit_{\mathbb{N}} b_{k(m,l)}$ , and so,  $q_{\overline{U_n}(\mathbf{u}),m,l} = \perp$ ; so Lemma 8 applies and we are done.

*For the inconsistency functionals.* Omitted. □

**Definability.** We are now in the position to prove the main result.

**Theorem 9.** *A partial continuous functional of type  $\alpha \rightarrow \mathbb{N}$  over  $\mathbb{N}$  and  $\mathbb{B}$  is computable if and only if it is recursive in  $\text{pcond}$  and  $\text{exist}$ .*

*Proof.* Since all  $\lambda$ -terms are computable and computable functionals are closed under application, it follows that any functional is computable if it is recursive in  $\text{pcond}$  and  $\text{exist}$ . For the converse, we consider here arrow types only. Let  $\Omega : \alpha_1 \rightarrow \dots \rightarrow \alpha_p \rightarrow \mathbb{N}$  be a computable functional. It will be represented as a primitive recursively enumerable set of atoms, that is,

$$\Omega = \{(U_{f_1(n)}, \dots, U_{f_p(n)}, b_{g(n)})\}_{n \in G_{\mathbb{N}}} ,$$

where, for each  $i = 1, \dots, p$ ,  $U_{f_i(n)}$  follows an enumeration of  $\text{Con}_{\alpha_i}$ ,  $b_{g(n)}$  follows an enumeration of  $\text{Con}_{\mathbb{N}}$ , and  $f_1, \dots, f_p, g$  are fixed primitive recursive functions.

For arbitrary  $\mathbf{u} \in \text{Ide}_{\alpha_1 \rightarrow \dots \rightarrow \alpha_p}$ ,  $v \in \text{Ide}_{\mathbb{N}}$ , let

$$q_{\mathbf{u}, n} := \bigvee_{i=1}^p \text{incns}_{\alpha_i}(u_i, f_i(n)) = \begin{cases} \text{tt} & \exists_{i=1}^p u_i \not\propto_{\alpha_i} U_{f_i(n)} \\ \text{ff} & \forall_{i=1}^p u_i \triangleright_{\alpha_i} U_{f_i(n)} \\ \perp & \text{otherwise} \end{cases} ,$$

$$q_{v, n} := \text{incns}_{\mathbb{N}}(v, g(n)) = \begin{cases} \text{tt} & v \not\propto_{\mathbb{N}} b_{g(n)} \\ \text{ff} & v \triangleright_{\mathbb{N}} b_{g(n)} \\ \perp & \text{otherwise} \end{cases} ,$$

and define a functional  $\omega : \alpha_1 \rightarrow \dots \rightarrow \alpha_p \rightarrow (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow G_{\mathbb{N}} \rightarrow \mathbb{N}$  by

$$\omega_{\mathbf{u}}(\psi)(n) := \text{pcond}(q_{\mathbf{u}, n}, \psi(n+1), \overline{b_{g(n)}}) \cup \text{pcond}(q_{\psi(n+1), n}, \perp, \psi(n+1)) .$$

We show that  $\Omega(\mathbf{u}) = \text{Y}(\omega_{\mathbf{u}})(0)$ . In particular, will show that both recursion on  $\omega_{\mathbf{u}}$  at 0 and  $\Omega(\mathbf{u})$  entail the very same information, that is,

$$\bigvee_{n \in G_{\mathbb{N}}} \Omega(\mathbf{u}) \triangleright_{\mathbb{N}} b_{g(n)} \leftrightarrow \text{Y}(\omega_{\mathbf{u}})(0) \triangleright_{\mathbb{N}} b_{g(n)} .$$

For the right direction, suppose that there exists an index  $n \in G_{\mathbb{N}}$  such that  $\Omega(\mathbf{u}) \triangleright_{\mathbb{N}} b_{g(n)}$ . This means that

$$\bigvee_{m \in G_{\mathbb{N}}} \bigvee_{i=1}^p u_i \triangleright_{\alpha_i} U_{f_i(m)} \wedge b_{g(m)} \triangleright_{\mathbb{N}} b_{g(n)} . \quad (2)$$

We claim that

$$\bigvee_{k \leq m} \omega_{\mathbf{u}}^{k+1}(\lambda x. \perp)(m-k) \triangleright_{\mathbb{N}} b_{g(m)} \quad (3)$$

and we prove it by induction on  $k$ . For  $k = 0$ , by (2), it is  $\omega_{\mathbf{u}}(\lambda x. \perp)(m) = \overline{b_{g(m)}} \triangleright_{\mathbb{N}} b_{g(m)}$ . For brevity, let  $v := \omega_{\mathbf{u}}^{k+2}(\lambda x. \perp)(m-k-1)$  and  $v_0 := \omega_{\mathbf{u}}^{k+1}(\lambda x. \perp)(m-k)$ ; the induction hypothesis is that  $v_0 \triangleright_{\mathbb{N}} b_{g(m)}$ . For  $k+1$  we have

$$v = \text{pcond}(q_{\mathbf{u}, m-k-1}, v_0, \overline{b_{g(m-k-1)}}) \cup \text{pcond}(q_{v_0, m-k-1}, \perp, v_0) .$$

We argue by cases on the outer boolean argument. If  $q_{\mathbf{u},m-k-1} = \text{tt}$ , then  $v := v_0 \triangleright_{\mathbb{N}} b_{g(m)}$  by induction hypothesis.

If  $q_{\mathbf{u},m-k-1} = \text{ff}$ , then  $u_i \triangleright_{\alpha_i} U_{f_i(m-k-1)}$  for all  $i$ , so  $\Omega(\mathbf{u}) \triangleright_{\mathbb{N}} b_{g(m-k-1)}$ . By (2) and Lemma 1 we get  $b_{g(m)} \diamondsuit_{\mathbb{N}} b_{g(m-k-1)}$ , and by Lemma 4 this yields

$$b_{g(m)} \triangleright_{\mathbb{N}} b_{g(m-k-1)} \vee b_{g(m-k-1)} \triangleright_{\mathbb{N}} b_{g(m)} \quad (4)$$

while  $v = \overline{b_{g(m-k-1)}} \cup \text{pcond}(q_{v_0,m-k-1}, \perp, v_0)$ . We distinguish cases on the inner boolean now. If  $q_{v_0,m-k-1} = \text{tt}$ , then  $v_0 \not\triangleright_{\mathbb{N}} b_{g(m-k-1)}$ ; if it were  $b_{g(m)} \triangleright_{\mathbb{N}} b_{g(m-k-1)}$ , by induction hypothesis, transitivity of entailment would yield  $v_0 \triangleright_{\mathbb{N}} b_{g(m-k-1)}$ , which is absurd; the dichotomy property (4) gives  $b_{g(m-k-1)} \triangleright_{\mathbb{N}} b_{g(m)}$ , and

$$v := \overline{b_{g(m-k-1)}} \cup \perp = \overline{b_{g(m-k-1)}} \triangleright_{\mathbb{N}} b_{g(m)} .$$

If  $q_{v_0,m-k-1} = \text{ff}$ , then  $v_0 \triangleright_{\mathbb{N}} b_{g(m-k-1)}$ , and

$$v := \overline{b_{g(m-k-1)}} \cup v_0 = v_0 \triangleright_{\mathbb{N}} b_{g(m)}$$

by induction hypothesis. If  $q_{v_0,m-k-1} = \perp$ , then  $v_0 \not\triangleright_{\mathbb{N}} b_{g(m-k-1)} \wedge v_0 \diamondsuit_{\mathbb{N}} b_{g(m-k-1)}$ ; if it were  $b_{g(m)} \triangleright_{\mathbb{N}} b_{g(m-k-1)}$ , we would again have a contradiction due to transitivity of entailment and induction hypothesis, so dichotomy (4) gives  $b_{g(m-k-1)} \triangleright_{\mathbb{N}} b_{g(m)}$ , and

$$v := \overline{b_{g(m-k-1)}} \cup \perp = \overline{b_{g(m-k-1)}} \triangleright_{\mathbb{N}} b_{g(m)} .$$

If now it is  $q_{\mathbf{u},m-k-1} = \perp$ , then  $u_i \diamondsuit_{\alpha_i} U_{f_i(m-k-1)}$  for all  $i$ , so by (2) and propagation we get  $U_{f_i(m-k-1)} \diamondsuit_{\alpha_i} U_{f_i(m)}$ . By definition of consistency in a function space, we get  $b_{g(m-k-1)} \diamondsuit_{\mathbb{N}} b_{g(m)}$ , and this yields the dichotomy property (4) again, so we distinguish cases on the inner boolean and argue as before.

We proved that  $v \triangleright_{\mathbb{N}} b_{g(m)}$  in all cases. Letting  $k = m$  in (3), we get

$$\omega_{\mathbf{u}}^{n+1}(\lambda x. \perp)(0) \triangleright_{\mathbb{N}} b_{g(m)} \Rightarrow \Upsilon(\omega_{\mathbf{u}})(0) \triangleright_{\mathbb{N}} b_{g(n)}$$

by (2) and the definition of the fixed point functional. So,  $\Upsilon(\omega_{\mathbf{u}})(0)$  entails all information of  $\Omega(\mathbf{u})$ .

Conversely, suppose that

$$\Upsilon(\omega_{\mathbf{u}})(0) \triangleright_{\mathbb{N}} b_{g(n)} . \quad (5)$$

We claim that

$$\exists_{m \in G_{\mathbb{N}}} . \Omega(\mathbf{u}) \triangleright_{\mathbb{N}} b_{g(m)} \wedge b_{g(m)} \triangleright_{\mathbb{N}} b_{g(n)} . \quad (6)$$

Suppose on the contrary that this is not the case. Then, for all  $m \in G_{\mathbb{N}}$ , if  $\Omega(\mathbf{u}) \triangleright_{\mathbb{N}} b_{g(m)}$  then  $b_{g(m)} \not\triangleright_{\mathbb{N}} b_{g(n)}$ . Now, the conditional clause gives  $q_{\mathbf{u},k} = \text{ff} \wedge b_{g(k)} \triangleright_{\mathbb{N}} b_{g(m)}$  for some  $k \in G_{\mathbb{N}}$ , which a fortiori gives

$$\omega_{\mathbf{u}}(\lambda x. \perp)(k) \triangleright_{\mathbb{N}} b_{g(k)} \wedge b_{g(k)} \triangleright_{\mathbb{N}} b_{g(m)} .$$

By transitivity we get  $\omega_{\mathbf{u}}(\lambda x. \perp)(k) \triangleright_{\mathbb{N}} b_{g(m)}$ , and so, all put together give

$$\Upsilon(\omega_{\mathbf{u}})(k) \triangleright_{\mathbb{N}} b_{g(m)} \rightarrow b_{g(m)} \not\triangleright_{\mathbb{N}} b_{g(n)}$$

for some  $k$ . But for  $m = n$  and  $k = 0$  we have a contradiction to (5). By transitivity of entailment and (6) we get  $\Omega(\mathbf{u}) \triangleright_{\mathbb{N}} b_{g(n)}$ . So,  $\Omega(\mathbf{u})$  entails all information of  $\Upsilon(\omega_{\mathbf{u}})(0)$ .  $\square$

## Acknowledgements

Many thanks go to Helmut Schwichtenberg for his impressive knack of coming up with insightful hints in minimal time. Sincere thanks are also due to the referees. The author was supported by a Marie Curie Early Stage Training fellowship (MEST-CT-2004-504029).

## References

1. Plotkin, G.D.: LCF considered as a programming language. *Theoret. Comput. Sci.* **5**(3) (1977/78) 223–255
2. Escardó, M.H.: PCF extended with real numbers: a domain-theoretic approach to higher-order exact real number computation. PhD thesis, University of London, Imperial College of Science, Technology and Medicine, Department of Computing (1997)
3. Schwichtenberg, H.: Classifying recursive functions. In Griffor, E., ed.: *Handbook of computability theory*. Volume 140 of *Stud. Logic Found. Math.* North-Holland, Amsterdam (1999) 533–586
4. Normann, D.: Computability over the partial continuous functionals. *J. Symbolic Logic* **65**(3) (2000) 1133–1142
5. Scott, D.S.: Domains for denotational semantics. In: Automata, languages and programming (Aarhus, 1982). Volume 140 of *Lecture Notes in Comput. Sci.* Springer, Berlin (1982) 577–613
6. Schwichtenberg, H.: Recursion on the partial continuous functionals. In Dimitracopoulos, C., Newelski, L., Normann, D., Steel, J., eds.: *Logic Colloquium '05*. Volume 28 of *Lecture Notes in Logic.*, Association for Symbolic Logic (2006) 173–201
7. Plotkin, G.D.:  $T^\omega$  as a universal domain. *J. Comput. System Sci.* **17**(2) (1978) 209–236
8. Stoltenberg-Hansen, V., Lindström, I., Griffor, E.R.: Mathematical theory of domains. Volume 22 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press (1994)
9. Karádais, B.A.: Atomicity, coherence of information, and pointfree structures. Submitted (2008)
10. Winskel, G.: Event structures. In: Advances in Petri nets 1986, Part II (Bad Honnef, 1986). Volume 255 of *Lecture Notes in Comput. Sci.* Springer, Berlin (1987) 325–392
11. Zhang, G.Q.: dI-domains as prime information systems. *Inform. and Comput.* **100**(2) (1992) 151–177
12. Santocanale, L.: A nice labelling for tree-like event structures of degree 3. In: CONCUR 2007 – Concurrency Theory. Volume 4703 of *Lecture Notes in Computer Science*. Springer, Berlin (2007) 151–165

## Appendix

**Scott topology, cartesian closure, syntax and semantics.** We collect here material which is preliminary to the previous exposition and hints at the theoretical background of the present work. Again, one may further refer to [6] and [3] (corresponding results of the latter, which refers to the flat case, are easy to adapt to our setting).

The set of ideals in an acis carries a natural topology based on the cones above its formal neighborhoods. In particular, for every acis  $\alpha$ ,  $\text{Kgl}_\alpha$  forms the basis of a topology on  $\text{Ide}_\alpha$ , called *Scott topology*. A collection of ideals  $\mathcal{U} \subseteq \text{Ide}_\alpha$  is an open set in the Scott topology if and only if  $u \subseteq v$  implies  $v \in \mathcal{U}$  for all  $u \in \mathcal{U}$  (*Alexandrov condition*), and for all  $u \in \mathcal{U}$  there is a  $U \subseteq^f u$  so that  $\bar{U} \in \mathcal{U}$  (*Scott condition*), or equivalently, if  $\mathcal{U} = \bigcup_{\bar{U} \in \mathcal{U}} \nabla U$ . Let  $f : \text{Ide}_\alpha \rightarrow \text{Ide}_\beta$  be a mapping between ideals, or an *ideal mapping*. It is *monotone* when it preserves inclusion, that is,  $u \subseteq v \rightarrow f(u) \subseteq f(v)$ . It is (Scott) *continuous* if and only if either of the following equivalent conditions holds: (a) the mapping is monotone and satisfies the *principle of finite support*, that is, if  $b \in f(u)$  then  $b \in f(\bar{U})$  for some  $U \subseteq^f u$ ; (b) the mapping is monotone and *commutes with directed unions*, that is,  $f(\bigcup_{u \in \mathcal{D}} u) = \bigcup_{u \in \mathcal{D}} f(u)$  for all directed sets  $\mathcal{D}$  of ideals of  $\alpha$ . Easy examples of continuous ideal mappings are identity and constant mappings, compositions of continuous mappings, application of ideals and the standard projections  $\pi_\alpha$ ,  $\pi_\beta$  of cartesian products  $\alpha \times \beta$ .

Let  $r \in \text{Ide}_{\alpha \rightarrow \beta}$  be an ideal and  $f : \text{Ide}_\alpha \rightarrow \text{Ide}_\beta$  a continuous ideal mapping. Define a continuous mapping  $\text{cm}(r) : \text{Ide}_\alpha \rightarrow \text{Ide}_\beta$  and an ideal set  $\text{is}(f) \in \text{Ide}_{\alpha \rightarrow \beta}$  respectively by

$$\begin{aligned} b \in \text{cm}(r)(u) &:\Leftrightarrow \exists_{U \in \text{Con}_\alpha} . U \subseteq^f u \wedge (U, b) \in r , \\ (U, b) \in \text{is}(f) &:\Leftrightarrow b \in f(\bar{U}). \end{aligned}$$

Indeed, through these assignments we obtain that the continuous mappings from  $\text{Ide}_\alpha$  to  $\text{Ide}_\beta$  are exactly the ideals of  $\alpha \rightarrow \beta$  [6]. Since  $\text{is}(\text{cm}(r)) = r$  and  $\text{cm}(\text{is}(f)) = f$ , we identify  $r$  with  $\text{cm}(r)$  and  $f$  with  $\text{is}(f)$ , and rely on the context to prevent ambiguity. Moreover, we allow ourselves to write  $f : \alpha \rightarrow \beta$  meaning  $f : \text{Ide}_\alpha \rightarrow \text{Ide}_\beta$ , or equivalently,  $f \in \text{Ide}_{\alpha \rightarrow \beta}$ .

Let  $\alpha$ ,  $\beta$  and  $\gamma$  be acises with  $T_\alpha \cap T_\beta = \emptyset$ . For every pair  $f : \gamma \rightarrow \alpha$ ,  $g : \gamma \rightarrow \beta$  of continuous mappings, there exists a unique continuous mapping  $h : \gamma \rightarrow \alpha \times \beta$  such that  $f = \pi_\alpha \circ h$  and  $g = \pi_\beta \circ h$ , given by  $h(u) := (f(u), g(u))$  for all  $u \in \text{Ide}_\gamma$  (universal property of cartesian products). Define the *cartesian product*  $f \times g : \alpha \times \beta \rightarrow \gamma \times \delta$  of two continuous mappings  $f : \alpha \rightarrow \gamma$  and  $g : \beta \rightarrow \delta$ , where  $T_\alpha \cap T_\beta = \emptyset$ , by  $f \times g(u, v) := (f(u), g(v))$ . A mapping  $f : \alpha \times \beta \rightarrow \gamma$  is continuous if and only if it is continuous in each component, that is, if and only if all sections  $f_\alpha^v : \alpha \rightarrow \gamma$ ,  $v$  fixed and all sections  $f_\beta^u : \beta \rightarrow \gamma$ ,  $u$  fixed, defined by  $f_\alpha^v(u) := f(u, v)$  and  $f_\beta^u(v) := f(u, v)$  are continuous.

Let  $\alpha$ ,  $\beta$ , and  $\gamma$  be acises. Define the *evaluation* mapping  $\text{eval} : (\alpha \rightarrow \beta) \times \alpha \rightarrow \beta$  by  $\text{eval}(f, u) := f(u)$  and the *currying* mapping  $\text{curry} : (\alpha \times \beta \rightarrow \gamma) \rightarrow (\alpha \rightarrow$

$(\beta \rightarrow \gamma)$ ) by  $\text{curry}(f)(u, v) := f(u, v)$ ; it is direct to see that the currying mapping is bijective. The evaluation and currying mappings are continuous. All of the above facts together provide that the acises together with continuous mappings between their function spaces form a *cartesian closed category*, with the triple  $(\emptyset, \emptyset, \emptyset)$  being the terminal information system.

The types we consider are  $\mathbb{N}$  and  $\mathbb{B}$  as base types, function types  $\alpha \rightarrow \beta$  and product types  $\alpha \times \beta$ . We build *simply-typed  $\lambda$ -terms* based on typed variables  $u^\alpha, v^\alpha, w^\alpha, \dots$  and typed constants  $c^\alpha$  by *application*  $(M^{\alpha \rightarrow \beta} N^\alpha)^\beta$  and  *$\lambda$ -abstraction*  $(\lambda u^\alpha. M^\beta)^{\alpha \rightarrow \beta}$ . As expected, we associate each type to the corresponding acis. For every  $\lambda$ -term  $M^\alpha$  whose variables are among  $u_1^{\alpha_1}, \dots, u_n^{\alpha_n}$  there is an ideal  $\llbracket M \rrbracket : \alpha_1 \times \dots \times \alpha_n \rightarrow \alpha$  such that

$$\begin{aligned}\llbracket u_i \rrbracket(u_1, \dots, u_n) &= u_i, \\ \llbracket MN \rrbracket(u_1, \dots, u_n) &= \llbracket M \rrbracket(u_1, \dots, u_n) (\llbracket N \rrbracket(u_1, \dots, u_n)), \\ \llbracket \lambda u. M \rrbracket(u_1, \dots, u_n)(v) &= \llbracket M \rrbracket(u_1, \dots, u_n, v).\end{aligned}$$

**Acises vs. other structures.** Let  $\alpha = (T, \text{Con}, \vdash)$  be a Scott information system in the sense of, say, [8]. Call it *atomic* if, whenever  $U \vdash b$ , it is  $\{a\} \vdash b$  for some  $a \in U$ , and *coherent* if for all  $U \subseteq^f T$ , it is  $U \in \text{Con}$  if and only if  $\{a, b\} \in \text{Con}$  for all  $a, b \in U$ . It is an easy exercise to see that a Scott information system which is atomic and coherent is exactly an acis in our sense. Moreover, let  $D = (D, \leq, \perp)$  be a domain (in the sense again of [8]). Call it *coherent* if  $\text{lub}_{i \in I} u_i \in D_c$  exactly when  $\text{lub}\{u_i, u_j\} \in D_c$  for all  $i, j \in I$ . One can show the following [9]. (a) Given an arbitrary information system  $\alpha$ , one can define an atomic information system  $\alpha^a$  and a coherent information system  $\alpha^c$ , so that  $\text{Ide}_\alpha \cong \text{Ide}_{\alpha^a}$  and  $\text{Ide}_\alpha \subseteq \text{Ide}_{\alpha^c}$ . So, to the extend that one is concerned with the ideals induced by an information system, one can choose to work with their more practical *atomic* version at no cost. On the other hand, coherent information systems are generally idealwise richer. (b) The domains induced by coherent information systems are exactly the coherent domains.

As pointed out by a referee, a structure that holds a strong resemblance to acises—more accurately, to Scott information systems in general—is an *event structure* in the sense of [10], a fact that has already been noticed elsewhere in the literature. Apart from the obvious difference in terms of axiomatization (for instance, in an event structure there is no ‘propagation of consistency through enabling’ required) the difference really lies in their quite disjoint perspective and intended meaning: information systems describe computation “logically” whereas event structures do it “temporally” [11]. Bringing the relation down to coherence, it is interesting that *coherent event structures* have also been considered in the literature (for example in [12]); these seem to correspond exactly to our acises. It is an interesting question whether these could interrelate in any fruitful way in the future.

# Safety Properties Verification for Pfaffian Dynamics

Margarita Korovina<sup>1</sup> and Nicolai Vorobjov<sup>2</sup>

<sup>1</sup> The University of Manchester, UK, and IIS SB RAS, Novosibirsk, Russia  
`korovina@brics.dk`,  
<http://www.brics.dk/~korovina>

<sup>2</sup> Department of Computer Science, University of Bath, UK  
`nrv@cs.bath.ac.uk`,  
<http://www.bath.ac.uk/~masnrv>

**Abstract.** In this paper we propose an algorithm for verification of safety properties of Pfaffian dynamical systems. We also show that this algorithm has an elementary (doubly-exponential) upper complexity bound.

## 1 Introduction

One of the important problems in the theory of dynamical systems is understanding of the behavior of a dynamical system with respect to safety properties. In other words it would be desirable for a given dynamical system to verify a safety property which states that "something bad does never happen", for examples, the power plant will never blow up, the reactor temperature will never exceed 100° C.

In mathematical settings this problem is formalised in the following way. We consider a continuous dynamical system  $\gamma : G_1 \times T \rightarrow G_2$ , where  $G_1 \subseteq \mathbb{R}^{k_1}$  is a set of control parameters,  $T$  is an interval of time and  $G_2 \subseteq \mathbb{R}^{k_2}$  is a state space. Let  $U$  be a set of control parameters. A safety property is formalised by an invariant which given by a condition  $\Phi$  on the states and requires that  $\Phi$  holds for all reachable states under the control  $U$ , i.e.  $\forall x \in U \forall t \in T \Phi(\gamma_x(t))$ . In this case we say *the subset  $U \subseteq G_1$  satisfies the invariant  $\Phi$  and the dynamic  $\gamma$  is safety under the control  $U$* . We assume that dynamical systems, invariants and sets of control parameters are defined by Pfaffian functions, either implicitly (via triangular systems of ordinary differential equations) or explicitly (by means of equations and inequalities involving *Pfaffian functions*). Such functions naturally arise in applications as real analytic solutions of triangular first order partial differential equations with polynomial coefficients, and include polynomials, algebraic functions, exponentials, and trigonometric functions in appropriate domains. Pfaffian functions form the largest natural class of real analytic functions which have a uniform description and an explicit characterisation of complexity of their representations in terms of *formats*.

Our goal is to characterise the subsets of control parameter space which satisfy a given invariant. In order to achieve our goal we use encoding trajectories

of a Pfaffian dynamical system by finite words [3, 7] and cylindrical cell decomposition for semi-Pfaffian sets [10, 4]. Based on this technique we construct an algorithm for safety properties verification for Pfaffian dynamical systems with an elementary exponential upper bound.

## 2 Basic Definitions and Notions

### 2.1 Pfaffian functions and related sets

In this section we overview the theory of Pfaffian functions and sets definable with Pfaffian functions. The detailed exposition can be found in the survey [4].

**Definition 1.** A Pfaffian chain of the order  $r \geq 0$  and degree  $\alpha \geq 1$  in an open domain  $G \subset \mathbb{R}^n$  is a sequence of real analytic functions  $f_1, \dots, f_r$  in  $G$  satisfying differential equations

$$\frac{\partial f_j}{\partial x_i} = g_{ij}(\mathbf{x}, f_1(\mathbf{x}), \dots, f_j(\mathbf{x})) \quad (1)$$

for  $1 \leq j \leq r$ ,  $1 \leq i \leq n$ . Here  $g_{ij}(\mathbf{x}, y_1, \dots, y_j)$  are polynomials in  $\mathbf{x} = (x_1, \dots, x_n, y_1, \dots, y_j)$  of degrees not exceeding  $\alpha$ .

A function

$$f(\mathbf{x}) = P(\mathbf{x}, f_1(\mathbf{x}), \dots, f_r(\mathbf{x})),$$

where  $P(\mathbf{x}, y_1, \dots, y_r)$  is a polynomial of a degree not exceeding  $\beta \geq 1$ , the sequence  $f_1, \dots, f_r$  is a Pfaffian chain of order  $r$  and degree  $\alpha$ , is called a Pfaffian function of order  $r$  and degree  $(\alpha, \beta)$ .

It is worth noting, apart from polynomials, the class of Pfaffian functions includes real algebraic functions, exponentials, logarithms, trigonometric functions, their compositions, and other major transcendental functions in appropriate domains (see [4, 5]). Now we introduce classes of sets definable with Pfaffian functions. In the case of polynomials they reduce to *semialgebraic* sets whose quantitative and algorithmic theory is treated in [1].

**Definition 2.** A set  $X \subset \mathbb{R}^n$  is called semi-Pfaffian in an open domain  $G \subset \mathbb{R}^n$  if it consists of the points in  $G$  satisfying a Boolean combination of some atomic equations and inequalities  $f = 0, g > 0$ , where  $f, g$  are Pfaffian functions having a common Pfaffian chain defined in  $G$ . A semi-Pfaffian set  $X$  is restricted in  $G$  if its topological closure lies in  $G$ .

**Definition 3.** A set  $X \subset \mathbb{R}^n$  is called sub-Pfaffian in an open domain  $G \subset \mathbb{R}^n$  if it is the image of a semi-Pfaffian set under a projection into a subspace.

In the sequel we will be dealing with the following subclass of sub-Pfaffian sets.

**Definition 4.** Suppose  $\bar{I} \subset \mathbb{R}$  is a closed interval. Consider the closed cube  $\bar{I}^{m+n}$  in an open domain  $G \subset \mathbb{R}^{m+n}$  and the projection map  $\pi : \mathbb{R}^{m+n} \rightarrow \mathbb{R}^n$ . A subset  $Y \subset \bar{I}^n$  is called restricted sub-Pfaffian if  $Y = \pi(X)$  for a restricted semi-Pfaffian set  $X \subset \bar{I}^{m+n}$ .

Note that a restricted sub-Pfaffian set need not be semi-Pfaffian.

**Definition 5.** Consider a semi-Pfaffian set

$$X := \bigcup_{1 \leq i \leq M} \{\mathbf{x} \in \mathbb{R}^n \mid f_{i1} = 0, \dots, f_{il_i} = 0, g_{i1} > 0, \dots, g_{ij_i} > 0\} \subset G, \quad (2)$$

where  $f_{is}, g_{is}$  are Pfaffian functions with a common Pfaffian chain of order  $r$  and degree  $(\alpha, \beta)$ , defined in an open domain  $G$ . Its format is the tuple  $(r, N, \alpha, \beta, n)$ , where  $N \geq \sum_{1 \leq i \leq M} (l_i + j_i)$ . For  $n = m + k$  and a sub-Pfaffian set  $Y \subset \mathbb{R}^k$  such that  $Y = \pi(\bar{X})$ , its format is the format of  $X$ .

*Remark 1.* In this paper we are concerned with complexities of computations, as functions of the format. In the case of Pfaffian dynamical systems these sizes and complexities also depend on the domain  $G$ . So far our definitions imposed no restrictions on an open set  $G$ , thus allowing it to be arbitrarily complex and to induce this complexity on the corresponding semi- and sub-Pfaffian sets. To avoid this we will always assume in the context of Pfaffian dynamical systems that  $G$  is “simple”, like  $\mathbb{R}^n$ , or  $I^n$  for open  $I \subseteq \mathbb{R}$ .

*Remark 2.* In this paper we construct and examine complexities of algorithms for verification of safety properties. In order to estimate the “efficiency” of a computation we need to specify more precisely a *model of computation*. As such we use a *real number machine* which is an analogy of a classical Turing machine but allows the exact arithmetic and comparisons on the real numbers. Since we are interested only in upper complexity bounds for algorithms, there is no need for a formal definition of this model of computation (it can be found in [2]). In some of our computational problems we will need to modify the standard real number machine by equipping it with an *oracle* for deciding feasibility of any system of Pfaffian equations and inequalities. An oracle is a subroutine which can be used by a given algorithm any time the latter needs to check feasibility. We assume that this procedure always gives a correct answer (“true” or “false”) though we do not specify how it actually works. An *elementary step* of a real number machine is either an arithmetic operation, or a comparison (branching) operation, or an oracle call. The *complexity* of a real number machine is the number of elementary steps it makes in the worst case until termination, as a function of the format of the input.

In the special case of semialgebraic sets, the oracle can be replaced by a proper real number machine, so the algorithm for checking of satisfiability of an invariant can be realized as a standard real number machine.

## 2.2 Cylindrical Cell Decompositions

Now we define cylindrical decompositions of semi- and sub-Pfaffian sets in a cube  $\bar{I}^n$ , where  $\bar{I}$  is a closed interval.

**Definition 6.** A cylindrical cell in  $\bar{I}^n$  is defined by induction as follows.

1. A cylindrical 0-cell in  $\bar{I}^n$  is an isolated point.
2. A cylindrical 1-cell in  $\bar{I}$  is an open interval  $(a, b) \subset \bar{I}$ .
3. For  $n \geq 2$  and  $0 \leq k < n$  a cylindrical  $(k+1)$ -cell in  $\bar{I}^n$  is either a graph of a continuous bounded function  $f : C \rightarrow \mathbb{R}$ , where  $C$  is a cylindrical  $(k+1)$ -cell in  $\bar{I}^{n-1}$  and  $k < n-1$ , or else a set of the form

$$\{(x_1, \dots, x_n) \in \bar{I}^n \mid (x_1, \dots, x_{n-1}) \in C \text{ and}$$

$$f(x_1, \dots, x_{n-1}) < x_n < g(x_1, \dots, x_{n-1})\},$$

where  $C$  is a cylindrical  $k$ -cell in  $\bar{I}^{n-1}$ , and  $f, g : C \rightarrow \bar{I}$  are continuous bounded functions such that  $f(x_1, \dots, x_{n-1}) < g(x_1, \dots, x_{n-1})$  for all points  $(x_1, \dots, x_{n-1}) \in C$ .

**Definition 7.** A cylindrical cell decomposition  $\mathcal{D}$  of a subset  $A \subset \bar{I}^n$  with respect to the variables  $x_1, \dots, x_n$  is defined by induction as follows.

1. If  $n = 1$ , then  $\mathcal{D}$  is a finite family of pair-wise disjoint cylindrical cells (i.e., isolated points and intervals) whose union is  $A$ .
2. If  $n \geq 2$ , then  $\mathcal{D}$  is a finite family of pair-wise disjoint cylindrical cells in  $\bar{I}^n$  whose union is  $A$  and there is a cylindrical cell decomposition of  $\pi(A)$  such that  $\pi(C)$  is its cell for each  $C \in \mathcal{D}$ , where  $\pi : \mathbb{R}^n \rightarrow \mathbb{R}^{n-1}$  is the projection map onto the coordinate subspace of  $x_1, \dots, x_{n-1}$ .

**Definition 8.** Let  $B \subset A \subset \bar{I}^n$  and  $\mathcal{D}$  be a cylindrical cell decomposition of  $A$ . Then  $\mathcal{D}$  is compatible with  $B$  if for any  $C \in \mathcal{D}$  we have either  $C \subset B$  or  $C \cap B = \emptyset$  (i.e., some subset  $\mathcal{D}' \subset \mathcal{D}$  is a cylindrical cell decomposition of  $B$ ).

**Definition 9.** For a given finite family  $f_1, \dots, f_N$  of Pfaffian functions in an open domain  $G$  we define its consistent sign assignment as a non-empty semi-Pfaffian set in  $G$  of the kind

$$\{\mathbf{x} \in G \mid f_{i_1} = 0, \dots, f_{i_{N_1}} = 0, f_{i_{N_1}+1} > 0, \dots, f_{i_{N_2}} > 0, f_{i_{N_2}+1} < 0, \dots, f_{i_N} < 0\},$$

where  $i_1, \dots, i_{N_1}, \dots, i_{N_2}, \dots, i_N$  is a permutation of  $1, \dots, N$ .

**Theorem 1.** [5, 9] Let  $f_1, \dots, f_N$  be a family of Pfaffian functions in an open domain  $G \subset \mathbb{R}^n$ ,  $G \supset \bar{I}^n$  having a common Pfaffian chain of order  $r$ , and degrees  $(\alpha, \beta)$ . Then there is an algorithm (with the oracle) producing a cylindrical cell decomposition of  $\bar{I}^n$  which is compatible with each consistent sign assignment of  $f_1, \dots, f_N$ . Each cell is a sub-Pfaffian set represented as a projection of a semi-Pfaffian set in DNF. The number of cells, the components of their formats and the complexity of the algorithm are less than

$$N^{(r+n)^{O(n)}} (\alpha + \beta)^{(r+n)^{O(n^3)}}.$$

We summarize main properties of Pfaffian functions in the following propositions.

- Pfaffian functions can be considered as generalisation of algebraic functions.
- Pfaffian functions have the uniform description and the explicit characterization of complexity of their representations.
- The class of Pfaffian functions includes  $\exp$ , trigonometrical functions defined in appropriate domains, and more generally solutions of a large class of differential equations.
- The structure  $\mathbb{IR} = \langle \mathbb{R}, +, *, 0, 1, <, \{f_1, \dots, f_n\} \rangle$ , where  $f_1, \dots, f_n$  are Pfaffian, is o-minimal, i.e. definable sets have only a finite number of connected definable components, in the other words, it has finiteness property.

### 3 Pfaffian Dynamical Systems

#### 3.1 Pfaffian Dynamics and Related Sets

We now recall definitions concerning Pfaffian dynamical systems.

**Definition 10.** Let  $G_1 \subset \mathbb{R}^{k_1}$  and  $G_2 \subset \mathbb{R}^{k_2}$  be open domains. A Pfaffian dynamical system is a map

$$\gamma : G_1 \times (-T, T) \rightarrow G_2$$

with a semi-Pfaffian graph, where  $G_1$  is a set of control parameters,  $(-T, T)$  is an interval of time, and  $G_2$  is a state space.

For a given  $\mathbf{x} \in G_1$  the set

$$\Gamma_{\mathbf{x}} = \{\mathbf{y} | \exists t \in (-T, T) (\gamma(\mathbf{x}, t) = \mathbf{y})\} \subset G_2$$

is called the trajectory (or evolution) determined by  $\mathbf{x}$ , and the graph

$$\widehat{\Gamma}_{\mathbf{x}} = \{(t, \mathbf{y}) | \gamma(\mathbf{x}, t) = \mathbf{y}\} \subset (-T, T) \times G_2$$

is called the integral curve determined by  $\mathbf{x}$ .

**Definition 11.** Let  $U \subseteq G_1$ . A set  $Inv \subseteq G_2$  is called invariant under the dynamical system  $\gamma$  and the control  $U$  if for all  $x \in U$  and for all  $t \in T$ ,  $\gamma_x(t) \in Inv$ .

In the next sections we investigate the behavior of a Pfaffian dynamical system with respect to a given semi-Pfaffian invariant.

#### 3.2 Encoding Trajectories by Words

We now introduce, following [3, 7], a technique of encoding trajectories of dynamical systems by words. Consider a Pfaffian dynamical system  $\gamma : G_1 \times (-T, T) \rightarrow G_2$ , where  $G_1 \subset \mathbb{R}^{k_1}$  and  $G_2 \subset \mathbb{R}^{k_2}$  are open domains, and a partition  $\mathcal{P} := \{P_1, \dots, P_s\}$  of  $G_2$  into  $s$  semi-Pfaffian sets  $P_j$ . Let the graph of  $\gamma$  and each set

$P_j$  have a format  $(r, N, \alpha, \beta, n)$ , where  $n \geq k_1 + k_2 + 1$ , and all Pfaffian functions involved have a common Pfaffian chain. Fix  $\mathbf{x} \in G_1$ . Define the set of points and open intervals in  $\mathbb{R}$ :

$$\begin{aligned} \mathcal{F}_{\mathbf{x}} := \{J \mid J \text{ is a point or an interval in } (-T, T) \text{ maximal w.r.t. inclusion for the} \\ \text{property } \exists i \in \{1, \dots, s\} \forall t \in J (\gamma(\mathbf{x}, t) \in P_i)\}. \end{aligned}$$

Let the cardinality  $|\mathcal{F}_{\mathbf{x}}| = r$  and  $y_1 < \dots < y_r$  be the set of representatives of  $\mathcal{F}_{\mathbf{x}}$  such that  $\gamma(\mathbf{x}, y_j) \in P_{i_j}$ . Then define the word  $\omega := P_{i_1} \cdots P_{i_r}$  in the alphabet  $\mathcal{P}$ . Informally,  $\omega$  is the list of names of elements of the partition in the order they are visited by the trajectory  $\Gamma_{\mathbf{x}}$ . In our setting  $\omega$  is called the *type of trajectory*  $\Gamma_{\mathbf{x}}$ . Introduce the set of words  $\Omega := \{\omega \mid \mathbf{x} \in G_1\}$ .

**Theorem 2.** [3, 7] *The set  $\Omega$  is finite and the number of different trajectory types of  $\gamma$  with respect to the partition  $\mathcal{P}$  is less than*

$$(sN)^{(r+n)^{O(n)}} (\alpha + \beta)^{(r+n)^{O(n^3)}} \quad (3)$$

**Theorem 3.** *There is a cell decomposition of the control parameter space  $G_1$  such that if  $\mathbf{x}_1$  and  $\mathbf{x}_2$  belong to the same cell then  $\Gamma_{\mathbf{x}_1}$  and  $\Gamma_{\mathbf{x}_2}$  are labelled by the same word.*

*Proof.* Consider the family  $\mathcal{F} = \{f_1, \dots, f_k\}$  of Pfaffian functions in the domain  $G_1 \times (-T, T) \times G_2$  consisting of all functions in variables  $\mathbf{x}, t, \mathbf{y}$  involved in the defining formulas for the graph of the map  $\gamma : (\mathbf{x}, t) \mapsto \mathbf{y}$ , and for all sets  $P_j$ . According to Theorem 1, there is a cylindrical decomposition  $\mathcal{D}$  of  $G_1 \times (-T, T) \times G_2$  with respect to the variables  $\mathbf{x}, t, \mathbf{y}$  having the following properties.

- 1)  $\mathcal{D}$  is compatible with each consistent sign assignment of  $f_1, \dots, f_k$ .
- 2) There are at most (3) cylindrical cells.
- 3) Each of these cells is sub-Pfaffian.
- 4)  $\mathcal{D}$  induces a cylindrical decomposition on  $G_1$  which we denote by  $\mathcal{E}$ .

We claim that for any cell  $C \in \mathcal{E}$  and any two points  $\mathbf{x}_1, \mathbf{x}_2 \in C$  the trajectories  $\Gamma_{\mathbf{x}_1}, \Gamma_{\mathbf{x}_2} \in G_2$  are intersecting sets  $P_1, \dots, P_s$  in the same order (i.e., are encoded by the same word from  $\Omega$ ). Indeed, let  $\pi : G_1 \times (-T, T) \times G_2 \rightarrow G_1$  be the projection on  $G_1$ . The decomposition  $\mathcal{D}$  induces cylindrical decompositions  $\mathcal{D}_1$  and  $\mathcal{D}_2$  on  $\pi^{-1}(\mathbf{x}_1)$  and  $\pi^{-1}(\mathbf{x}_2)$  respectively. In particular, each of the integral curves  $\widehat{\Gamma}_{\mathbf{x}_1}$  and  $\widehat{\Gamma}_{\mathbf{x}_2}$  is decomposed into a sequence of alternating points and open intervals. Due to basic properties of cylindrical decomposition, there is a natural bijection  $\psi : \mathcal{D}_1 \rightarrow \mathcal{D}_2$  such that

- (i) the restriction of  $\psi$  to the set of all cells in  $\widehat{\Gamma}_{\mathbf{x}_1}$  is a bijection onto the set of all cells in  $\widehat{\Gamma}_{\mathbf{x}_2}$ ;
- (ii) for each  $1 \leq j \leq s$  the restriction of  $\psi$  to the set of all cells in  $(-T, T) \times P_j \cap \pi^{-1}(\mathbf{x}_1)$  is a bijection onto the set of all cells in  $(-T, T) \times P_j \cap \pi^{-1}(\mathbf{x}_2)$ .

(iii) the bijection  $\psi$  preserves the order in which cells appear in the trajectories.

It follows that if a cell  $B \in \mathcal{D}_1$  is a subset of  $\widehat{\Gamma}_{\mathbf{x}_1} \cap ((-T, T) \times P_j)$  for some  $1 \leq j \leq s$ , then  $\psi(B) \subset \widehat{\Gamma}_{\mathbf{x}_2} \cap ((-T, T) \times P_j)$ . Moreover, if for cells  $B_1, B_2 \in \mathcal{D}_1$  there exist  $t_1, t_2 \in (-T, T)$  such that  $t_1 < t_2$  and  $\gamma(\mathbf{x}_1, t_1) \in B_1 \wedge \gamma(\mathbf{x}_1, t_2) \in B_2$  then there exist  $t'_1, t'_2 \in (-T, T)$  such that  $t'_1 < t'_2$  and  $\gamma(\mathbf{x}_2, t'_1) \in \psi(B_1) \wedge \gamma(\mathbf{x}_2, t'_2) \in \psi(B_2)$ . The claim is proved.

It follows that the cardinality of  $\Omega$  does not exceed the cardinality of  $\mathcal{E}$  which does not exceed the cardinality of  $\mathcal{D}$  which in turn is at most (3).

## 4 An Algorithm for Safety Properties Verification

Consider a Pfaffian dynamical system  $\gamma : G_1 \times (-T, T) \rightarrow G_2$ , a semi-Pfaffian subset of control parameters  $U \subseteq G_1$  and a semi-Pfaffian invariant  $Inv \subseteq G_2$ . Let the graph of  $\gamma$  and the sets  $U, Inv$  have a format  $(r, N, \alpha, \beta, n)$ , and all Pfaffian functions involved have a common Pfaffian chain.

**Theorem 4.** *There is an algorithm which checks whether the control  $U$  satisfies the invariant  $Inv$ . The complexity of this algorithm does not exceed*

$$(sN)^{(r+n)^{O(n)}} (\alpha + \beta)^{(r+n)^{O(n^3)}} \quad (4)$$

*Proof.* First the algorithm produces the set of words  $\Omega$  corresponding to the Pfaffian dynamical system  $\gamma : G_1 \times (-T, T) \rightarrow G_2$  and the partition  $\mathcal{P} = \{P_1, P_2\}$ , where  $P_1 = Inv$  and  $P_2 = G_2 \setminus Inv$ . Consider the family of Pfaffian functions in the domain  $G_1 \times (-T, T) \times G_2$  consisting of all functions in variables  $\mathbf{x}, t, \mathbf{y}$  involved in the defining formulas for the graph of the map  $\gamma : (\mathbf{x}, t) \mapsto \mathbf{y}$ , for the set  $U$ , and for the partition  $\mathcal{P}$ . According to Theorem 1, there is a cylindrical decomposition  $\mathcal{D}$  with respect to  $(\mathbf{x}, t, \mathbf{y})$  which is compatible with this family and consists of at most (4) cylindrical cells.

This cell decomposition  $\mathcal{D}$  induces the cell decomposition  $\mathcal{E}$  (see the proof of Theorem 3). Using the oracle, which decides feasibility of any system of Pfaffian equations and inequalities, the algorithm selects the cells from  $\mathcal{D}$  which are subsets of  $\{(\mathbf{x}, t, \mathbf{y}) | \mathbf{y} = \gamma(\mathbf{x}, t)\}$ . Denote the set of the selected cells by  $\mathcal{B}$ . Observe that for any fixed  $\mathbf{x}' \in G_1$  the set  $\bigcup_{B \in \mathcal{B}} B \cap \{(\mathbf{x}, t, \mathbf{y}) | \mathbf{x} = \mathbf{x}'\}$  coincides with the integral curve  $\widehat{\Gamma}_{\mathbf{x}'}$ . Then the algorithm determines the order in which the cells  $B \in \mathcal{B}$  intersected with  $\{(\mathbf{x}, t, \mathbf{y}) | \mathbf{x} = \mathbf{x}'\}$  appear in the trajectory  $\Gamma_{\mathbf{x}'}$ .

More precisely, for each pair of distinct cells  $B_1, B_2 \in \mathcal{B}$  the algorithm decides, using the oracle, whether

$$\exists \mathbf{x} \exists t_1 \exists t_2 \exists \mathbf{y}_1 \exists \mathbf{y}_2 ((\mathbf{x}, t_1, \mathbf{y}_1) \in B_1 \wedge (\mathbf{x}, t_2, \mathbf{y}_2) \in B_2 \wedge (t_1 < t_2)).$$

For a given  $C \in \mathcal{E}$ , after all pairs of cells are processed we get the ordered set of cells  $B_1, \dots, B_k$  in  $\mathcal{D}$  such that for any  $1 \leq i \leq k$  and any  $\mathbf{x}' \in C$  the sequence of points and intervals

$$B_1 \cap \{(\mathbf{x}, t, \mathbf{y}) | \mathbf{x} = \mathbf{x}'\}, \dots, B_k \cap \{(\mathbf{x}, t, \mathbf{y}) | \mathbf{x} = \mathbf{x}'\}$$

forms the integral curve  $\widehat{I}_{x'}$ . By the definition of cylindrical decomposition, for any pair  $B_i, P_j$  either  $B_i \subset (C \times (-T, T) \times P_j)$  or  $B_i \cap (C \times (-T, T) \times P_j) = \emptyset$ . The algorithm uses the oracle to decide for every pair which of these two cases takes place. As the result, the sequence  $B_1, \dots, B_k$  becomes partitioned into subsequences of the kind

$$(B_1, \dots, B_{k_1}), (B_{k_1+1}, \dots, B_{k_2}), \dots, (B_{k_{\ell-1}+1}, \dots, B_k),$$

where for any  $i$ ,  $0 \leq i \leq \ell - 1$ , the cells  $B_{k_i+1}, \dots, B_{k_{i+1}}$  lie in  $C \times (-T, T) \times P_{j_i}$  for some  $j_i$ , while  $B_{k_i} \cap C \times (-T, T) \times P_{j_i} = \emptyset$  and  $B_{k_{i+1}+1} \cap C \times (-T, T) \times P_{j_i} = \emptyset$ . Then the word  $\omega := P_{j_0} \cdots P_{j_{\ell-1}}$  corresponds to the cell  $C$ . Considering all cells in  $\mathcal{E}$  the algorithm finds  $\mathcal{O}$ .

Then the algorithm collects all cells from  $\mathcal{E}$  such that their union is  $U$ . If all of these cells corresponds to the word  $P_1$ , then  $\gamma$  is safety under the control  $U$ . This completes the description of the algorithm.

A straightforward analysis shows that the complexity of the algorithm does not exceed (4), taking into account the bounds from Theorem 1.

## References

1. S. Basu, R. Pollack and M.-F. Roy, *Algorithms in Real Algebraic Geometry*, Springer, Berlin-Heidelberg, 2003.
2. L. Blum, F. Cucker, M. Shub, and S. Smale, *Complexity and Real Computation*, Springer, New York, 1997.
3. T. Brihaye, C. Michaux, C. Riviere, C. Troestler, On o-minimal hybrid systems, in: *Hybrid Systems: Computation and Control*, R. Alur, G. J. Pappas, (Eds.), LNCS, **2993**, Springer, Heidelberg, 2004, 219–233.
4. A. Gabrielov, N. Vorobjov, Complexity of computations with Pfaffian and Noetherian functions, in: *Normal Forms, Bifurcations and Finiteness Problems in Differential Equations*, Yu. Ilyashenko et al., (Eds.), NATO Science Series II, **137**, Kluwer, 2004, 211–250.
5. A. Gabrielov, N. Vorobjov, Complexity of cylindrical decompositions of sub-Pfaffian sets, *J. Pure and Appl. Algebra*, **164**, 1–2, 2001, 179–197.
6. A. Khovanskii, *Fewnomials*, Number 88 in Translations of Mathematical Monographs. American Mathematical Society, Providence, RI, 1991.
7. M. Korovina and N. Vorobjov, Pfaffian hybrid systems. In *Springer Lecture Notes in Comp. Sci.*, volume 3210 of *Computer Science Logic '04*, 2004, 430–441.
8. M. Korovina and N. Vorobjov, Upper and lower Bounds on Sizes of Finite Bisimulations of Pfaffian Hybrid Systems. In *Proceedings of CiE'06*, invited talk, LNCS 3988, 2006, 235–241.
9. S. Pericleous, N. Vorobjov, New complexity bounds for cylindrical decompositions of sub-Pfaffian sets, in: *Discrete and Computational Geometry. Goodman-Pollack Festschrift*, B. Aronov et al. (Eds.), Springer, 2003, 673–694.
10. L. van den Dries, *Tame Topology and O-minimal Structures*, Number 248 in London Mathematical Society Lecture Notes Series. Cambridge University Press, Cambridge, 1998.

# On Extending Wand’s Type Reconstruction Algorithm to Handle Polymorphic Let<sup>\*</sup>

Sunil Kothari and James L. Caldwell

Department of Computer Science  
University of Wyoming  
Laramie, WY 82071-3315, USA  
`{skothari,jlc}@cs.uwyo.edu`

**Abstract.** We have extended Wand’s type reconstruction algorithm to polymorphic let by extending the constraint language and by using a multi-phase unification algorithm in the constraint solving phase. We show the correctness of our approach by extending the Wand’s soundness and completeness results. We have validated our approach against other popular type reconstruction algorithms by implementing OCaml prototypes and running them on non-trivial examples.

## 1 Introduction

The general type reconstruction problem can be formulated as:

Given a well-formed term  $M$  without any types, does there exist a type  $\tau$  and a type environment<sup>1</sup>  $\Gamma$  such that a judgment  $\Gamma \vdash M : \tau$  is valid?

Type reconstruction is a popular feature in modern functional programming languages. Underlying any type reconstruction algorithm is a set of rules encoding a type system. One of the most widely used type systems is the Hindley-Milner (HM) type system, first mentioned in [Mil78] by Milner, but discovered independently by Hindley [Hin69]. Various type reconstruction algorithms [Mil78,DM82,LY98] have been proposed to implement the HM type system. Many of these algorithms are characterized by intermittent constraint generation and constraint solving. But over the years, focus has shifted to algorithms having a clear separation of constraint generation and constraint solving phases [Hee05,PR05,Wan87]. This separation leads to better error messages [Hee05] when the constraint set is unsatisfiable (since a larger set of constraints is available to reason about the error). Moreover, the separation provides a clean ab-

---

<sup>\*</sup> This material is based upon work supported by the National Science Foundation under Grant No. NSF CNS-0613919.

<sup>1</sup> We assume the initial type environment is empty since we are dealing with closed terms.

straction of the various substitution-based algorithms<sup>2</sup> since most well known algorithms are specific instances of various constraint solving strategies.

The type inference involving polymorphic let construct is a non-trivial problem. In fact, in the worst case, it is a DEXPTIME<sup>3</sup>-complete and PSPACE-hard problem [PJ89,Mai89] in the level of nested lets. Moreover, the literature on constraint-based type reconstruction is sparse and uneven. For example, Pierce's book [Pie02] has no references on how to handle ML-Let construct in constraint-based algorithms, HM(X) [SOW97,PR05] requires specialized knowledge, whereas Aiken and Wimmers [AW93] use subtyping constraints. Furthermore, none of the literature, in our view, describes it as a *direct* extension to well known Wand's algorithm [Wan87].

The Helium compiler [HLI03] is known for giving good quality error messages and a very simple constraint representation is used for handling the let construct<sup>4</sup>. This paper describes an approach where Helium's constraint representation is used to handle let polymorphism. Our approach builds upon Wand's algorithm, and our proofs rely on the soundness and completeness of Wand's algorithm. We have validated our approach with some of the known type reconstruction algorithms [Kot07]. In summary, our contributions are:

1. A new algorithm extending Wand's algorithm [Wan87] to include polymorphic let.
2. New soundness and completeness proofs for Wand's system and the extended system using a novel desugaring of polymorphic lets.

The rest of this paper is organized as follows: Section 2 reviews the previous methods for inferring the type of the let construct. Section 3 introduces the concepts and terminologies needed for this paper. Section 4 gives an overview of Wand's algorithm and states soundness and completeness theorems. Section 5 describes the changes needed for the extension. Section 6 gives an overview of the correctness proofs. Section 7 summarizes our current work.

## 2 Literature Review

We review some of the constraint-based algorithms in their handling of the let construct. A detailed survey of substitution-based algorithms is available in [Kot07]. **Wand** [Wan87] looked at the type inference problem as a type-erasure: whether it is decidable that a term of the untyped lambda calculus is the image under type-erasing of a term of the simply typed lambda calculus, and presented a type reconstruction algorithm. This was the first successful attempt at separating constraint generation from constraint solving phase. However, extending the algorithm to handle the let construct remained a future work<sup>5</sup>. **Heeren**

---

<sup>2</sup> Throughout this paper we term *substitution-based algorithms* as those algorithms which intermix constraint generation and constraint solving, whereas algorithms with a clear separation are termed *constraint-based algorithms*.

<sup>3</sup>  $\text{DTIME}(2^{n^{O(1)}})$

<sup>4</sup> Personal communication from Bastiaan Heeren.

<sup>5</sup> Personal communication from Mitchell Wand.

[Hee05] suggested three constraint representations to handle the let construct; each equally expressive but differing in constraint solving.

**Approach 1. Qualification of type constraints:** Type schemes contain a constraint component as part of its type as shown by the following grammar:

$$\sigma ::= \forall \vec{\alpha}.\sigma \mid C \Rightarrow \tau$$

For example, an expression  $\lambda x.x$  can be assigned a type  $\tau_1 \rightarrow \tau_2$  under the constraint  $\tau_1 \equiv \tau_2$ . So the type of the expression is  $\forall \tau_1, \tau_2. \tau_1 \equiv \tau_2 \Rightarrow \tau_1 \rightarrow \tau_2$ . In many ways, this approach is similar to HM(X)[SOW97], Pottier and Rémy's account of ML type inference [PR05], and Jones's qualified types [Jon95].

**Approach 2. Type scheme variables as placeholders:** The type constraint language is extended to take into account generalization and instantiation of type schemes by means of type scheme variables. The constraint language is given by the following grammar:

$$\mathcal{C} ::= \tau_1 \equiv \tau_2 \mid \sigma := GEN(\Gamma, \tau) \mid \tau := INST(\sigma)$$

Our approach uses a slightly modified version of the above constraint representation. We know of no published account of the constraint solving phase for this representation, although Heeren does mention in his thesis that a special substitution that maps type scheme variables to type schemes is needed for this representation. Our algorithm does not require any such substitution.

**Approach 3. Using implicit instance constraints:** This approach merges the generalization and instantiation constraints in the above approach. The constraint language is given by the following grammar:

$$\mathcal{C} ::= \tau_1 \equiv \tau_2 \mid \tau_1 \leq_M \tau_2$$

This representation is described, in detail, in Heeren's thesis [Hee05].

### 3 Preliminaries

In this paper, we assume familiarity with the notion of types, functional programming and the lambda calculus. The terms considered here are pure untyped lambda terms given by the following grammar:

$$A ::= x \mid MN \mid \lambda x.M$$

We follow the usual conventions for lambda calculus: arrow types associate to the right, function applications associate to the left and application binds more tightly than abstraction. The types for untyped lambda terms is given by the following grammar:

$$\tau ::= \alpha \mid \tau_1 \rightarrow \tau_2$$

The type is either a type variable or a function type. We call any expression formed by following the above grammar as a *type expression*. We follow the convention that  $\alpha, \beta$  denote type variables, whereas  $\tau$  denotes a type expression. A *type environment*, denoted by  $\Gamma$ , maps type variables to type expressions. The set of *free type variables* of a type expression  $\tau$  is denoted by  $FTV(\tau)$  and those of a type environment  $\Gamma$  is denoted by  $FTV(\Gamma)$ . A term and its type is related by an *assertion*, denoted by  $\Gamma \vdash M : \tau$ , where  $M$  is a term,  $\tau$  is a type and  $\Gamma$  is a type environment. We denote type environment where  $x$  is not in the domain of  $\Gamma$  by  $\Gamma \setminus x$ . A *derivation* of an assertion  $\Gamma \vdash M : \tau$  is a finite tree of assertions,

where the root is  $\Gamma \vdash M : \tau$ . Every interior node in the tree is related to its parent by an instance of one of the non-axiom type rules and every leaf node is an instance of an axiom type rule. In this paper, we consider three different type systems: the Hindley-Milner type system illustrated in Fig. 1, Wand's system illustrated in Fig. 2, and the extended Wand system illustrated later in Section 5. We denote a judgment in a particular system by a subscript. For example,  $\Gamma \vdash_{HM} M : \tau$  is a HM judgment.

$$\begin{array}{c} \frac{}{\Gamma \vdash_{HM} x : \tau} \text{ where } x : \tau \in \Gamma \quad (\text{HM-Var}) \\ \frac{\Gamma \setminus x \cup \{x : \tau_1\} \vdash_{HM} M : \tau_0}{\Gamma \vdash_{HM} \lambda x. M : \tau_1 \rightarrow \tau_0} \quad (\text{HM-Abs}) \\ \frac{\Gamma \vdash_{HM} M : \tau_1 \rightarrow \tau \quad \Gamma \vdash_{HM} N : \tau_1}{\Gamma \vdash_{HM} MN : \tau} \quad (\text{HM-App}) \end{array}$$

**Fig. 1. Hindley-Milner type system**

$$\begin{array}{c} \frac{}{\Gamma, \{\alpha \stackrel{e}{=} \tau\} \vdash_W x : \alpha} \text{ where } x : \tau \in \Gamma \text{ and } \alpha \text{ is fresh} \quad (\text{W-Var}) \\ \frac{(\Gamma \setminus x) \cup \{x : \alpha\}, E \vdash_W M : \beta}{\Gamma, E \cup \{\tau \stackrel{e}{=} \alpha \rightarrow \beta\} \vdash_W \lambda x. M : \tau} \text{ where } \alpha, \beta \text{ are fresh} \quad (\text{W-Abs}) \\ \frac{\Gamma, E_1 \vdash_W M : \alpha \rightarrow \tau \quad \Gamma, E_2 \vdash_W N : \alpha}{\Gamma, E_1 \cup E_2 \vdash_W MN : \tau} \text{ where } \alpha \text{ is fresh} \quad (\text{W-App}) \end{array}$$

**Fig. 2. Wand's type system**

A *substitution* is a mapping from type variables to type expressions. Let  $\rho_1, \rho_2$  be two substitutions then *substitution composition*  $\rho_1 \circ \rho_2$  is defined extensionally as  $\forall \alpha. (\rho_1 \circ \rho_2)\alpha \stackrel{\text{def}}{=} \rho_1(\rho_2\alpha)$ , where  $\alpha$  is a type variable. Substitution composition is associative but non-commutative. A substitution  $\rho$  is *idempotent* if  $\rho \circ \rho = \rho$ . We treat all the substitutions as idempotent substitutions. A type  $\tau'$  is a substitution instance of a type  $\tau$  if and only if  $\tau' = \rho\tau$ , for some substitution  $\rho$ . Substitution application to a type environment  $\Gamma$ , denoted by  $\rho\Gamma$ , is defined as  $\{x : \rho\tau \mid x : \tau \in \Gamma\}$ .

In Wand's system, constraints plays a central role. Specifically, Wand's algorithm generates equality constraints, referred to as *e-constraints*, and are denoted by  $\tau_1 \stackrel{e}{=} \tau_2$ , where  $\tau_1, \tau_2$  are type expressions. An e-constraint  $\tau_1 \stackrel{e}{=} \tau_2$  is *solvable* if there exists a substitution  $\rho$  such that  $\rho\tau_1 = \rho\tau_2$ . More formally, we denote solvability of a constraint by  $\models$  (read solve). We write  $\rho \models \tau_1 \stackrel{e}{=} \tau_2$ , if  $\rho\tau_1 = \rho\tau_2$ . A type judgment in Wand's system is given as  $\Gamma, E \vdash_W M : \alpha$ , where  $E$  denotes a constraint set. Sometimes we elide the constraint component of the judgment to simplify the presentation. In that case, we denote a judgment by  $\Gamma \vdash_W M : \tau$ , where  $E$  is implicit and so there is a substitution generated by solving  $E$  such that  $\rho\alpha = \tau$ .

## 4 Wand's Algorithm

Next we briefly discuss Wand's algorithm [Wan87] and state the soundness and completeness theorems. Central to Wand's algorithm is the notion of *action table*, which takes as input a type system and a term and generates equational constraints. For untyped lambda calculus, the type system is shown in Fig. 2. Let  $G$  denote a set of assertions (also called goals) and  $E$  a set of equational constraints. Then the algorithm sketch is given as:

**Input.** A term  $M_0$  of  $\Lambda$ .

**Initialization.** Set  $E = \emptyset$  and  $G = \{(\Gamma_0, M_0, \alpha_0)\}$ , where  $\alpha_0$  is a fresh type variable and  $\Gamma_0$  is an empty environment.

**Loop Step.** If  $G = \emptyset$  then return  $E$  else choose a sub-goal  $(\Gamma, M, \tau)$  from  $G$ , delete the sub-goal from  $G$  and add to  $E$  and  $G$  new verification conditions and subgoals generated by the action table<sup>6</sup>.

**Solve Constraints.** Unify constraints in  $E$ .

We can now describe the soundness and completeness results as stated by Wand, but before that we introduce some terminology. We denote a goal by a 3-tuple, i.e.  $(\Gamma, M, \tau)$ , and we write  $\rho \models g$  i.e.  $\rho \models (\Gamma, M, \tau)$  if and only if  $\rho \Gamma \vdash_{HM} M : \rho \tau$ . We write  $\rho \models G$  if and only if  $\forall g \in G. \rho \models g$ . Similarly, if  $E$  is a set of e-constraints, we write  $\rho \models E$  if and only if  $\forall e \in E. \rho \models e$ . Finally, we say  $\rho$  solves  $(E, G)$ , where  $(E, G)$  result from applying Wand's algorithm, if and only if  $\rho \models E$  and  $\rho \models G$ .

**(Soundness)**  $\forall \rho. \rho \models (E, G) \Rightarrow \rho \Gamma_0 \vdash_{HM} M_0 : \rho \tau_0$

**(Completeness)**  $\Gamma \vdash_{HM} M_0 : \tau \Rightarrow (\exists \rho. \rho \models (E, G)) \wedge \Gamma = \rho \Gamma_0 \wedge \tau = \rho \tau_0$

We have reformulated Wand's statement of completeness and soundness and made them more abstract (by eliminating the goal set). The reformulated theorems are used later in the proofs of soundness and completeness of the extended proof system. Our statement of the soundness and completeness are given below:

**Theorem 1 (Soundness).** *If there is a derivation of  $\Gamma_0, E \vdash_W M_0 : \tau_0$  generating constraint set  $E$  then, for any  $\rho$  such that  $\rho \models E$ ,  $\rho \Gamma_0 \vdash_{HM} M_0 : \rho \tau_0$  is derivable.*

**Theorem 2 (Completeness).** *If there is a derivation of  $\Gamma \vdash_{HM} M_0 : \tau$ , then for any  $\rho, \tau_0, \Gamma_0$ , such that  $\text{dom}(\rho) = (\text{FTV}(\Gamma_0) \cup \{\tau_0\})$ ,  $\rho \Gamma_0 = \Gamma$ , and  $\rho \tau_0 = \tau$  then there exists a derivation of  $\Gamma_0, E \vdash_W M_0 : \tau_0$ , and there exists a substitution  $\rho'$  such that  $\rho \subseteq \rho'$  and  $\rho' \models E$ .*

Both these theorems are proved in [KC07].

## 5 Extended Wand's Algorithm

In this section we describe the changes needed to incorporate the let construct. First, we enrich our term and type language. We call the extended language *Core-ML* [MH88], and it is defined as:

$$\text{Core-ML} ::= x \mid MN \mid \lambda x. M \mid \text{let } x = M \text{ in } N$$

---

<sup>6</sup> See Section 5 for action table behavior for untyped lambda calculus.

The let expression  $\text{let } x = M \text{ in } N$  is not merely a syntactic sugar for  $(\lambda x.N)M$ . For instance, the term  $\text{let } i = \lambda x.x \text{ in } i \ i$  is typable but the desugared term  $(\lambda i.i \ i)(\lambda x.x)$  is not typable. This is true for both Haskell and ML type reconstruction algorithms. To handle polymorphic types introduced by let-expressions, the type syntax is extended with type scheme variable and type scheme as shown below:

$$\sigma ::= \alpha^* \mid \forall \vec{\alpha}.\tau$$

A *type scheme*, denoted by  $\forall \vec{\alpha}.\tau$ , is a type where zero or more type variables are universally quantified. We denote a type scheme variable by annotating a type variable with a “\*”. *Generalizing* a type  $\tau$  with respect to a type environment  $\Gamma$  entails quantifying over the free variables of  $\tau$  that are not free in  $\Gamma$ . Thus  $\text{gen}(\Gamma, \tau) \stackrel{\text{def}}{=} \forall \vec{\alpha}.\tau$ , where  $\vec{\alpha} = FTV(\tau) - FTV(\Gamma)$ . On the other hand, *instantiation* of a type scheme involves replacing the quantified variables by fresh type variables and is given by  $\text{inst}(\forall \vec{\alpha}.\tau) \stackrel{\text{def}}{=} \tau[\alpha_1 := \beta_1, \dots, \alpha_n := \beta_n]$ , where  $\beta_1, \dots, \beta_n$  are fresh type variables.

Next, we extend the constraint language  $\mathcal{C}$  to include two other kinds of constraints as shown below:

$$\mathcal{C} ::= \begin{array}{c} \tau \stackrel{e}{=} \tau \\ (\text{e-constraint}) \end{array} \mid \begin{array}{c} \tau \stackrel{s}{=} \Gamma \alpha^* \\ (\text{s-constraint}) \end{array} \mid \begin{array}{c} \alpha^* \stackrel{i}{=} \tau \\ (\text{i-constraint}) \end{array}$$

A *s-constraint*  $\tau \stackrel{s}{=} \Gamma \alpha^*$  expresses the fact that  $\alpha^*$  denotes a type scheme obtained by generalizing a type  $\tau$  with respect to the environment  $\Gamma$ . An *i-constraint*  $\alpha^* \stackrel{i}{=} \tau$  expresses the fact that  $\tau$  is constrained to be an instantiated value of the type scheme denoted by  $\alpha^*$ . Wand's type system is now extended with two rules to account for i&s-constraints. The new type system is called *extended Wand's system* and a judgment in the extended system is denoted by the subscript  $W^+$ .

$$(\text{W-Var-i}) \frac{}{\Gamma, \{\alpha^* \stackrel{i}{=} \tau\} \vdash_{W^+} x : \alpha^*} \text{ where } x : \tau \in \Gamma$$

$$(\text{W-Let}) \frac{\Gamma, E_1 \vdash_{W^+} M : \alpha_1 \quad (\Gamma \setminus x) \cup \{x : \alpha_2^*\}, E_2 \vdash_{W^+} N : \tau}{\Gamma, E_1 \cup E_2 \cup \{\alpha_1 \stackrel{s}{=} \alpha_2^*\} \vdash_{W^+} \text{let } x = M \text{ in } N : \tau} \text{ where } \alpha_1, \alpha_2^* \text{ are fresh}$$

Apart from extending the type rules, we also had to extend the notion of satisfiability and substitution application to a constraint. First, we describe some notations used in the description of satisfiability. We use the notation  $E_{\alpha^*}$  to denote a set of i-constraints related<sup>7</sup> to a s-constraint  $\tau_0 \stackrel{s}{=} \Gamma \alpha^*$ . From this point onwards, we think of a s-constraint and related i-constraint as a pair  $(\tau_0 \stackrel{s}{=} \Gamma \alpha^*, E_{\alpha^*})$ ; the first component being the s-constraint and the second component being the list of related i-constraint(s). We use the symbol  $\leq$  to express the notion of an instance. Specifically,  $\tau \leq \sigma$  expresses the fact that  $\tau$  is an instance of  $\sigma$  in the sense that  $\tau$  is obtained by instantiating all the bound variables of  $\sigma$ . This notion can then be used to express the satisfiability of *i&s* constraints. We say

<sup>7</sup> A s-constraint is *related* to an i-constraint if they share the same type scheme variable.

$\rho$  satisfies  $(\tau_0 \stackrel{s}{=} \Gamma \alpha^*, E_{\alpha^*})$  if  $\forall (\alpha^* \stackrel{i}{=} \tau_1) \in E_{\alpha^*} . \rho \tau_1 \leq \rho(\text{gen}(\Gamma, \tau_0))$ . Substitution application to a pair of s-constraint and related i-constraints is defined as:  $\rho(\tau_0 \stackrel{s}{=} \Gamma \alpha^*, E_{\alpha^*}) \stackrel{\text{def}}{=} (\rho\tau_0 \stackrel{s}{=} \rho\Gamma \alpha^*, \{\rho\tau \stackrel{i}{=} \alpha^* \mid (\tau \stackrel{i}{=} \alpha^*) \in E_{\alpha^*}\})$ .

Next, we sketch the constraint generation phase<sup>8</sup> for Core-ML. The algorithm sketch remains the same except that  $E$  now is a list<sup>9</sup> of equational constraints instead of a set. The behavior of action table for Core-ML is same as that for untyped lambda calculus except for the variable (a slight modification) and the let case as shown below:

**Case  $(\Gamma, x, \tau_0)$ .** If  $x$  is bound to a type scheme variable  $\alpha^*$  in  $\Gamma$ , i.e.  $x : \alpha^* \in \Gamma$ , then add  $\alpha^* \stackrel{i}{=} \tau_0$  to  $E$  else add  $\tau_0 \stackrel{e}{=} \tau_1$  (where  $x : \tau_1 \in \Gamma$ ) to  $E$ .

**Case  $(\Gamma, MN, \tau_0)$ .** Let  $\alpha$  be a fresh type variable. Generate subgoals  $(\Gamma, M, \alpha \rightarrow \tau_0)$  and  $(\Gamma, N, \alpha)$ , and add to  $G$ .

**Case  $(\Gamma, \lambda x. M, \tau_0)$ .** Let  $\alpha$  and  $\beta$  be two fresh type variables. Generate equation  $\tau_0 \stackrel{e}{=} \alpha \rightarrow \beta$  and sub-goal  $((\Gamma \setminus x) \cup \{x : \alpha\}, M, \beta)$ , and add to  $E$  &  $G$  respectively.

**Case  $(\Gamma, \text{let } x = M \text{ in } N, \tau_0)$ .** Let  $\alpha_1, \alpha_2^*$  be fresh type variables. Append<sup>10</sup>

$E_l @ [\alpha_1 \stackrel{s}{=} \Gamma \alpha_2^*] @ E_r$  to list  $E$ , where  $E_l, E_r$  are obtained by recursively calling the extended algorithm on  $(\Gamma, M, \alpha_1)$  and  $((\Gamma \setminus x) \cup \{x : \alpha_2^*\}, N, \tau_0)$  respectively.

The next few paragraphs highlight the constraint solving phase. This phase consists of two distinct unification phases: Phase I and Phase II. We first give an informal description of both the phases and follow it with a formal description. In the first phase, e-constraints are unified. Note that if there are no i&s-constraints, i.e. the term is a pure lambda term, then our Phase I mirrors the constraint solving phase for Wand's algorithm. In the second phase, a s-constraint and related i-constraints are chosen and transformed to e-constraints and unified using the Phase I unification. Let  $E = E_e @ E_{i\&s}$  be the constraint list obtained from the constraint generation phase, where  $E_e$  denotes a list containing e-constraints,  $E_{i\&s}$  denotes a list containing i&s-constraints. The constraint solving algorithm, *SOLVE*, integrates the two unification phases as follows:

$$\begin{aligned} \text{SOLVE}(E) = \\ \text{let } \rho_1 = \text{unify}_1 E_e \text{ in} \\ \rho_1 \circ (\text{unify}_2 \rho_1(E_{i\&s})) \end{aligned}$$

The first phase, *unify*<sub>1</sub> is defined as:

$$\begin{aligned} \text{unify}_1(E) = \\ \text{unify}_1((\alpha \stackrel{e}{=} \beta) :: E) &= \text{if } \alpha = \beta \text{ then } \text{unify}_1 E \\ \text{unify}_1((\alpha \stackrel{e}{=} \tau) :: E) &= \text{if } \alpha \text{ occurs in } \tau \text{ then raise Failure} \\ \text{unify}_1((\alpha \stackrel{e}{=} \tau) :: E) &= (\alpha \mapsto \tau) \circ \text{unify}_1(E[\alpha := \tau]) \\ \text{unify}_1((\tau \stackrel{e}{=} \alpha) :: E) &= \text{unify}_1((\alpha \stackrel{e}{=} \tau) :: E) \\ \text{unify}_1((\tau_1 \rightarrow \tau_2 \stackrel{e}{=} \tau_3 \rightarrow \tau_4) :: E) &= \text{unify}_1((\tau_1 \stackrel{e}{=} \tau_3) :: ((\tau_2 \stackrel{e}{=} \tau_4) :: E)) \end{aligned}$$

<sup>8</sup> We denote this constraint generation phase by *Wand*<sup>+</sup> in Fig. 3.

<sup>9</sup> This is needed to preserve the order of s-constraints.

<sup>10</sup> This will ensure that we solve the leftmost innermost let first.

The second phase,  $unify_2$ , is defined as:

$$\begin{aligned} unify_2(E' @ E) &= \text{let } \rho_1 = unify_1 E'' \\ &\quad \text{in } \rho_1 \circ unify_2(\rho_1 E) \\ &\quad \text{where } E'' = \{inst(gen(\Gamma, \tau_1)) \stackrel{\text{e}}{=} \tau_2 \mid (\alpha^* \stackrel{i}{=} \tau_2) \in E_{\alpha^*}\} \\ &\quad \text{and } E' = [(\tau_0 \stackrel{s}{=} \alpha^*), E_{\alpha^*}] \\ unify_2[] &= Id \end{aligned}$$

The first phase of our constraint solving algorithm is very similar to the algorithm proposed by Martelli and Montanari [MM82], which is known to be exponential<sup>11</sup> in the size of the input in the worst case [BS01]. Therefore, the first phase of the unification is linear while our implementation, which does not use the efficient DAG representation, is exponential in the size of the input in the worst case. The second phase of the algorithm involves calling Phase I algorithm as many times as the number of s-constraints, *i.e.*, the number of let-bound variables and there can be only  $O(n)$  of those. Therefore, in the worst case, the total time required by the algorithm is  $O(2^n)$ , where  $n$  is the size of the term.

## 6 Extension Correctness

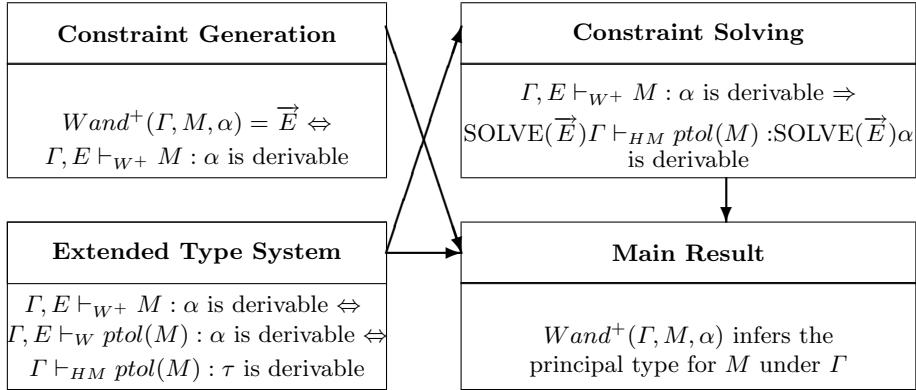
For the correctness result, we need a function to transform polymorphic lets to pure lambda terms since Wand's type system has no notion of let. We call this function  $ptol$ . Note that  $ptol$  is a type and value preserving transformation [KC07]. We introduce two additional notations before formally describing  $ptol$ . First, we use  $N[x]$  to denote one hole (denoted by  $[]$ ) in a context (denoted by  $N$ ) filled with the let-bound variable ( $x$  in this case). Second, we use the notation  $|FV(N)|_x$  to denote the count<sup>12</sup> of free occurrence of  $x$  in  $N$ . Notice that we transform only the body of the let since we assume the let-binding is let-free. This assumption is strictly not necessary since there is a type and value preserving transformation mentioned by Mairson [Mai89], which can make a let-binding let-free. We use the assumption to simplify our proofs but its certainly not a restriction on our algorithm. With the above notations, we describe  $ptol$  below:

$$\begin{aligned} ptol(x) &= x \\ ptol(\lambda x. M) &= \lambda x. ptol(M) \\ ptol(MN) &= (ptol(M) ptol(N)) \\ ptol(\text{let } x = M \text{ in } N[x]) &= \text{let } N_1 = ptol(N) \text{ in} && \text{if } |FV(N)|_x \leq 1 \\ &\quad (\lambda x. N_1[x]) M \\ ptol(\text{let } x = M \text{ in } N[x]) &= \text{let } N_1 = ptol(N) \text{ in} && \text{if } |FV(N)|_x > 1 \\ &\quad ptol(\text{let } x_1 = M \text{ in let } x = M \text{ in } N_1[x_1]) \\ &\quad \text{where } x_1 \text{ is a fresh variable.} \end{aligned}$$

<sup>11</sup> Linear if the input and output are represented as directed acyclic graph (DAG) [PW76].

<sup>12</sup> We use the *conservative* notion of occurrence of a let-bound variable rather than the actual types to differentiate between a monomorphic and polymorphic let.

The correctness is given by the proof sketch in Fig. 3. The most complicated proof was showing that the extended type system was sound and complete with respect to the Hindley-Milner type system via Wand’s type system. The detailed arguments involved in the correctness, the proofs of various theorems and lemmas, and a detailed discussion on the novel desugaring of polymorphic lets to pure lambda terms can be found in [KC07].



**Fig. 3. Correctness proof overview**

## 7 Conclusions and Future Work

The extension of Wand’s algorithm is a non-trivial extension and requires careful handling of constraints. Our algorithm is a *direct* extension of Wand’s algorithm and our soundness and completeness proofs rely on completeness and soundness of Wand’s algorithm. The main idea behind our algorithm is to preserve the order of generated s-constraints (as described in the action table for Core-ML) and use a multi-phase unification, while preserving this order, in the constraint solving phase. We have validated our approach by running the examples mentioned in this paper on Alg. W [Mil78,DM82], Alg. M [LY98] and Alg. J [Mil78]. An implementation of our algorithm and other popular type reconstruction algorithms in OCaml is available online at <http://www.cs.uwyo.edu/~skothari>. We are working on a formalization of the proofs in CoQ.

## Acknowledgments

Thanks to Bastiaan Heeren for a detailed response to author’s query regarding the constraint representations mentioned in his thesis. We also want to thank anonymous referees for their detailed comments and suggestions (on an earlier draft of this paper), which greatly improved the presentation of this paper.

## References

- [AW93] A. Aiken and E. L. Wimmers. Type inclusion constraints and type inference. In *Functional Programming Languages and Computer Architecture*, pages 31–41, 1993.
- [BS01] F. Baader and W. Snyder. Unification theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 8, pages 445–532. Elsevier Science, 2001.
- [DM82] L. Damas and R. Milner. Principal type-schemes for functional programs. In *POPL ’82*, pages 207–212, New York, 1982. ACM Press.
- [Hee05] B. Heeren. *Top Quality Type Error Messages*. PhD thesis, Universiteit Utrecht, 2005.
- [Hin69] J. R. Hindley. The principal type-scheme of an object in combinatory logic. *Trans. American Math. Soc.*, 146:29–60, 1969.
- [HLI03] B. Heeren, D. Leijen, and A. IJzendoorn. Helium, for learning haskell. In *Haskell ’03: Proceedings of the 2003 ACM SIGPLAN workshop on Haskell*, pages 62–71, New York, NY, USA, 2003. ACM Press.
- [Jon95] M. P. Jones. *Qualified types: theory and practice*. Cambridge University Press, New York, NY, USA, 1995.
- [KC07] S. Kothari and J. L. Caldwell. Wand’s Algorithm Extended for the Polymorphic ML-Let. Technical report, University of Wyoming, 2007.
- [Kot07] S. Kothari. Type Reconstruction Algorithms: A Survey. Technical report, University of Wyoming, 2007.
- [LY98] O. Lee and K. Yi. Proofs about a folklore let-polymorphic type inference algorithm. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 20(4):707–723, 1998.
- [Mai89] H. G. Mairson. Deciding ML typability is complete for deterministic exponential time. In *Proc. of the 16th ACM Sym. Principles of Programming Languages*, pages 382–401, 1989.
- [MH88] J. C. Mitchell and R. Harper. The essence of ML. In *POPL ’88: Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 28–46, 1988.
- [Mil78] R. Milner. A theory of type polymorphism in programming. *Journal of computer and system sciences*, pages 348–375, 1978.
- [MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282, 1982.
- [Pie02] B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.
- [PJ89] P.C.Kanellakis and J.C.Mitchell. Polymorphic unification and ML typing. In *6th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 105–115. ACM Press, 1989.
- [PR05] F. Pottier and D. Rémy. The essence of ML type inference. In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 10, pages 389–489. MIT Press, 2005.
- [PW76] M. S. Paterson and M. N. Wegman. Linear unification. In *STOC ’76: Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 181–186, New York, NY, USA, 1976. ACM.
- [SOW97] M. Sulzmann, M. Odersky, and M. Wehr. Type inference with constrained types. In *Fourth International Workshop on Foundations of Object-Oriented Programming (FOOL 4)*, 1997.
- [Wan87] M. Wand. A simple algorithm and proof for type inference. *Fundamenta Informaticae*, 10:115–122, 1987.

# Simulations Between Tilings

Gr  gory Lafitte<sup>1</sup> and Michael Weiss<sup>2,\*</sup>

<sup>1</sup> Laboratoire d' Informatique Fondamentale de Marseille (LIF), CNRS – Aix-Marseille Universit  , 39, rue Joliot-Curie, F-13453 Marseille Cedex 13, France

<sup>2</sup> Centre Universitaire d' Informatique, Universit   de Gen  ve, Battelle b  timent A  
7 route de Drize, CH-1227 Carouge, Switzerland

**Abstract.** Wang tiles are unit size squares with colored edges. To know whether a given finite set of Wang tiles can tile the plane while respecting colors on edges is undecidable. Robinson's tiling is a self-similar tiling where the computation of a Turing machine can be carried out. We introduce finer notions of simulation between tilings, giving rise to a finer notion of universality. We show how to use this construction to be able to simulate a tile set by another tile set through Turing machines. We study the computability of some problems related to simulation. Finally, we show how to build a tile set that can simulate all the periodic tile sets.

## 1 Introduction

Wang was the first to introduce in [13] the study of tilings with colored tiles. A tile is a unit size square with colored edges. Two tiles can be assembled if their common edge has the same color. A tiling consists in assembling tiles from a tile set (a finite set of tiles) on the grid  $\mathbb{Z}^2$ .

Wang conjectured that if a tile set tiles the plane, then it tiles it in a periodic way. In [3], Berger was the first to construct an aperiodic tile set, *i.e.*, a tile set which tiles the plane only in an aperiodic way. This proof showed that Wang's conjecture is false and that the domino problem is undecidable. This domino problem asks if one can decide whether given a tile set, there exists a tiling of the plane generated by this tile set. Simplified constructions of aperiodic tile sets can be found in [12] and later [1].

The main argument of this proof was to simulate the behavior of a given Turing machine with a tile set, in the sense that the Turing machine  $M$  stops on an instance  $w$  if and only if the tile set  $\tau_{\langle M, w \rangle}$  does not tile the plane. Therefore, one of the most important fact concerning tilings is that tilings can constitute a Turing equivalent computation model. This computation model is particularly relevant as a model of computation on the plane.

Tilings have been studied for different purposes: some researchers have used tilings as a tool for studying mathematical logical problems [1], others have studied the different kinds of tilings that one tile set can produce [5,6,12,10], or defined tools to quantify the regular structure of a tiling [7,2].

---

\* This author has been supported by the FNS grant 200020-113257.

The most used construction to simulate Turing machines with tile sets is Robinson's tiling. In [12], Robinson has built a tile set that generates only self-similar aperiodic tilings. The construction lays on a hierarchy of squares of ever increasing sizes. In each of this square, some zones can be used to simulate the behavior of a Turing machine. We show how to use this construction to be able to simulate a tile set by another tile set through Turing machines. In [10], notions of simulation and reduction between tilings and tile sets have lead to notions of universality for tilings and completeness for tile sets. From these definitions of simulation, we define finer notions of simulation between tile sets (totally and exactly). These new notions of simulation give rise to a finer notion of universality. We use our construction to study the computability of problems related to simulation. Then we use this construction to build tile sets that generate tilings with specific properties, *e.g.*, universality for a certain class.

The main result of the paper is to use Robinson's construction to build a recursive aperiodic tile set that totally simulates each and every periodic tile set and only them, *i.e.*, a tile set that is totally universal for periodic tile sets. This is an interesting tile set since the problem to know whether a tile set is periodic is undecidable. We then generalize this construction and give sufficient conditions for tile sets classes to be totally simulated by one tile set.

In section 2, we recall the basic notions of tilings and simulation between tile sets and give two definitions of simulation, the total and the exact simulations, and a definition of universality, which are finer than the ones introduced in [10]. In section 3, we show how we can simulate a Turing machine which stops on every input independently of the time and space needed for the computation. We use our construction to simulate a tile set with another tile set through the computation of a Turing machine in Robinson's tiling . In section 4, we study the computability of problems related to simulation and show that the problem to know whether a recursive tiling  $P$  reduces to another recursive tiling  $Q$  is  $\Sigma_2$ -complete. In the last section, we build a tile set that totally simulates each and every periodic tile set and only them. We then generalize this result to give sufficient conditions for a tile set class to be totally simulated by one tile set.

## 2 Notions of simulation

We start by recalling the basic notions of tilings. A tile is an oriented unit size square with colored edges from  $C$ , where  $C$  is a finite set of colors. A tile set is a finite set of tiles. To tile consists in placing the tiles of a given tile set on the grid  $\mathbb{Z}^2$  such that two adjacent tiles share the same color on their common edge. Since a tile set can be described with a finite set of integers, then we can enumerate the tile sets, and  $\tau_i$  designates the  $i^{th}$  tile set.

Let  $\tau$  be a tile set. A tiling  $P$  generated by  $\tau$  is called a  $\tau$ -tiling. It is associated to a tiling function  $f_P$  where  $f_P(x, y)$  gives the tile at position  $(x, y)$  in  $P$ . When we say that we superimpose the tiles of a tile set  $\tau$  on the tiles of a tile set  $\tau'$ , we mean that for any tile  $t \in \tau$  and any tile  $t' \in \tau'$ , we build a tile  $u = t \times t'$  where the colors of the sides of  $u$  are the cartesian product of the colors of the

sides of  $t$  and  $t'$ . Then two tiles  $u_1 = t_1 \times t'_1$  and  $u_2 = t_2 \times t'_2$  match if and only if  $t_1$  and  $t_2$  match and  $t'_1$  and  $t'_2$  match.

Different notions of reduction have been introduced in [10]. We recall some of the notions relative to these reductions and we refer the reader to this paper for detailed explanations and properties.

A pattern is a finite tiling. If it is generated by  $\tau$ , we call it a  $\tau$ -pattern. A finite set of rectangular  $\tau$ -patterns of same size is a  $\tau$ -pattern set. By analogy with tilings, to tile with a pattern set consists in placing the patterns on a regular subgrid of  $\mathbb{Z}^2$  in such a way that the connection between two patterns respects the local constraint of color matching. We call a tiling  $P$  generated by a pattern set  $M$ , a  $M$ -tiling. If  $M$  is a set of  $\tau$ -patterns, then for any  $M$ -tiling  $P$ , there exists a  $\tau$ -tiling  $Q$  that is a representation of  $P$  at the unit tile level.

From this remark we obtain notions of simulation. We say that our pattern tiling  $P$  simulates a tiling  $P'$  if there exists a mapping  $R$  from the patterns of  $P$  to the tiles of  $P'$  such that if we replace the patterns of  $P$  by their corresponding tiles given by  $R$ , then we obtain  $P'$ . In such a case, we write  $P' \leqslant^R P$  and say that  $P'$  reduces to  $P$ .

In [10], the following definitions were introduced: a tiling  $P$  is strongly universal if for any tile set  $\tau$ , there exists a  $\tau$ -tiling  $Q$  such that  $Q \leqslant P$  and a tile set  $\tau$  is complete if for any tile set  $\tau'$  and any  $\tau'$ -tiling  $Q$  there exists a  $\tau$ -tiling  $P$  such that  $Q \leqslant P$ .

The following definitions present finer notions of simulation:

**Definition 1.** Let  $\tau$  and  $\tau'$  be two tile sets. We say that  $\tau$  totally simulates  $\tau'$  if there exist  $a, b \in \mathbb{Z}$  and a reduction  $R$  from the  $a \times b$  patterns of  $\tau$  to the tiles of  $\tau'$  such that the two following conditions are respected:

1. for any  $\tau'$ -tiling  $Q$ , there exists a  $\tau$ -tiling  $P$  such that  $Q \leqslant^R P$ ,
2. for any  $\tau$ -tiling  $P$ , there exists a  $\tau'$ -tiling  $Q$  such that  $Q \leqslant^R P$ .

We denote it by  $\tau' \leqslant_t \tau$  (or  $\tau' \leqslant_t^R \tau$  to specify the reduction  $R$ ).

If  $\tau' \leqslant_t \tau$ , then there exists a reduction  $R$  such that any  $\tau$ -tiling can be cut in rectangle patterns of size  $a \times b$  such that if one replaces these patterns by their corresponding tiles given by  $R$  then one obtains a  $\tau'$ -tiling. And the set of all  $\tau'$ -tilings that reduce to a  $\tau$ -tiling is exactly the set of all  $\tau$ -tilings. The total simulation is thus more specific than the simulation introduced in [10]. In this way,  $\tau$  can be seen as a tile set which *computes* in a same way than  $\tau'$ .

To be able to study these notions of simulation, we now describe some specific aspects of the classical Robinson construction that we will use later on.

### 3 Some specific aspects of Robinson's construction

We study here the implication of the complexity of Turing machines on their simulation by Robinson's construction.

Since Berger's proof of the undecidability of the domino problem, we know how to simulate a Turing machine with a tiling. For a detailed explanation of the construction, we refer the reader to [1].

We now show that the simulation in Robinson's tiling of the computation of a given Turing machine, which stops on all input, does not depend on the time and space needed for the computation.

Notice that, a problem could occur if, given a Turing machine  $M$  with  $L(M)$  recursive, we wanted to simulate in the  $n^{th}$  square of Robinson's tiling (the square of size  $2^{2n} - 1$ ) the computation of  $M$  on the  $n^{th}$  word. Since in the  $n^{th}$  square of Robinson's tiling, we have a square of size  $2^n + 1$  dedicated to the computation, if  $M$  uses more than  $2^n + 1$  times or spaces to compute the result, then the space×time diagram of  $M$  on this  $n^{th}$  input will not fit in the  $n^{th}$  square.

Without loss of generality, we can suppose that  $M$  stops after  $(2^n + 1) \times s(n)$  spaces and  $k \times (2^n + 1) \times s(n)$  times. If we simulate this Turing machine in Robinson's tiling stretched vertically by a factor  $k$ , *i.e.*, where we have replaced the tiles of Robinson's tile set by rectangular patterns of size  $1 \times k$ , then the space×time diagram of the computation of  $M$  on the  $n^{th}$  word will enter exactly in the square of size  $((2^n + 1) \times s(n)) \times (k \times (2^n + 1) \times s(n))$ . We have to force the computation to be carried out in this rectangle.

The first line of a rectangle of computation in Robinson's tiling is filled in a non-deterministic way with a number of 1's followed by a 2 and then by  $\square$ 's (representing the empty symbol). The goal is to check whether  $1^n$  is the greatest input on which computation can be carried out in this rectangle. To do this, we simulate in parallel the computation of  $M$  on the input  $1^n$  and the computation of a modification of  $M$ , say  $M'$ , which works as  $M$  except that a 2 is seen as a 1. Therefore, if  $M$  computes on  $1^n$ , then  $M'$  simulates  $M$  computing on  $1^{n+1}$ . So,  $1^n$  is the greatest input whose computation can be simulated in this rectangle if and only if the computation on  $1^n$  enters in the rectangle and if the computation on  $1^{n+1}$  does not.

To be sure that we are in this case, if the computation of  $M$  on  $1^n$  stops, then a special color is sent by the final state to the north side of the rectangle in which the computation is made, which forces the whole border of this rectangle to be marked with this special color. Then we just have to add the condition that the computation of  $M'$  on  $1^n 2$  matches with the special colored border if and only if it does not reach a final state.

Therefore, a rectangle can be filled if and only if the input wrote on the first line is the greatest input whose computation can be simulated in this rectangle. At the end of our tiling process, our tile set simulates the computations of  $M$  on every input.

Thus, the simulation of a Turing machine in a tiling does not depend on the space and time needed to stop.

## 4 Computability of simulation

In this section, we study the computability of problems related to simulation. We define the following problems:

**Problem TSR** (Tile sets reduction)

- Input: Tile sets  $\tau, \tau'$ .
- Question: Does a  $\tau$ -tiling  $P$  and a  $\tau'$ -tiling  $Q$  exist such that  $P \leq Q$ ?

**Problem C-NESS** (Completeness)

- Input: Tile set  $\tau$ .
- Question: Is  $\tau$  complete?

**Problem TR** (Tilings reduction)

- Input: Two recursive tilings  $P, Q$ .
- Question: Does  $P \leq Q$ ?

We first show that the halting problem reduces to problem TR, using the above construction. The construction of this proof can be used to prove the same result for the other two problems. At the end of this section, we will show the exact location of problem TR in the arithmetical hierarchy.

**Theorem 1.** *Problem TR is undecidable.*

*Proof.* We show that the halting problem reduces to problem TR.

Let  $P$  be a deterministic tiling, *i.e.*, a tiling generated by a tile set that tiles the plane in an unique way, such that  $P$  does not reduce to Robinson's tiling. Let  $M$  be a one-tape Turing machine and  $w$  an input word. We build a recursive tiling  $Q$  such that  $P \leq Q \Leftrightarrow M(w) \downarrow$ . We consider the one-tape Turing machine  $M'$  which takes as input the empty word  $\epsilon$  and has the following behavior: it writes  $w$  on the tape and simulates  $M$ .

Let  $\tau_{M'}$  be the tile set which simulates  $M'$ . We superimpose  $\tau_{M'}$  on Robinson's tiling in such a way that the computation of  $M'$  is simulated in the free zones of the squares of size  $2^{2n} - 1$  in Robinson's tiling with the special condition that if the computation reaches a final state before reaching the square perimeter, then a special color is sent to the north side of the rectangle in which the computation is made, which forces the whole border of the rectangle to be marked with this special color.

Thus, a square is marked with the special color if and only if the square is big enough to allow the computation to reach a final state. Of course, if  $M$  stops on  $w$  using time  $t$  and space  $s$ , then the square of size  $2^{2m} - 1$ , where  $m \gg \max(\log_2(t), \log_2(s))$ , is big enough to allow the computation to reach a final state.

Therefore, if  $M(w) \downarrow$ , then there exists an integer  $n_0$  such that any square of size  $2^{2n} - 1$  can be filled with the computation of  $M'$  for all  $n \geq n_0$  and the squares of size  $2^{2l} - 1$  are squares without computation for any  $l < n_0$ .

In [10], we have shown how to build a Turing machine  $M$  which generates space $\times$ time diagrams which are isomorphic to a tile set  $\tau$ . We refer the reader to this paper for detailed explanations. Now, we consider the Turing machine  $M_P$  which simulates the tiles of the tiling  $P$  and the tile set  $\tau_P$  which simulates  $M_P$ . We superimpose  $\tau_P$  on  $\tau_{M'}$  with the condition that the computation of the tiles of  $P$  can begin if and only if the square in which the computation is made is marked with the special color. Then a square of size  $2^{2l} - 1$  can be a pattern that simulates a tile of  $P$  if and only if:

- the square is big enough to allow the simulation of  $M'$  to reach a final state,
- and the square is big enough to compute a tile of  $P$ .

If these two conditions are respected, then the squares of size  $2^{2l} - 1$  are the core of the patterns that simulate the tiles of  $P$ . We denote by  $Q$  the tiling generated by the tile set we have built until now, with the special condition that any square big enough to allow the simulation of  $M'$  does carry out the simulation. Since  $P$  is a deterministic tiling, then the tile can be assembled in a unique way and thus, if only one square can be filled with a pattern representing a tile of  $P$ , then  $Q$  simulates the tiling  $P$ . To be sure that  $P$  simulates  $Q$ , we impose that the pattern centered around  $(0, 0)$  in  $Q$  simulates the tiles of  $P$  at position  $(0, 0)$ .

We have to check that  $Q$  is a recursive tiling, *i.e.*, there exists a Turing machine which given as input two integers  $a, b$ , outputs the tile of  $P$  at position  $(a, b)$ . This machine has the following behavior:

- Find the smallest square containing position  $(a, b)$  in Robinson's tiling.
- If this position is out of a square of size  $2^{2n} - 1$ , output the tile of Robinson's tiling at position  $(a, b)$ .
- Otherwise, compute in the square of Robinson's tiling the tile set which simulates  $M'$  superimposed on the tile set which computes the tiles of  $P$ .
- If the computation reaches a final state and gives a pattern simulating a tile of  $P$  before reaching the square perimeter, then fill any square of size  $2^{2n} - 1$  with the computation of the patterns simulating the tiles of  $P$ , beginning by the one centered around the origin until we arrive to the square containing the position  $(a, b)$ .
- Output the tile at position  $(a, b)$ .
- If the computation does not halt in the square, output the tile of Robinson's tiling at position  $(a, b)$ .  $\square$

By using the same kind of construction, we show that the halting problem reduces to problems TSR and C-NESS:

**Corollary 1.** *Problem C-NESS and TSR are undecidable.*

We can precisely locate problem TR in the arithmetical hierarchy. The next result shows that in fact, problem 3 is equivalent to problem  $Fin = \{ i \mid \text{The Turing machine with code } i \text{ accepts a finite language } L(M_i) \}$  which is  $\Sigma_2$ -complete.

**Theorem 2.** *Problem TR is  $\Sigma_2$ -complete.*

*Proof.* [Problem TR  $\in \Sigma_2$ ]: Let  $P$  and  $Q$  be two recursive tilings. Then  $P \leq Q$  if there exist two integers  $a, b$  such that for any integer  $n$ , the  $n \times n$  pattern of  $P$  centered around the origin reduces to the  $na \times nb$  pattern of  $Q$  centered around the origin.

Therefore, problem TR can be defined as a  $\exists\forall$  arithmetical property.

[ $Fin \leq$  Problem TR]: Let  $i$  be the code of a Turing machine. We transform it in a pair of recursive functions  $(f, g)$  which represent two tilings  $P_f$  and  $P_g$  such that  $P_f \leq P_g$  if and only if  $L(M_i)$  is finite.

Let  $P_f$  be a deterministic tiling. We explain the behavior of the tiling function  $g$ . We use a tile set, say  $\tau$ , composed of a unicolor blue tile and four tiles with three sides blue and one red. In [10] it has been shown that this tile set can simulate any tile set with infinitely many different reductions. At step one, we begin to simulate  $P_{f(i)}$  with  $\tau$  by computing the tiles closest to  $(0, 0)$  and in parallel of this computation, we enumerate the words accepted by  $M_i$ . If no word is accepted, then  $g$  keeps tiling the plane with patterns that simulate  $P_f$ . If a word is accepted, we pass to the next step.

At step  $n$ , we have already tiled a rectangular pattern  $A$  that simulates a rectangular pattern  $B$  of  $P_f$ . We do not want anymore this pattern  $A$  to be a simulation of the pattern  $B$  of  $P_f$ . We thus consider a larger enough square around  $A$  that we tile with the unicolor tile to guarantee that  $A$  does not simulate  $B$  anymore. Let  $A'$  be this square. We start the simulation of  $P_f$ , using the pattern  $A'$  already tiled. In parallel, we still enumerate the words accepted by  $M_i$ . If no word is accepted, then  $g$  goes on simulating  $P_f$ . If a new word is accepted by  $M_i$ , we go on to the next step.

Therefore,  $g$  produces a tiling that simulates  $P_f$  if and only  $M_i$  accepts a finite set of words. If  $L(M_i)$  is infinite, the simulation never finishes and  $P_g$  does not simulate  $P_{f(i)}$ .  $\square$

In the following theorem, we give an upper bound on the computability of the C-NESS and TSR problems.

**Theorem 3.** *Problems C-NESS and TSR are respectively  $\Pi_3$  and  $\Sigma_2$ .*

*Proof.* i) In [10], it has been shown that a tile set is complete if it can generate a weakly dense universal tiling, *i.e.*, a tiling  $P$  such that for any tile set  $\tau$ , there exist two integers  $a, b$  and a reduction function  $R$  from the  $a \times b$  patterns of  $P$  to the tiles of  $\tau$ , such that the tiling  $P'$ , obtained by replacing the  $a \times b$  patterns of  $P$  by their corresponding tiles of  $\tau$  given by  $R$ , contains any  $\tau$ -pattern.

Thus, we have the following description of the problem C-NESS:  $\tau$  is complete if and only if for any tile set  $\tau'$ , there exist two integers  $a, b$  such that for any integer  $n$  and for any  $\tau'$ -pattern  $m'$  of size  $n$ , there exists an  $na \times nb$   $\tau$ -pattern to which  $m'$  reduces.

Therefore, problem C-NESS can be defined as a  $\forall\exists\forall$  arithmetical property.

- ii) This problem can be defined as follows: there exists a  $\tau$ -tiling  $P$  which reduces to a  $\tau'$ -tiling  $Q$  if and only if there exist two integers  $a, b$  such that for any  $n$ , there exists an  $n \times n$   $\tau$ -pattern which reduces to an  $na \times nb$   $\tau'$ -pattern.

Therefore, problem TSR can be defined as a  $\exists\forall$  arithmetical property.  $\square$

In the next section, we show how this construction of simulating a tile set by another tile set through Turing machines can be a powerful tool to build tile sets that generate tilings with specific properties.

## 5 Total simulation of all periodic tile sets

In this section, we build a recursive aperiodic tile set that totally simulates each and every periodic tile set and only them. This tile set is thus totally universal for periodic tile sets. This is an interesting tile set since the problem to know whether a tile set is periodic is undecidable. We then generalize this construction and give sufficient conditions for tile sets classes to be totally simulated by one tile set.

**Theorem 4.** *The set of periodic tile sets can be totally simulated by one tile set, i.e., there exists a tile set  $\tau$  such that for any periodic tile set  $\tau'$ ,  $\tau' \leq_t \tau$ .*

*Proof.* First of all, let  $f$  be the projection of a recursive one-one function from  $\mathbb{N}$  to  $\mathbb{N}^2$  such that  $f(n) \leq n$ . Thus, the set  $\{f(1), f(2), \dots\}$  contains infinitely many times any integer. We can make the assumption that  $f$  can be computed by a Turing machine  $M_f$  in time and space  $t$  with unary input.

We now build a one-tape Turing machine  $M$  with the following behavior: it takes as input a word written in unary ( $1^n$ ), computes  $1^{f(n)}$  by simulating  $M_f$ , and then simulates the tile set  $\tau_{f(n)}$  to enumerate the rectangular patterns generated by this tile set. If a periodic pattern is found, then the computation stops and never halts otherwise.

We now simulate this Turing machine in Robinson's tiling stretched vertically by a factor 2. We have seen that the simulation of the computation of a Turing machine does not depend on the time and space needed to stop. In the square of size  $2^n - 1$  we simulate  $M_f$  on the greatest input  $1^m$  whose computation fits in this square. It thus outputs  $1^{f(m)}$ . Since Robinson's tiling is stretched with a factor 2, then we still have remaining spaces where we can compute  $M$ .

We simulate  $M$  in this tiling with the special condition that if a final state is reached, then a special color is sent to the north side of the rectangle in which the computation is made, which forces the whole border of the rectangle to be marked with this special color. If no final state is reached, then the computation keeps going on until it matches with the north side of the rectangle and the special color cannot be added to the sides of this rectangle.

This special color triggers the beginning of the computation of a new Turing machine, say  $N$ , in this rectangle. The behavior of  $N$  is the following: an input is an integer  $x$  - the index of a tile set - and an integer  $y$  - the index of a color.

We check whether  $y$  is a valid color of the tile set  $\tau_x$ . If yes, we choose in a non-deterministic way a tile of  $\tau_x$  with south color  $y$ , and we simulate this tile, *i.e.*, we build a space $\times$ time diagram which is a pattern representing this tile.

We simulate  $N$  in our rectangle with the following conditions:

- The computation of  $N$  can begin if and only if the rectangle is marked with the special color.
- The first line of the rectangle marked with the special color is filled in a non-deterministic way with the letters of the alphabet. The computation of  $N$  will say whether the input was valid or not, and if it is not, then the computation stops and the tiling cannot be completed to constitute a full tiling of the plane.
- Since the choice of  $x$  and  $y$  are made in a non-deterministic way, we have to check that  $x$  is the code of the tile set that  $M_f$  has computed before (even if it was not actually computed *before* since in our tiling the computations are made simultaneously). To do this, the bits of  $x$  are sent to the north and have to match with the value of  $1^{f(n)}$  to let the computation go on.

Therefore, at the end, the computation of  $N$  is carried out in the rectangle if and only if  $M$  has stopped, *i.e.*, if the tile set  $\tau_{f(n)}$  is periodic, and if and only if  $N$  has simulated a tile of  $\tau_{f(n)}$ .

The final stage consists in sending the colors of the simulation of the tiles of  $\tau_{f(n)}$  outside the rectangle to be sure that two Robinson rectangles simulating the tiles of  $\tau_{f(n)}$  match if and only if the tiles they represent match.

Since the index of any tile set is enumerated infinitely many times by  $f$ , then if a tile set  $\tau$  is periodic, then there exist infinitely many squares big enough to carry out the computation of  $M$  to find a periodic  $\tau$ -pattern and to stop, and thus, to trigger the beginning of the simulation of the tiles of  $\tau$ . Therefore, the periodic tile sets are totally simulated by our tile set.  $\square$

In this proof, we have used the fact that the property is recursively enumerable, and that no tile set, that does not tile the plane, has the property, here being periodic. If a tile set that does not tile the plane has the property, then when we simulate the tiles of this tile set, we arrive at a step where no more tiles can be simulated and the tiling process goes unfinished. We say that a property is *decent* if the class of tile sets satisfying it is recursively enumerable and if this property can not be verified by any tile set that does not tile the plane. This is equivalent to having a recursively enumerable property such that if a tile set can generate a pattern having this property, then it tiles the plane. Thus, we have the following generalization:

**Theorem 5.** *Let  $P$  be a decent property, then the set  $L_P = \{ i \mid \tau_i \text{ satisfies } P \}$  can be totally simulated by one tile set.*

This is equivalent to say that there exists a tile set which is totally universal for  $L_P$ . We finish this section by giving some classes of tile sets that can be totally simulated by one tile set:

**Corollary 2.** *The class of tile sets that generate a period using all of its tiles can be totally simulated by one tile set.*

The next corollary shows that the class of tile sets that exactly reduces to a tile set that tiles the plane can be totally simulated by one tile set:

**Corollary 3.** *Let  $\tau$  be a tile set that tiles the plane. The following set can be totally simulated by one tile set:*

$$L = \{ i \mid \tau_i \leqslant_{\equiv} \tau \}$$

*Proof.*  $L$  is recursively enumerable, since it is enough to enumerate the patterns generated by  $\tau_i$  and to find whether they are isomorphic to the tiles of  $\tau$ . Since  $\tau$  tiles the plane, no tile set that does not tile the plane satisfies the property.  $\square$

## References

1. ALLAUZEN (C.) and DURAND (B.), *The Classical Decision Problem*, appendix A: “Tiling problems”, p. 407–420. Springer, 1996.
2. BALLIER (A.), DURAND (B.) and JEANDEL (E.), *Structural Aspects of Tilings*, in *Proceedings of the Symposium on Theoretical Aspects of Computer Science*, Dagstuhl Seminar Proceedings n° **08001**, p. 61–72, 2008.
3. BERGER (R.), « The undecidability of the domino problem », *Memoirs of the American Mathematical Society*, vol. **66**, 1966, p. 1–72.
4. CERVELLE (J.) and DURAND (B.), « Tilings: recursivity and regularity », *Theoretical Computer Science*, vol. **310**, n° 1-3, 2004, p. 469–477.
5. CULIK II (K.) and KARI (J.), « On aperiodic sets of Wang tiles », in *Foundations of Computer Science: Potential - Theory - Cognition*, p. 153–162, 1997.
6. DURAND (B.), LEVIN (L. A.) and SHEN (A.), « Complex tilings », in *Proceedings of the Symposium on Theory of Computing*, p. 732–739, 2001.
7. DURAND (B.), « Tilings and quasiperiodicity », *Theoretical Computer Science*, vol. **221**, n° 1-2, 1999, p. 61–75.
8. DURAND (B.), « De la logique aux pavages », *Theoretical Computer Science*, vol. **281**, n° 1-2, 2002, p. 311–324.
9. HANF (W. P.), « Non-recursive tilings of the plane. I », *Journal of Symbolic Logic*, vol. **39**, n° 2, 1974, p. 283–285.
10. LAFITTE (G.) and WEISS (M.), « Universal Tilings », in *Proceedings of the Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science n° **4393**, p. 367–380, 2007.
11. MYERS (D.), « Non-recursive tilings of the plane. II », *Journal of Symbolic Logic*, vol. **39**, n° 2, 1974, p. 286–294.
12. ROBINSON (R.), « Undecidability and nonperiodicity for tilings of the plane », *Inventiones Mathematicae*, vol. **12**, 1971, p. 177–209.
13. WANG (H.), « Proving theorems by pattern recognition II », *Bell System Technical Journal*, vol. **40**, 1961, p. 1–41.
14. WANG (H.), « Dominoes and the  $\forall \exists \forall$ -case of the decision problem », in *Proceedings of the Symposium on Mathematical Theory of automata*, p. 23–55, 1962.

# Discrete Non Determinism and Nash Equilibria for Strategy-Based Games

Stéphane Le Roux\*

École normale supérieure de Lyon, Université de Lyon, LIP, CNRS, INRIA, UCBL  
<http://perso.ens-lyon.fr/stephane.le.roux/>

**Abstract.** Not all strategic games have Nash equilibria, so Nash defined probabilistic equilibria, which exist in all strategic games. However, the probability approach fails for the slightly more general *abstract strategic games* that are defined in this paper. Instead, this paper uses *discrete non determinism*, which yields relevant notions of *equilibrium with guaranteed existence*. These equilibria are payoff-wise *efficient and easy to find*. Moreover, the discrete approach still works for much more general games supporting both sequential and simultaneous decision-making.

**Key words:** Abstract strategic games, games in graphs, discrete non determinism, Nash equilibrium, fixed-point, constructive existence, efficient algorithm.

## 1 Introduction

Not all finite strategic games have (pure) Nash equilibria. To cope with this, Nash [8] defined probabilistic equilibria and proved their existence, *i.e.* he derived from a finite game a continuous game that has a (pure) Nash equilibrium.

This paper summarises [5], but without the proofs: it generalises strategic games by replacing real-valued payoff functions with abstract outcomes; it translates the notion of Nash equilibrium from the real-valued to the abstract setting, by replacing the usual order over the reals with one binary relation per agent in order to account for his preferences over outcomes, as in [9]. However like in the real-valued case, some *abstract strategic games* have no Nash equilibrium.

Unlike for real-valued games, using probabilities to weaken the notion of strategy is irrelevant in this abstract setting, as detailed in [5] or [6]. So, this paper says that an agent *does not choose* (resp. *may choose*) a strategy instead of saying that he chooses the strategy with probability 0 (resp.  $p > 0$ ). The set of strategies that may be chosen by an agent is named a discrete non deterministic (nd for short) strategy, similarly to [4]. Unlike Nash's probability approach, this weakening keeps things discrete and finite, *i.e.* easy to handle.

Five relevant extensions of Nash equilibrium, named *nd Nash equilibrium*, are defined as follows on the *nd strategy profiles*: First, define *nd games* and their *nd equilibria*, and five embeddings of abstract strategic games into nd games. Then,

---

\* Now postdoc at INRIA-Microsoft Research. I thank Pierre Lescanne for his comments on the draft of this paper, and a reviewer for his suggestions.

define the nd Nash equilibria as the preimages of the nd equilibria by the five embeddings respectively. In the five cases, nd Nash equilibrium existence follows nd equilibrium existence, which follows a preliminary pre-fixed point result.

The constructive proofs of existence of this paper give (exact) algorithms computing nd Nash equilibria. Time complexity happens to be polynomial for the five embeddings (yet might be exponential for other embeddings). In the real-valued setting, playing according to the computed nd Nash equilibrium may give better average payoff than playing randomly. Together with a robustness argument, it shows that these nd Nash equilibria can serve as recommendations, whereas an example shows that probabilistic Nash equilibria cannot.

This paper also defines multi strategic games, *i.e.* graphs whose nodes are abstract strategic games. A play in such a game is a sequence of local plays of strategic games each of whose outcome points to the next node. Multi strategic games can thus model within a single game both sequential and simultaneous decision-making mechanisms. Embedding them into nd games also provides them with a notion of non deterministic Nash equilibrium, with an existence result.

Section 2 defines abstract strategic games and their Nash equilibria; section 3 states a pre-fixed point result; section 4 defines nd games and their nd equilibria, and states existence results; section 5 embeds abstract strategic games into nd games in five ways, thus providing the formers with five notions of nd Nash equilibrium, with existence results. An example and efficiency results follow; informal section 6 defines multi strategic games and embeds them into nd games, thus providing them with a notion of nd Nash equilibrium.

**Conventions:** the notation  $P \stackrel{\Delta}{=} Q$  defines  $P$  as coinciding with  $Q$ .

Let  $E = \prod_{i \in I} E_i$  be a cartesian product. For  $e$  in  $E$ , let  $e_i$  be the  $E_i$ -component of  $e$ . Let  $E_{-i}$  denote  $\prod_{j \in I - \{i\}} E_j$ . For  $e$  in  $E$ , let  $e_{-i}$  be  $(\dots, e_{i-1}, e_{i+1}, \dots)$  the projection of  $e$  on  $E_{-i}$ . For  $x$  in  $E_i$  and  $X$  in  $E_{-i}$ , define  $X; x$  in  $E$  as  $(X; x)_i \stackrel{\Delta}{=} x$  and for all  $j \neq i$ ,  $(X; x)_j \stackrel{\Delta}{=} X_j$ .

## 2 Abstract Strategic Games

This section defines abstract strategic games and their Nash equilibria.

Abstract strategic games are strategic games whose real-valued payoff functions are replaced with abstract outcomes. In addition for each agent, an *arbitrary* binary relation accounts explicitly for the agent's preference over outcomes. These relations replace the (implicit) usual total order over the real numbers.

**Definition 1 (Abstract strategic games)** *Abstract strategic games are 4-tuples  $\langle \mathcal{A}, S, P, (\triangleleft_a)_{a \in \mathcal{A}} \rangle$  where:*

- $\mathcal{A}$  is a non-empty set of agents,
- $S = \bigotimes_{a \in \mathcal{A}} S_a$  is the Cartesian product of non-empty sets of strategies,
- $P : S \rightarrow O_c$  is a function mapping strategy profiles to given outcomes,
- $\triangleleft_a$  is a relation over outcomes, and  $o \triangleleft_a o'$  says that agent  $a$  prefers  $o'$  to  $o$ .

For  $s$  and  $s'$  in  $S$ ,  $s \triangleleft_a s'$  stands for  $P(s) \triangleleft_a P(s')$ , by abuse of notation.

Below, the left-hand strategic game is real-valued, as the right-hand one is abstract. Both games involve agent  $v$  (with strategies  $v_1$  and  $v_2$ ) and agent  $h$ .

	$h_1$	$h_2$		$h_1$	$h_2$
$v_1$	$v \mapsto 0, h \mapsto 2$	$v \mapsto 1, h \mapsto 1$		$v_1$	$oc_1$
$v_2$	$v \mapsto 0, h \mapsto 1$	$v \mapsto 2, h \mapsto 0$		$v_2$	$oc_3$

The following notion of happiness (first item of definition 2) generalises the traditional one, *i.e.* they coincide on real-valued strategic games. Happiness is also characterised using the notion of best response (second item). As usual, Nash equilibrium is defined through happiness for all agents (third item).

**Definition 2 (Happiness, Nash equilibrium)** *Let  $g = \langle \mathcal{A}, S, P, (\triangleleft_a)_{a \in \mathcal{A}} \rangle$ .*

$$\begin{aligned} Happy_g(a, s) &\stackrel{\Delta}{=} \forall s' \in S, \neg(s' \triangleleft_a s \wedge s \triangleleft_a s') \\ Happy_g(a, s) &\Leftrightarrow s_a \in BR_a(s_{-a}) \stackrel{\Delta}{=} \{x \in S_a \mid \forall y \in S_a, \neg(s_{-a}; x \triangleleft_a s_{-a}; y)\} \\ Eq_g(s) &\stackrel{\Delta}{=} \forall a \in \mathcal{A}, Happy_g(a, s) \end{aligned}$$

For instance in the abstract strategic game before definition 2, the profile  $(v_1, h_1)$  is a Nash equilibrium *iff*  $\neg(oc_1 \triangleleft_v oc_3)$ ,  $\neg(oc_1 \triangleleft_h oc_2)$ ,  $\neg(oc_1 \triangleleft_v oc_1)$ , and  $\neg(oc_1 \triangleleft_h oc_1)$ . Also, note that there is a natural embedding of real-valued strategic games into abstract ones that preserves and reflects Nash equilibria.

### 3 A Preliminary Pre-Fixed Point Result

This section proves a pre-fixed-point result, *i.e.* the existence of a  $y$  such that  $y \preceq F(y)$  for all  $F$  and  $\preceq$  that comply with given constraints.

First, meet semi-lattices are defined below like in the literature.

**Definition 3 (Meet semi-lattice)** *A meet semi-lattice is a partially ordered set  $(S, \preceq)$  each of whose 2-element subset has a greatest  $\preceq$  lower bound.*

In the rest of this section,  $(S, \preceq)$  is a generic meet semi-lattice with least element  $m$ ;  $inf$  is the infimum function related to  $\preceq$ ;  $F$  is a generic function from  $S$  to  $S$ . As defined below, a meeting point is an  $x$  in the lattice such that every sequence decreasing from  $x$  is not too much "scattered" by  $F$ .

**Definition 4 (Meeting point)** *Let  $x$  be in  $S - \{m\}$ . If for all  $x_1 \dots x_n$  such that  $m \neq x_1 \preceq \dots \preceq x_n \preceq x$ , we have  $inf(F(x_1), \dots, F(x_n), x) \neq m$ , then  $x$  is said to be a  $F$ -meeting point, and one writes  $M_F(x)$ .*

The  $F$ -meeting point predicate is preserved by the greatest lower bound operator used with the meeting point and its image by  $F$ , as stated below.

**Lemma 5**  $M_F(x) \Rightarrow M_F \circ inf(x, F(x))$ .

If there is a  $F$ -meeting point but no infinite strictly decreasing sequence, iteration of lemma 5 yields a non-trivial pre fixed point of  $F$ , as stated below.

**Lemma 6** *If  $\preceq$  is well-founded and if there exists a  $F$ -meeting point, then there exists a  $F$  pre fixed point different from  $m$ , *i.e.*  $y \preceq F(y)$  for some  $y \neq m$ .*

## 4 Non Deterministic Games

This section defines non deterministic games, nd games for short, and their nd equilibria. Three results of existence of nd equilibrium follow. In this paper, these definitions and results are merely general intermediate tools.

Informally, nd games generalise real-valued or abstract strategic games in three ways: First, a nd game involves a *set I* of "strategic games". Second, for each  $i$  in  $I$ , each agent  $a$  selects *one or more* "pure strategies" in  $S_a^i$ . This discrete non determinism amounts to forgetting the probabilities of a probabilistic strategy while remembering whether or not they equal zero. Third, there is no payoff function or abstract outcome, but each agent is given a *best-response function* telling what his best nd strategy against his opponents' nd strategies is.

**Definition 7 (Nd games)** A nd game is a pair  $\langle (S_a^i)_{a \in \mathcal{A}}, (BR_a)_{a \in \mathcal{A}} \rangle$  where:

- $I$  is a non-empty set of indices and  $\mathcal{A}$  is a non-empty set of agents.
- For all  $a$  in  $\mathcal{A}$ ,  $BR_a$  is a function from  $\Sigma_{-a}$  to  $\Sigma_a$ ,  
where  $\Sigma = \bigotimes_{a \in \mathcal{A}} \Sigma_a$  and  $\Sigma_a = \bigotimes_{i \in I} \mathcal{P}(S_a^i) - \{\emptyset\}$  and  $\Sigma_{-a} = \bigotimes_{a' \in \mathcal{A} - \{a\}} \Sigma_{a'}$ .

Elements of  $\Sigma_a$  (resp.  $\Sigma$ ) are called nd strategies for  $a$  (resp. nd strategy profiles).

In the rest of this section,  $g = \langle (S_a^i)_{a \in \mathcal{A}}, (BR_a)_{a \in \mathcal{A}} \rangle$  is a generic nd game,  $\sigma$  is a generic nd profile in the corresponding  $\Sigma$ , and  $a$  is a generic agent.

In definition 8, agent  $a$  is happy with  $\sigma$  if his own nd strategy  $\sigma_a$  is included in his nd best response against his opponents' combined nd strategies, i.e.  $BR_a(\sigma_{-a})$ . So, informally, an agent is happy if any "pure strategy" that he selects is a "pure best response" against others' nd strategies. This inclusion generalises the membership that is used in happiness characterisation of definition 2. Like for Nash equilibria, nd equilibria amount to happiness for all agents.

**Definition 8 (Happiness and non deterministic equilibrium)**

$$\begin{aligned} Happy_g(\sigma, a) &\triangleq \sigma_a \subseteq BR_a(\sigma_{-a}) \Leftrightarrow \forall i \in I, \sigma_{a,i} \subseteq (BR_a(\sigma_{-a}))_i \in \mathcal{P}(S_a^i) \\ Eq_g(\sigma) &\triangleq \forall a \in \mathcal{A}, Happy_g(\sigma, a) \\ Eq_g(\sigma) &\Leftrightarrow \sigma \subseteq BR(\sigma) \text{ where } BR(\sigma) \triangleq \bigotimes_{a \in \mathcal{A}} BR_a(\sigma_{-a}) \end{aligned}$$

Below, a meet lattice is identified within the nd game, and invoking lemma 6 yields a first equilibrium existence result.

**Lemma 9** The poset  $(\Sigma \cup \{\emptyset\}, \subseteq)$  is a meet semi-lattice with least element  $\emptyset$ , and if there exists a BR-meeting point in  $\Sigma$ , there exists a nd equilibrium for  $g$ .

The next equilibrium existence result invokes lemma 9 and the mentionned  $\sigma$  being a BR-meeting point. Lemma 10 is invoked in the rest of the paper.

**Lemma 10** Assume that for all agents  $a$ , for all  $\gamma_1 \dots \gamma_n$  in  $\Sigma_{-a}$ , if  $\gamma_n \subseteq \dots \subseteq \gamma_1 \subseteq \sigma_{-a}$  then  $\cap_{1 \leq k \leq n} BR_a(\gamma_k) \cap \sigma_a \neq \emptyset$ . Then, the game  $g$  has a nd equilibrium.

Also, equilibria are preserved when "increasing" the best response functions.

**Lemma 11** Let  $g' = \langle (S_a^i)_{a \in \mathcal{A}}, (BR'_a)_{a \in \mathcal{A}} \rangle$  be another nd game such that for all  $a$  in  $\mathcal{A}$ , for all  $\gamma$  in  $\Sigma_{-a}$ ,  $BR_a(\gamma) \subseteq BR'_a(\gamma)$ . Then  $Eq_g(\sigma) \Rightarrow Eq_{g'}(\sigma)$ .

## 5 Non Deterministic Nash Equilibria

Like real-valued strategic games, not all abstract ones have Nash equilibria; unlike for real-valued strategic games, using probabilities to weaken the notion of strategy is irrelevant for abstract ones, as detailed in [5]. This section invokes the concept of discrete non determinism of section 4 instead. Informally, a *nd* strategy for an agent of an abstract strategic game is a non-empty subset of the set of his (pure) strategies. Then, if a strategy belongs to the *nd* strategy, the agent may eventually choose it; if it does not, he will not. Conceptually, *nd* strategies lie between strategies and probabilistic strategies, while being simpler than the latters. Indeed, a probabilistic strategy induces a *nd* strategy, but it further specifies that the agent may choose strategies with given probabilities.

By embeddings into *nd* games, this section provides abstract strategic games with five notions of *non deterministic Nash equilibrium*. Existence results follow. This section also discusses the complexity of finding such equilibria, gives an example, and shows efficiency of these equilibria.

**Definitions and existence:** The following definition extends orders over functions' codomains to orders over functions, in a Pareto style.

**Definition 12** Let  $f$  and  $f'$  be functions of type  $A \rightarrow B$ , and let  $A'$  be a subset of  $A$ . Let  $\prec$  be an irreflexive relation over  $B$ , and let  $\preceq$  be its reflexive closure.

$$\begin{aligned} f \preceq^{A'} f' &\triangleq \forall x \in A', f(x) \preceq f'(x) \\ f \prec^{A'} f' &\triangleq f \preceq^{A'} f' \wedge \exists x \in A', f(x) \prec f'(x) \end{aligned}$$

For instance if both functions are represented by vectors of naturals with the usual total order,  $(1, 1) < (1, 2)$ ,  $(0, 2) < (1, 2)$  and  $(0, 2) < (1, 3)$ , but  $(2, 0) \not\prec (1, 3)$ . The extension preserves strict partial ordering, as stated below.

**Lemma 13** If  $\prec$  is a strict partial order over  $B$ , then the derived  $\prec^A$  over functions of type  $A \rightarrow B$  is also a strict partial order.

The following lemma is proved by induction on the mentioned  $n$ .

**Lemma 14** Let  $E$  be a finite set of functions of type  $A \rightarrow B$ . Let  $\prec$  be irreflexive and transitive over  $B$ , and let  $\preceq$  be its reflexive closure. For any  $\emptyset \neq A_0 \subseteq \dots \subseteq A_n \subseteq A$  there is an  $f$  that is maximal in  $E$  wrt all the  $\prec^{A_i}$ .

In the rest of this subsection  $g = \langle \mathcal{A}, S, P, (\triangleleft_a)_{a \in \mathcal{A}} \rangle$  is a generic abstract strategic game,  $a$  is a generic agent, and  $\gamma$  is a generic element in the corresponding  $\Sigma_{-a}$ . Also, the  $\triangleleft_a$  are strict partial orders, i.e. irreflexive and transitive.

Five embeddings of  $g$  into *nd* games are defined below. The five images of  $g$  have *nd* equilibria by lemmas 10, 11 and 14. This provides abstract strategic games with five definitions of *non-deterministic Nash equilibrium* and guarantees of existence. In the notation  $s \triangleleft_a^\gamma s'$  below, the strategies  $s$  and  $s'$  are seen as functions from  $S_a$  to the outcomes and  $\gamma$  corresponds to  $A'$  in definition 12.

**Lemma 15** For  $i$  between 0 and 4,  $\langle (S_a)_{a \in \mathcal{A}}, (BR_a^i)_{a \in \mathcal{A}} \rangle$  is a nd game (derived from  $g$ ), and it has a nd equilibrium that is named nd Nash equilibrium for  $g$ .

$$\begin{aligned} BR_a^0(\gamma) &\triangleq \{s \in S_a \mid \forall s' \in S_a, \neg(s \triangleleft_a^{S-a} s') \wedge \\ &\quad \forall s' \in S_a, \neg(s \triangleleft_a^\gamma s') \wedge \\ &\quad \exists c \in \gamma, \forall s' \in S_a, \neg(s \triangleleft_a^{\{c\}} s')\} \\ BR_a^1(\gamma) &\triangleq \{s \in S_a \mid \forall s' \in S_a, \neg(s \triangleleft_a^{S-a} s')\} \\ BR_a^2(\gamma) &\triangleq \{s \in S_a \mid \forall s' \in S_a, \neg(s \triangleleft_a^\gamma s')\} \\ BR_a^3(\gamma) &\triangleq \{s \in S_a \mid \exists c \in \gamma, \forall s' \in S_a, \neg(s \triangleleft_a^{\{c\}} s')\} \\ BR_a^4(\gamma) &\triangleq \{s \in S_a \mid \forall s' \in S_a, \exists c \in \gamma, \neg(s \triangleleft_a^{\{c\}} s')\} \end{aligned}$$

The  $BR_a^i$  for  $i = 1, 2, 3$  correspond to the three conjuncts of  $BR_a^0$ . (Note that  $BR_a^1(\gamma)$  does not depend on  $\gamma$ .) Also,  $BR_a^4$  is even more generous than  $BR_a^2$ . Depending on the application, a user of this theory may choose either of these definitions or design his own definition of nd Nash equilibrium. Note that  $BR^2$  and  $BR^4$  somewhat relate to the notions of dominated strategy, studied in [2] and [7], and rationalizability, studied in [1] and [10]. These notions are more recently discussed in [3], for instance.

The rest of the subsection discusses a few properties of these equilibria. First, the equilibria related to  $BR^1$  form a simple structure, as stated below.

**Lemma 16** The nd equilibria related to  $BR^1$  are the elements of  $\Sigma$  that are included in  $\bigotimes_{a \in \mathcal{A}} \{s \in S_a \mid \forall s' \in S_a, \neg(s \triangleleft_a^{S-a} s')\}$ .

The following lemma states that the nd Nash equilibria related to  $BR^3$  or  $BR^4$  define a Cartesian union lattice.

**Lemma 17** If equilibrium is defined through either  $BR^3$  or  $BR^4$ , then  $Eq_g(\sigma) \wedge Eq_g(\sigma') \Rightarrow Eq_g(\sigma \cup^\times \sigma')$ , where  $\sigma \cup^\times \sigma'$  is the smallest element of  $\Sigma$  containing both  $\sigma$  and  $\sigma'$ .

**Algorithmic time complexity:** The constructive proof of lemma 15 yields algorithms finding a nd Nash equilibrium for each  $BR^i$ . Computations relate to iteration of lemma 5, i.e. successive calls to  $BR^i$  starting with  $\gamma = S$ , the full nd strategy profile. For the five  $BR^i$ , overall complexity in calls to preferences is polynomial in the number of profiles  $|S|$ : indeed, a call to  $BR^i$  dismisses at least one strategy of one agent (unless the nd profile is a nd Nash equilibrium), so there are at most  $|S|$  such calls; each of them calls the  $BR_a^i$  which decides whether or not each strategy of agent  $a$  is a best response (if agent  $a$  has only one strategy, no need to call  $BR_a^i$ ); there are at most  $|S|$  such decisions per call to  $BR^i$ . For  $BR^0$ , this decision requires at most  $3 \times |S|$  calls to a  $\triangleleft_a$ . So, finding a  $BR^0$ -equilibrium (similarly  $BR^i$ -equilibrium) is at most cubic in  $|S|$ .

**Example:** The strategic game below uses naturals: the profile  $(v_1, h_2)$  induces payoff 3 for agent  $v$  and 2 for  $h$  and preferences are the usual order over naturals. Let us apply the  $BR^0$ -equilibrium algorithm to the game, starting with the full nd profile  $(\{v_1, v_2, v_3, v_4, v_5\}, \{h_1, h_2, h_3, h_4, h_5\})$ , as meant by the underlining.

	<u><math>h_1</math></u>	<u><math>h_2</math></u>	<u><math>h_3</math></u>	<u><math>h_4</math></u>	<u><math>h_5</math></u>		<u><math>h_1</math></u>	<u><math>h_2</math></u>	<u><math>h_3</math></u>	<u><math>h_4</math></u>	<u><math>h_5</math></u>		<u><math>h_1</math></u>	<u><math>h_2</math></u>	<u><math>h_3</math></u>	<u><math>h_4</math></u>	<u><math>h_5</math></u>
<u><math>v_1</math></u>	0,0	3,2	2,2	2,1	3,0	<u><math>v_1</math></u>	0,0	3,2	2,2	2,1	3,0	<u><math>v_1</math></u>	0,0	3,2	2,2	2,1	3,0
<u><math>v_2</math></u>	3,3	0,2	0,0	2,1	3,2	<u><math>v_2</math></u>	3,3	0,2	0,0	2,1	3,2	<u><math>v_2</math></u>	3,3	0,2	0,0	2,1	3,2
<u><math>v_3</math></u>	2,2	1,0	0,1	3,1	0,2	<u><math>v_3</math></u>	2,2	1,0	0,1	3,1	0,2	<u><math>v_3</math></u>	2,2	1,0	0,1	3,1	0,2
<u><math>v_4</math></u>	1,0	1,2	1,2	1,2	1,0	<u><math>v_4</math></u>	1,0	1,2	1,2	1,2	1,0	<u><math>v_4</math></u>	1,0	1,2	1,2	1,2	1,0
<u><math>v_5</math></u>	2,0	1,0	0,0	0,1	0,0	<u><math>v_5</math></u>	2,0	1,0	0,0	0,1	0,0	<u><math>v_5</math></u>	2,0	1,0	0,0	0,1	0,0

A priori,  $v$  (resp.  $h$ ) may play either of the  $v_i$  (resp.  $h_i$ ). In this context,  $h_5$  (resp. row  $v_5$ ) is smaller than  $h_1$  (resp.  $v_3$ ) according to agent  $h$  (resp.  $v$ ). So  $h_5$  (resp.  $v_5$ ) is not a best response of  $h$  (resp.  $v$ ). Also,  $v_4$  is not a best response of  $v$  because for each  $h_i$ , row  $v_4$  is smaller than some other row. So, the combined best responses against the full nd profile are  $v_1$  to  $v_3$  and  $h_1$  to  $h_4$ , as shown in the second picture above. Now, a priori  $v$  may play either  $v_1$  or  $v_2$  or  $v_3$ . So  $h_4$  is not a best response of  $h$  since for each row  $v_1$  to  $v_3$ ,  $h_4$  is smaller than some other column. So the nd profile "shrinks" to the third one above.

	<u><math>h_1</math></u>	<u><math>h_2</math></u>	<u><math>h_3</math></u>	$h_4$	$h_5$		<u><math>h_1</math></u>	<u><math>h_2</math></u>	$h_3$	$h_4$	$h_5$						
<u><math>v_1</math></u>	0,0	3,2	2,2	2,1	3,0	<u><math>v_1</math></u>	0,0	3,2	2,2	2,1	3,0	<u><math>v_1</math></u>	0,0	3,2	2,2	2,1	3,0
<u><math>v_2</math></u>	3,3	0,2	0,0	2,1	3,2	<u><math>v_2</math></u>	3,3	0,2	0,0	2,1	3,2	<u><math>v_2</math></u>	3,3	0,2	0,0	2,1	3,2
$v_3$	2,2	1,0	0,1	3,1	0,2	$v_3$	2,2	1,0	0,1	3,1	0,2	$v_3$	2,2	1,0	0,1	3,1	0,2
$v_4$	1,0	1,2	1,2	1,2	1,0	$v_4$	1,0	1,2	1,2	1,2	1,0	$v_4$	1,0	1,2	1,2	1,2	1,0
$v_5$	2,0	1,0	0,0	0,1	0,0	$v_5$	2,0	1,0	0,0	0,1	0,0	$v_5$	2,0	1,0	0,0	0,1	0,0

Similarly, for  $h_1$ ,  $h_2$  and  $h_3$ , row  $v_3$  is smaller than  $v_1$  or  $v_2$ , hence the left-hand nd profile above. Since  $h_3$  is smaller than  $h_2$  now, this yield the right-hand irreducible nd profile above, which is a nd Nash ( $BR^0$ -)equilibrium for the game.

**Recommendation:** Consider the following class of games  $G$ , where each \* can be either 1 or  $-1$ , and where the preferences are along the usual order  $-1 < 1$ .

	$h_1$	$h_2$
$v_1$	*, *	*, *
$v_2$	*, *	*, *

For each agent, the arithmetic mean of its payoff over its four payoffs for all games in  $G$  is 0, by a "symmetry" argument. However for each agent, the arithmetic mean of its payoff over its payoff that are inside the nd Nash equilibrium for all games in the class is  $3/8$ , by a few combinatorial arguments.

**Lemma 18** *For a game  $g$  in  $G$ , let  $eq(g)$  be the nd Nash  $BR^2$ -equilibrium built by the proof of lemma 15.*

$$\frac{1}{|G|} \times \sum_{g \in G} \frac{1}{|eq(g)|} \times \sum_{s \in eq(g)} P(s, v) = \frac{3}{8}$$

Probabilistic Nash equilibria cannot serve as recommendations to agents on how to play because even for strategic games over rational numbers there is no algorithm that finds such an equilibrium and that is robust to strategy renaming. Below, let  $v$  (resp.  $h$ ) model a real-life situation with the game to the left (resp. right). Each game has only two probabilistic Nash equilibria, namely  $((v_1 \mapsto \frac{1}{2}, v_2 \mapsto \frac{1}{2}), (h_1 \mapsto \frac{1}{2}, h_2 \mapsto \frac{1}{2}))$  and  $((v_3 \mapsto \frac{1}{2}, v_2 \mapsto \frac{1}{2}), (h_3 \mapsto \frac{1}{2}, h_2 \mapsto \frac{1}{2}))$ . However, there is no way for agents to collectively choose one of them in a non-cooperative setting.

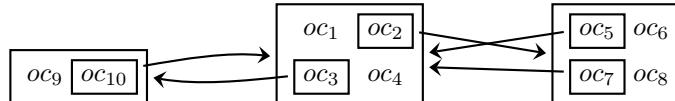
	$h_1$	$h_2$	$h_3$		$h_3$	$h_2$	$h_1$
$v_1$	3, 2	2, 3	0, 0		3, 2	2, 3	0, 0
$v_2$	2, 3	3, 2	2, 3		2, 3	3, 2	2, 3
$v_3$	0, 0	2, 3	3, 2		0, 0	2, 3	3, 2

This paper's equilibrium computation is robust. Since it is more efficient than random play, it can serve as a recommendation.

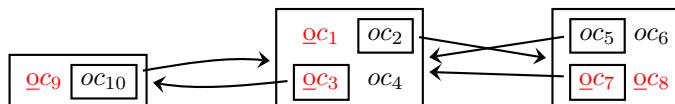
## 6 Multi Strategic Games

This section introduces multi strategic games and related notions informally. Such a game is a graph whose nodes are "abstract strategic games". For each node, each agent has to choose a *node strategy* once for all. These combine into a *node profile* that yields an abstract outcome and points to another node .

The 2-agent game below has three nodes (big rectangles). Consider a play starting at (or reaching) the central node. For instance, if  $v$  and  $h$  both choose their first strategy at this node, the outcome is  $oc_1$  and the play reaches the central node again; if  $h$  chooses his second strategy instead, the outcome is  $oc_2$  and the play reaches the right node, hence the small rectangle and the arrow.



A strategy for an agent is the combination of his node strategies, and a profile is the combination of agents' strategies. Nd node strategies allow an agent to choose several node strategies at a same node. A nd profile is depicted below. At the central node, underlining says  $h$  selects his first strategy and  $v$  selects both.

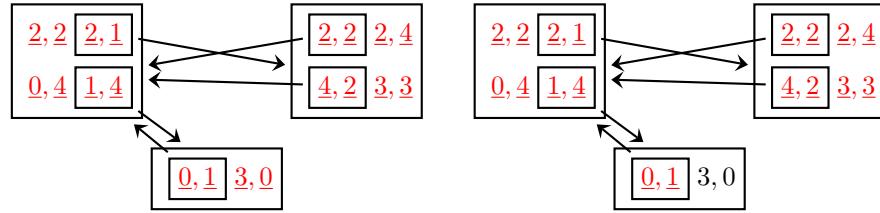


From each node, a nd profile induces a set of infinite sequences of outcomes. Above, the nd profiles induces the sequence  $oc_9^\omega$  at the left-hand node, and the sequences  $oc_1^k, oc_3, oc_9^\omega$ , for  $k$  in  $\mathbb{N}$ , at the central node.

As described in [5], embedding such games into nd games also provides a non-trivial notion of nd Nash equilibrium with guaranteed existence. The constructive

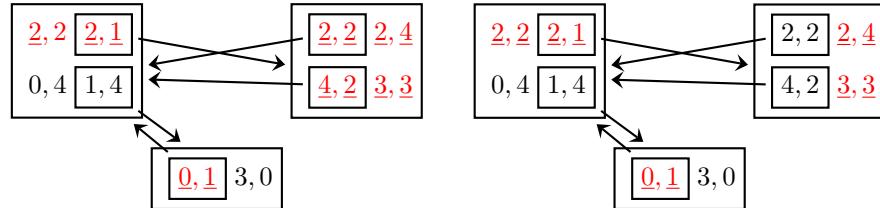
proof yields a search algorithm exemplified below. The game involves agents  $v$  and  $h$ . Agent  $v$  chooses the rows and gets the first figures of the payoff functions.

At first, all agents consider all of their options, as suggested by the underlining. At the bottom node, only agent  $h$  has a decision to make. If he chooses right, he gets  $0^\omega$  (and  $v$  gets  $3^\omega$ , but  $h$  does not take it into account). If  $h$  chooses left, he gets an infinite sequences of positive numbers whatever  $v$ 's strategy may be, which is better than 0 at any stage of the sequence. So  $h$  dismisses for good his right strategy at the bottom node, as depicted on the right-hand side below.

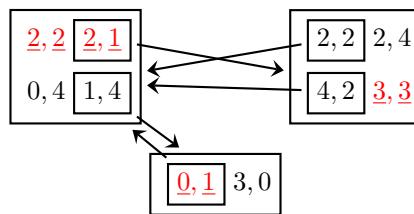


Now agent  $v$  considers the top-left node. If he chooses his bottom strategy, induced sequences use only 0 and 1. If he choose his top strategy, induced sequences use numbers that are greater than 1, which is better. So  $v$  dismisses for good his top strategy at this node, as depicted on the left-hand side below.

Then agent  $h$  considers the top-right node. If he chooses his left (resp. right) strategy, induced sequences use only 1 and 2 (resp. 3 and 4). So agent  $h$  dismisses his left strategy at the top-right node, as depicted on the right-hand side below.



Eventually, agent  $v$  dismisses his top strategy at the top-right node, which yields the nd Nash equilibrium below. So, for each agent, fore each node, the agent cannot get a better set of sequences by changing his strategy.



The universal existence of such equilibria actually follows an embedding of multi strategic games into nd games. However, other more subtle notions of equilibrium with guaranteed existence may be defined.

## 7 Conclusion

Using probability to weaken the notion of strategy and guarantee existence of weak Nash equilibria works for real-valued strategic games but not for the slightly more general abstract strategic games. Using discrete non determinism works in both settings and enables (at least) five different relevant notions of discrete non deterministic equilibrium. Some of these notions are actually related to the concept of dominated strategy. However, this concept had not led to discrete Nash equilibria in the past literature, perhaps because probabilities seemed more accurate than discrete non determinism or because no alternative to Nash's approach seemed to be required. Since elimination of dominated strategies gives information about the probabilistic Nash equilibria of a strategic game, studying discrete Nash equilibria may actually give information about probabilistic Nash equilibria. In addition, the discrete approach is better in at least four respects: First, it still works in an even more general setting, namely multi strategic games, which combine sequential and simultaneous decision-making. Second, the constructive proof of existence builds one specific equilibrium that can serve as a recommendation since it seems payoff-wise efficient. However, more work is needed about payoff-wise efficiency. Third, the proof/algorithm has a (low) polynomial time complexity for the five notions, which makes equilibria easy to find among exponentially many candidates. Fourth, some other notions of discrete equilibrium might yield NP-complete problems, which suggests that finding discrete non deterministic equilibria also lies on the boundary of P.

## References

1. B Douglas Bernheim. Rationalizable strategic behavior. *Econometrica*, 52:1007–1028, 1984.
2. D Gale. A theory of n-person games games with perfect information. *Proceedings of the National Academy of Sciences of the United States of America*, 39:496–501, 1953.
3. J Hillas and E Kohlberg. Foundations of strategic equilibrium. In R.J. Aumann and S. Hart, editors, *Handbook of Game Theory with Economic Applications*. 2002.
4. S Le Roux. Non-determinism and Nash equilibria for sequential game over partial order. In *Proceedings Computational Logic and Applications, CLA '05*. DMTCS Proceedings, 2006. <http://www.dmtcs.org/pdffiles/dmAF0106.pdf>.
5. S Le Roux. Discrete non determinism and nash equilibria for strategy-based games. Research report <http://hal.inria.fr/inria-00195397/fr>, INRIA, 2007.
6. S. Le Roux. *Generalisation and formalisation in game theory*. PhD thesis, Ecole Normale Supérieure de Lyon, 2008.
7. RD Luce and H Raiffa. *Games and Decisions*. John Wiley and Sons, New York, 1957.
8. J Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36:48–49, 1950.
9. MJ Osborne and A Rubinstein. *A Course in Game Theory*. The MIT Press, 1994.
10. DG Pearce. Rationalizable strategic behavior and the problem of perfection. *Econometrica*, 52:1029–1050, 1984.

# **Combining Concept Maps and Petri Nets to Generate Intelligent Tutoring Systems**

Maikel León, Isis Bonet, and Zenaida García

Department of Computer Science  
Central University of Las Villas  
Highway to Camajuaní km 5.5  
Santa Clara, Cuba  
Phone: (53) (42) 281515  
Fax : (53) (42) 281608  
[{mle, isisb, zgarcia}@uclv.edu.cu](mailto:{mle, isisb, zgarcia}@uclv.edu.cu)

**Abstract.** The use of pedagogical methods with the technologies of the information and communications produce a new quality that favors the task of generating, transmitting and sharing knowledge. Such is the case of the pedagogical effect that produces the use of the Concept Maps, which constitute a tool for the management of knowledge, an aid to personalize the learning process, to exchange knowledge, and to learn how to learn. In this paper the authors present a new approach to elaborate Intelligent Tutoring Systems, where the techniques of Concept Maps and Artificial Intelligence are combined, using Petri Nets as theoretical frame, for the student model. The pedagogical model that controls the interaction between the apprentice and the generated Intelligent Tutoring Systems is implemented by Petri Nets. The Petri Nets transitions are controlled by conditions that refer to the apprentice model. The firing of these transitions produces actions that update this apprentice model. These conditions are automatically included into the pedagogical model and the teacher has only to specify the contents of the domain model.

**Key words:** Intelligent Tutoring Systems, Petri Nets, Student Model, Artificial Intelligence, Concept Maps.

## **1 Introduction**

To construct and to share knowledge, to learn significantly, and to learn how to learn, are ideas on which researchers have been pondering for a long time as well as the use of tools that would allow taking these aspirations into practice. To achieve this, different techniques and strategies have been used. Concept Maps (CM) provide a schematic summary of what is learned and they order it in a hierachic range. The knowledge is organized and represented in all the levels of abstraction, the general knowledge goes to the upper part and the most specific one goes to the lower [1]. Over the last few years the CM have increasingly got a great popularity and its integration with the technologies of the information and the communications have become a very important element in the plans for the improvement of the teaching-learning process. The main application of the CM has had effect in the teaching-learning process, which is its basic intention, they are based on an instrument that combines the scientific rigidity with simplicity and flexibility, producing a general

approval in the audience of students and professionals; this represents an important nexus between pedagogy and technology in a huge field of applications. Also, it constitutes an important aid for those who generate, transmit, store, and divulge information and knowledge and it comprises an important tool to obtain a highest practical value in the systems of the teaching-learning process [2].

Thus, Petri Nets (PN) are a graphical and mathematical modeling tool. It consists of places, transitions, and arcs that connect them. Input arcs connect places with transitions, while output arcs start at a transition and end at a place. There are other types of arcs, e.g. inhibitor arcs. Places can contain tokens; the current state of the modeled system (the marking) is given by the number (and type if the tokens are distinguishable) of tokens in each place. Transitions are active components. They model activities which can occur (the transition fires), thus changing the state of the system (the marking of the PN). Transitions are only allowed to fire if they are enabled, which means that all the preconditions for the activity must be fulfilled (there are enough tokens available in the input places). When the transition fires, it removes tokens from its input places and adds some at all of its output places. The number of tokens removed/added depends on the cardinality of each arc. The interactive firing of transitions in subsequent markings is called token game [3]. PN are a promising tool for describing and studying systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic. As a graphical tool, PN can be used as a visual-communication aid similar to flow charts, block diagrams, and networks. In addition, tokens are used in these nets to simulate the dynamic and concurrent activities of systems. As a mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models governing the behavior of systems [4].

To study performance and dependability issues of systems it is necessary to include a timing concept into the model. There are several possibilities to do this for a PN; however, the most common way is to associate a firing delay with each transition. This delay specifies the time that the transition has to be enabled, before it can actually fire. If the delay is a random distribution function, the resulting net class is called stochastic PN. Different types of transitions can be distinguished depending on their associated delay, for instance immediate transitions (no delay), exponential transitions (delay is an exponential distribution), and deterministic transitions (delay is fixed). The concept of PN has its origin in Carl Adam Petri's dissertation Kommunikation mit Automaten, submitted in 1962 to the faculty of Mathematics and Physics at the Technische Universität Darmstadt, Germany.

Till now the Intelligent Tutoring Systems (ITS) have demonstrated its effectiveness in diverse domains. However, its construction implies a complex and intense work of engineering of knowledge, which prevents an effective use of it. In order to eliminate these deficiencies there appears the target to construct the author's hardware which facilitates the construction of these systems to all users (not necessarily expert) in the computer field, in particular to those instructors who master a certain matter of education [5]. The field of the ITS is characterized by the application of Artificial Intelligence techniques, to the development of the teaching-learning process assisted by computers. If the key feature of an ITS is the aptitude to adapt itself to the student, the key component of the system is the Student Model, where the information associated to the student is stored. This information must be inferred by the system depend-

ing on the information available: previous data, response to questions, etc. This process of inference is identified as diagnosis, and is undoubtedly the most complicated process inside an ITS, who uses the Artificial Intelligence techniques to represent knowledge, to shape the human reasoning, to emphasize the learning by means of the action, to combine experiences of resolution and discovery, to be able to solve problems by their own, to formulate diagnoses and to provide explanations. So, they count on a bank of instruction strategies which helps to decide what and how to inform to the student to get an effective direction.

The aim of this paper is to present a new approach to elaborate ITS using a CM, questionnaires able to catch the cognitive and affective state of the student (Student Model) and by using PN to allow the student to sail in an oriented way according to their knowledge and not in a free way as the CM are conceived. And it is the capacity to adapt dynamically to the development of the student's learning what makes this system intelligent. The pedagogical model is controlled by PN that access and updates the apprentice model, while presenting the contents of the domain model. The PN are automatically generated from specifications defined by the teacher using a graphical interface and compiled into a neural net that actually implement the ITS. An important point in the proposed approach is that the teacher does not need to specify the interaction with the apprentice model; this interaction is automatically included in the PN using the prerequisite and difficulty orders previously defined by the teacher.

## 2 Motivations for Concept Maps

Concept Maps provide a framework for capturing experts' internal knowledge and making it explicit in a visual, graphical form that can be easily examined and shared. CM constitute a tool of great profit for teachers, investigators of educational topics, psychologists, sociologists and students in general, as well as for other areas especially when it is necessary to manage with large volumes of information. They have become a very important element in the plans for the improvement of the ITS and they have also extended its use to other areas of the human activity in which both management and the use of knowledge take up a preponderant place. If to define a CM is relatively simple, it is simpler to understand the meaning of the definition. The Concept Maps were defined by Novak, his creator, as a skill that represents a strategy of learning, a method to get the gist of a topic and a schematic resource to represent a set of conceptual meanings included in a set of propositions [2].

It is necessary to point out that there is not only one model of CM, several may exist. The important point is the relations that are established between the concepts through the linking words to form propositions that configure a real value on the studied object. For such a reason, in a concept there may appear diversity of real values. In fact, it turns very difficult to find two exactly equal CM, due to the individual character of knowledge. The CM can be described under diverse perspectives: abstract, visualization, and conversation. Since significant learning is reached more easily when the new concepts or conceptual meanings are included under wider concepts, the most used CM are the hierachic ones, the most general and inclusive con-

cepts are placed in the upper part of the map, and the progressively more specific and less inclusive concepts, in the lower part. The subordinated relations among concepts may change in different fragments of learning, so in a CM, any concept may rise up to the top position, and keep a significant propositional relation with other concepts of the map. The use of CM designed by the professor increases both learning and retention of scientific information. The students produce maps as learning tools. Considering that the CM constitute an explicit and clear representation of the concepts and propositions, they allow both teachers and students to exchange points of view on the validity of a specific propositional link and to recognize the missing connections in the concepts that suggest the need of a new learning. For this reason, this skill has complemented so favorably with the practice of distance learning which presupposes that students and teachers are not physically in the same place at the same time.

Concept Maps have particular characteristics that make them amenable to smart tools [2]. These include:

1. Concept Maps have structure: By definition, more general concepts are presented at the top with more specific concepts at the bottom. Other structural information, e.g. the number of ingoing and outgoing links of a concept, may provide additional information regarding a concept's role in the map.
2. Concept Maps are based on propositions: every two concepts with their linking phrase forms a "unit of meaning". This propositional structure distinguishes Concept Maps from other tools such as Mind Mapping and The Brain, and provides semantics to the relationships between concepts.
3. Concept Maps have a context: A Concept Map is a representation of a person's understanding of a particular domain of knowledge. As such, all concepts and linking phrases are to be interpreted within that context.
4. Concepts and linking phrases are as short as possible, possibly single words.
5. Every two concepts joined by a linking phrase form a standalone proposition. That is, the proposition can be read independently of the map and still "make sense".
6. The structure is hierarchical and the root node of the map is a good representative of the topic of the map.

The students who analyze CM will have a wider basic knowledge; therefore, they will be more prepared to solve problems in comparison to those students who learn by memorizing. The CM have turned into a useful instrument for teacher training and for the student's understanding of diverse subjects. CM constitute an instrument that merges the scientific rigidity with the simplicity and flexibility. It represents an aid for those who generate, transmit, store and spread information and knowledge. They also constitute an important tool to achieve a practical value, especially in the Artificial Intelligence systems.

### 3 Proposed Model

The ITS that are created correspond with a CM, with the particular feature that in some of its nodes there appears a questionnaire, capable to get the cognitive and affective state of the student and able to guide his navigation, creating this way an “Intelligent” CM. At the first level of the authoring interface, the teacher must specify the set of pedagogical units that define each curriculum and, at the second level, the set of problems that define each pedagogical unit. The elements of these sets present a partial order and associated prerequisites [6]. To specify these partial orders and prerequisites the teacher disposes of a graphical interface that allows the construction of graphs. In these graphs, an edge from node  $n_1$  to node  $n_2$  means that  $n_2$  has  $n_1$  as a prerequisite. Each node  $n$  may have the following input edges:

- None: node  $n$  has no prerequisite and can be executed anytime. This is the case only for initial nodes.
- One: node  $n$  has only one node as prerequisite and this one must be executed before node  $n$  is available for execution.
- Two or more necessary edges: node  $n$  has several prerequisite nodes and all of them must be executed, before node  $n$  is available for execution.
- Two or more alternative edges: node  $n$  has several prerequisite nodes but only one of them must be executed before node  $n$  is available for execution.

Necessary and alternative edges may occur simultaneously in the same node. Nodes and the different types of edges may be combined in a complex graph, according to the intended course sequences, but they must satisfy one restriction: each graph must have only one initial node and only one final node. This restriction seems reasonable, because pedagogical units and problems should have one begin and one end point and it also assures that the graphs of the two levels can be combined into a one level PN. Figure 1 shows a prerequisite graph. Note the AND constraint (with nodes  $n_5$  and  $n_6$  before and node  $n_6$  after) associated with necessary edges and the OR constraint (with nodes  $n_6$  and  $n_7$  before and node  $n_8$  after) associated with alternative edges.

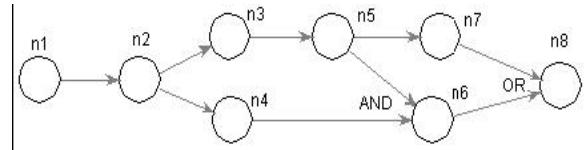


Fig. 1. Prerequisites Graph

The first step in the generation of the final code of an ITS using the proposed authoring tool is to compile each prerequisite graph into a PN that represents the pedagogical model. The pedagogical model defines the control strategy of the interaction between the apprentice and the TA associated with each sub-domain and is specified by the teacher in the form of curriculum and pedagogical unit graphs. To represent the pedagogical model, we use an Object Petri Net (OPN) [4]. An OPN is defined by a control structure (places, transitions and arcs connecting places to/from transitions), by the data structure of its tokens and by its semantics, that defines how the token player should execute the OPN, given an initial marking. The OPN places can represent either pedagogical unit, if the graph corresponds to the first level of the interface, or problems, if it corresponds to the second level.

The net structure is directly obtained from the graphs specified by the teacher and the token data structure is defined by the adopted apprentice model. As the tokens have the class Apprentice the marking of the OPN is associated with the current activity of the apprentice and also with the possible future activities. A transition fires when the apprentice has finished the interactions associated with its input nodes. The token is then moved to the output places of the transition, meaning that the apprentice is ready to begin the activities associated with them. A prerequisite graph (curriculum or pedagogical unit) is automatically translated into an OPN structure. Each configuration of nodes and edges in a prerequisite graph is associated with an OPN fragment that implements it and these fragments are combined into a single Petri net that implements the graph.

For instance, a simple sequence of nodes, see figure 2(a), is implemented by one transition ( $t_{1,2}$ ), one input place ( $n_1$ ) and one output place ( $n_2$ ). One node with two output edges, see figure 2(b), is implemented by a fork transition ( $tf_{1,2}$ ) with one input place ( $n_1$ ) and two output places ( $n_2$  and  $n_3$ ). After the activity associated with the input place is finished, the apprentice can do either the activity associated with one output place or the activity associated with the other. A node with two input necessary edges, see figure 2(c), is implemented by a join transition ( $tj_{1,2}$ ) with two input places ( $b_{1,3}$  and  $b_{2,3}$ ) and one output place ( $n_3$ ). The places  $b_{1,3}$  and  $b_{2,3}$  are buffer places and do not correspond to pedagogical units or problems. They are necessary because after the activity associated with either one of the places  $n_1$  or  $n_2$  is finished, that information must be stored in the corresponding buffer and only when the activity associated with the other place is also finished (and also stored in the corresponding buffer) the activity associated with the output place ( $n_3$ ) can be performed. For this, we need two extra transitions ( $tb_{1,3}$  and  $tb_{2,3}$ ) connecting, respectively, the input places  $n_1$  and  $n_2$  to the buffers  $b_{1,3}$  and  $b_{2,3}$ . Finally, see figure 2(d), a node with two alternative input edges is represented by two transitions ( $t_{1,3}$  and  $t_{2,3}$ ), with input places  $n_1$  and  $n_2$ , respectively, and both with output place  $n_3$ . Transitions  $t_1$  and  $t_2$  are used to remove the apprentice token from places  $n_1$  and  $n_2$ , respectively; in the case the activities associated with  $n_3$  have already been performed, avoiding in this way the repetition of  $n_3$ . Figure 3 shows the OPN obtained from the prerequisite graph of figure 1.

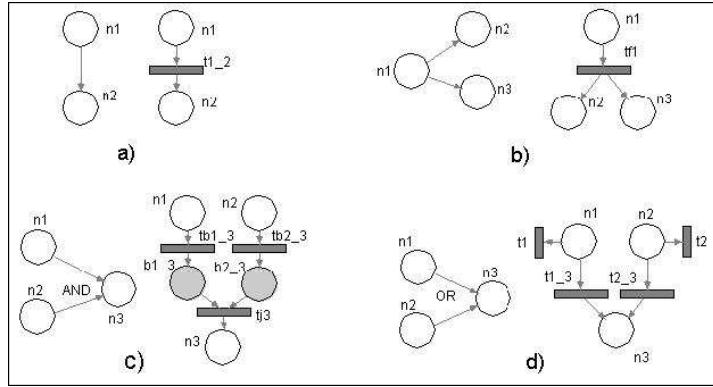


Fig. 2. Graph Topologies and Petri Nets

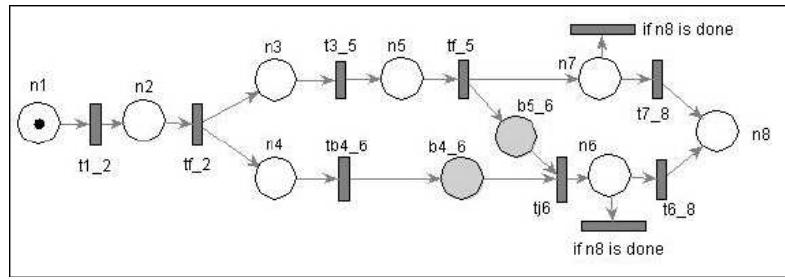


Fig. 3. Structure of an OPN

The semantic of an OPN is the particular way the token player executes this net given an initial marking. An ordinary token player (used for PN with binary tokens) verifies, from an initial marking, whether there are enabled transitions. If there is just one, the token player fires it. If there is more than one, it chooses one of them according to a priority order, or arbitrarily if no priority order is specified. In an OPN token player, because the token is associated with a data structure, the transitions can have pre-conditions that refer to the token attributes and/or to external variables [4]. The apprentice token attributes that are relevant to the definition of algorithm are name, doing and tobedone and the relevant external variables are next, that contains the next activity to be performed, and answer, that receives the interaction support unit returned values. We say that the apprentice, identified by name, is doing a problem if the apprentice is interacting with one of its associated interaction support units. When the apprentice finishes the interaction with one support unit, the unit returns one of the following navigation commands: halt, continue, repeat, that is stored in the external variable answer.

The command halt means that the apprentice wants to interrupt the present session with the ITS, the command repeat means that the next activity should be another interaction support unit of the same problem and the command continue means that the current problem is finished and a next one must be chosen, either by the coordinator or by the apprentice. The apprentice is asked to choose the next problem, when there is more than one problem in tobedone. Because the apprentice must choose only

one problem to do next, only one transition can fire at a given moment [7]. The coordinator initializes the associated PN with a marking where the apprentice token is stored in the place associated with the first problem of the first pedagogical unit and calls the OPN token player.

#### 4 Petri Nets and Neural Networks. A possible solution

PN are powerful and versatile tools for modeling, simulating, analyzing and designing of complex workflow systems. This topic mainly discusses a hybrid approach using neural network and PN in the formal model of an ITS. Neural networks are used in different fields; classification is one of problems where they are commonly used [8]. PN are alternative tools for the study of non-deterministic, concurrent, parallel, asynchronous, distributed or stochastic systems [9]. They can model systems in an easy and natural way. Furthermore, the PN approach can be easily combined with other techniques and theories such as object-oriented programming, fuzzy theory, neural networks, etc [10]. These modified PN are widely used in computing, manufacturing, robotic, knowledge based systems, process control, as well as in other kinds of engineering applications. Since PN offer advantages to model systems and can interact with other techniques easily, it would be advantageous to model neural networks starting from PN models, which allow not only the design adjustment but also the initialization of the neural network weights [11]. Following the algorithm proposed by Xiaou Li and Wen Yu in [4], we can model a neural network starting from a PN with the application of weighty production rules in the algorithm. See the following example (figure 3).

The following weighting production rules are obtained starting from the example of Figure 3 and following the algorithm described in [4]:

1. If P1 and P4 then P5 (w1, w4)
2. If P1 and P4 then P6 (w1, w4)
3. If P1 and P4 then P12 (w1, w4)
4. If P5 or P6 then P9 ( $\mu_4$ ,  $\mu_{5-9}$ )
5. If P6 and P1 then P15 (w6, w1)
6. If P9 then P10 ( $\mu_6$ )

The learning algorithm of the neural networks obtained is the same as the back-propagation of multilayer neural networks. The main idea is that all layer weights can be updated through the backpropagation algorithm if certainty factors of all sink places are given. A complex neural network can be divided into several sub-networks starting from the modular design of an original PN. With this system a teacher can edit an ITS starting from a CM that will compiled into a PN, the neural network that control the system interaction is constructed starting from the PN. The initial modelling PN will allow us to design a feedforward neural network with the backpropagation learning algorithm. It is even possible to create several neural networks starting

from only one PN, taking into account its possible modules [12]. To split neural networks in modules makes their training and performance easier.

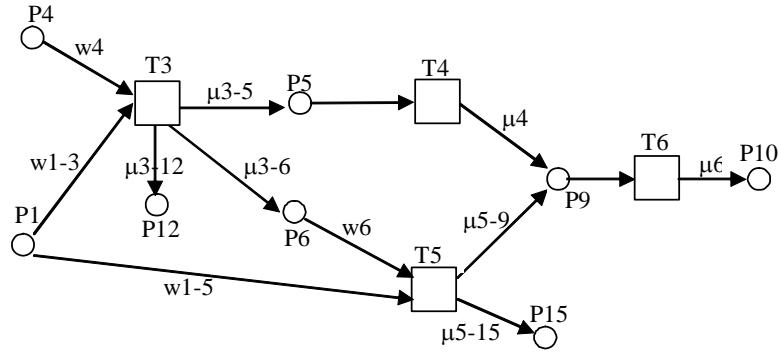


Fig. 3. Example of Petri Net

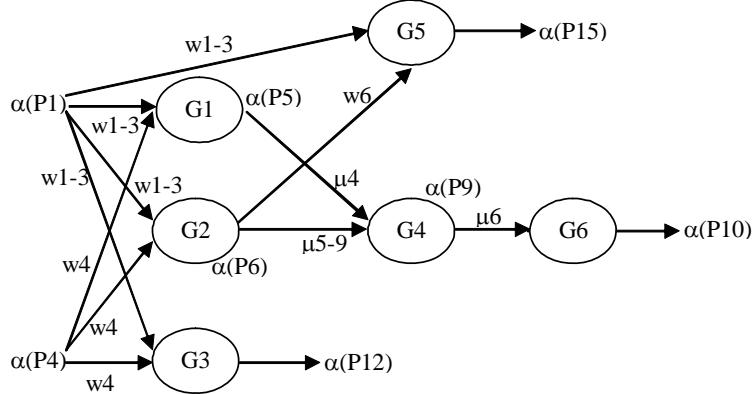


Fig. 4. A neural network model obtained from the Petri Net of Figure 3

## 5 Conclusions

With this work we propose a new Student Model that could be considered in the construction of ITS, where are combined the facilities of the Concept Maps for the organization of the knowledge and the potentiality of the Petri Nets that allows to the student to sail in an oriented way according to their knowledge and not in a free way as the CM are conceived. The proposed idea has been applied successful in the elaboration of ITS by users (not necessarily expert in the Computer Science field).

The suggested hybrid system combines PN and neural networks using the advantages of PN in order to overcome the neural network deficiencies concerning their original design and definition of their initial weights.

## References

1. Cañas Alberto J., Marco Carvalho. (2004). Concept Maps and AI: an Unlikely Marriage? SBIE 2004: Simpósio Brasileiro de Informática na Educação Manaus, Brasil.
2. Martínez, N., León M. (2006). Mapas Conceptuales y Redes Bayesianas en los Sistemas de Enseñanza/Aprendizaje Inteligentes. Congreso Internacional de Informática Educativa. Costa Rica.
3. Xiao-Qiang Liu, Min Wu, Jia-Xun Chen. (2002). Knowledge aggregation and navigation highlevel petri nets-based in e-learning. In Proceedings International Conferenceon Machine Learning and Cybernetics, volume 1, pages 420– 425.
4. Li X., Yu W. (2000). Dynamic Knowledge Inference and Learning under Adaptive Fuzzy Petri Net Framework, IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and reviews, Vol. 30, No. 4.
5. Esposito F., Ferilli S., Basile A., Di Mauro N. (2006). Inference of abduction theories for handling incompleteness in first-order learning. Knowledge and Information Systems (KAIS) Journal, 1-27, ISSN: 0219-1377.
6. Brusilovsky P. (2000). Adaptative hypermedia: From intelligent tutoring systems to web-based education. Lecture Notes in Computer Science. 5th International Conference on ITS. Montreal, Canada.
7. Murray T. (1999). Authoring intelligent tutoring systems: An analysis of the state of the art. In Journal of Artificial Intelligence and Education, volume 10.
8. Hilera, J. (1995). Redes neuronales artificiales: Fundamentos, modelos y aplicaciones. Ed. Addison-Wesley Iberoamericana, 1995.
9. Yu X., Chen G., Cheng S. (1995). Dynamic Learning Rate Optimization of the Backpropagation Algorithm, IEEE Transaction on Neural Networks, Vol. 6, No. 3, pp. 669-677.
10. Bello, R. (2002). Aplicaciones de la Inteligencia Artificial. Ediciones de la Noche, Guadalajara, Jalisco, México. ISBN: 970-27-0177-5.
11. Liu W., Hong J. (2000). Re-investigating Dempster's Idea on Evidence Combination. Knowledge and Information Systems: An International Journal, Springer-Verlag, Vol.2, No.2, pp.223-241, ISSN 0219-1377.
12. Cooley J. (1999). Data Preparation for mining World Wide Web browsing patterns. Journal of Knowledge and Information Systems.

# Query-Optimal Oracle Turing Machines for Type-2 Computations

Chung-Chih Li

School of Information Technology  
Illinois State University  
Normal, IL 61790, USA

**Abstract.** With the notion of optimal programs defined in terms of Blum's complexity measure, the well known speed-up theorem states that there exist computable functions that do not have optimal programs. For type-2 computation, we argue that the same speed-up theorem can be obtained as long as the complexity measure satisfies Blum's complexity measure under the Oracle Turing Machine model. In the present paper, we consider the oracle query as a dynamic complexity measure. Since the query complexity is not a Blum's complexity measure, we have to characterize the notions of optimal programs in a very different way. We give three notions of query-optimal programs in different strength and argue that the stronger the notion of query-optimal programs we define, the more difficult to have query-optimal programs. In other words, with a stronger definition of query-optimal programs, one can prove an easier version of the speed-up theorem.

**Key words:** Type-2 Complexity Theory, Type-2 Computation, Query Optimal Programs, Speed-up Theorems

## 1 Introduction

Searching for the best algorithm to solve a concerned computable problem is a primary task for computer scientists. However, the speed-up theorem tells us that, theoretically, we cannot succeed on all problems. In fact, the speed-up phenomena have been extensively studied by mathematicians for more than a half century, which in logical form was first remarked by Gödel in terms of the length of theorem proving [5, 6, 13]. Blum [1, 2] re-discovered the speed-up phenomenon in the context of computational complexity.

Blum's speed-up theorem is a result of his axiomatization of complexity measures. Fix a programming system for Turing machines and let  $\varphi_i$  denote the function computed by the  $i^{th}$  Turing machine and  $\Phi_i$  denote the cost function associated to it. Blum considers that dynamic complexity measures should meet the following two requirements: for any  $i, x, m \in \mathbf{N}$ , (i)  $\varphi_i$  is convergent on  $x$  (denoted by  $\varphi_i(x) \downarrow$ ) if and only if  $\Phi_i$  is convergent on  $x$  (denoted by  $\Phi_i(x) \downarrow$ ) and (ii)  $\Phi_i(x) = m$  is effectively decidable. The two axioms have successfully lifted the study of complexity theory to an abstract level with rich results that

are independent from any specific machine models. At type-1, the meaning of “more efficient programs” is intuitive: For a computable function  $f$ , we say that program  $j$  is more efficient than program  $i$  if  $\varphi_i = \varphi_j = f$  and for all but finitely many  $x$  we have  $\Phi_j(x) < \Phi_i(x)$ . Precisely, Blum [1, 2] formulates the speed-up theorem as follows: For any recursive function  $r$ , there exists a recursive function  $f$  such that

$$(\forall i : \varphi_i = f) (\exists j : \varphi_j = f) (\overset{\infty}{\forall} x) [r(\Phi_j(x)) \leq \Phi_i(x)]. \quad (1)$$

There are many sources for detailed proofs and discussions regarding this curious speed-up phenomenon, e.g., [1–4, 6, 11–14]. We shall skip all irrelevant details due to the space constraints.

When we try to lift the complexity theory to type-2 computation, we want to know whether or not the same speed-up phenomenon can be described in terms of some complexity measures that are sensible in the context of type-2 computation. Clearly, time and space remain meaningful in type-2 computation. Since time and space are Blum’s complexity measures, we have not faced too many difficulties in translating the original proofs of the speed-up theorem and its variants into type-2 (see [9]). For query complexity, on the other hand, the two Blum’s axioms fail. As a result, we can’t reuse the same concepts and techniques but develop our new ones in order to investigate this specular speed-up phenomenon.

## 2 Type-2 Computation Using Oracle Turing Machines

In this section we will provide necessary notations for type-2 computation; detailed discussion about the type-2 complexity theory can be found in [7–10]. We consider natural numbers as type-0 objects and functions over natural numbers as type-1 objects. For type-2 objects, they are *functionals* that take as inputs and produce as outputs type-0 or type-1 objects. We consider objects of lower type as special cases of higher type, i.e., type-0  $\subset$  type-1  $\subset$  type-2. Without loss of generality we restrict our type-2 functionals to the standard type  $\mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$ , where  $\mathcal{T}$  is the set of total functions and  $\rightarrow$  means possibly partial.

We use the Oracle Turing Machine (OTM) as our standard computing formalism for type-2 computation. An OTM is a Turing machine equipped with a function oracle. We say that a type-2 functional  $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$  is computable if there is an OTM that computes  $F$  as follows. Before the OTM begins to run, the type-1 argument should be presented to the OTM as an oracle. Every OTM has two extra tapes called query-tape and answer-tape for oracle queries and oracle answers, respectively. During the course of the computation, the OTM may enter a special state called query-state where the oracle will read the query left on the query-tape and return its answer to the answer-tape. We can fix a programming system  $\langle \widehat{\varphi}_i \rangle_{i \in \mathbf{N}}$  for OTM’s and let  $\widehat{M}_e$  denote the OTM with index  $e$  that computes  $\widehat{\varphi}_e$ .

Let  $\mathcal{F}$  denote the set of finite functions over natural numbers, i.e.,  $\sigma \in \mathcal{F}$  iff the domain of  $\sigma$  is a subset of  $\mathbf{N}$  and its cardinality is in  $\mathbf{N}$ . Given  $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$ ,

$F(f, x) \downarrow = y$  denotes that  $F$  is defined at  $(f, x)$  and its value is  $y$ . Since if  $F$  is computable,  $F$  must be *continuous* (i.e., *compact* and *monotone*), it follows that if  $F(f, x) \downarrow = y$ , there must be  $\sigma \in \mathcal{F}$  with  $\sigma \subset f$  such that for every extension  $\eta$  of  $\sigma$  (i.e.,  $\sigma \subseteq \eta$ ), we have  $F(f, x) \downarrow = F(\sigma, x) \downarrow = F(\eta, x) \downarrow = y$ . We say that  $(\sigma, x)$  and  $(\eta, x)$  are locking fragments of  $F$  on  $(f, x)$ . For any  $\sigma \in \mathcal{F}$ , let  $((\sigma))$  denote the set of total extensions of  $\sigma$ , i.e.,  $((\sigma)) = \{f \in \mathcal{T} \mid \sigma \subset f\}$ . Also, if  $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$ , let  $((\sigma, x)) = \{(f, x) \mid f \in ((\sigma))\}$ . If  $\sigma_1$  and  $\sigma_2$  are consistent, we have  $((\sigma_1)) \cap ((\sigma_2)) = ((\sigma_1 \cup \sigma_2))$ ; otherwise,  $((\sigma_1)) \cap ((\sigma_2)) = \emptyset$ . The union operation  $((\sigma_1)) \cup ((\sigma_2))$  is conventional.

### 3 Query-optimal Programs

Only in type-2 computation can we use the number of queries made throughout the course of the computation as a dynamic complexity measure. We are interested in whether or not there is a *query-optimal program* for any given type-2 computable functional; if there is not, can we have a speed-up theorem in terms of query-complexity. It turns out that the query-complexity fails to satisfy Blum's two axioms. Consequently, the arguments in the original proof of the speed-up theorem are no longer valid. We thus need a new approach to understand this curious phenomenon. In this section we define a few alternative notions of query-optimal programs. These notions must be sensible and intuitive.

#### 3.1 Unnecessary Queries

We shall argue that some intuitive notions of query-optimal programs are not flexible enough to derive interesting results. We say that an oracle query is necessary if its answer returned from the oracle will affect the result of the computation. Intuitively, a query-optimal program should be a program that does not make any unnecessary queries. Unfortunately, this notion of query-optimal programs is too strong. It is easy to argue that there are functionals that always make unnecessary oracle queries. Consider the following functional  $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$  defined by,

$$F(f, x) = \begin{cases} f(0) + 1 & \text{if } \varphi_x(x) \downarrow \text{ in } f(0) \text{ steps;} \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

Clearly,  $F$  is computable and total. Fix any  $a$  such that  $\varphi_a$  is diverged on  $a$  (denoted by  $\varphi_a(a) \uparrow$ ). Then, on input  $(f, a)$ , the value of  $f(0)$  only affects the speed of computing  $F(f, a)$ . Thus,  $F(f, a) = 0$  for any  $f \in \mathcal{T}$ , and hence  $(\emptyset, a)$  is the minimal locking fragment of  $F$  on  $(f, a)$ . That means any queries made during the computation of  $F$  on  $(f, a)$  are unnecessary. By contradiction, if there were an OTM that would not make any unnecessary queries for  $F$ , one could modify such OTM to solve the halting problem, which is impossible. Therefore, the type-2 functional defined in (2) can't be computed by an OTM without making unnecessary queries. From this example, it is clear that to require query-optimal

programs to be ones that do not make unnecessary queries is too restricted. We will loose the requirements by allowing a small amount of unnecessary queries. By “a small amount”, we means “a compact set” in some topology determined by the concerned computation. We formalize our idea in the next subsection.

### 3.2 Unnecessary Queries in Compact Sets

We use  $Q(s, e, f, x)$  to denote the collection of queries made during the run time of OTM  $\widehat{M}_e$  on  $(f, x)$  up to  $s$  steps. Formally,

**Definition 1.** Let  $e$  be a  $\widehat{\varphi}$ -program. Define  $Q : (\mathbf{N} \times \mathbf{N} \times \mathcal{T} \times \mathbf{N}) \rightarrow \mathcal{F}$  by:  $Q(s, e, f, x) = \tau$ , where  $\tau \in \mathcal{F}$  with  $\tau \subset f$  and  $\text{dom}(\tau)$  is the collection of queries made during the course of the computation of OTM  $\widehat{M}_e$  on  $(f, x)$  in  $s$  steps. ■

Clearly,  $Q(s, e, f, x)$  is monotone in  $s$ , i.e.,  $s \in \mathbf{N}$ ,  $Q(s, e, f, x) \subseteq Q(s+1, e, f, x)$ .

**Definition 2.** Let  $e, x \in \mathbf{N}$  and  $f \in \mathcal{T}$ . We say that  $Q(\cdot, e, f, x)$  is defined in the limit if and only if there exist  $s \in \mathbf{N}$  and  $\tau \in \mathcal{F}$  such that, for every  $s' \geq s$ ,

$$Q(s, e, f, x) = Q(s', e, f, x) = \tau.$$

In the case above, we write  $\lim_{s \rightarrow \infty} Q(s, e, f, x) \downarrow = \tau$ . ■

We wish to treat the set of queries an OTM made as a dynamic complexity measure, but it fails to satisfy Blum’s two axioms for complexity measure. That is,  $\lim_{s \rightarrow \infty} Q(s, e, f, x) \downarrow$  does not mean that  $\widehat{\varphi}_e(f, x) \downarrow$ , and hence Blum’s first axiom fails. Moreover, for some  $\sigma \in \mathcal{F}$  with  $\sigma \subset f$ ,  $\lim_{s \rightarrow \infty} Q(s, e, f, x) = \sigma$  is not necessarily decidable, and hence Blum’s second axiom fails too.

Let  $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$  be computable. Since we mean to characterize some  $\widehat{\varphi}$ -programs  $e_1, e_2, \dots, e_n$  that compute the same functional  $F$  in terms of queries, the topology  $\mathbb{T}(\dots)$  defined in [7–10] cannot serve as the relative topology in which we need a workable notion of compactness. This is because  $\widehat{\varphi}_{e_1} = \widehat{\varphi}_{e_2} = \dots = \widehat{\varphi}_{e_n} = F$  and thus  $\mathbb{T}(\widehat{\varphi}_{e_1}, \widehat{\varphi}_{e_2}, \dots, \widehat{\varphi}_{e_n}) = \mathbb{T}(F)$ , and hence the complexity of unnecessary queries cannot be described by the compact sets in  $\mathbb{T}(F)$ . The product topology,  $\mathbb{T} \times \mathbf{N}$  (Baire and the Discrete topologies), is fine enough to describe the needed compact sets but it suffers the same problem as discussed in [7–10]. We thus introduce another family of topologies as follows.

**Definition 3.** Given  $e_1, e_2, \dots, e_n \in \mathbf{N}$ , let  $B(e_1, e_2, \dots, e_n) \subseteq \mathcal{T} \times \mathbf{N}$  be defined as follows:  $(\sigma, a) \in B(e_1, e_2, \dots, e_n)$  if and only if there exists some  $f \in \mathcal{T}$  such that, for each  $i \in \{1, \dots, n\}$ , we have

$$\lim_{s \rightarrow \infty} Q(s, e_i, f, a) \downarrow = \sigma_i,$$

and  $\sigma = \bigcap_{i \in \{1, \dots, n\}} \sigma_i$ . Let  $\mathbb{Q}(e_1, \dots, e_n)$  be the topology on  $\mathcal{T} \times \mathbf{N}$  defined by the basic open sets in  $\mathcal{B}$ , where

$$\mathcal{B} = \{((\sigma, x)) \mid (\sigma, x) \in B\}.$$

■

Since a  $\widehat{\varphi}$ -program  $e$  may make unnecessary queries outside the domains of its minimal locking fragments, it follows that the topology  $\mathbb{Q}(e)$  may be finer than the topology  $\mathbb{T}(\widehat{\varphi}_e)$  which is defined by the minimal locking fragments of  $\widehat{\varphi}_e$ . That is,  $\mathbb{T}(\widehat{\varphi}_e) \subseteq \mathbb{Q}(e)$ . In other words, if  $e$  is a  $\widehat{\varphi}$ -program for  $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$ , then  $\mathbb{T}(F) \subseteq \mathbb{Q}(e)$ . On the other hand, if  $\mathbb{Q}(e) \subset \mathbb{T}(F)$ , then we can conclude that  $e$  cannot be a  $\widehat{\varphi}$ -program for  $F$ . In general, we have

$$\mathbb{T}(\widehat{\varphi}_i, \widehat{\varphi}_j) \subseteq \mathbb{Q}(i, j).$$

For convenience, we define the following notations. Let  $\text{card}(S)$  be the cardinality of set  $S$ . Suppose  $i$  and  $j$  are two  $\widehat{\varphi}$ -programs that compute the same functional, i.e.,  $\widehat{\varphi}_i = \widehat{\varphi}_j$ . Define

$$\begin{aligned} Q_{[i < j]} &= \left\{ (f, x) \mid \text{card}(\lim_{s \rightarrow \infty} Q(s, i, f, x)) < \text{card}(\lim_{s \rightarrow \infty} Q(s, j, f, x)) \right\}, \\ Q_{[i=j]} &= \left\{ (f, x) \mid \text{card}(\lim_{s \rightarrow \infty} Q(s, i, f, x)) = \text{card}(\lim_{s \rightarrow \infty} Q(s, j, f, x)) \right\}, \\ Q_{[i > j]} &= \left\{ (f, x) \mid \text{card}(\lim_{s \rightarrow \infty} Q(s, i, f, x)) > \text{card}(\lim_{s \rightarrow \infty} Q(s, j, f, x)) \right\}. \end{aligned}$$

The following definition makes use of the compactness in the topology we just mentioned to define the meaning of “better”  $\widehat{\varphi}$ -programs in terms of query-complexity.

**Definition 4 (Query Speed-up).** Let  $\widehat{\varphi}_i = \widehat{\varphi}_j$ .

1. We say that  $j$  is a **weakly** query sped-up version of  $i$  if and only if

$$Q_{[i < j]} \text{ is compact and } Q_{[i > j]} \text{ is noncompact in } \mathbb{Q}(i, j).$$

If  $j$  is a **weakly** query sped-up version of  $i$ , we write  $\widehat{\varphi}_i \prec_Q^* \widehat{\varphi}_j$ .

2. We say that  $j$  is a **strongly** query sped-up version of  $i$  if and only if

$$Q_{[i < j]} \cup Q_{[i=j]} \text{ is compact in } \mathbb{Q}(i, j).$$

If  $j$  is a **strongly** query sped-up version of  $i$ , we write  $\widehat{\varphi}_i \ll_Q^* \widehat{\varphi}_j$ . ■

Suppose that we are to write a new  $\widehat{\varphi}$ -program  $j$  for  $\widehat{\varphi}_i$ . Intuitively, if the new  $\widehat{\varphi}$ -program  $j$  can improve the query-complexity on a “large” (noncompact) set of inputs and leaves a “small” (compact) set of inputs on which  $j$  queries more than  $i$  does, we say that  $j$  is a *weakly* query sped-up version of  $i$ . Note that, we do not care how many inputs on which the query-complexity remains unchanged. Since  $\mathbb{T}(\widehat{\varphi}_j) \subseteq \mathbb{Q}(i, j)$ , it follows that, if  $Q_{[i < j]}$  is compact in  $\mathbb{Q}(i, j)$ , so is it in  $\mathbb{T}(\widehat{\varphi}_j)$ , and hence we can patch  $\widehat{\varphi}$ -program  $j$  on  $Q_{[i < j]}$  so that the patched  $\widehat{\varphi}$ -program will never query more than  $i$  does. Whereas, if  $\widehat{\varphi}$ -program  $j$  is a *strongly* query sped-up version of  $i$ , then the set on which  $j$  does not improve its query-complexity must be “small” (compact). Note that, if  $Q_{[i < j]} \cup Q_{[i=j]}$  is compact in  $\mathbb{Q}(i, j)$ , then  $Q_{[i > j]}$  must be noncompact in  $\mathbb{Q}(i, j)$ . Clearly, for  $\widehat{\varphi}_i = \widehat{\varphi}_j$ , we have the following implication:

$$\widehat{\varphi}_i \ll_Q^* \widehat{\varphi}_j \implies \widehat{\varphi}_i \prec_Q^* \widehat{\varphi}_j.$$

**Definition 5 (Query-optimal  $\widehat{\varphi}$ -programs).** Let  $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$  be a computable functional. Let  $e$  be a  $\widehat{\varphi}$ -program for  $F$ .

1. We say that  $e$  is an **absolute** query-optimal  $\widehat{\varphi}$ -program for  $F$  if and only if, on every  $(f, x) \in \mathcal{T} \times \mathbf{N}$ , the OTM  $\widehat{M}_e$  does not make unnecessary queries during the course of computing  $\widehat{\varphi}_e(f, x)$ .
2. We say that  $e$  is a **strong** query-optimal  $\widehat{\varphi}$ -program for  $F$  if and only if there is no  $\widehat{\varphi}$ -program  $i$  for  $F$  such that,  $\widehat{\varphi}_e \prec_Q^* \widehat{\varphi}_i$ .
3. We say that  $e$  is a **weak** query-optimal  $\widehat{\varphi}$ -program for  $F$  if and only if there is no  $\widehat{\varphi}$ -program  $i$  for  $F$  such that,  $\widehat{\varphi}_e \preccurlyeq_Q^* \widehat{\varphi}_i$ . ■

Definition 5 gives us three versions of query-optimal  $\widehat{\varphi}$ -programs, where version 1 is the strongest notion and version 3 is the weakest one. Suppose that  $e$  is a  $\widehat{\varphi}$ -program for  $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$ . The following implications are straightforward:

1. If  $e$  is an absolute query-optimal  $\widehat{\varphi}$ -program for  $F$ , then it is also a strong query-optimal  $\widehat{\varphi}$ -program for  $F$ .
2. If  $e$  is a strong query-optimal  $\widehat{\varphi}$ -program for  $F$ , then it is also a weak query-optimal  $\widehat{\varphi}$ -program for  $F$ .

Moreover, if  $\mathbb{T}(\widehat{\varphi}_e) = \mathbb{Q}(e)$ , then  $e$  is an absolute query-optimal  $\widehat{\varphi}$ -program for  $\widehat{\varphi}_e$ . We shall argue that the stronger the notion of *query-optimum*, the easier we can obtain a speed-up theorem. In other words, it is more difficult to obtain a query-optimal OTM when the notion of query-optimum becomes stronger. However, we do not know if the speed-up phenomenon still exists under our weakest notion of query-optimum; neither do we know how far can one weaken the notion of query-optimal programs to do away the speed-up phenomenon while keeping the notion nontrivial. We give partial results in the next section.

## 4 Query-optimal Programs

With a fixed complexity measure that satisfies Blum's axioms, the original speed-up theorem states that there is a total computable function without optimal program for it. Although query-complexity does not satisfy the Blum's axioms, each of the three versions of query-optimal programs gives rise to the following questions:

1. Does every computable functional  $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$  always have a query-optimal  $\widehat{\varphi}$ -program for it?
2. If the answer to the first question is negative, then can we construct the query-speedable functional?
3. If a given  $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$  does have a query-optimal  $\widehat{\varphi}$ -programs for it, then can we uniformly construct a query-optimal program for  $F$  from an arbitrary  $\widehat{\varphi}$ -program for  $F$ ?
4. Suppose there is no query-optimal  $\widehat{\varphi}$ -program for  $\widehat{\varphi}_{e_0} : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$ . Can we effectively construct an infinite sequence of query-speeded-up version of  $\widehat{\varphi}$ -programs from  $e_0$ ? That is, can we have  $e_0, e_1, e_2, \dots$  such that, for each  $i \in \mathbf{N}$ ,  $e_{i+1}$  is a query-speeded-up version of  $e_i$ ?

We have a positive answer to the second question with respect to the *absolute* and *strong* versions of query-optimal programs. Thus, a negative answer to the first question follows immediately. Compared to the original *nonconstructive* speed-up theorem, the speedable functional for the second question is very simple. However, we do not know if there is a computable  $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$  that can forever strongly speed-up, i.e., we do not know if there is a computable  $F : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$  without a weak query-optimal  $\widehat{\varphi}$ -program. The answer to the third question is negative given by Theorem 4. The fourth question is still open. That is, given  $\widehat{\varphi}_{e_0} : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$  that is known to be speedable, we do not know if there is an effective way to construct a sequence of  $\widehat{\varphi}$ -programs such that,

$$\widehat{\varphi}_{e_0} \prec_Q^* \widehat{\varphi}_{e_1} \prec_Q^* \widehat{\varphi}_{e_2} \prec_Q^* \dots$$

or similarly,

$$\widehat{\varphi}_{e_0} \prec_Q^* \widehat{\varphi}_{e_1} \prec_Q^* \widehat{\varphi}_{e_2} \prec_Q^* \dots.$$

Let  $\varphi_e(x) \downarrow_s$  denote the case that the Turing Machine  $M_e$ , on  $x$ , halts in  $s$  steps. We now define a useful functional  $K : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$  in the following.

**Definition 6.** Define  $K : \mathcal{T} \times \mathbf{N} \rightarrow \mathbf{N}$  by

$$K(f, x) = \begin{cases} \varphi_x(x) + 1 & \text{if } \varphi_x(x) \downarrow_s, \text{ where } s = \sum_{i=0}^{f(0)} f(i); \\ 0 & \text{otherwise.} \end{cases}$$

■

Let  $\mathbf{H}$  denote the set  $\{x | \varphi_x(x) \downarrow\}$ . For any  $(f, x) \in \mathcal{T} \times \mathbf{N}$ , if  $x \notin \mathbf{H}$ , then any  $\widehat{\varphi}$ -program that computes  $K$ , on input  $(f, x)$ , may ask arbitrarily many unnecessary queries, where the number of queries depends on the value of  $f(0)$ . It is easy to prove that there is no  $\widehat{\varphi}$ -program for  $K$  with all unnecessary queries removed (Theorem 1). Even if we lower the standard to strong query-optimal programs, we still cannot have a query-optimal  $\widehat{\varphi}$ -program for  $K$  (Theorem 2). Theorem 3 states that  $K$  does have a weak query-optimal program for it.

**Theorem 1.**  $K$  has no absolute query-optimal  $\widehat{\varphi}$ -program.

Proof: We show that from an absolute query-optimal  $\widehat{\varphi}$ -program for  $K$ , if any, one can construct a solution to the halting problem.

Suppose, by contradiction,  $\widehat{\varphi}_e$  is an absolute query-optimal  $\widehat{\varphi}$ -program for  $K$ . Clearly, if  $x \notin \mathbf{H}$ , then for any  $f \in \mathcal{T}$ ,  $K(f, x) = 0$ . On the other hand, if  $x \in \mathbf{H}$ , there must be two distinct  $f, g \in \mathcal{T}$  such that  $\widehat{\varphi}_e(f, x) \neq \widehat{\varphi}_e(g, x)$ . Thus, some queries to the oracles  $f$  and  $g$  must be made. Therefore, given any  $x \in \mathbf{N}$ , we can run  $\widehat{\varphi}_e(\emptyset^{\sim 0}, x)$  first, where  $\emptyset^{\sim 0}$  is the zero everywhere function. Then, if no queries were made during the course of computation, we know that  $x \notin \mathbf{H}$ . Hence,  $\mathbf{N} - \mathbf{H}$  is recursively enumerable, a contradiction. □

**Theorem 2.**  $K$  has no strong query-optimal  $\widehat{\varphi}$ -program.

Proof: Suppose, by contradiction,  $\widehat{\varphi}_e$  is a strong query-optimal  $\widehat{\varphi}$ -program for  $K$ . Let  $i \notin \mathbf{H}$  such that, for any  $f \in \mathcal{T}$ , the OTM,  $\widehat{M}_e$ , on  $(f, i)$ , will make oracle queries at points  $0, 1, 2, \dots, f(0)$ . Such  $i$  must exist, otherwise the halting problem can be solved by a similar argument given for the proof of Theorem 1. We construct a *weakly* sped-up version of  $e$  as shown in Figure 1.

```

Program
  input  $f : \mathcal{T}, x : \mathbf{N}$ ;
  if  $x = i$  and  $f(0)$  is even
    then output 0;
    else output  $\widehat{\varphi}_e(f, x)$ ; /* by running  $\widehat{\varphi}$ -program  $e$  on  $(f, x)$  */
End program

```

**Fig. 1.** A Weakly Sped-up Version of  $\widehat{\varphi}_e$

Let the index of the  $\widehat{\varphi}$ -program in Figure 1 be  $p$ . Clearly, if  $x \neq i$  or if  $f(0)$  is odd, then the two  $\widehat{\varphi}$ -programs,  $e$  and  $p$ , will perform the same computation, and hence the two make the same oracle queries. The two computations differ on the following set:

$$\{(f, i) \mid f(0) \text{ is even}\}. \quad (3)$$

Clearly, the set above is noncompact in  $\mathbb{Q}(e, p)$ . According to the program  $p$  and the assumption on  $i$ , the following set is the same as (3),

$$\left\{(f, i) \mid \text{card}(\lim_{s \rightarrow \infty} Q(s, e, f, i)) > \text{card}(\lim_{s \rightarrow \infty} Q(s, p, f, i))\right\}.$$

Thus,  $Q_{[e>p]}$  is noncompact in  $\mathbb{Q}(e, p)$ . Moreover,  $Q_{[e<p]} = \emptyset$ , and hence  $Q_{[e<p]}$  is compact in  $\mathbb{Q}(e, p)$ . It follows that  $p$  is a weakly sped-up version of  $e$ , i.e.,  $\widehat{\varphi}_e \prec_Q^* \widehat{\varphi}_p$ . Therefore,  $e$  cannot be a strong query-optimal  $\widehat{\varphi}$ -program for  $K$ .  $\square$

**Theorem 3.** *There is a weak query-optimal  $\widehat{\varphi}$ -program for  $K$ .*

Sketch of Proof: It is impossible to strongly speed-up any  $\widehat{\varphi}$  forever. Otherwise, we can reach two  $\widehat{\varphi}$  programs  $e$  and  $p$  such that,  $\widehat{\varphi}_e = \widehat{\varphi}_p = K$  and  $\widehat{\varphi}_e \prec_Q^* \widehat{\varphi}_p$ . Then, we can compare the queries made during the course of the computations between  $\widehat{\varphi}_e$  and  $\widehat{\varphi}_p$  to solve the halting problem. Therefore, a weak query-optimal  $\widehat{\varphi}$ -program for  $K$  must exist. We skip details of the proof.  $\square$

However, we do not know if there is a computable type-2 functional such that, there is no weak query-optimal  $\widehat{\varphi}$ -program for it. In other words, we do not if there is a speed-up phenomenon associated with the notion of weak query-optimum.

Suppose we know that a query-optimal program for  $\widehat{\varphi}_e$  does exist. In general, there is no effective way to construct a query-optimal version of  $e$  in any strength, i.e., absolute, strong, and weak query-optimum. We state this in the following theorem, which is the strongest version in the sense that the weakest version of query-optimal programs is considered.

**Theorem 4.** *There is no recursive  $r : \mathbf{N} \rightarrow \mathbf{N}$  such that, for every  $\widehat{\varphi}$ -program  $e$ , if  $\widehat{\varphi}_e$  is total and there is a weak query-optimal program for it,  $r(e)$  is a weak query-optimal  $\widehat{\varphi}$ -program for  $\widehat{\varphi}_e$ .*

Proof: By contradiction, suppose such recursive  $r : \mathbf{N} \rightarrow \mathbf{N}$  exists. Consider the following program shown in Figure 2. Suppose that, by the recursion theorem, the index of the  $\widehat{\varphi}$ -program is  $e$ .

```

Program e
  input  $f : \mathcal{T}, x : \mathbf{N}$ ;
   $Q \leftarrow Q(f(0), r(e), f, x)$ ;
  if  $\widehat{\varphi}_{r(e)}(f, x)$  converges in  $f(0)$  steps
    then output  $f(\max(\text{dom}(Q)) + 1)$ ;
    else output  $f(1)$ ;
End program e

```

**Fig. 2.** A  $\widehat{\varphi}$ -program with index  $e$  for Theorem 4

According to the construction, it is clear that  $e$  is a total  $\widehat{\varphi}$ -program. By assumption,  $r(e)$  is a weak query-optimal version of  $e$ . On any  $(f, x) \in \mathcal{T} \times \mathbf{N}$ , the OTM  $\widehat{M}_{r(e)}$  will never halt in  $f(0)$  steps, otherwise the OTM  $\widehat{M}_{r(e)}$  does not compute  $\widehat{\varphi}_e$ . This is because if the OTM  $\widehat{M}_{r(e)}$  on  $(f, x)$  halts in  $f(0)$  steps,  $Q$  must contain the complete collection of queries made by the OTM  $\widehat{M}_{r(e)}$  on  $(f, x)$ . But  $\widehat{\varphi}_e(f, x)$  is the value of  $f$  at a point not in  $Q$ .

Thus, we conclude that for every  $(f, x) \in \mathcal{T} \times \mathbf{N}$ ,  $\widehat{\varphi}_e(f, x) = f(1)$ . But if that is the case,  $f(0)$  will not be queried by the OTM  $\widehat{M}_{r(e)}$ , namely, the value of  $f(0)$  does not affect the computation of  $\widehat{M}_{r(e)}$  at all. Thus, if  $f(0)$  is sufficiently large,  $\widehat{\varphi}_{r(e)}(f, x)$  will converge in  $f(0)$  steps, and hence  $\widehat{\varphi}_e(f, x) = f(\max(\text{dom}(Q)) + 1)$ . Thus, the OTM  $\widehat{M}_{r(e)}$  does not compute  $\widehat{\varphi}_e$ .

Therefore,  $r(e)$  cannot be a weak query-optimal version of  $\widehat{\varphi}$ -program  $e$ .  $\square$

## 5 Conclusion

The oracle query is a unique dynamic complexity measure in type-2 computation. Although Blum's complexity measure has created a rich chapter in machine independent complexity theory, it is not appropriate to impose the same requirement to query complexity. We therefore provide new notions in order to describe the concept of query-optimal programs in type-2 computation. We obtain a speed-up theorem under a reasonable notion of query-optimal programs, which means that there exists a computable type-2 functional that does not have a query optimal OTM for it. Our version of speed-up theorem is stronger than the classical one in a sense that the speedable functional,  $K$ , does not depend on the speedup factor, i.e., the computable function  $r$  in equation (1).

Clearly, there are many open questions that deserve further invitation. For example, can we uniformly construct a query-spaced-up program for our functional  $K$ ? Do we have a speed-up theorem under our weaker notions of query-optimal programs? Some classical questions can also be asked. For examples, do we have a union theorem? gap theorem? compression theorem? Or do we have a complexity hierarchy characterized by the query-complexity? Since Blum's two axioms cannot be applied to the query-complexity, the approach and results must be very different from the classical ones. It seems to us that the framework proposed in this paper may be a feasible direction for computer science theorists to study the query complexity closely.

## References

1. Manuel Blum. A machine-independent theory of the complexity of recursive functions. *Journal of the ACM*, 14(2):322–336, 1967.
2. Manuel Blum. On effective procedures for speeding up algorithms. *Journal of the ACM*, 18(2):290–305, 1971.
3. Critian Calude. *Theories of Computational Complexity*. Number 35 in Annals of Discrete Mathematics. North-Holland, Elsevier Science Publisher, B.V., 1988.
4. Nigel Cutland. *Computability: An introduction to recursive function theory*. Cambridge, New York, 1980.
5. Martin Davis, editor. *The Undecidable*. Raven Press, New York, 1965.
6. Kurt Gödel. Über die längte der beweise. *Ergebnisse eines Math. Kolloquiums*, 7:23–24, 1936. Translation in [5], pages 82–83, “On the length of proofs.”.
7. Chung-Chih Li. Asymptotic behaviors of type-2 algorithms and induced Baire topologies. In *Proceedings of the Third International Conference on Theoretical Computer Science*, pages 471–484, Toulouse, France, August 2004.
8. Chung-Chih Li. Clocking type-2 computation in the unit cost model. In Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John V. Tucker, editors, *Proceedings of Computability in Europe, CiE 2006: Logical Approaches to Computational Barriers, CSR 7-2006*, pages 182–192, Swansea, UK, June/July 2006.
9. Chung-Chih Li. Speed-up theorems in type-2 computation. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *Proceedings of Computability in Europe, CiE 2007: Computation and Logic in the Real World*, pages 478–487, Siena, Italy, June 2007. Springer, LNCS 4497.
10. Chung-Chih Li and James S. Royer. On type-2 complexity classes: Preliminary report. In *Proceedings of the Third International Workshop on Implicit Computational Complexity*, pages 123–138, Aarhus, Denmark, May 2001.
11. A. R. Meyer and P. C. Fischer. Computational speed-up by effective operators. *The Journal of Symbolic Logic*, 37:55–68, 1972.
12. Joel I. Seiferas. Machine-independent complexity theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 163–186. North-Holland, Elsevier Science Publisher, B.V., 1990. MIT press for paperback edition.
13. P. Van Emde Boas. Ten years of speed-up. *Proceedings of the Fourth Symposium Mathematical Foundations of Computer Science 1975*, pages 13–29, 1975. Lecture Notes in Computer Science.
14. Klaus Wagner and Gerd Wechsung. *Computational Complexity*. Mathematics and its applications. D. Reidel Publishing Company, Dordrecht, 1985.

# From Hilbert's Program to a Logic Tool Box

J. A. Makowsky

Department of Computer Science  
Technion – Israel Institute of Technology  
Haifa, Israel  
[janos@cs.technion.ac.il](mailto:janos@cs.technion.ac.il)  
[www.cs.technion.ac.il/~janos](http://www.cs.technion.ac.il/~janos)

For Witek Marek, first mentor,  
then colleague and true friend,  
at the occasion of his 65th birthday.

**Abstract.** In this paper I discuss what, according to my long experience, every computer scientist should know from logic. We concentrate on issues of modeling, interpretability and levels of abstraction. We discuss how the minimal toolbox of logic tools should look like for a computer scientist who is involved in designing and analyzing reliable systems. We shall conclude that many classical topics dear to logicians are less important than usually presented, and that less known ideas from logic may be more useful for the working computer scientist.

The following text is not a scientific paper. It is really a prose version of a set of slides in which I present my ideas on the subject. I have presented a first version of these slides at the LPAR'07 conference in Yerevan, Armenia, in October 2007. I do hope that I will finally turn these thoughts into a proper scholarly paper. In these sketchy notes I mostly give references to monographs, and not the original papers.

## The Students I Have in Mind

I want to examine what we should teach from logic to our *non-specialized undergraduate students*. I mean, what does *every* graduate of Computer Science have to learn in/from logic? The current syllabus is often justified more by *the traditional narrative* than by *the practitioner's needs*. The practitioner's needs are determined by what he needs to understand his own activity in dealing with his computing environment. As a computer/computing engineer he should be aware of the inherent difference between *consumer products* and *life-critical hardware and software*. The occasional failure of consumer goods is beneficial to the functioning of the Fordistic consumer society in as it maintains the consumption cycles needed for its functioning. The failure of life-critical products is disastrous

for all the parties involved. Life-critical products have to be properly *specified, verified, tested and certified* before they can be released. The practitioner therefore needs a basic understanding of what it means to properly specify, test, verify and possibly certify a product.

Practically speaking,

- he should understand the meaning and implications of modeling his environment as precise mathematical objects and relations;
- he should understand and be able to distinguish intended properties of this modeling and side-effects;
- he should be able to discern different level of abstraction;
- he should master the (non-formalized) language of sets and second order logic which enables him to speak about the modeled objects;
- he should understand what it means to prove properties of modeled objects and relations;
- but he should also understand the *inherent limitations* of what can be achieved, and of his own activity.

## 1 Sets and the Logical Foundations of Mathematics

Whether we like or not depends on our philosophical position, if we at all have one, but it is a fact supported by a large social consensus, that the language of sets is the most used and most accepted way of modeling mathematical objects. A very convincing discussion, why sets are used that way, is given in [BG08]. We are used to model automata of all sorts including Turing machines as tuples of sets, functions and relations. We do the same when we discuss behavior of hardware and software, when we prove properties of modeled artefacts, and when we show that certain combination of properties of such artefacts cannot be achieved.

The emergence of the language of sets goes back to the work of G. Cantor and G. Frege, who both felt the need to put mathematics on new rigorous foundations upon which the growing edifice of real and complex analysis could be built. Cantor initiated the use of sets for modeling natural, real and complex numbers and their functions, and Frege wanted to derive the rules of set formation from logic. Frege's program intended to derive the foundations of mathematics from logical principles. It derived set theory as the universal data structure for modeling mathematical objects from logic. The history of logic in the years between 1850 and 1950 is the history of successes and failures of Frege's program. This history forms the traditional narrative along which we are used to teach logic. I will argue that this narrative is misleading as far as the working mathematician or computer scientist or engineer is concerned.

Let us look at this traditional narrative the way I see it. We start by paraphrasing the history of Logicism from Frege to Gödel, and further to the re-evaluation of Frege's program.

### Act I: Cantors Paradise

- First G. Cantor (1874 - 1884) created the Paradise of Sets.
- Then G. Frege (1879) created the modern Logical Formalisms, including the *correct binding rules for quantification*, and
- set out to lay the Foundations of Mathematics with his *Die Grundgesetze der Arithmetik*, Volume1 (1893), see [Bur05]
- The book was not well received. Only G. Peano, author of *The principles of arithmetic, presented by a new method* (1889), [Ken73] wrote a positive review of it.

### Act II: Paradise lost

- On 16 June 1902, Bertrand Russell pointed out, with great modesty, that the Russell paradox gives a *contradiction* in Frege's system of axioms.
- ... and with Russel's paradox started the *crisis of the Foundations of Mathematics*,
- G. Cantor had sensed this, when he noticed trouble with the "set of all sets" and his notion of cardinality.

Let  $V$  be the set of all sets. Then its power set  $P(V)$  is a subset of  $V$ . But Cantor proved that the cardinality of the power set  $P(A)$  of a set  $A$  is always strictly bigger than the cardinality of  $A$ . On the other hand the cardinality of a subset of  $A$  is at most the cardinality of the set  $A$ , a contradiction.

### Act III: Hilbert's Program

D. Hilbert around 1920 designs a program to provide a secure foundations for all mathematics. In particular this should include<sup>1</sup>:

- *Formalization* of all mathematics: all mathematical statements should be written in a precise formal language, and manipulated according to well defined rules.
- *Completeness*: a proof that all true mathematical statements can be proved in the formalism.
- *Consistency*: a proof that no contradiction can be obtained in the formalism of mathematics. This consistency proof should preferably use only "finitistic" reasoning about finite mathematical objects.
- *Conservation*: a proof that any result about "real objects" obtained using reasoning about "ideal objects" (such as uncountable sets) can be proved without using ideal objects.
- *Decidability*: there should be an algorithm for deciding the truth or falsity of any mathematical statement.

---

<sup>1</sup> This subsection is a quote from Wikipedia. Its author is unknown.

### Hilbert's Logic Lectures

In 1928, D. Hilbert and W. Ackermann publish *Grundzüge der theoretischen Logik*, [HA28, HA49, HA50]. Here are the points of interest to us:

- The Logic in question is *Second Order Logic*.
- What we call *First Order Logic*, is called there the *restricted calculus*.
- They *prove* soundness of the calculus, and ask the question of *completeness*.

The book is soon translated into English, French and Russian and remains the most widely used reference for more than thirty years. K. Gödel, as a graduate student, reads the book in 1928. The original book contains several technical mistakes which are fixed in subsequent editions. The first English edition [HA50] gives credit to A. Church and W. Quine for pointing out some mistakes in the second German edition.

The book states as the main problem of Logic its axiomatization and proofs of the

- Independence of the axioms
  - Consistency of the axioms
  - Completeness of the axioms
  - The decision problem of the consequence relation

### Act IV: Rise and Fall of Hilbert's Program

#### Initial successes:

- Leopold Löwenheim (1915), Thoralf Skolem (1920), Mojzesz Pressburger (1929), Alfred Tarski (1930), Frank Plumpton Ramsey (1930), László Kalmár (1939) and many others prove partial *decidability* results for fragments of Logic, and for Arithmetic, Algebra, Geometry.
- In 1929 Kurt Gödel proves the *completeness* of the Hilbert-Ackermann axiomatization of the the *restricted* (first order) calculus.

#### Final blows:

- 1931 K. Gödel proves that every recursive theory which contains arithmetic is *incomplete*.
- 1931 K. Gödel proves that every recursive consistent theory which contains arithmetic cannot prove its own consistency.
- 1936 Alonzo Church and Alain Turing show that already for the *restricted* calculus with free relation variables the set of tautologies is not computable (but is semi-computable). Hence, they gave a negative solution to the Decision Problem.

The most comprehensive account of solvable and unsolvable cases of the Decision Problem can be found in [BGG97].

### **Act V: Clarifications and Repairs**

Out of the ashes rise the four classical sub-disciplines of mathematical logic:

**Set Theory** arises from work by E. Zermelo, D. Mirimanoff, J. von Neumann, A. Fränkel, K. Gödel and P. Bernays. Alternative approaches were developed by, among others, W. Quine, W. Ackermann, and J.L. Kelley and A.P. Morse, and more recently, by P. Aczel.

Set theory, in contrast to using sets in mathematical practice, is mostly concerned with settling questions around the axiom of choice and cardinal arithmetic, or in formulating alternatives, such as the axiom of determinacy, and in clarifying their impact on questions in topology and analysis. Today, set theory is a highly specialized branch of technical mathematics with little impact on computer science.

**Proof Theory** arises from work by W. Ackermann, G. Gentzen, J. Herbrand, D. Hilbert and P. Bernays.

It has developed into a full-fledged theory of proofs, comprising the analysis of (transfinite) consistency proofs in terms of ordinals and fast-growing functions, program extraction from proofs, and resource analysis related to provability. It also plays an important role in all aspects of automated reasoning, an important branch of Artificial Intelligence.

**Recursion Theory** arises from work by E. Post, J. Herbrand, K. Gödel, A. Church, A. Turing, H. Curry. Recursion theory developed at one side into degree theory, classifying non-computable functions according to their different levels of complexity, at the other side it developed into Computability Theory and has become one of the pillars of Computer Science education in its own right.

**Model theory** arises from work by T. Skolem, A. Tarski, A. Robinson, R. Fraïssé and A. Mal'cev.

Two main directions evolve, classification theory, and a more algebraic and geometric theory, linking model theory with algebraic geometry and number theory. Although finite model theory has its early origin, it was through Automata Theory, Database Theory and Complexity Theory that it evolved into its own discipline with a legitimate place in advanced Computer Science education.

... and for long this remained the classical divide of **Mathematical Logic**.

J. Shoenfield's monograph [Sho67] is possibly the only monograph covering all aspects of Mathematical Logic up to the boundaries of research of his time. Since then the four classical disciplines pursue their own paths, and among the younger generations of researchers it cannot be taken for granted that they have studied the four disciplines in depth.

### **Act VI: 100 years later - Fixing Frege**

If only G. Frege had not been so scared by B. Russel's letter. C. Wright, P. Geach and H. Hodes suggested, and G. Boolos proved (1987) that a modified

Frege program actually is feasible, [Boo98a,Boo98c,Boo98b]. They noted that the famous contradiction stemmed from the axiom which states, roughly, that the extension of any concept is a set. However, this axiom is only used to derive an abstraction principle, called Hume's principle, which states, again roughly, that two extensions have the same cardinality if and only there is a bijection.

So we have for the modified Frege system:

**Frege:** The Peano Postulates can be deduced in dyadic second order logic from Hume's principle and suitable definitions of the natural numbers (Frege's Arithmetic).

**Boolos:** Frege's arithmetic is interpretable in second order Peano Arithmetic.

Some (the Neo-Logicists) argue that this justifies a revival of Logicism. But it also creates new problems. A thorough discussion of the pros and cons of Neo-Logicism can be found in J. Burgess [Bur05]. A thorough discussion of *abstraction principles* similar to Hume's Principle can be found in K. Fine [Fin02].

That much for the "big crisis".

At least the set theory needed for the foundations of Computer Science can be derived from logical principles.

## 2 The Foundations of Mathematics and Computer Science

What Frege and Russel and Whitehead had in mind, viz. to build the foundations of mathematics from scratch, was done in a more intelligible (but still not too user friendly) way by E. Landau in his *Foundations of Analysis* first published in German in 1930, and in English in 1951, with many reprints, the latest in 1999 with a German-English vocabulary by the American Mathematical Society, [Lan99]. In this book he explicitly constructs the real and complex numbers from the standard model of Peano arithmetic. Landau's style is very dry and concise, and the text was written for mature mathematicians. A more pedagogical version of the same constructions can be found in S. Feferman's [Fef64], which I also would love to see reprinted. One can view such a foundation of Analysis as a *pragmatic* version of Frege's program. Roughly, one proceeds as follows:

- One starts with a cumulative hierarchy of sets, based on the empty set alone (or with urelements) and natural set construction principles which allow to construct also infinite sets.
- Then one defines inductively the natural numbers with a successor function, and the sets of finite words over a (not necessarily) finite alphabet (a set), with an append operation for each element of the alphabet.

- One then proceeds with defining the number systems  $\mathbb{N}$ ,  $\mathbb{Z}$ ,  $\mathbb{Q}$  and their arithmetic operations *inductively* and using *quotient structures*.
- Then one constructs the reals  $\mathbb{R}$ , using Dedekind cuts.
- In similar ways one constructs other structures, say, groups, fields, topological spaces, Banach spaces, Lie algebras, which are specified *axiomatically*.
- *Existence* of axiomatically defined objects had to be established by an *explicit sequence of set construction steps within the cumulative hierarchy*.
- Clearly, one can apply the same methods to model objects of the computing world, such as automata, formal languages, programs, data structures, etc.

One *should* adapt Landau's way for modeling the basic data structures of Computer Science. I have attempted to do this in the course *Sets and Logic for Computer science*, which we teach in the third semester in our Computer Science undergraduate program.

### Set Theory in Computer Science

Besides using set theory for modeling purposes, the computer scientist uses only few ingredients of set theory:

1. The Cantor-Bernstein Theorem to prove equicardinality,
2. The fact that a countable union of countable sets is countable,
3. The fact that the cardinality of the power set of a set  $A$  is always bigger than that of  $A$ ,
4. The relationship between the termination of processes and well-orderings.
5. The Recursion Theorem.
6. Some Fixed-Point Theorems.

The first three of these go ultimately back to Cantor's original work and are very basic. The Recursion Theorem and Fixed Point Theorems should be taught in a more advanced course.

### Recursion Theory vs Computation Theory

Recursion Theory got its name for a good reason: The computable functions over the natural numbers were defined recursively, and early Recursion Theory consisted in studying the strength of various proposed recursion schemes. Recursion schemes can be replaced by register machines. Computability Theory studies usually the computable relations and functions over sets of words. The three approaches are inter-translatable, but they are not the same. It is a pity that teaching all these complementary notions of computability is not always part of the Computer Science curriculum. Be that as it may, Computation Theory has emancipated itself from Logic and in Computer Science the two are often taught independently. A good exception is the monograph [Pap94].

### Proof Theory

Proof theory evolved around the question what type of consistency proofs are at all possible. As a spin-off the field of deduction-based automated reasoning and automated theorem proving came into being. Proof theory is also used in the foundations of programming languages, where it generated a rich literature of deep insights into the nature of programming. Some basic principles of automated reasoning do belong into a beginner's course on Artificial Intelligence, and some basic facts about functional programming do belong into a basic course on Programming Languages. However, only little of this rich material is suitable for the undergraduate student I have in mind. A comprehensive survey of Proof Theory is [Bus98].

### Model Theory

The main tools of classical model theory almost all derive from the compactness theorem and variations of the model existence theorem. Using these tools one proves preservation theorems, the omitting types theorem and develops a general understanding of the possible structure of models of first order theories. I have described how to use these tools in the Computer Science context, in database theory, the foundations of Logic Programming, and the specification of data types, [Mak84,Mak92]. It turned out, however, that the Compactness Theorem is mostly suitable when dealing with infinite structures, and the most prominent application of the Compactness Theorem in Computing is Herbrand's Theorem with its ramifications in Automated Reasoning and Logic Programming. The most important tools from Model Theory in algorithmic applications are the Ehrenfeucht-Fraïssé games and the Feferman-Vaught Theorem and its variations. The former is omnipresent in Finite Model Theory, cf. [EF95,Lib04] and the a survey of the uses of the latter can be found in [Mak04]. For other failures of classical theorems of First Order Logic when restricted to the finite case, cf. [Gur88].

One should add here that the combination of the Compactness Theorem with the Ehrenfeucht-Fraïssé games leads to Lindström's characterization of First Order Logic. The attempts to develop an *Abstract Model Theory* are documented in the monumental [BF85]. This line of reasoning had a considerable impact on Finite Model Theory and Descriptive Complexity in providing techniques for defining logics which capture complexity classes. For the advanced student applications of Finite Model Theory to Computer Science are surveyed in [GKL<sup>+</sup>07].

### The classical textbooks in Logic

The available undergraduate texts of Logic for Computer Science follow too often the narrative of the em Rise and Fall of Hilbert's Program. They emphasize the classical Hilbertian topics.

- Logic is needed to resolve the paradoxes of set theory.

- First Order Logic is THE LOGIC due to its completeness theorem.
- The main theorems of logic are the *Completeness Theorem* and the *Compactness Theorem*
- The tautologies of First Order Logic are not recursive.
- Arithmetic Truth is not recursive enumerable.
- One cannot prove CONSISTENCY within rich enough systems.

This is NOT what a Practitioner  
of Computing Sciences NEEDS !

Other texts are often written with a very special agenda reflecting the author's research interest or his particular tastes. Finally, there are texts which are really written with the undergraduate Computer Science student in mind. But then they are often either written specifically with programmers in mind, and do not deal with the data modeling issues<sup>2</sup>. Logic is first of all language in which we express ourselves before we prove statements. We first have to formulate a *specification*, a *database query*, an *intermediate assertion*, a *loop invariant*, before we prove them to hold, to be valid, or for two of them to be equivalent or not. Logic deals with **definability** issues as much as with **provability** issues, something which in the Hilbertian tradition is easily forgotten. An introductory text in Logic for Computer Science should choose its topics in way, that the student meets the topics taught again in later courses. When we teach Linear Algebra in the first year of a mathematics curriculum, most of the topics reappear in vector analysis, differential equations, physic, statistics etc. We have to build our syllabus of logic keeping this in mind.

### 3 So what Does a Practitioner of Computing Sciences Need?

We distinguish between knowledge of *theoretical orientation* and *practical knowledge*, which consists of *tools* and *skills*. In our case this means:

#### Theoretical orientation:

- *awareness* that our domain of discourse is an *idealized world of artefacts* which models fairly accurately the artefacts which allow us to run and interact with computing machinery.
- *awareness* of the different levels of abstractions.
- *awareness* that in this world of artefacts there are *a priori limitations*. Not everything is realizable, computable, etc.

---

<sup>2</sup> The recent book by R. Bornat[Bor05] is a lovely introduction to Logic for programmers.

**Practical knowledge:**

- *tools* which allow us to *model new artefacts*, whenever they arise;
- *tools* which allow us to *prove properties* of the modeled artefacts.

The student needs a carefully adapted blend of the *practical Frege* program, with the knowledge of its *limitations*. He needs both *proficiency* and *performance* in his practical knowledge.

## 4 Lessons from 150 years of History

I have spent so much space reviewing what I consider noteworthy in the evolution of the Logicians program because I do want to draw some lessons from it which are *not foundational* but *practical*. I would like the B.Sc. graduates of Computer Science to be familiar with the following:

**Lesson 1. Modeling the world**

**Our scientific language:** Natural Language enhanced by precise use of boolean operations, quantification and the use of naive language of sets.

**Our universal data structure:** A cumulative world of sets.

**Modeling the world:** We model *all* artefacts of our computing world by constructed objects in the world of sets.

**Modeling involves side effects:**

Modeled artefact have properties not intended.

**Digression: the ordered pair:** Ordered pairs could be introduced via an abstraction principle:

$(x, y) = (x', y')$  if and only if  $x = x'$  and  $y = y'$ .

But usually the ordered pair is modeled directly:

N. Wiener:  $(x, y)_W := \{\{x\}, \emptyset\}, \{\{y\}\}$ ,

K. Kuratowski:  $(x, y)_K = \{\{x\}, \{x, y\}\}$ ,

Simplified :  $(x, y)_S = \{x, \{x, y\}\}$ .

Now one has to verify that  $(x, y) = (x', y')$  if and only if  $x = x'$  and  $y = y'$ .

All the proposed versions do satisfy this, but the proofs differ. The simplified version requires the axiom of foundation. Kuratowski's version is the accepted definition today. But all definitions have **side effects**, e.g.,  $x$  is an element of  $\{x, \{x, y\}\}$  but not of  $\{\{x\}, \{x, y\}\}$ . Proving properties of objects which depend on the use of ordered pairs should not use these side effects, but only the defining property.

The distinction between specified properties and side effects should be taught early on!

**Fixing levels of abstraction:** Introducing structures, and fixing which sets are not further to be analyzed.

A graph is a pair  $< V, E >$ .

A finite automaton is a tuple  $< S, \Sigma, R, I, T >$ .

Like in the foundations of Analysis, as practiced by R. Dedekind, E. Landau and N. Bourbaki, we need the *precise language mix* of *normalized natural language* augmented by the *language of sets* to *model* the *idealized artefacts* of computer science. To model the artefacts we also need basic tools.

**Artefacts:**

strings, concatenation, natural numbers,  
 graphs, relational structures stacks, arrays;  
 circuits, Turing machines, register machines;  
 specification and programming languages,

**Tools:** Inductive definitions, proofs by induction;  
 enumerations,  
 proving countability and uncountability;  
 well-orderings (for termination)

Is this not "too denotational" ? ....  
 ... our friends may ask.

Yes, this approach *does* map everything into *sets*. But "truth" does not necessarily *presuppose* a world of sets. Truth in the sense of Frege's world is defined by the laws (introduction and elimination rules) of logic and of the Fregean constructs. It does leave your *foundational options* open ...

**Lesson 2. Modeling Computability and its limitations (when modeled)**

We have already said that computability is usually taught in a separate course, be it together with formal languages or with an introduction to basic complexity theory. Nevertheless, our student should understand that computability is modeled over different domains, computing operations, resource restrictions.

**Natural numbers and recursion:**

The original definition of the set of *recursive functions*.

**Natural numbers and register machines:**

Close to early programming languages.

**Turing machines and words:** Close to assembly languages.

**Other models:** Logic programs, Lambda calculus, cellular automata, quantum computing

Showing their equivalence involves modeling also

- translation between the domains;
- translations between programs (interpreters and compilers).

Here I want to stress The different basic structures involved, and their bi-interpretability. In terms of knowledge of orientation and practical knowledge we have:

**Orientation:**

*Not everything* is computable.

*Not everything* is feasibly computable.

**Tools:**

- Using the non-solvability of the Halting Problem to prove non-computability.
- Using different types of reducibilities (and simulations).

I have observed that even my colleagues are sometimes imprecise: The Church Turing Hypothesis is often carelessly invoked. There is also a trend to say *computable* when actually one means *feasibly computable*, where *feasibly computable* may mean computable in *deterministic* polynomial time, sometimes computable in polynomial time with *randomized algorithms*.

It is also important to *distinguish* between complexity classes defined as *equivalence classes of problems* under certain reductions, and sometimes defined as classes of decision (counting, approximation) problems solvable in a specific computational model.

Using polynomial time Turing reductions, the class  $[SAT]_T$  of problems reducible to  $SAT$  is of the first type,  $NP$  is of the second type, and  $NP = [SAT]_T$  is a theorem. In the case of counting problems  $[\#SAT]_T$  is of the first type,  $\#P$  is of the second type, and  $\#P = [\#SAT]_T$  is not true. Using reductions in First Order Logic we still have  $[SAT]_{FOL} = NP$ , but not every problem  $X$ , which is  $NP$ -complete with respect to polynomial time Turing reductions satisfies  $[X]_{FOL} = [SAT]_{FOL}$ .

It is important to insist that slogans are replaced by precise definitions.

### Lesson 3. Modeling Syntax and Semantics

We look at Propositional, First Order, Second Order Logic, or any other logic of *assertions*. Again we model them in our framework of sets.

**Syntax:**

The syntax is an inductively defined set of words, the well formed expressions.

**Semantics:** *Structures* are interpretations of the basic non-logical symbols.

*Assignments* are interpretations of the variables. The *meaning function* associates with structures, assignments, and formulas a truth value.

What is the meaning of an assertion ?

**Without free variables:** The meaning of an assertion is a *truth value*.

But this is misleading!

**With free variables:** The meaning of an assertion is the set of interpretations of its free variables. In the case of first order variables only it is a *relation*. As in Classical Geometry one speaks of the *geometrical lieu* of all points satisfying an equation, we can speak of the *logical lieu* defined by a formula. In modern data base parlance this is called a *query*.

We define usually *logical validity* via truth values. It would be preferable to define *validity* and *logical consequence* directly for formulas *with free variables*.

Our student is more likely to meet in the sequel of courses formulas with free variables that just formulas without.

### Do we need the Completeness Theorem?

For the **practical knowledge** we need:

- The semantic notion of *logical consequence*.
- Enough basic logical equivalences to prove the *Prenex Normal Form Theorem (PNF)*.
- Introduction and elimination rules for quantifiers (via constants).
- A *game theoretic* interpretation of formulas in PNF.

For the **knowledge of orientation** we might state (but not prove) the Completeness Theorem for our redundant set of manipulation rules.

Here are the arguments for and against proving the Completeness Theorem in the first course of Logic.

The classical argument pro:

- Completeness and its corollary, *Compactness* is at the heart of logic.

My arguments against:

- None of these are part of the practical knowledge we aim at.
- The proof of the Completeness Theorem is a waste of time at the expense of teaching more the important skills of understanding the manipulation and meaning of formulas.
- First Order Logic is not privileged in our context. We deal very often with finite structures, where the Completeness Theorem is not true.
- *Second Order Logic* anyhow is the natural logic we work in, and not taking that seriously confuses the student.

We should instead concentrate on understanding quantification

As **tools** we need to

- Read, write and *understand the meaning* of First Order **and** Second Order formulas.
- Understand the relationship between *projection of relations* and first order quantification.
- Understand that *Relational Calculus* and First Order Logic are really the same (i.e., bi-interpretable).
- Introduce immediately after the proof of the Prenex Normal Form Theorem the *Ehrenfeucht-Fraïssé Game*, and proceed to show the easy direction of the *Ehrenfeucht-Fraïssé Theorem*, i.e., if a formula (say in Prenex Normal Form) of quantifier rank  $k$  is true in one but not in another structure, then we can derive from the formula a winning strategy for player I (the spoiler) for the game with  $k$  moves.
- Play with the *game interpretation* of quantifiers to analyze the *amount of quantification* needed to express, say "there exists at least  $n$  elements  $x$  such that  $\phi(x)$ ".
- One can even point out that (easy direction) of *Ehrenfeucht-Fraïssé Theorem* holds also for Second Order Logic.

#### Lesson 4. Limitations of formalisms: Definability

Before we find time to prove the Completeness Theorem, I would like the students to understand the difference between First Order (FO) and Second Order (SO) Logic.

- Look at the statement  
 "There are an equal number of  $x$   
 with  $P(x)$  and with  $Q(x)$ "  
 where  $P, Q$  are unary predicate symbols.  
 This is expressible in SO but not in FO, and we can even show the proof having the Ehrenfeucht-Fraïssé Games available.
- We can even be daring, and show that connectedness on finite graphs is SO-definable, but not FO-definable.
- In the natural numbers  $\mathbb{N}$ , *multiplication* is SO-definable, but not FO-definable, using addition only. However, multiplication is FO-definable using addition and squaring.  
 The negative result we cannot prove in an undergraduate course, as we need the decidability of FO Pressburger Arithmetic. But we can explain it.

### Lesson 5. Interpretability and Reducibility

Again before we use our time prove the Completeness Theorem I would like the students to understand what it means that a structure is FO-interpretable in another structure. Let look at the case of the natural numbers  $\mathfrak{N}$  and the integers  $\mathfrak{Z}$  with their arithmetic operations.

The integers  $\mathfrak{Z}$  with their arithmetic are *FO-interpretable* inside the natural numbers  $\mathfrak{N}$  with their arithmetic.

To get the interpretation we define a new structure from  $\mathfrak{N}$ , called a *transduction*  $T(\mathfrak{N})$ , and which will be isomorphic to  $\mathfrak{Z}$ , as follows.

- The new universe consists of equivalence classes of pairs of natural numbers such that  $(x, y) \sim (x', y')$  iff  $x + y' = x' + y$ .
- The new equality is this equivalence.
- The new addition is the old addition on representatives.
- Same for multiplication.

$T$  is a semantic map. Its syntactic counterpart is the *interpretation*  $S : Formulas \rightarrow Formulas$ , defined as follows:

For any SO-formula  $\phi$  we let  $S(\phi)$  be the result of substituting the *new* definitions of addition and multiplication and equality for the corresponding symbols. In the exact definition one has to be careful with the renaming of free variables.

$S$  and  $T$  are intimately related:

$$\mathfrak{Z} = T(\mathfrak{N}) \models \phi \text{ iff } \mathfrak{N} \models S(\phi)$$

which is the *Fundamental Property of Transductions and Interpretations*.

In the same way we can see that

- The Cartesian product is interpretable in the disjoint union.
- Many graph transformations are given as transductions.
- All implementations of one data structure in another are of this form.
- Transductions and interpretations are everywhere

## 5 The Fundamental Properties of SO and FO

In teaching our students to think and speak Second Order Logic, we should teach

- that isomorphic structures satisfy the same SO sentences;
- the Fundamental Property of Transductions and Interpretations;
- the Prenex Normal Form Theorem and its visualization as a two person game.

and we should practice thinking in SO as the natural language of specifying properties of modeled artefacts.

### The Fundamental Properties of FO

Besides the properties of SO we have

The *Ehrenfeucht-Fraïssé Theorem*:

Two structures *can be distinguished* by a sentences of quantifier depth  $k$  iff Player I (the spoiler) can force a win in the EF-game of length  $k$ .

or, equivalently

Two structures *cannot be distinguished* by a sentences of quantifier depth  $k$  iff Player II (the duplicator) can force a win in the EF-game of length  $k$ .

We say that two structures are  $k$ -isomorphic if Player II can force a win in the EF-game of length  $k$ .

Furthermore:

$k$ -isomorphism is preserved under the formation of disjoint unions of structures.

Modified versions also hold for Monadic Second Order Logic, but *not* for SO.

### Combining EF-Games and Interpretations

Combining games and interpretations gives a very powerful tool to compute the meaning function of a FO formula in a complex structure by reducing this computation to simpler structures.

If  $G$  is obtained from graphs  $H_1, H_2$  by applying disjoint unions, Cartesian products, and first order definable transductions  $T_1, T_2$ , say

$$G = T_1(H_1 \times T_2(H_2))$$

then the truth of the formulas of quantifier rank  $k$  in  $G$  is uniquely and effectively determined by the the truth of the formulas of quantifier rank  $k$  which hold in  $H_1$  and  $H_2$ .

This is the *Feferman-Vaught Theorem*. It allows us to compute the meaning function for FO-formulas (or MSO-formulas) of composite structures by reducing its computation to the meaning functions of the formulas on the components. I have surveyed how to use the Feferman-Vaught Theorem in Computer Science in my paper in [Mak04].

## 6 My Logic Tool Box

So we finally come to the description of the **Logic Tool Box** I would like to give to our students. Tools to do what, you will ask. Tools to think *rigorously* in order to approach the disciplines of programming and information processing, tools to model accurately new artefacts, as they occur, tools to grasp the scope

of abstraction and modularity. Our students are not logicians. Logic per se is not their main interest. Logic is for Computer Science what Hygienics is to Medicine. They should learn **rigorous informal reasoning before** they learn to model this kind of reasoning as *formalized proof sequences*. Needless to say that each tool comes with a **required skill** how to use it.

My Logic Tool Box contains:

#### **Modeling tools:**

- Basic set construction principles;
- Inductive definitions;
- Proofs by induction;
- Basic cardinality arguments.

#### **Logic tools:**

- Propositional Logic and its axiomatization.
- Second Order Logic as the main formalism to express properties of the modeled artefacts.
- The semantic notion of logical consequence and validity.
- Validity over finite structures.
- Quantifier manipulation rules.
- Skolem functions.
- The Fundamental Property of Transductions and Interpretations.
- First Order Logic as an amenable fragment of Second Order Logic.
- The *Ehrenfeucht-Fraïssé Theorem* and its refinements.
- The Feferman-Vaught Theorem and its variations.

We said before that the Completeness Theorem for First Order Logic holds only, if we define validity over all First Order Structures. For Second Order Logic one would have to explain the difference between Henkin's notion of validity and standard second order quantification. Just stating the Completeness Theorem for First Order Logic misleads the student, and explaining its true subtleties may be beyond the undergraduate level.

#### **Where these tools work**

I have chosen the Logic Tool Box with a view on the courses our student has to take during his undergraduate studies. Ideally, the course I have in mind, **Sets and Logic for Computer Science**, should be taught in the second or third semester. The student should have studied already **Discrete Mathematics** and **Algorithms and Data Structures**, so the teacher can rely on the intuition of the students, and the examples developed in these courses. The course should play the same role as the course Number Systems used to play when it was still customary to teach it, cf. [Fef64],

The modeling skills taught in our course should help him in the following (usually compulsory) courses:

- Automata and Formal Languages
- Introduction to Computability

- Database Systems
- Graph Algorithms
- Principles of Programming Languages
- Computer Architecture
- Introduction to Artificial Intelligence

The more specialized topics of Logic should be taught in advanced courses: A course **Advanced Topics in Logic** could have three parts, covering the Completeness and Incompleteness Theorem, Ordinals and Termination, and Temporal and Modal Logic. Other topics belong there where they are really used, in the courses on **Verification**, **Automated Theorem Proving**, **Principles of Logic Programming**, **Database Theory**, **Functional Programming** and so forth.

## 7 What was omitted?

I have not included in my discussion what is called in the standard classification Non-classical Logics. These logics can also be modeled using set-theoretic tools, and indeed they are. When they find applications to Computer Science, as Temporal Logic [MP95] or the Logic of Knowledge, [FHMV95], they also find their way into more advanced courses.

I have not included in my discussion two classical concerns of the debate around the Foundations of Mathematics and Computer Science: Epistemology and degrees of constructivism. A delightful and insightful presentation and discussion of these matters from a contemporary point of view can be found in [Sha00]. Although I tend to be a Platonist, viewing mathematical concepts as *real*, I am aware of the difficulties inherent in this position, cf. [Mad90, Mad98]. I am also aware of the social and cultural mechanisms at work which strongly influence how science evolves, cf. [Wil81]. However, I strongly object to the arguments which take the social and cultural mechanisms at work as a justification for the erroneous claim that scientific truth is purely social and cultural.

From a more pragmatic point of view I tend to be a Formalist, viewing the observable part of the mathematical and logical enterprise as happening on (virtual) paper written with (virtual) pencils. Concerning the degrees of constructivism I subscribe to, I only want to remark that often *non-constructive* is confused with *lack of detail*. The axiom of choice is an example of *lack of detail*. I assume that the choice function exists and I want to proceed from there, filling in the details (implementation) later, or leaving them to others. Software engineering always proceeds like this and is not considered non-constructive even by the most extreme constructivists. Using a cardinality argument to prove the existence of, say, transcendental numbers, expander graphs or other combinatorial objects, is considered non-constructive, but can be explained in the same way.

Our students, however, should rather follow the advise of the Rabbinic Sages, who admonish us not to study Kabbala (Jewish Mysticism) before the mature age of forty years and before serious exposure to the more down-to-earth matters of Talmud and Torah. Our students should view the Philosophy of Mathematics

and of Computer Science as something to be left for later. Children do not question linguistic principles before they learn their first language. Scientists should not question Science before they master the craft.

### Acknowledgments

I would like to thank N. Francez, D. Giorgetti, S. Halevy and D. Hay for stimulating discussions and suggestions about how to teach Logic to Computer Science students.

I would like to thank the Trade Union of University Professors (Irgun HaSegel) of Israel for giving me time to prepare this paper. At the moment of completion we were in the forth week of our teaching strike.

### References

- [BF85] J. Barwise and S. Feferman, editors. *Model-Theoretic Logics*. Perspectives in Mathematical Logic. Springer Verlag, 1985.
- [BG08] A. Blass and Y. Gurevich. Why sets? In A. Avron, N. Dershowitz, and A. Rabinowich, editors, *Pillars of Computer Science: Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *Lecture Notes in Computer Science*, page In press. Springer, 2008.
- [BGG97] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer-Verlag, 1997.
- [Bool98a] G. Boolos. The consistency of Frege’s “foundations of arithmetic”. In *Logic, Logic, Logic*, pages 182–201. Harvard University Press, 1998.
- [Bool98b] G. Boolos. *Logic, Logic, Logic*. Harvard University Press, 1998.
- [Bool98c] G. Boolos. On the proof of Frege’s theorem. In *Logic, Logic, Logic*, pages 275–90. Harvard University Press, 1998.
- [Bor05] R. Bornat. *Proof and Disproof in Formal Logic*. Number 2 in Oxford Texts in Logic. Oxford University Press, 2005.
- [Bur05] J.P. Burgess. *Fixing Frege*. Princeton University Press, 2005.
- [Bus98] S. Buss, editor. *Handbook of Proof Theory*, volume 137 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science Publishers, 1998.
- [EF95] H.D. Ebbinghaus and J. Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer, 1995.
- [Fef64] S. Feferman. *The number systems: foundations of algebra and analysis*. Addison-Wesley, 1964.
- [FHMV95] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.
- [Fin02] K. Fine. *The Limits of Abstraction*. Oxford University Press, 2002.
- [GKL<sup>+</sup>07] E. Grädel, P. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Vardi, Y. Venema, and S. Weinstein. *Finite Model Theory and its Applications*. Springer, 2007.
- [Gur88] Y. Gurevich. Logic and the challenge of computer science. In E. Börger, editor, *Trends in Theoretical Computer Science*, Principles of Computer Science Series, chapter 1. Computer Science Press, 1988.
- [HA28] D. Hilbert and W. Ackermann. *Grundzuge der theoretischen Logik*. Springer, 1928.
- [HA49] D. Hilbert and W. Ackermann. *Grundzuge der theoretischen Logik, 3rd edition*. Springer, 1949.

- [HA50] D. Hilbert and W. Ackermann. *Principles of Mathematical Logic*. Chelsea Publishing Company, 1950.
- [Ken73] H.C. Kennedy. What Russel learned from Peano. *Notre Dame Journal of Formal Logic*, 14:3:367–372, 1973.
- [Lan99] E. Landau. *Die Grundlagen der Analysis*. American Mathematical Society, 1999.
- [Lib04] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.
- [Mad90] P. Maddy. *Realisms in Mathematics*. Oxford University Press, 1990.
- [Mad98] P. Maddy. *Naturalisms in Mathematics*. Oxford University Press, 1998.
- [Mak84] J.A. Makowsky. Model theoretic issues in theoretical computer science, part I: Relational databases and abstract data types. In G. Lolli and al., editors, *Logic Colloquium '82*, Studies in Logic, pages 303–343. North Holland, 1984.
- [Mak92] J.A. Makowsky. Model theory and computer science: An appetizer. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, chapter I.6. Oxford University Press, 1992.
- [Mak04] J.A. Makowsky. Algorithmic uses of the Feferman-Vaught theorem. *Annals of Pure and Applied Logic*, 126:1–3, 2004.
- [MP95] Z. Manna and A. Pnueli. *Temporal verification of reactive systems*. Springer, 1995.
- [Pap94] C. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.
- [Sha00] S. Shapiro. *Thinking about Mathematics*. Oxford University Press, 2000.
- [Sho67] J. Shoenfield. *Mathematical Logic*. Addison-Wesley Series in Logic. Addison-Wesley, 1967.
- [Wil81] R.L. Wilder. *Mathematics as a Cultural System*. Pergamon Press, 1981.

# From Program Verification to Certified Binaries\*

## The Quest for the Holy Grail of Software Engineering

Angelos Manousaridis, Michalis A. Papakyriakou, and Nikolaos S. Papaspyrou

National Technical University of Athens  
School of Electrical and Computer Engineering  
Software Engineering Laboratory  
Polytechnioupoli, 15780 Zografou, Athens, Greece  
`{amanous, mpapaky, nickie}@softlab.ntua.gr`

**Abstract.** The long tradition of formal program verification and the more recent frameworks for proof-carrying code share a common goal: the construction of certified software. In this paper, mainly through a simple motivating example, we describe our vision of a complete hybrid system that combines the two approaches. We discuss the feasibility of such an ambitious project and report on progress made so far.

**Key words:** Formal methods, type systems and type theory, certified code, proof-preserving compilation.

## 1 Introduction

*Program verification* aims at formally proving the correctness of a computer program, with respect to a certain formal specification or property. As a research field of computer science, program verification is well into the fourth decade of its existence. However, it can hardly be argued that it is often adopted in practice by software engineers, except for verifying mission-critical systems. For the vast majority of software systems, quality assurance amounts to dynamic testing, which unfortunately can produce no guarantees. In this respect, *software engineering*, defined as “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software” [1], is still very far from reaching the maturity of other branches of engineering.

Several formal logics, their majority greatly influenced by Hoare Logic [2], have been proposed in combination with programming languages as the vehicles for program verification [3]. Most of the proposed approaches advocate a clear separation between the language in which specifications are given (e.g. first-order predicate logic), the programming language, and the methodology and the tools—if any—that support the construction of proofs.

---

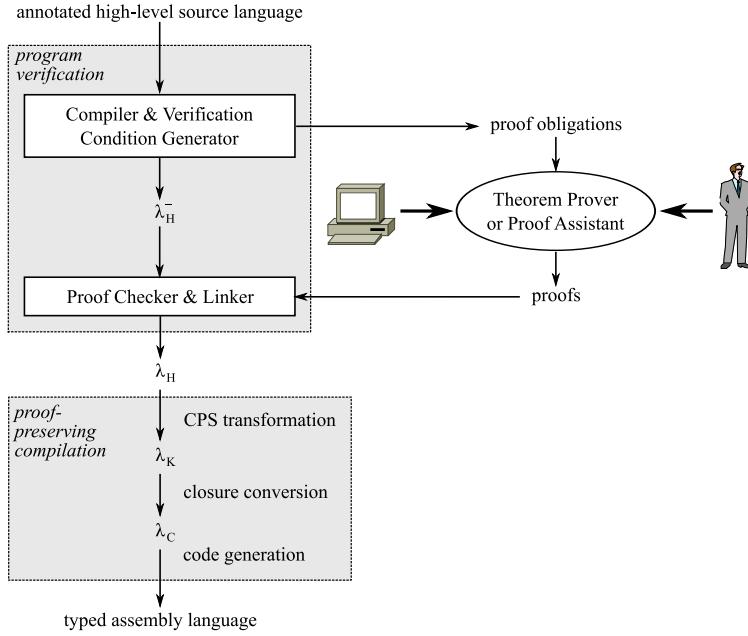
\* This work has been funded by the research programme ΠΕΝΕΔ (grant number 03ΕΔ 330), cofinanced by public expenditure (75% by the European Social Fund and 25% by the Greek Ministry of Education, General Secretariat of Research and Technology) and by the private sector, under measure 8.3 of the Operational Programme “Competitiveness” in the European Union’s 3rd Community Support Framework.

*Proof-carrying code* [4, 5] and *foundational proof-carrying code* [6] are general frameworks, expressing a relatively modern philosophy towards the verification of low-level (e.g. machine language) programs. A *certified binary* is a value (a function, a data structure, or a combination of both) together with a proof that the value satisfies a given specification. Certified binaries are essential in modern distributed computer systems, where executable code is transferred among computing devices that do not necessarily trust one another. The recipient of a certified binary does not need to trust the producer: the proof can be mechanically checked and, once found valid, it is known beyond doubt that the binary conforms to its specification. Existing compilers that produce certified binaries have mostly focused on simple memory and control-flow safety properties. Although the two frameworks are general enough to express arbitrary program properties, in the general case not much is known on how to construct certified binaries or how to automatically generate them from high-level source programs.

More recently, type-theoretic frameworks for constructing, composing and reasoning about certified software have been proposed [7, 8], based on the “formulae as types” principle [9]. The type-theoretic approach provides an embedding of logic in the type system of the programming language: program properties are encoded in types and proof checking is reduced to type checking. In analogy to a type-preserving compiler, which uses a typed intermediate [10] and a typed assembly language [11] and propagates type information from the source program down to the lower-level equivalent programs, a type-theoretic framework for certified binaries can support *proof-preserving compilation*. Provided that a common logic (type language) is used for expressing properties and proofs, from the source language to the target language, certified binaries can be generated by compiling previously verified source programs. This is important, because high-level programs are easier to reason about than low-level programs.

Still, in a type-theoretic framework such as that proposed by Shao *et al.* [7], constructing a proof of correctness even for a small program, written in an appropriate high-level source language, is far from simple. As the logic is part of the programming language (more accurately, part of the type language), the proof must be *embedded* in the code and has to be constructed at the same time with it. Although this has long been proposed as the “right way” to produce software [12, 13], it is not popular with programmers, who generally prefer to write down their algorithm first and then (if ever) prove its correctness. Furthermore, if something in the specification changes, the code has to change as well and, sometimes, the modifications can be substantial in size even if the code’s operational behaviour does not change.

Due to the complexity of the type languages used in the type theoretic frameworks that support proof-preserving compilation, type inference (or proof inference) is in general undecidable. The type system of the source language cannot do miracles. It can therefore be argued that, although the embedding of logic in the programming language is appropriate for the lower-level languages used by the compiler, it is not appropriate for the source language, in which the task of constructing the proof is—more or less—the programmers’ responsibility.



**Fig. 1.** Overview of a hybrid system for generating certified binaries.

In this paper, we register our dream of a hybrid system (half based on traditional program verification and half type-theoretic). Although similar dreams must be common among computer scientists who advocate program verification and proof-carrying code, it seems that they are still rather far from becoming reality. We outline our experience—limited, so far—in building such a system. If it turns out that, with the assistance of appropriate program verification tools, programmers are able to prove the correctness of their programs (we want to be optimistic and believe this hypothesis to be true), such a system can be thought of as the Holy Grail of software engineering.

## 2 A Hybrid System for Generating Certified Binaries

A hybrid system for generating certified binaries from annotated source programs can be structured in two layers, as depicted in Fig. 1. First, a “programmer-friendly” program verification layer assists programmers in constructing valid proofs for their source programs according to the specifications that they have set. In this layer, specifications and proofs are separate from the actual code and an ordinary, general-purpose programming language can be used.

The program verification layer can follow the methodology suggested in the work of Filliatre *et al.* related to the *Why* software verification platform [14, 15]. The source code, written in any from a variety of languages, must be annotated

with specifications (preconditions, postconditions, invariants, etc.) in some appropriate logic. It is then given to a tool serving two purposes: (i) to compile the source code into a lower-level intermediate language  $\lambda_H^-$ ; and (ii) to generate proof obligations that must be proved, in order to verify the correctness of the source code w.r.t. its specifications.

The language  $\lambda_H^-$  can be thought of as a typed intermediate language, such as  $\lambda_H$  in the paper by Shao *et al.* [7], with *some proofs missing*. The missing proofs are exactly those corresponding to the generated proof obligations. A variety of tools (from automatic theorem provers to human-driven proof assistants) can be used to discharge the proof obligations and generate the missing proofs. Subsequently, the intermediate program in  $\lambda_H^-$  can be automatically “linked” with the constructed proofs, resulting in a  $\lambda_H$  program. An additional type/proof checking step can be performed, to ensure the correctness of the “linked” program.

The second layer of the hybrid system consists of a type/proof preserving compiler, which transforms the program in  $\lambda_H$  and produces a certified binary. This compiler performs type/proof preserving program transformations that produce progressively lower-level code. Shao *et al.* have shown how to perform type preserving CPS transformation and closure conversion on  $\lambda_H$  (and call the intermediate languages  $\lambda_K$  and  $\lambda_C$  respectively). One or more type-preserving “code generation” steps are required to obtain a certified binary in the form of a (machine dependent or independent) typed assembly language.

It should be noted that the *trusted computing base*, i.e. the piece of software that the recipient of a certified binary must blindly trust (not shown in Fig. 1), consists only of a type/proof checker for the typed assembly language and a (type/proof erasing) translator to native assembly language. Both pieces of software are of moderate size and relatively easy to build.

### 3 A Motivating Example

In this section we present a small case study: the construction of a certified binary from a C function annotated with its specification. The example is intentionally chosen to be very simple, so that self-contained equivalent programs in  $\lambda_H$  and  $\lambda_K$  can fit in this paper.

Consider a function `root` that calculates the *integer square root* of an integer number  $n$ , i.e. the greatest integer  $r$  with the property  $r^2 \leq n$ . A naïve C program that implements this function is the following:

```
int root (int n) {
    int y = 0;
    while ((y+1)*(y+1) <= n) y++;
    return y;
}
```

Following the notation used by the *Why* verification platform and the verification tool *Caduceus* for C programs [14, 15], the same program annotated with the function’s pre- and postcondition and the loop invariant is given in Fig. 2.

```

//@ predicate leRoot(int r, int x) { r >= 0 && r*r <= x }
//@ predicate isRoot(int r, int x) { leRoot(r, x) && (r+1)*(r+1) > x }

/*@ requires n >= 0
  @ ensures isRoot(\result, n)
  */
int root (int n) {
    int y = 0;
    //@ invariant leRoot(y, n)
    while ((y+1)*(y+1) <= n) y++;
    return y;
}

```

**Fig. 2.** The example program, annotated with its specification.

```

root  ▷  ∀n:Z. ∀n*: (n ≥ 0). sint n → ∃x:Z. ∃x*: isRoot x n. sint x
=  poly n:Z. poly n*: (n ≥ 0). lambda n:sint n.
  (fix loop: ∀y:Z. ∀y*: leRoot y n. sint y → ∃x:Z. ∃x*: isRoot x n. sint x.
   poly y:Z. poly y*: leRoot y n. lambda y:sint y.
   if [♣,♣] ((y + cint [1]))2 > n,
      p1* . pack (y, pack (♣, y) as ∃y*: isRoot y n. sint y) as
      ∃x:Z. ∃x*: isRoot x n. sint x,
      p2* . loop [y + 1] [♣] (y + cint [1]))
  [0] [♣] cint [0]

```

**Fig. 3.** The  $\lambda_H^-$  term with the missing proofs that correspond to proof obligations (♣).

The program in Fig. 2 is subsequently compiled to the  $\lambda_H^-$  program of Fig. 3. Readers not familiar with the syntax of  $\lambda_H$  will probably find it hard to decipher the code. However, two things are obvious. First, the specifications in Fig. 2 have been translated to the types that are embedded in the term of Fig. 3. For instance, the type of `root` itself contains a direct translation of the function's pre- and postconditions. Second, there are five parts of this code, marked with the symbol ♣, that are missing. The first of these five is the predicate associated with the condition of the `if` expression. The remaining four are proofs that have to be constructed externally. Four proof obligations are therefore produced by the verification condition generator.

The proof obligations must now be discharged, either by an automatic theorem prover or by a proof assistant. Suppose that the second alternative is used and the human prover provides the code given in Fig. 4 for the Coq<sup>1</sup> proof assistant [16]. The missing parts of Fig. 3 can then be filled in, resulting in the  $\lambda_H$  program of Fig. 5.

---

<sup>1</sup> Coq uses the *Calculus of Inductive Constructions* (CIC) as its type language and, for this reason, Coq proofs can be directly embedded in  $\lambda_H$ , which is also based on CIC. Other theorem provers or proof assistants can be used instead, but the resulting proofs would then have to be translated to CIC. It is worth mentioning that all proof obligations were easily proved automatically (by `auto`) in Isabelle/HOL.

```

Definition leRoot (r : Z) (x : Z) := (r >= 0 /\ r*r <= x)%Z.
Definition isRoot (r : Z) (x : Z) := leRoot r x /\ ((r+1)*(r+1) > x)%Z.

Definition decidable (P : Prop) (b : bool) := if b then P else ~P.

Lemma geDecidablePrf: forall n m : Z, decidable (n >= m)%Z (Zge_bool n m).
intros; unfold decidable, Zge, Zge_bool;
case (Zcompare n m); [ discriminate | auto | discriminate ].

Lemma gtDecidablePrf: forall n m : Z, decidable (n > m)%Z (Zgt_bool n m).
intros; unfold decidable, Zgt, Zgt_bool;
case (Zcompare n m); [ discriminate | discriminate | auto ].

Lemma Z_ge_refl: forall n : Z, (n >= n)%Z.
auto with zarith.

Lemma Zplus_ge_compat:
forall n m p q : Z, (n >= m -> p >= q -> n + p >= m + q)%Z.
intros n m p q; intros H1 H2; apply Zle_ge; apply Zplus_le_compat;
apply Zge_le; assumption.

```

**Fig. 4.** Coq code, useful in discharging the proof obligations.

```

root  ▷  ∀n:Z. ∀n*: (n ≥ 0). sint n → ∃x:Z. ∃x*:isRoot x n. sint x
=  poly n:Z. poly n*: (n ≥ 0). lambda n:sint n.
  (fix loop: ∀y:Z. ∀y*:leRoot y n. sint y → ∃x:Z. ∃x*:isRoot x n. sint x.
   poly y:Z. poly y*:leRoot y n. lambda y:sint y.
   if [decidable((y + 1)2 > n),gtDecidablePrf (y + 1)2 n] (
    (y + cint[1])2 > n,
    p1* . pack(y, pack(conj y* p1*, y) as ∃y*:isRoot y n. sint y) as
    ∃x:Z. ∃x*:isRoot x n. sint x,
    p2*. loop [y + 1] [conj(Zplus.ge_compat y 0 1 0
      (proj1 y*)
      (geDecidablePrf 1 0))
      (Znot_gt_le(y + 1)2 n p2*)] (y + cint[1]))
   [0] [conj(Z_ge_refl 0) (Zge_le n 0 n*)] cint[0]

```

**Fig. 5.** The  $\lambda_H$  term with the “linked” proofs.

Proof-preserving compilation phases can now be applied to the  $\lambda_H$  program. However, after CPS transformation, the size and complexity of the resulting program are too much for the human reader. The  $\lambda_K$  program obtained by the CPS transformation of the  $\lambda_H$  program of Fig. 5 and after some simple optimizations (such as constant propagation and beta contraction) is given in Fig. 6 at the end of this paper. To increase readability, the types of continuation parameters have been omitted from the  $\lambda_K$  program. In subsequent phases, still lower-level programs are obtained. The corresponding  $\lambda_C$  program, after a naïve closure conversion, is a few hundred lines long when expressed in the same textual format. To obtain an efficient implementation, it is essential to invent and implement proof-preserving compiler optimizations and to find a compact representation for proofs.

## 4 Conclusion

The realization of a complete hybrid system for constructing certified binaries requires the implementation of the system's two main software layers. Both for program verification and for type-based proof-preserving compilation, there is still a long way to go. However, in order to exploit the feasibility of such a system, we have used existing techniques and tools. To verify high-level source programs and produce proof obligations, a platform such as *Why/Caduceus* can be used. The integration of such a platform with a compiler from the source language to  $\lambda_H^-$  and the implementation of a proof checker and linker are still future work.

So far, we have partially implemented a proof-preserving compiler in OCaml, manipulating programs in the set of languages described by Shao *et al.* [7]. As an implementation of the type language (CIC) we have used the Coq proof assistant, whose source code is freely available. In this way, we can build on Coq's rich set of proof libraries. Our system is incompetent with long and complex source programs. There is much to be done before such an approach to software verification can be applied to real software.

## References

1. IEEE: Standard Glossary of Software Engineering Terminology. IEEE Standard 610.12-1990.
2. Hoare, C.A.R.: An axiomatic basis for computer programming. *Communications of the ACM* **12**(10) (1969) 576–585
3. Cousot, P.: Methods and logics for proving programs. In van Leeuwen, J., ed.: *Formal Models and Semantics*. Volume B of *Handbook of Theoretical Computer Science*. Elsevier Science Publishers B.V., Amsterdam, The Netherlands (1990) 843–993
4. Necula, G.: Proof-carrying code. In: *Proceedings of the 24th ACM Symposium on the Principles of Programming Languages*. (1997) 106–119
5. Necula, G.: Compiling with Proofs. PhD thesis, Carnegie Mellon University (1998)
6. Appel, A.W.: Foundational proof-carrying code. In: *Proceedings 16th IEEE Symposium on Logic in Computer Science*. (2001) 247–258

```
(lambda k. k
  (poly n:Z. lambda k. k
    (poly n*: (n ≥ 0). lambda k. k
      (lambda x_arg:sint n × Kc(∃x:Z. ∃x*:isRoot x n. sint x).
        let n = sel [N_lt_prop 0 2] (x_arg, cnat [0]) in
        let k0 = sel [N_lt_prop 1 2] (x_arg, cnat [1]) in
        (fix loop [y:Z] (k:Kc(∀x*:leRoot y n. sint y → ∃x:Z. ∃x*:isRoot x n. sint x)). k
          (poly y*:leRoot y n. lambda k. k
            (lambda x_arg:sint y × Kc(∃x:Z. ∃x*:isRoot x n. sint x).
              let y = sel [N_lt_prop 0 2] (x_arg, cnat [0]) in
              let k1 = sel [N_lt_prop 1 2] (x_arg, cnat [1]) in
              let z1 = y + cint [1] in
              let z2 = z1 * z1 in
              let z3 = z2 > n in
              if [decidable((y + 1)^2 > n), gtDecidablePrf (y + 1)^2 n] (z3,
                p1*. k1 (pack (y, pack (conj y* p1*, y) as K(∃y*:isRoot y n. sint y)) as
                  K(∃x:Z. ∃x*:isRoot x n. sint x)),
                p2*. loop [y + 1] (lambda k. k [conj (Zplus_ge_compat y 0 1 0 (proj1 y*)
                  (geDecidablePrf 1 0))
                  (Znot_gt_le (y + 1)^2 n p2*)]
                  (lambda k. let z1 = y + cint [1] in k (z1, k1)))))))[0]
                (lambda k. k [conj (Z_ge_refl 0) (Zge_le n 0 n*)] (lambda k. k (cint [0], k0)))))))
              )))))))))
```

**Fig. 6.** The  $\lambda_K$  term, after the proof-preserving CPS transformation and some optimizations.

7. Shao, Z., Trifonov, V., Saha, B., Papaspyrou, N.: A type system for certified binaries. ACM Transactions on Programming Languages and Systems **27**(1) (2005) 1–45
8. Crary, K., Vanderwaart, J.C.: An expressive, scalable type theory for certified code. In: Proceedings of the 7th ACM International Conference on Functional Programming. (2002) 191–205
9. Howard, W.A.: The formulae-as-types notion of constructions. In Seldin, J.P., Hindley, J.R., eds.: To H. B. Curry: Essays on Computation Logic, Lambda Calculus and Formalism. Academic Press, Boston, MA (1980) 479–490
10. Harper, R., Morrisett, G.: Compiling polymorphism using intensional type analysis. In: Proc. 22nd ACM Symp. on Principles of Prog. Lang. (1995) 130–141
11. Morrisett, G., Walker, D., Crary, K., Glew, N.: From System F to typed assembly language. In: Proc. 25th ACM Symp. on Principles of Prog. Lang. (1998) 85–97
12. Dijkstra, E.W.: A Discipline of Programming. Prentice Hall (1976)
13. Gries, D.: The Science of Programming. Springer-Verlag (1981)
14. Filliâtre, J.C.: Why: A multi-language multi-prover verification tool. Research Report 1366, LRI, Université Paris Sud (March 2003)
15. Filliâtre, J.C., Marché, C.: The Why/Krakatoa/Caduceus platform for deductive program verification. In: Computer Aided Verification. Volume 4590 of LNCS. Springer (2007) 173–177
16. The Coq Proof Assistant Reference Manual, URL: <http://coq.inria.fr/>

# Limiting Recursion, FM–representability, and Hypercomputations

Marcin Mostowski

Department of Logic  
Institute of Philosophy, Warsaw University  
[m.mostowski@uw.edu.pl](mailto:m.mostowski@uw.edu.pl)

**Abstract.** We consider various methodologically well motivated notions which appear to be essentially different, but surprisingly capture the same class of relations. These are: the notion of FM–representability (representability in Finite Models), statistical representability, limiting recursion or algorithmic learning with empty data, and decidability by accelerating Turing machines by computations of length  $\omega$ . Finally, we give a new result characterizing FM–representability in poor finite arithmetics, weaker than divisibility arithmetic, but stronger than coprimality arithmetic.

## 1 Introduction

Two classical bounds determined by computations were given in the thirties by *recursive* — relations which can be decided by an algorithmic process, and *recursively enumerable* — relations which can be positively decided by finding proofs. Thirty years later — as a result of our computer experience — another important bound was postulated: *practical computability*. Edmond’s thesis (called also: the feasibility thesis) — which identifies practically computable notions with those PTIME–computable — is still one of the crucial methodological motivations of our research in computational complexity.

However, in about the same time we were led to recognize another important bound falling far outside recursivity. It was motivated by our attempts of computerising some of our intellectual activities, namely learning from finite samples. In his classical paper [3] Mark Gold formulated fundamental ideas of algorithmic learning theory. However in his earlier paper [2] he gave an idea of *limiting recursion*. In the same issue of the Journal of Symbolic Logic [21] Hilary Putnam independently postulated a similar explication for the notion of learnability. As a matter of fact identifying learnability with limiting recursion was Putnam’s idea. Nevertheless this notion seems to be a good explication for the idea of algorithmic learning with no input data. Surprisingly this notion coincides with some other methodologically motivated notions. These are FM–representability (arithmetical expressibility in potentially infinite domains), statistical representability, and decidability by hypercomputations of length  $\omega$ .

Aristotle (in his Physics [1]) about 360 BC noticed the difference between actual and potential infinities. Potentially infinite are collections which are finite

but unbounded, in the sense that they can be always enlarged. Actually infinite are collections containing infinitely many members. More than 2000 years later in his talk Hilbert ([6]) recalled this idea stressing its importance in foundations of mathematics.<sup>1</sup>

Currently the idea is reflected — among others — in difference in ways of thinking in classical and finite model theory. One of the crucial problems in the latter is the question which notions can be expressed when we restrict possible interpretations to finite models. We try — under such restriction — to express various properties of computations, formulae, and some other combinatorial objects. Then we need a general method of determining the class of the notions meaningful in finite models.

As an answer to this problem the author (in [14]) considered the notion of FM-representability. A relation on natural numbers is FM-representable by a given formula if each finite part of its characteristic function is correctly described by this formula in all sufficiently large finite models. In Aristotelian spirit it can be treated as an answer to the question: what can be meaningfully described in a finite but potentially infinite world?

\* \* \*

In this paper we discuss the notion of FM-representability and other equivalent notions. Our research follow the ideas of the paper [14] and research done in collaboration with Konrad Zdanowski ([19], [18], and [27]).

The paper is partially self contained. Only the last part about FM-representability in poor arithmetics is dependent on other papers ([17] and [18]). Giving all the details would force us to repeat all the technical machinery which was very carefully elaborated there.

## 2 Finite arithmetics

We say that  $\mathcal{A}$  is an arithmetical model if it is of the form  $\mathcal{A} = (\mathbb{N}, R_1, \dots, R_s)$ , where  $\mathbb{N}$  is the set of natural numbers and  $R_1, \dots, R_s$  are relations on  $\mathbb{N}$ .

We consider finite initial fragments of arithmetical models. Namely, for every  $n \in \mathbb{N} - \{0\}$ , by  $\mathcal{A}_n$  we denote the model  $\mathcal{A}_n = (\{0, \dots, n-1\}, R_1^n, \dots, R_s^n)$ ,

---

<sup>1</sup> Hilbert claimed there that lack of actual infinity is a characteristic feature of finitistic mathematics in opposition to ideal mathematics which should be grounded by representing it in the form of an axiomatic theory. His explanation of an idea of finitistic mathematics — from the point of our current knowledge — allows various nonequivalent explications. Later it was interpreted according to Gödel's results by identifying *finitistic* with expressible in primitively recursive arithmetic (see [24]). Nevertheless it would be interesting to investigate Hilbert's idea just by taking lack of actual infinity as the defining feature of finitistic mathematics considered as the part of mathematics which is well defined and free of inconsistent intuitions. In this way finitistic mathematics would be identified with mathematics restricted to FM-representable notions.

where  $R_i^n$  is the restriction of  $R_i$  to the set  $\{0, \dots, n-1\}$ . The family of models  $\{\mathcal{A}_n : n = 1, 2, 3, \dots\}$  is denoted by  $FM(\mathcal{A})$ .

The most common arithmetical models are defined as algebraic structures, e.g. with addition and multiplication. However — as a rule — initial segments of natural numbers are not closed under arithmetical operations.<sup>2</sup> Therefore we treat e.g. addition and multiplication as ternary relations.

Let  $\varphi(x_1, \dots, x_p)$  be an arithmetical formula and  $b_1, \dots, b_p \in \mathbb{N}$ . We say that  $\varphi$  is satisfied by  $b_1, \dots, b_p$  in all sufficiently large finite models of  $FM(\mathcal{A})$ , what is denoted by  $FM(\mathcal{A}) \models_{sl} \varphi[b_1, \dots, b_p]$ , if there is  $k \in \mathbb{N}$  such that for all  $n \geq k$   $\mathcal{A}_n \models \varphi[b_1, \dots, b_p]$ . When no ambiguity arises we will write  $\models_{sl} \varphi[b_1, \dots, b_p]$  instead of  $FM(\mathcal{N}) \models_{sl} \varphi(b_1, \dots, b_p)$ , where  $\mathcal{N} = (\mathbb{N}, +, \times)$ .

Traditionally an arithmetical model  $\mathcal{A}$  is called an arithmetical domain — with possible qualifications, e.g. addition domain, multiplication domain, divisibility domain, or coprimality domain, with obvious meaning pointing at the relations considered. The corresponding class of finite models  $FM(\mathcal{A})$  is called FM-domain or a finite arithmetic possibly with appropriate qualifications.

The research area devoted to study logical properties of FM-domains is also called finite arithmetic. A report of our state of knowledge in this area can be found in [11].

### 3 Representing concepts in finite models

One of the main questions related to finite arithmetics is the problem of FM-representability in a given FM-domain.

Let  $\varphi(x_1, \dots, x_n)$  be a formula and  $S \subseteq \mathbb{N}^n$ . We say that  $\varphi(x_1, \dots, x_n)$  FM-represents  $S$  in  $FM(\mathcal{A})$  if for all  $a_1, \dots, a_n \in \mathbb{N}$  the following two conditions are satisfied:

1. if  $S(a_1, \dots, a_n)$  then  $FM(\mathcal{A}) \models_{sl} \varphi(a_1, \dots, a_n)$ ,
2. if  $\neg S(a_1, \dots, a_n)$  then  $FM(\mathcal{A}) \models_{sl} \neg \varphi(a_1, \dots, a_n)$ ,

The idea of this definition is that a formula  $\varphi$  FM-represents a relation between natural numbers if for any given finite fragment of this relation  $\varphi$  correctly describes this fragment in all sufficiently large finite initial segments of  $\mathcal{A}$  (for both positive and negative cases). We say that a relation is FM-representable in a given FM-domain if it is FM-representable in this FM-domain by some formula.

This notion and its basic properties was presented in the paper [14]. Originally it was motivated by studying truth definitions in finite models (FM truth definitions).<sup>3</sup> Tarski's undefinability of truth theorem (see [25]) essentially re-

---

<sup>2</sup> In earlier papers (see e.g. [14]) we assumed that operations take as their value the greatest element MAX of the initial segment when they are not defined. However this approach does not make things clearer.

<sup>3</sup> See [14] and the later papers [15, 7]. Leszek Kołodziejczyk successfully applied the method for computational complexity questions in [8, 9]. See also recent discussion of the idea in [11].

quires expressibility of some syntactical relations in the model considered. Application of Tarski's idea in finite models also requires expressibility of some relations in finite models. FM-representability just gives a proper notion of expressibility for finite models.

Let us observe that except for the trivial case initial segments of natural numbers are not closed on pairing function. Therefore we cannot restrict our attention to sets — unary relations. However some arguments can be given only for the unary case when no essential difference will follow from considering arbitrary arities.

### 3.1 FM representability theorem

In this section we consider the problem of characterizing of FM-representable relations. We begin with recalling classical characterizations of  $\Delta_2^0$  arithmetical relations (for the proof see e.g. [22] or [23]).

**Theorem 1 (The characterisation of  $\Delta_2^0$  arithmetical relations).** *Let  $R$  be a relation on natural numbers. Then the following are equivalent:*

1.  *$R$  is recursive with recursively enumerable oracle; ( – in terms of oracle machines)*
2.  *$R$  is of degree  $\leq \mathbf{0}'$ ; ( – in terms of Turing degrees)*
3.  *$R$  is recursive in the limit (see [2]), in the other words there is a recursive sequence of recursive relations  $S_0, S_1, S_2, \dots$  such that  $R = \lim_{n \rightarrow \infty} S_n$ ; ( – in terms of limits)*
4.  *$R$  is  $\Delta_2^0$  in arithmetical hierarchy. ( – in terms of arithmetical definability)*

The equivalence with condition 3 is also known as the limit lemma.

$$R = \lim_{n \rightarrow \infty} S_n$$

means that the limit of the characteristic functions  $S_n$  exists, for all arguments, and it is 1 exactly when  $R$  holds for these arguments. In other words it is equivalent to the conjunction of the following two conditions:

$$\forall a_1, \dots, a_k (R(a_1, \dots, a_k) \equiv \exists m \forall n > m \ S_n(a_1, \dots, a_k))$$

and

$$\forall a_1, \dots, a_k (\neg R(a_1, \dots, a_k) \equiv \exists m \forall n > m \ \neg S_n(a_1, \dots, a_k)).$$

Before stating the FM-representability theorem we consider its easier version — namely FM-representability of recursively enumerable relations.

**Proposition 1.** *Each recursively enumerable relation is FM-representable.*

*Proof.* Let us consider Kleene  $T$  predicate such that  $T(e, n, c)$  if and only if  $e$  is a number of a Turing machine and  $c$  is a code of a finished computation of this machine with the input  $n$ . It is known that a proper arithmetical formula

can be chosen in such a way that all quantifiers are bounded by “ $< c$ ”. If  $R$  is recursively enumerable then  $R = W_e$ , for some  $e$ , where  $W_e$  is the set of inputs for which  $e$  halts. Then  $R$  is FM-represented by the formula  $\exists c \ T(\bar{e}, n, c)$ .<sup>4</sup>

Now we are ready to characterize just FM-representable relations.

**Theorem 2 (The FM-representability theorem, M. Mostowski, [14]).**

*Let  $R$  be a relation on natural numbers. Then all the conditions of theorem 1 are equivalent to the following:  $R$  is FM-representable (in  $\text{FM}(\mathbb{N}, \times, +)$ ).*

*Proof.* Let us observe that both cases positive and negative for FM-representability are defined by  $\Sigma_2^0$  formulae. It follows that all FM-representable relations are  $\Delta_2^0$ -definable.

Following [14] we will show that all relations recognized by oracle Turing machines with recursively enumerable oracles are FM-representable. Let  $M_1$  be an oracle deterministic Turing machine recognising  $n$ -ary relation<sup>5</sup>  $R$  and using a recursively enumerable oracle  $S$ . Moreover,  $S$  is given by a deterministic Turing machine  $M_2$  such that  $M_2$  halts on a given input  $n$  exactly when  $n \in S$ . Our FM-representing formula  $\varphi(x_1, \dots, x_n)$  for  $R$  can be formulated as follows:

*there is an accepting computation of  $M_1$  on input  $x_1, \dots, x_n$  such that each oracle question  $y$  is answered positively exactly when  $\psi(y)$ ,*

where  $\psi(y)$  is an arithmetical formula FM-representing the set  $S$ , it exists by proposition 1.

For each given input  $a_1, \dots, a_n$  we can find a model of size sufficient to contain the unique  $M_1$ -computation  $c$  and all required witnesses for oracle questions put by  $c$  for accepting them correctly as members of  $S$  by the formula  $\psi(y)$ .

All the claims up to now have assumed that we consider only representability by first order arithmetical formulae and FM-domain  $\mathcal{N}$ . In [14] we have considered much stronger logic — namely finite order logic. However it was also observed there that taking any logic stronger than first order logic we do not obtain more FM-representable relations, provided the logic has decidable “truth in a finite model” relation. A logic  $L$  has decidable “truth in a finite model” relation if the relation “ $M \models \varphi$ ” is recursive for arguments: a finite model  $M$  and  $L$ -formula  $\varphi$ .

**Theorem 3.** *Let  $\mathcal{A}$  be a model on natural numbers having all relations recursive and  $L$  be a logic with decidable “truth in a finite model” relation. Then the relations FM-representable in  $\text{FM}(\mathcal{A})$  by  $L$ -formulae are  $\Delta_2^0$  arithmetically definable.*

---

<sup>4</sup> Let us observe that by the Matiyasevich theorem recursively enumerable sets are exactly sets definable by purely existential arithmetical formulae. This gives slightly easier argument.

<sup>5</sup> In finite models we cannot restrict to sets of natural numbers because finite models are not closed on pairing function.

### 3.2 Statistical representability

The notion of statistical representability was proposed by Konrad Zdanowski (see [19] and [27]) with the intention of comparing it with FM-representability.

By  $\mu_n(\varphi(a_1, \dots, a_k))$  we mean the density of the models satisfying  $\varphi(a_1, \dots, a_k)$  between models of size not greater than  $n$ , that is

$$\mu_n(\varphi(a_1, \dots, a_k)) = \frac{1}{n} \text{card}\{s : 1 \leq s \leq n \text{ and } \mathcal{N}_s \models \varphi(a_1, \dots, a_k)\}.$$

A relation  $R \subseteq \mathbb{N}^k$  is statistically representable if there is arithmetical formula  $\varphi(x_1, \dots, x_k)$  such that for each  $a_1, \dots, a_k \in \mathbb{N}$ :

- the limit  $\mu(\varphi(a_1, \dots, a_k)) = \lim_{n \rightarrow \infty} \mu_n(\varphi(a_1, \dots, a_k))$  exists;
- if  $R(a_1, \dots, a_k)$  then  $\mu(\varphi(a_1, \dots, a_k)) > \frac{1}{2}$ ;
- if  $\neg R(a_1, \dots, a_k)$  then  $\mu(\varphi(a_1, \dots, a_k)) < \frac{1}{2}$ .

**Theorem 4. The statistical representability theorem, K. Zdanowski, [19] and [27]**

*Let  $R$  be a relation on natural numbers. Then all the conditions of theorem 1 and theorem 2 are equivalent to the following:  $R$  is statistically representable.*

*Proof.* By the definition each FM-representing formula statistically represents the same relation. On the other hand if a formula  $\varphi(x_1, \dots, x_k)$  statistically represents a relation  $R$  then a formula  $\psi(x_1, \dots, x_k)$  saying that “majority of  $x$ -s satisfy  $0 < x \wedge \varphi^{\leq x}(x_1, \dots, x_k)$ ” FM-represents the same relation, where the formula  $\varphi^{\leq x}(x_1, \dots, x_k)$  is obtained by bounding all quantifiers in  $\varphi(x_1, \dots, x_k)$  by the condition  $\leq x$ . Obviously the logic with majority quantifier has decidable “truth in a finite model” relation. Then by theorem 3 the relation  $R$  is FM-representable.

### 3.3 Learnable relations

In his currently classical paper Gold [3] considers learning algorithms supplying a natural framework for algorithmic modelling the phenomenon of learning grammar of a language given by finite samples. The algorithm in his sense has to identify on the basis of finite samples a grammar in such a way that the identification has to stabilise on a correct grammar after finite number of guesses. In a similar way we can think of learning mathematical notions. However in this case no empirical data are required. What we need is more and more work. So at each stage  $t$  we have some answer but after finitely many stages the answer is stabilising.

We say that  $\mathcal{A}$  is a *mathematical learning algorithm* for a relation  $R \subseteq \mathbb{N}^k$  if

- $\mathcal{A}$  works with inputs  $a_1, \dots, a_k, t \in \mathbb{N}$ ,
- $R(a_1, \dots, a_k)$  if and only if there is  $s$  such that for all  $t > s$   $\mathcal{A}$  for the input  $a_1, \dots, a_k, t$  answers “YES”,

- $\neg R(a_1, \dots, a_k)$  if and only if there is  $s$  such that for all  $t > s$   $\mathcal{A}$  for the input  $a_1, \dots, a_k, t$  answers “NO”.

We say that a relation  $R$  is mathematically learnable if there is a mathematical learning algorithm for  $R$ .

**Theorem 5 (Mathematical learnability theorem).**

*Let  $R$  be a relation on natural numbers. Then all the conditions of theorem 1 and theorems 2, 4 are equivalent to the following:  $R$  is mathematically learnable.*

*Proof.* Let us observe that a mathematical learning algorithm  $\mathcal{A}$  for  $R$  gives a recursive sequence of relations  $S_0, S_1, S_2, \dots$  such that

$$R = \lim_{n \rightarrow \infty} S_n,$$

where  $S_n$  is the relation computed when we fix  $t = n$ .

### 3.4 Relations decidable by Zeno machines

Now we will consider a characterisation of  $\Delta_2^0$  arithmetical relations in terms of hypercomputations. Currently hypercomputations have vast literature (see e.g. [5], [4], [20]). In spite of our idea of potential infinity hypercomputations essentially use actual infinity.<sup>6</sup> We restrict our interest here to so called Zeno machines which are the simplest (hyper)computing devices.

A Zeno machine — called also accelerating Turing machine — is defined in the same way as the simplest Turing machine. It has an infinite tape consisting of  $\omega$  cells. The first cell (0 cell) is used for giving outputs: 0 or 1. It is accelerating because it carries out each next step of computation two times quicker than the previous one. So its computation consists of  $\omega$  steps. If the first step is done in 1 second then all the computation will be finished after 2 seconds. After that we look at the first cell, if it contains 1 then the answer is “YES”, and if it contains 0 then the answer is “NO”. When the content of the first cell stabilises after finitely many moves then the situation is clear — it contains the stabilised character. Otherwise, when this value is changed infinitely many times, Potgieter [20]<sup>7</sup> says that it does not halt, but Hamkins [4] says that it takes the supremum value = 1. These two approaches are not equivalent, and the Hamkins’ approach seems to be slightly artificial. Then we assume that if the machine infinitely many times changes its output then the output is undefined.

We say that a relation is *Zeno decidable* if its characteristic function can be computed by a Zeno machine. The following characterises Zeno decidable relations.

---

<sup>6</sup> For discussion of these two views in the context of computational power see [16].

<sup>7</sup> The requirement for the halting condition in [20] that also the head stabilises is an obvious mistake, because in this case Zeno machines would be equivalent to Turing machines.

**Theorem 6 (Zeno decidability theorem).**

*Let  $R$  be a relation on natural numbers. Then all the conditions of theorem 1 and theorems 2, 4, 5 are equivalent to the following:  $R$  is Zeno decidable.*

*Proof.* It is easy to check by writing down the halting condition for Zeno machines that each Zeno decidable relation is  $\Delta_2^0$  arithmetically definable.

Let us consider a relation  $R$  decidable by a Zeno machine  $Z$ . Firstly, let us observe that we can assume that in each computation of  $Z$  there are infinitely many steps in which the machine writes something in the first cell. This can be achieved by adding after each instruction a journey to the first cell, writing the same value as that read, and then going back. So we consider an algorithm  $A$  taking as inputs all inputs of  $Z$  and additionally  $t$ . The algorithm simulates the behaviour of  $Z$  and counts how many times writing on the first cell was done, if it was done  $t$  times then it halts and answers “YES” if in the first cell is 1 and “NO” otherwise. So the defined algorithm is — by the assumptions on  $Z$  — a mathematical learning algorithm for  $R$ .

The idea of Zeno machine can be easily generalised for computations of length defined by any countable ordinal. It is observed in [5] that arithmetical truth can be computed by accelerating Turing machines in  $\omega^2$  steps. An easy generalisation of the above theorem gives another proof for this fact.

## 4 FM–representability in poor arithmetics

In this section we consider FM–representability in some FM–domains with poor arithmetical notions. The weakest known such notion, which is sufficient for FM–representability of all  $\Delta_2^0$  relations, is the relation of divisibility.

**Theorem 7 (FM–representability for divisibility FM–domain, [17]).** *Let  $R$  be a relation on natural numbers. Then all the conditions of theorem 1 and theorems 2, 4, 5, 6 are equivalent to the following:  $R$  is FM–representable in the divisibility FM–domain.*

*Proof.* We give here only a sketchy argument, for details we refer to [17]. The result is based on the observation that the product of any two coprime numbers can be defined in terms of divisibility. Therefore we can define the standard ordering on initial segments of finite models by saying that the first number can be multiplied by something by which the second number cannot be multiplied. In this way in each finite divisibility model we can reconstruct sufficiently large model of divisibility and ordering. However, it is known (see [13]) that addition and multiplication are definable in finite models in terms of divisibility and ordering.

Now let us consider arithmetics which are too poor for obtaining full FM–representability theorem. We start with the observation by Michał Krynicki and Konrad Zdanowski.

**Theorem 8 (FM-representability in arithmetic of addition, [12]).** *Relations which are FM-representable in  $\text{FM}((\mathbb{N}, +))$  are just the relations definable in  $(\mathbb{N}, +)$ .*

*Proof.* It follows from more general fact (see [27]) that if  $\mathcal{A}$  is an arithmetical domain with standard ordering then the FM-representable relations in the corresponding FM-domain are exactly the  $\Delta_2^0$ -definable relations in  $\mathcal{A}$ . However in arithmetic of addition we have elimination of quantifiers then each definable relation is  $\Delta_2^0$ -definable.

Now we are going to characterise FM-representability in coprimality FM-domain. Firstly we need a few auxiliary notions.

Let  $\sim$  be an equivalence relation on  $\mathbb{N}$  and let  $R \subseteq \mathbb{N}^k$ . We say that  $\sim$  is a congruence relation for  $R$  if for all  $a_1, \dots, a_k, b_1, \dots, b_k \in \mathbb{N}$  such that  $a_i \sim b_i$  for,  $i = 1, \dots, k$ ,

$$(a_1, \dots, a_k) \in R \iff (b_1, \dots, b_k) \in R.$$

For  $a, b \in \mathbb{N}$ , we define  $a \approx b$  if  $a$  and  $b$  have the same prime divisors, that is  $\forall x (x \perp a \equiv x \perp b)$ , where  $x \perp y$  means that  $x$  and  $y$  have no common prime divisors. A relation  $R \subseteq \mathbb{N}^k$  is coprimality invariant if  $\approx$  is a congruence for  $R$ .

We know that the structure  $(\mathbb{N}, \perp)$  has many automorphisms and only coprimality invariant relations are preserved by these automorphisms. In particular any two powers of the same prime can be interchanged. Moreover, this property determines definability also on initial segments. Therefore only coprimality invariant relations can be FM-represented. It follows, by theorem 3, that only  $\Delta_2^0$  arithmetically definable coprimality invariant relations can be FM-represented in FM-domain of coprimeness.

**Theorem 9 (FM-representability in arithmetic of coprimeness, [18]).**  *$R$  is FM-representable in  $\text{FM}((\mathbb{N}, \perp))$  if and only if  $R$  is FM-representable in  $\text{FM}(\mathcal{N})$  and  $R$  is coprimality invariant.*

*Proof.* We skip details from [18], giving only general idea and those technicalities which are needed for the next argument. By a similar trick as in the proof of theorem 7 we can define the standard ordering between primes on sufficiently large initial segments. Then using properties of the distribution of primes we encode pairs of primes and compare lengths of sequences of primes. This allows to interpret relations  $R_+$  and  $R_\times$  in coprimality finite models. These relations are coprimality invariant versions of  $R'_+$  and  $R'_\times$  defined as follows:

- $R'_+(p_i, p_j, p_k)$  if and only if  $i + j = k$ ,
- $R'_\times(p_i, p_j, p_k)$  if and only if  $i \times j = k$ ,

where  $p_i$  is the  $i$ -th prime. So we define  $R_+(x, y, z)$  as  $\exists x' \approx x \exists y' \approx y \exists z' \approx z R'_+(x', y', z')$  and similarly  $R_\times$ .

In this way we can FM-represent all FM-representable relations on indices of primes. We have to transfer them into other relations. For this let us observe

that our method of defining ordering works also for products of finite sets of primes (of course up to equivalence  $\approx$ ). By  $ind(x)$  we mean the index of  $x$  in this ordering. Then we define the relation  $W$  such that

$$(x, y) \in W \text{ if and only if } y \approx p_{ind(x)}.$$

In [18] it is shown that the relation  $W$  is FM-representable in coprimality domain and this essentially finishes the proof.

Korec observed in [10] that between coprimality and divisibility we have infinitely many relations ordered according to their definability power.

Let  $n \in \mathbb{N} - \{0\}$  or  $n = \infty$ . We define the following relations on natural numbers:

- $x|_n y$  if and only if for each prime  $q$  and  $k \leq n$ , if  $q^k|n$  then  $q^k|y$ .
- $x \approx_n y$  if and only if  $x|_n y$  and  $y|_n x$ .

The relation  $|_\infty$  is just divisibility  $|$  and  $\approx_\infty$  is the identity. The relation  $|_1$  is mutually definable with coprimality.

The following theorem generalizes the above results on FM-representability in  $FM((\mathbb{N}, |))$  and in  $FM((\mathbb{N}, \perp))$ .

**Theorem 10.** *For any  $n > 0$  or  $n = \infty$ , a relation  $R \subseteq \mathbb{N}^r$  is FM-representable in  $FM((\mathbb{N}, |_n))$  if and only if  $R$  is  $\Delta_2^0$ -definable and the relation  $\approx_n$  is a congruence for  $R$ .*

*Proof.* We will show only how to modify the proof of theorem 9 for obtaining this generalisation. We fix  $n \neq 0, \infty$ . Of course the relation  $\perp$  is definable by  $|_n$ , then we can assume that all required relations are definable provided they use only coprimality. Thus the ordering we define in a similar way,  $x \prec_n y$  means that there is  $z$  coprime with both  $x$  and  $y$  such that there is  $w$  divisible by  $z$  and each power  $p^i$  of prime  $p \neq z$  ( $i \leq n$ ) divides  $w$  exactly when  $p^i$  divides  $x$ , but no such  $w$  exists for  $y$  and  $z$ . The main difference is that now the ordering is defined up to  $\approx_n$  instead of  $\approx$ .

The only relation which should be defined essentially in a different way is  $W$ . This is so because now  $ind(x)$  is the index of  $x$  in a more subtle ordering. However also in this case the argument from [18] can be repeated in extenso.

## 5 Summarizing FM representability theorem

In this final section we collect together all the results about the notions equivalent to FM-representability in one theorem.

**Theorem 11 (The FM-representability theorem and equivalent notions).** *Let  $R$  be a relation on natural numbers. Then the following are equivalent:*

1.  *$R$  is recursive with recursively enumerable oracle; ( – in terms of oracle machines)*

2.  $R$  is of degree  $\leq \mathbf{0}'$ ; ( $-$  in terms of Turing degrees)
3.  $R$  is  $\Delta_2^0$  in arithmetical hierarchy; ( $-$  in terms of arithmetical definability)
4.  $R$  is recursive in the limit; ( $-$  in terms of limits)
5.  $R$  is FM-representable (in  $\text{FM}(\mathbb{N}, \times, +)$ ); ( $-$  just FM-representability)
6.  $R$  is statistically representable; ( $-$  in terms of density)
7.  $R$  is mathematically learnable; ( $-$  in terms of algorithmic learning)
8.  $R$  is Zeno decidable; ( $-$  in terms of hypercomputations)
9.  $R$  is FM-representable in the divisibility FM-domain.

## References

1. Aristotle. *Physics*. about 360 BC.
2. E. M. Gold. Limiting recursion. *The Journal of Symbolic Logic*, 30:28–48, 1965.
3. E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
4. J. D. Hamkins. Infinitary computability with infinite time Turing machines. In B. Cooper, B. Loewe, and L. Torenvliet, editors, *Proceedings of the conference Computability in Europe*, volume 3526 of *Lecture Notes in Computer Science*, pages 180–187. Springer, 2005.
5. J. D. Hamkins and A. Lewis. Infinite time Turing machines. *The Journal of Symbolic Logic*, 65:567–604, 2000.
6. D. Hilbert. Über das Unendliche. *Mathematische Annalen*, 95:161–190, 1926.
7. L. Kołodziejczyk. A finite model-theoretical proof of a property of bounded query classes within ph. *The Journal of Symbolic Logic*, 69:1105–1116, 2004.
8. L. Kołodziejczyk. Truth definitions in finite models. *The Journal of Symbolic Logic*, 69:183–200, 2004.
9. L. A. Kołodziejczyk. *Truth definitions and higher order logics in finite models*. PhD thesis, Warsaw University, 2005.
10. I. Korec. A list of arithmetical structures complete with respect to first-order definability. *Theoretical Computer Science*, 257:115–151, 2001.
11. M. Krynicki, M. Mostowski, and K. Zdanowski. Finite arithmetics. *Fundamenta Informaticae*, 81(1–3):183–202, 2007.
12. M. Krynicki and K. Zdanowski. Theories of arithmetics in finite models. *Journal of Symbolic Logic*, 70(1):1–28, 2005.
13. T. Lee. Arithmetical definability over finite structures. *Mathematical Logic Quarterly*, 49:385–393, 2003.
14. M. Mostowski. On representing concepts in finite models. *Mathematical Logic Quarterly*, 47:513–523, 2001.
15. M. Mostowski. On representing semantics in finite models. In A. Rojszczak<sup>†</sup>, J. Cachro, and G. Kurczewski, editors, *Philosophical Dimensions of Logic and Science*, pages 15–28. Kluwer Academic Publishers, 2003.
16. M. Mostowski. Potential infinity and the Church Thesis. *Fundamenta Informaticae*, 81(1–3):241–248, 2007.
17. M. Mostowski and A. Wasilewska. Arithmetic of divisibility in finite models. *Mathematical Logic Quarterly*, 50(2):169–174, 2004.
18. M. Mostowski and K. Zdanowski. Coprimality in finite models. In Luke Ong, editor, *Computer Science Logic: 19th International Workshop, CSL 2005*, volume 3634 of *Lecture Notes in Computer Science*, pages 263–275. Springer, 2005.

19. M. Mostowski and K. Zdanowski. *FM*–representability and beyond. In B. Cooper, B. Loewe, and L. Torenvliet, editors, *Proceedings of the conference Computability in Europe*, volume 3526 of *Lecture Notes in Computer Science*, pages 358–367. Springer, 2005.
20. P. H. Potgieter. Zeno machines and hypercomputation. *Theoretical Computer Science*, 358:23–33, 2006.
21. Hilary Putnam. Trial and error predicates and the solution to a problem of mostowski. *Journal of Symbolic Logic*, 30(1):49–57, 1965.
22. J. R. Shoenfield. *Recursion Theory*. Lectures Notes in Logic. Springer–Verlag, 1993.
23. R. Soare. *Recursively enumerable sets and degrees*. Springer-Verlag, 1987.
24. W. W. Tait. Finitism. *Journal of Philosophy*, 78:524–546, 1981.
25. A. Tarski. *Pojęcie prawdy w językach nauk dedukcyjnych*. Nakładem Towarzystwa Naukowego Warszawskiego, 1933. English version in [26].
26. A. Tarski. The concept of truth in formalized languages. In J. H. Woodger, editor, *Logic, semantics, metamathematics*, pages 152 – 278. Oxford at The Clarendon Press, 1956. translated from German by J. H. Woodger.
27. K. Zdanowski. *Arithmetics in finite but potentially infinite worlds*. PhD thesis, Warsaw University, 2005.

# Using Tables to Construct Non-Redundant Proofs

Vivek Nigam

INRIA & LIX/École Polytechnique, Palaiseau, France  
`nigam@lix.inria.fr` \*

**Abstract.** Proofs containing more than one subproof for a common subgoal are less preferred in frameworks such as Proof Carrying Code, where proofs are stored and communicated, than proofs that don't contain such redundancies. In this paper, we show how (cut-free) proofs can be transformed into proofs containing cuts and where no atom is proved twice, called *non-redundant proofs*. Two main questions arise when trying to construct these non-redundant proofs: First, which cut-formulas should be used; Second, where to perform cut rules. Some advances in proof theory, namely, our better understanding of focused proofs, allows us to propose the following answers: We use only atomic subgoals of the original proof; and we place cut rules only at the end of the *asynchronous phases*. The backbone of a non-redundant proof is a tree, called *tree of multicut derivations (tmcd)*, where a node is a derivation containing only multicut rules, and an edge represents the provability dependency between a subgoal introduced by a node's multicut rule and another (tree of) multicut derivation. We show how to obtain a **tmcd** from an existing proof.

## 1 Introduction

Frameworks such as Proof Carrying Code [9], where mobile codes are sent with proofs assuring that these codes satisfy certain properties, provide “real world” concerns, not only for *provability*, but also for the *shape* and *format* of proofs. In these frameworks, since proofs need to be stored and communicated, proofs have to attend certain engineering aspects; for instance, the size of proofs is relevant; more precisely, smaller proofs are preferred.

A redundant proof is a proof that contains more than one *non-trivial* subproof of the same atom. For example, consider the proofs that the 12th Fibonacci number is 144 (*fib* 12 144) and obtained from the following logic specification  $\{\text{fib } 1 \ 1, \text{fib } 2 \ 1, \forall XYZ. [\text{fib } X \ Y \wedge \text{fib } (X + 1) \ Z \supset \text{fib } (X + 2) \ (Y + Z)]\}$ . Two types of proofs can be distinguished: one where a *forward chaining* behavior is

---

\* I thank Dale Miller, Miki Hermann, David Baelde, and anonymous reviewers for their helpful comments and discussions. This work has been supported in part by INRIA through the “Equipes Associées” Slimmer and by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2005-015905 MOBIUS project.

adopted, and another where a *backward chaining* behavior is adopted. In the frameworks previously mentioned, the former linear size non-redundant proof is preferred to the latter exponential size redundant proof.

We propose a procedure to construct a non-redundant sequent calculus proof from a redundant sequent calculus proof. This is done by collecting (or *tabling*) from an existing proof a set of atomic lemmas, called *table*. These lemmas are then used as cut formulas to construct a non-redundant proof containing cuts. The use of cuts in non-redundant proofs is not surprising; it is well known that, when compared with cut proofs, cut-free proofs are potentially bigger (also known as the *cut-elimination blow-up*).

There are two main problems to be addressed: (1) which lemmas should be used; and (2) when to use these lemmas, that is, while constructing the non-redundant proof, when should a cut be used. Our answers for these question lie in the structure of *focused proofs*.

By distinguishing rules that are invertible, called asynchronous rules, from rules that are not invertible, called synchronous rules, focused proofs are organized in two alternating phases: *asynchronous phases*, where asynchronous rules are eagerly applied; and *synchronous phases*, where a formula is picked, or *focused* on, and synchronous rules are applied hereditarily to its subformulas (for more about focused proofs, we invite the reader to [1]). Some recent advances on our understanding about focused proofs in classical and intuitionistic logics [5] and about the effect of *atomic polarities*<sup>1</sup> in the shape of proofs [2, 7], provides us with the machinery necessary to propose the following answers to the previous questions:

1) We collect (or table), in the existing proof, all the atomic subgoals. This restriction allows us to construct non-redundant proofs with a polarized cut-rule [7], where an atomic cut-formula has negative polarity in one branch of the proof tree and positive polarity on the other branch of the proof tree. As we investigate elsewhere [7], by using this polarized cut-rule, it is possible to mix a forward chaining behavior with a backward chaining behavior, what enforces some subgoals not to be re-proven;

2) The idea is to use the lemmas as close as possible to the root of the tree, so that if a lemma is to be proved again later in the tree, then it would already be available in the set of hypothesis of the sequent and allow to immediately complete the proof with an initial rule. However, it may happen that a lemma can't be proved right from the bottom of the tree and should be introduced into the proof only when there is an increment in the set of hypothesis of a sequent (by for example, a right implication rule). We show that focusing provides the discipline necessary to identify the places in a proof tree where new lemmas should be introduced, namely, at the end of the *asynchronous phases*.

This paper is structured as follows: we introduce in Section 2 some key concepts related to *focusing* and introduce the intuitionistic system  $LJF^t$  and the use of tables to specify *multicut derivations* (*mcd*). In Section 3, we specify

---

<sup>1</sup> In a focused system, atoms are assigned either positive or negative polarity. This assignment is necessary to organize focused proofs.

how to extract *tree of multicut derivations* (**tmcd**), that is the backbone to construct non-redundant proofs, from different types of proofs, namely Horn Theory proofs, Uniform proofs, and  $LJF^t$  proofs. In Section 4, we show and discuss some experimental results, and finally in Section 5, we finish with some concluding remarks.

## 2 Preliminaries

### 2.1 $LJF^t$

In focused proof systems, formulas are classified as *positive* and *negative*. The formulas *true*,  $\perp$ , or the formulas with main connective  $\wedge$ ,  $\vee$ , or  $\exists$  are positive, while the remaining formulas are negative. This classification is natural since for negative formulas, their right introduction rules are invertible, while this is not necessarily the case for positive formulas. Focused proof systems capitalize on this classification by organizing focused proofs in two phases: the *asynchronous phases*, where only invertible rules are applied, and *synchronous phase*, where non-invertible rules are applied hereditarily to a formula and its subformulas. Notice that backtracking is only necessary in the synchronous phase. In many focused systems, such as  $LJF$ , proposed by Liang and Miller in [5], this classification is extended to atoms by assigning arbitrarily their polarities.

The Figure 1 depicts the inference rules in  $LJF$ , where four different types of sequents can be identified: (1) The sequent  $[\Gamma] - A \rightarrow$  is a *right-focusing* sequent (the focus is  $A$ ); (2) The sequent  $[\Gamma] \xrightarrow{A} [R]$ : is a *left-focusing* sequent (with focus on  $A$ ); (3) The sequent  $[\Gamma], \Theta \longrightarrow \mathcal{R}$  is an *unfocused sequent*. Here,  $\Gamma$  contains negative formulas and positive atoms, and  $\mathcal{R}$  is either in brackets, written as  $[R]$ , or without brackets; (4) The sequent  $[\Gamma] \longrightarrow [R]$  is an instance of the previous sequent where  $\Theta$  is empty.

Asynchronous phases use the third type of sequent above (the unfocused sequents): in that case,  $\Theta$  contains positive or negative formulas. If  $\Theta$  contains positive formulas, then an introduction rule (either  $\wedge_l, \exists_l$  or  $false_l$ ) is used to decompose it; if it is negative, then the formula is moved to the  $\Gamma$  context (by using the  $\|_l$  rule). The end of the asynchronous phase is represented by the fourth type of sequent. Such a sequent is then established by using one of the decide rules,  $D_r$  or  $D_l$ . The application of one of these decide rules then selects a formula for focusing and switches proof search to the *synchronous phase* or *focused phase*. This focused phase then proceeds by applying sequences of inference rules on focused formulas: in general, backtracking may be necessary in this phase of search. Moreover, according to which phase rules can be performed, we classify the rules  $\wedge_l, \exists_l, false_l, \supset_r, \forall_r, \|_l, \|_r, \wedge_r^-$  as *asynchronous rules*, and the remaining rules as *synchronous rules*.

As pointed out elsewhere [5, 2, 7], the atomic polarities play an important role in the shape of the proofs, without affecting in no way provability. For instance, if all atoms have positive polarity, only proofs with a forward chaining behavior are possible, and on the other hand, if all atoms have negative polarity, only proofs with a backward chaining behavior are possible, for example *uniform proofs* [6].

$$\begin{array}{c}
\frac{[N, \Gamma] \xrightarrow{N} [R]}{[N, \Gamma] \longrightarrow [R]} D_l \quad \frac{[\Gamma] - P \rightarrow}{[\Gamma] \longrightarrow [P]} D_r \quad \frac{[\Gamma], P \longrightarrow [R]}{[\Gamma] \xrightarrow{P} [R]} R_l \quad \frac{[\Gamma] \longrightarrow N}{[\Gamma] - N \rightarrow} R_r \\
\\
\frac{}{[\Gamma] \xrightarrow{A_n} [A_n]} I_l \quad \frac{}{[\Gamma, A_p] - A_p \rightarrow} I_r \quad \frac{[\Gamma, N_a], \Theta \longrightarrow \mathcal{R}}{[\Gamma], \Theta, N_a \longrightarrow \mathcal{R}} \mathbb{I}_l \quad \frac{[\Gamma], \Theta \longrightarrow [P_a]}{[\Gamma], \Theta \longrightarrow P_a} \mathbb{I}_r \\
\\
\frac{}{[\Gamma], \Theta, \perp \longrightarrow \mathcal{R}} \text{false}_l \quad \frac{[\Gamma], \Theta \longrightarrow \mathcal{R}}{[\Gamma], \Theta, \text{true} \longrightarrow \mathcal{R}} \text{true}_l \quad \frac{}{[\Gamma] - \text{true} \rightarrow} \text{true}_r \\
\\
\frac{[\Gamma], \Theta, A, B \longrightarrow \mathcal{R}}{[\Gamma], \Theta, A \wedge B \longrightarrow \mathcal{R}} \wedge_l \quad \frac{[\Gamma] - A \rightarrow \quad [\Gamma] - B \rightarrow}{[\Gamma] - A \wedge B \rightarrow} \wedge_r \quad \frac{[\Gamma] - A \rightarrow \quad [\Gamma] \xrightarrow{B} [R]}{[\Gamma] \xrightarrow{A \supset B} [R]} \supset_l \\
\\
\frac{[\Gamma] \xrightarrow{A_i} [R]}{[\Gamma] \xrightarrow{A_1 \wedge^- A_2} [R]} \wedge_l^- \quad \frac{[\Gamma], \Theta \longrightarrow A \quad [\Gamma], \Theta \longrightarrow B}{[\Gamma], \Theta \longrightarrow A \wedge^- B} \wedge_r^- \quad \frac{[\Gamma], \Theta, A \longrightarrow B}{[\Gamma], \Theta \longrightarrow A \supset B} \supset_r \\
\\
\frac{[\Gamma], \Theta, A \longrightarrow \mathcal{R}}{[\Gamma], \Theta, \exists y A \longrightarrow \mathcal{R}} \exists_l \quad \frac{[\Gamma] - A[t/x] \rightarrow}{[\Gamma] - \exists x A \rightarrow} \exists_r \quad \frac{[\Gamma] \xrightarrow{A[t/x]} [R]}{[\Gamma] \xrightarrow{\forall x A} [R]} \forall_l \quad \frac{[\Gamma], \Theta \longrightarrow A}{[\Gamma], \Theta \longrightarrow \forall y A} \forall_r
\end{array}$$

**Fig. 1.** *LJF*: Here,  $\Gamma$  is a set of formulas,  $\Theta$  is a list of formulas,  $A_n$  denotes a negative atom,  $A_p$  a positive atom, and  $P$  a positive formula,  $N$  a negative formula,  $N_a$  a negative formula or an atom, and  $P_a$  a positive formula or an atom. All other formulas are arbitrary and  $y$  is not free in  $\Gamma, \Theta$  or  $R$ .

*LJF*<sup>t</sup> capitalizes on the observation that atomic polarities can be arbitrarily assigned, and extends *LJF* in two ways. (1) Extends the *LJF* sequents with a polarity context,  $\mathcal{P}$ , which specifies all the positive atoms in a sequent. An atom in a sequent,  $\mathcal{P}; \Gamma \longrightarrow [R]$ , is positive if and only if  $A \in \mathcal{P}$ ; (2) extends *LJF* with the following polarized multicut rule:

$$\frac{\mathcal{P}; \Gamma \longrightarrow [A_1] \quad \dots \quad \mathcal{P}; \Gamma \longrightarrow [A_n] \quad \mathcal{P} \cup \Delta; \Gamma \cup \Delta \longrightarrow [R]}{\mathcal{P}; \Gamma \longrightarrow [R]} mc.$$

Where  $\Delta = \{A_1, \dots, A_n\}$  is a set of atoms. The multicut rule is the only rule that can change the polarity context in a proof.

**Remarks:** (1) The results in this paper could be easily applied to (focused) classic logics, such as *LKF* [5]. (2) Here we only consider focused proofs; however, it seems possible to apply the results obtained here to a more general setting where non-focused proofs are considered, by using methods such as in [8] to convert non-focused proofs to focused proofs, but this is left out of the scope of this paper. (3) We use interchangeably the same names for the rules in *LJF* and *LJF*<sup>t</sup>, but always being clear from the context to which system we refer to.

In the next subsection, we use this polarized multicut rule to construct multicut derivations that are the basic element used to construct non-redundant proofs.

## 2.2 Tables as Multicut Derivations

We consider a table as a partially ordered finite set of atoms.

**Definition 1.** A table is a tuple  $\mathcal{T} = \langle \mathcal{A}, \prec \rangle$ , where  $\mathcal{A}$  is some finite set of atoms, and  $\prec$  is a partial order relation over the elements of  $\mathcal{A}$ .

In a table, each atom represents, intuitively, a provable sub-goal necessary in the proof of a sequent (say  $\Gamma \longrightarrow G$ ), and the order relation the provability dependency between the atoms, that is, if  $A \prec B$  then  $A$  is a subgoal used to prove the goal  $B$ .

The next definition specifies a derivation composed only of multicut rules, represented by a table.

**Definition 2.** Let  $\mathcal{T} = \langle \mathcal{A}, \prec \rangle$  be a table. The multicut derivation for  $\mathcal{T}$  and the sequent  $\mathcal{S} = \Gamma \longrightarrow G$ , written as  $mcd(\mathcal{T}, \mathcal{S})$ , is defined inductively as follows: if  $\mathcal{A}$  is empty, then  $mcd(\mathcal{T}, \mathcal{S})$  is the derivation containing just the sequent  $\mathcal{S}$ . Otherwise, if  $\{A_1, \dots, A_n\}$  is the collection of  $\prec$ -minimal elements in  $\mathcal{A}$  and if  $\Pi$  is the multicut derivation for the smaller table  $\langle \mathcal{A} \setminus \{A_1, \dots, A_n\}, \prec \rangle$  and the sequent  $\Gamma, A_1, \dots, A_n \longrightarrow G$ , then  $mcd(\mathcal{T}, \mathcal{S})$  is the derivation

$$\frac{\Gamma \longrightarrow A_1 \quad \dots \quad \Gamma \longrightarrow A_n \quad \Pi, A_1, \dots, A_n \longrightarrow G}{\Gamma \longrightarrow G} mc$$

Multicut derivations are always open derivations (that is, they contain leaves that are not proved). A proof of a multicut derivation is any (closed) proof that extends this open derivation.

Elsewhere [7], we investigated the use of tables for obtaining non-redundant proofs in the Horn fragment. We used the following observation:

**Proposition 1.** [7] Let  $\Gamma$  be a set of Horn clauses,  $A \in \mathcal{P} \cap \Gamma$ , and  $\Xi$  be an arbitrary LJF<sup>t</sup> proof tree for  $\mathcal{P}; [\Gamma] \dashv_G \vdash$ . Then every occurrence of a sequent with right-hand side the atom  $A$  is the conclusion of an  $I_r$  rule.

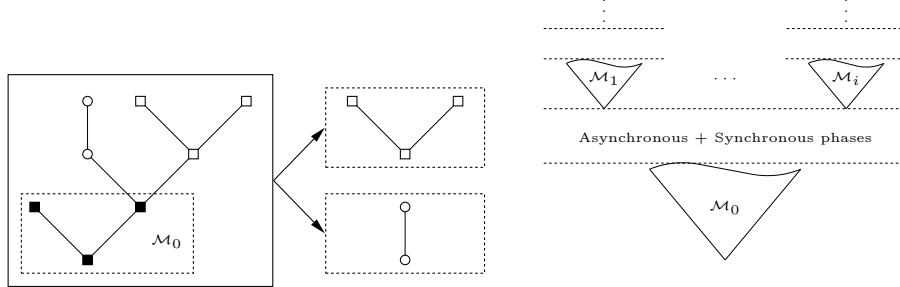
If in a proof of  $\Xi$ , there are several non-trivial proofs for the subgoal  $A$ , that is, proofs that contain more than an inference rule, one could table  $A$  and construct the corresponding multicut derivation for this table and construct a non-redundant proof. For example, when comparing the two derivations below, the left derivation could have several non-trivial subproofs for  $A$ , while the right derivation must have only one non trivial proof for  $A$ : the proof of the cut's left branch.

$$\frac{\Gamma \longrightarrow A \quad \Gamma \longrightarrow G}{\Gamma \longrightarrow A \wedge G} \implies \frac{\mathcal{P}; [\Gamma] \longrightarrow [A] \quad \mathcal{P} \cup \{A\}; [\Gamma, A] \longrightarrow [A \wedge G]}{\mathcal{P}; [\Gamma] \longrightarrow [A \wedge G]} mc.$$

In the next sections, we use *mcds* to construct non-redundant proofs. To represent a *mcd* when we specify the algorithms to extract non-redundant proofs, we use the following data type  $mcd ::= \text{sequent}^* \text{atom list}$ <sup>2</sup>.

---

<sup>2</sup> We use a list of atoms representing the topological sort of a table's partial ordering.



**Fig. 2.** The left figure illustrates of how the function `buildTmcder` extracts a **tmcder** from a proof tree. The right figure depicts the general architecture of a completed **tmcder** proof. The triangles represent **mcds** and the dashed lines represents end of asynchronous phases.

### 3 Tree of Multicut Derivations - **tmcder**

The backbone of a non-redundant proof is a tree, called *tree of multicut derivations* (**tmcder**), where a node is a derivation containing only multicut rules, and an edge represents the provability dependency between a subgoal introduced by a node's multicut rule and another (tree of) multicut derivation. The architecture of a **tmcder** is depicted in Figure 2. The idea is that each multicut derivation is only used when the context is augmented with new atoms or new positive formulas. Since contexts can only be augmented by the asynchronous rules  $\forall_r, \exists_l$ , and  $\supset_r$ <sup>3</sup> and asynchronous rules are invertible, focusing provides a natural way to identify where to place a multicut derivation, namely, whenever an asynchronous phase ends and the context is augmented with a new positive formula or a new atom. While focused cut-free proofs are structured in two alternating phases, asynchronous phases and synchronous phases, focused cut proofs are structured with one more phase, called *cut phase*, appearing always between an asynchronous phase and the following synchronous phase.

In the following subsections, we specify algorithms to extract **tmcders** from proofs in Horn Theory, from Uniform Proofs [6], and from *LJF* proofs. We represent **tmcders** by the following data type:  $\text{tmcder} ::= \text{atom}^* \text{ mcd }^* \text{ tmcder list}$ .

#### 3.1 Horn Theory

A characteristic of uniform Horn Theory proofs, such as the proofs generated by Prolog, is that contexts never changes. Therefore, all proved atoms in such a proof are provable from the initial context. This property enables us to construct a **tmcder** with only one node, obtained from the function `buildTable` shown in Figure 3. This function uses the function `atomPOT` that performs a postorder traversal (i.e., process a nodes premises before processing the node), and uses

<sup>3</sup> The first two rules augment the context with a new eigenvariable, and the last rule augments the context with some formula.

```

buildTable : tree → mcd
  input:  $\Xi$ 
    (rootOf( $\Xi$ ), eliRed(atomPOT  $\Xi$ ))
where:
atomPOT : tree → atom list
  input: Node(seq,  $b_1, \dots, b_n$ )
    if seq =  $[\Gamma] \longrightarrow [A]$ 
      then atomPOT  $b_1 :: \dots ::$ 
        atomPOT  $b_n :: A$ 
      else atomPOT  $b_1 :: \dots ::$ 
        atomPOT  $b_n$ 
getSCT : context → tree → tree
  input:  $\Gamma$ , Node(seq,branches)
    if  $\Gamma$  = contextOf(seq)
      then
        Node(seq, map (getSCT  $\Gamma$ ) branches)
      else Nil
getChT : context → form → tree list → (form * tree) list
  input:  $\Gamma, F, \text{Node}(\text{seq}, \text{branches})::\text{list}$ 
    if seq =  $[\Gamma] \longrightarrow [A]$ 
      then (getChT  $\Gamma A$  branches) :: (getChT  $\Gamma F$  list)
    else if seq =  $[\Gamma'] \longrightarrow [\cdot]$  and  $\Gamma \neq \Gamma'$ 
      then ( $F$ , Node(seq,branches)) :: (getChT  $\Gamma F$  list)
    else (getChT  $\Gamma F$  branches) :: (getChT  $\Gamma F$  list)
buildTmcd : tree → tmcd
  input:  $\Xi$ 
    eliRed(tmcdAux Empty  $\Xi$ )
where:
tmcdAux : (form * tree) → tmcd
  input: ( $A, \Xi$ )
    (A, buildTable(getSCT (contextOf( $\Xi$ ))  $\Xi$ ),
     map tmcdAux (getChT (contextOf( $\Xi$ )) A ( $\Xi :: []$ )))

```

**Fig. 3.** Functions used to extract a **tmcd** from a goal directed proof. Here the functions: contextOf returns the bracket context of a sequent; map applies a function to all the elements of a list.

the function eliRed that retains only the first occurrence of any repeated atomic formula in a list of formulas.

The correctness of this algorithm can be shown by a simple induction and can be found elsewhere [7]. It is easy to show that buildTable extracts a *mcd* from a Horn Theory proof in time  $\mathcal{O}(n)$ , where  $n$  is the size of the input proof.

### 3.2 Uniform Proofs

We now specify how to extract a tree of multicut derivations from a finite (goal-directed) proof tree, that is, *LJF* proofs where all atoms are assigned negative polarity. In this more general class of proofs, it can happen that, after an asynchronous phase, the context is augmented with a new positive formula or with a new atom. Hence, differently from the Horn Theory case, not all atomic subgoals are provable from the initial context, and therefore, a **tmcd** with more than one node will be the backbone of the non-redundant proof of an uniform proof.

The function buildTmcd, shown in Figure 3, extracts a **tmcd** from an uniform proof. We use the illustration in Figure 2 to explain how this function works. The three different kinds of nodes (filled squares, ellipses, and blank squares) represent sequents with different bracket contexts. First, the subtree with the filled squares is extracted, by using the function getSCT, shown in Figure 3, and, from this subtree, the multicut derivation  $\mathcal{M}_0$  is constructed by using the function buildTable. Second, by using the function getChT, also shown in Figure 3, the remaining children trees are extracted together with an atom, appearing in  $\mathcal{M}_0$ , representing the provability dependency of two nodes of the extracted **tmcd**. Third, buildTmcd is recursively applied to each child tree. Fourth, as done with in the Horn Theory case, we eliminate redundancies with the function eliRed as follows: let  $\mathcal{M}_a$  be a multicut derivation containing the atom  $A$ , and let  $\mathcal{M}_d$  be a descendent multicut derivation of  $\mathcal{M}_a$ . If  $\mathcal{M}_d$  has the base sequent  $[\Gamma] \longrightarrow [A]$  then we obtain a new **tmcd**, by removing the **tmcd**'s subtree with

root  $\mathcal{M}_d$ ; or if  $A$  is in  $\mathcal{M}_d$ , then we obtain a new **tmcd** by removing  $A$  from  $\mathcal{M}_d$  and its possible descendent subtrees from the tree of multicut derivations. There is a final step that is not shown here concerning the sequent's polarity context,  $\mathcal{P}$  and context  $\Gamma$ , which need to be augmented with the previous application of multicut rules. This can be done in a straightforward way, by traversing the obtained **tmcd**.

We prove the following correctness result by induction on the height of the input tree  $\Xi$ .

**Proposition 2.** *Let  $\Xi$  be a uniform LJF proof and let  $\tau = \text{buildTmcd}(\Xi)$  be the tree of multicut derivations obtained from  $\Xi$ . Then  $\tau$  can be completed to a proof by adding derivations containing only one  $D_l$  rule.*

As argued before, the synchronous rules are not invertible and hence, have an inherent *don't know non-determinism*. In particular, the  $D_l$  rule has higher degree of don't know non-determinism, since it is usually the case that the context of sequents contain several formulas and therefore, when applying the  $D_l$  rule, an interpreter might need to backtrack more often to this rule and choose another formula to focus on. Hence, the proposition above states that not only **tmcds** are correct, but also that one can complete a **tmcd** in an optimal way, since an interpreter only needs to search for derivations containing only one  $D_l$  rule.

### 3.3 LJF proof

In the previous subsection, we considered only uniform *LJF* proofs, that is, proofs where all atoms have negative polarity. We now extend the results obtained before to all *LJF* proofs where there can also be atoms with positive polarity. Now, there are two main differences with respect to uniform *LJF* proofs: (1) **Initial Right Rule** - Initial right rules can end the proof; (2) **Reaction Left with Atomic Formula** - By allowing atoms with positive polarity, proofs can perform forward chaining steps. For instance, consider the following derivation where the atom  $A$  has positive polarity:

This type of derivation is not possible to occur in a uniform proof, since there, the only rule that introduces atoms focused in the left is the initial left rule.

$$\frac{\frac{\frac{[\Gamma, A] \longrightarrow [G']}{} R_r, []_r}{[\Gamma] \xrightarrow{-G} [\Gamma]} \quad \frac{[\Gamma] \xrightarrow{A} [G']}{} \supset_l}{[\Gamma] \xrightarrow{G \supset A} [G']}$$

We change the functions *atomPOT*, *getSCT*, and *getChT*, to accommodate these differences. For the first difference, namely that initial right rules can finish the proof, it suffices to table the positive atom used to finish the proof, and therefore, in the extracted **tmcd**, this atom will have its polarity changed to positive<sup>4</sup>. For the second difference, namely that atomic reaction left rules can happen, we table the forward chained atom performing, in the extracted **tmcd**, the following transformation:

---

<sup>4</sup> Remember that at the base of any **tmcd**, all atoms are assigned negative polarity.

$$\begin{array}{c}
 \frac{\Xi}{[\Gamma, A] \longrightarrow [G']} R_r, []_l \quad \frac{\mathcal{P}; [\Gamma] -_G \rightarrow \mathcal{P}; [\Gamma] \xrightarrow{A} [A]}{\mathcal{P}; [\Gamma] \xrightarrow{G \supset A} [A]} I_l \\
 \frac{[\Gamma] -_G \rightarrow [\Gamma] \xrightarrow{A} [G']}{[\Gamma] \xrightarrow{G \supset A} [G']} D_l \qquad \Rightarrow \qquad \frac{\mathcal{P}; [\Gamma] \xrightarrow{G \supset A} [A] D_l}{\mathcal{P}; [\Gamma] \longrightarrow [A]} \qquad \frac{\mathcal{P}'; [\Gamma, A] \longrightarrow [G']}{\mathcal{P}; [\Gamma] \longrightarrow [G']} mc
 \end{array}$$

However, a new question arises: where in the **tmcd** should we perform a cut with  $A$  as cut formula. The answer is to insert this cut before inserting the cuts with the atomic subgoals appearing in  $\Xi$  because to prove these subgoals it might be necessary to use  $A$ .

Accordingly, we add new cases, shown in Figure 4, to the functions atomPOT, getSCT, and getChT. To atomPOT, the first new case inserts a positive atom focused on the left to the beginning of the list of atomic subgoals; the second new case inserts to the list of atomic subgoals any positive atom that is used to finish a proof with an initial right rule. The function getSCT, that is used to extract subtrees used to construct **tmcd**'s nodes, is modified so that extracted subtrees include atomic reaction left rules. Since the subtree extracted by getSCT is modified, getChT is modified to extract correctly the children subtrees.

```

function: atomPOT = ...
function: getSCT = ...
function: getChT = ...

else if seq =  $[\Gamma] \xrightarrow{A} [G]$  and  $A \notin \Gamma$  then  $A :: \text{atomPOT } b_1 :: \dots ::$ 
      atomPOT  $b_n$ 
else if seq =  $[\Gamma] -_A \rightarrow$  then  $A ...$ 
else if seq =  $[\Gamma] \xrightarrow{A} [G]$  then Node(seq, map (getSCT ( $\Gamma \cup \{A\}$ )) branches) ...
else if seq =  $[\Gamma] -_A \rightarrow$  then (getChT ( $\Gamma \cup \{A\}$ ) F branches) :: (getChT ( $\Gamma \cup \{A\}$ ) F list) ...

```

**Fig. 4.** New cases added to the functions shown in Figure 2 to handle all LJF proofs.

We prove the following correctness result by induction on the height of the input tree  $\Xi$ .

**Proposition 3.** *Let  $\Xi$  be an LJF proof and let  $\tau = \text{buildTmcd}(\Xi)$  be the tree of multicut derivations obtained from  $\Xi$ . Then  $\tau$  can be completed to a proof by adding derivations containing only one  $D_l$  rule.*

The complexity of buildTmcd lies in the comparison of two context (e.g. contextof(seq) =  $\Gamma$ ). Considering an upperbound,  $s$ , on the length of formulas, we can show that buildTmcd extracts a **tmcd** from a  $LJF^t$  proof in time  $\mathcal{O}(ns(\log n))$ , where  $n$  is the size of the input proof. This is done by assigning an ordering to clauses, represented by strings; for instance by using the ASCII numbering. To determine that two sequents have the same context, it suffices to sort the contexts of the sequents and then, check one by one the equivalence of the clauses in the contexts.

## 4 Experiments

As the experimental results in the following table shows, by completing the **tmcd** extracted from the original tree ( $\Xi$ ), we obtain a considerably smaller proof.

Also, the execution time of `buildTmcd` is much lesser than the time needed to find the original proof.

	Size - number of nodes	Time - ms				
		$\Xi$	proof from <b>tmcd</b>	buildTmcd( $\Xi$ )	find( $\Xi$ )	complete <b>tmcd</b>
5 <sup>th</sup> fib	15	15		2.8	45.0	50
8 <sup>th</sup> fib	67	27		3.1	85.0	120
11 <sup>th</sup> fib	287	39		4.8	330.0	170

In the case for the 8<sup>th</sup> Fibonacci, the time needed to complete a **tmcd** is more than of finding the original proof. However, we observed in our experiments that it takes a constant time of 20 ms to complete one of the open branches of a **tmcd** and there are only a linear number of them, that is, one open branch for each subgoal. Therefore, we expect that while the time to search for proofs exponentially grows with the Fibonacci number, the time of completing the extracted **tmcd** should increase linearly.

## 5 Conclusions

This paper presents an approach, from a proof theoretic point of view, for reducing size of proofs through redundancy elimination. By a careful study of focused proofs, we propose a new structure, called tree of multicut derivations, to be the backbone for the construction of non-redundant proofs. The theoretical and experimental results in this papers suggests that the procedure proposed can be used to obtain smaller proofs. Further research is needed to investigate the plausibility of this approach in the Proof Carrying Code framework.

## References

1. Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. of Logic and Computation*, 2(3):297–347, 1992.
2. Kaustuv Chaudhuri, Frank Pfenning, and Greg Price. A logical characterization of forward and backward chaining in the inverse method. In Proceedings of IJCAR’06, 2006.
3. Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In Proceedings of ICFP ’00, 2000.
4. Radha Jagadeesan, Gopalan Nadathur, and Vijay Saraswat. Testing concurrent systems: An interpretation of intuitionistic logic. In Proceedings of FSTTCS, 2005.
5. Chuck Liang and Dale Miller. Focusing and polarization in intuitionistic logic. In Proceedings of CSL’07, 2007.
6. Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.
7. Dale Miller and Vivek Nigam. Incorporating tables into proofs. In Proceedings of CSL’07, 2007.
8. Dale Miller and Alexis Saurin. From proofs to focused proofs: a modular proof of focalization in linear logic. In Proceedings of CSL’07, 2007.
9. George C. Necula. Proof-carrying code. In Proceedings of POPL’97, 1997.

# Classifying the Phase Transition Threshold for Unordered Regressive Ramsey Numbers

Florian Pelupessy and Andreas Weiermann

Vakgroep Zuivere Wiskunde en Computer algebra  
Krijgslaan 281 Gebouw S22, 9000 Ghent, Belgium  
`{weierman, pelupessy}@cage.ugent.be`

**Abstract.** Following ideas of Richer (2000) we introduce the notion of unordered regressive Ramsey numbers or unordered Kanamori-McAloon numbers. We show that these are of Ackermannian growth rate. For a given number-theoretic function  $f$  we consider unordered  $f$ -regressive Ramsey numbers and classify exactly the threshold for  $f$  which gives rise to the Ackermannian growth rate of the induced unordered  $f$ -regressive Ramsey numbers. This threshold coincides with the corresponding threshold for the standard regressive Ramsey numbers. Our proof is based on an extension of an argument from a corresponding proof in a paper by Kojman, Lee, Omri and Weiermann 2007.

**Key words:** regressive Ramsey numbers, Ackermann function, unordered canonical Ramsey theorem, Kanamori McAloon theorem

## 1 Introduction

There exist three basic and well known infinitary Ramsey principles which give rise to independence results for PA: Ramsey's theorem, the canonical Ramsey theorem (Erdős Rado) and the regressive Ramsey theorem (Kanamori McAloon) [8,14,13,12]. The latter two principles make essential use of a pre-existing order on the natural numbers in order to speak about min-colourings, max-colourings and min-homogeneous sets, etc. The three basic infinitary Ramsey-principles give rise to finitary Ramsey principles which can be formulated in the language of arithmetic: the finite Ramsey principle with a suitable largeness condition (i.e. the Paris-Harrington principle [14]), the Kanamori McAloon principle [8] and the finite canonical Ramsey principle with a suitable largeness condition. It turns out that all these finite versions make essential use of the standard  $<$ -relation and one might wonder if it is possible to find strong principles which do not depend so intrinsically on the less than relation.

An interesting approach to this question can be obtained from a recent paper by Richer [17] about unordered canonical Ramsey numbers and their asymptotic classification. It is quite natural to extend Richer's approach to the context of strong Ramsey principles and in this paper we do this for the Kanamori McAloon Ramsey theorem.

We consider the phase transition for unordered regressive Ramsey numbers. For simplicity we limit ourselves to only colourings of pairs (graphs) and expect

that the result extends to higher dimensions (hyper graphs) using appropriate bounds from the fast growing hierarchy. We also expect that our results generalize to the unordered canonical Ramsey theorem with a suitable largeness condition. This and other related questions will be investigated jointly in a bigger research project with A. Bovykin, L. Carlucci, G. Lee et al.

This contributes to a general research program of the second author about phase transitions in logic and combinatorics (see, for example, [19,20,21,22,23] for more information).

We identify the natural numbers with their corresponding sets of predecessors and use  $C(u, v)$  instead of  $C(\{u, v\})$  for a colouring  $C : [R]^2 \rightarrow \mathbb{N}$ . Here  $[R]^2$  denotes the set of pairs of unequal elements from  $R$ . No ordering of  $u, v$  is implied here and when known we will give the relative order of the two elements. Denote the collection of subsets of size  $m$  of a given set  $R$  with  $P_m(R)$ . Given a number-theoretic function  $f$  let us call a colouring  $C$  of pairs  *$f$ -regressive* if  $C(u, v) \leq f(u)$  for all  $u, v$  with  $u < v$ . Call an ordered set  $(H, \prec)$   $\min_{\prec}$ -*homogeneous* for  $C$  if  $C(x, y) = C(x, z)$  for all  $x, y, z$  in  $H$  with  $x \prec y, x \prec z$ . Then given  $m$  there exists a least number  $R := uKM_f(m)$  such that for all  $f$ -regressive colourings  $C : [R]^2 \rightarrow \mathbb{N}$  there exists an  $H \in P_m(R)$  and a linear ordering  $\prec$  on  $H$  such that  $H$  is  $\min_{\prec}$ -homogeneous for  $C$ .

The class of *primitive recursive functions* is the smallest class of functions  $\mathbb{N}^d \rightarrow \mathbb{N}$  which contains the constant functions, projections and successor function and is closed under composition and recursion. We call a function *Ackermannian* if it eventually dominates every primitive recursive function. Define the *Ackermann function*  $A$  as follows:

$$\begin{aligned} A_0(i) &:= i + 1 \\ A_{n+1}(i) &:= (A_n)^{(i)}(i) \\ A(i) &:= A_i(i) \end{aligned}$$

The Ackermann function is Ackermannian. It is easy to see that for constant functions  $f$  the function  $uKM_f$  is primitive recursive and so in between the constant function and the identity function there will be phase transition from being primitive recursive to Ackermannian of  $uKM_f$ . Roughly speaking, for  $f(i) = \sqrt[k]{i}$ , the function  $uKM_f$  is Ackermannian whereas for  $f(i) = \log(i)$  the function  $uKM_f$  is still elementary recursive. In a final step we let  $k$  in  $\sqrt[k]{i}$  depend on  $i$ . We show that function  $uKM_f$  is still primitive recursive for any  $d$  if  $f(i) \leq {}^{A_d^{-1}(i)}\sqrt{i}$  but becomes Ackermannian if  $f(i) \geq {}^{A^{-1}(i)}\sqrt{i}$ .

## 2 Upper Bounds

We use results from the Kanamori-McAloon principle to derive upper bounds on the unordered case.

**Theorem 1 (uKM).** *For every  $m$  there exists an  $R$  such that for every  $f$ -regressive colouring  $C : [R]^2 \rightarrow \mathbb{N}$  there exists an  $H \in P_m(R)$  with linear order  $\prec \subseteq H^2$  which is  $\min_{\prec}$ -homogeneous for  $C$ .*

*Proof.* Let  $KM_f(m)$  be the minimal  $\bar{R}$  such that for every  $f$ -regressive colouring  $C : [R]^2 \rightarrow \mathbb{N}$  there exists an  $H \in P_m(R)$  which is  $\min_{<}$ -homogeneous for  $C$ . (see [8] for proof of existence of such  $\bar{R}$ )

For given  $f$  take  $R = KM_f(m)$  and an  $f$ -regressive colouring  $C$ . Then there exists  $H \in P_m(R)$  which is  $\min_{<}$ -homogeneous for  $C$ , hence taking for  $\prec$  the ordering  $<$  suffices to make  $H$   $\min_{\prec}$ -homogeneous for  $C$ .

Notation:  $uKM_f(m) :=$  the smallest such  $R$ . The proof of this theorem also delivers upper bounds on  $uKM_f$ .

**Theorem 2.**  $uKM_f$  is primitive recursive if  $f$  is:

1. a constant function,
2.  $i \mapsto \log i$ ,
3.  $i \mapsto \sqrt[A_d^{-1}(i)]{i}$ .

*Proof.* Because  $uKM_f(m) \leq KM_f(m)$  and the primitive recursive functions are closed under bounded search it suffices that  $KM_f$  is primitive recursive. For proof of that for all three cases see [2].

### 3 Lower Bounds

For the lower bound it is not possible to easily transfer earlier results. We modify the proofs for  $KM_f$  from [2] and [10] to suit the problem that allowing differing orderings on  $H$  gives. As a preliminary we begin with some results about primitive recursive functions.

For  $s > 0$ , define the following sequence:

$$\begin{aligned} A_0^s(i) &:= i + 1 \\ A_{n+1}^s(i) &:= (A_n^s)^{(\lfloor \sqrt[s]{i} \rfloor)}(i) \\ A^s(i) &:= A_i^s(i) \end{aligned}$$

Note that for  $s = 1$  this is the Ackermann function.

For  $R_c^2(i)$  we take the minimal  $R$  such that for every colouring  $C : [R]^2 \rightarrow c$  there exists  $Y \in P_i(R)$  such that  $Y$  is  $C$ -homogeneous. ( $C$  is constant on  $[Y]^2$ ).

**Lemma 1.** 1. The Ackermann function is Ackermannian.  
 2. If the composition of two non-decreasing functions is Ackermannian and one of those is primitive recursive, then the other is Ackermannian.  
 3.  $(i, c) \mapsto R_c^2(i)$  is primitive recursive.

*Proof.* A proof of the first two statements can be found in [1] and [11], for a proof of the latter one see [7].

We now give a lower bound for  $uKM_f$  which should ensure that it is Ackermannian. This proof rests on two ideas, namely the use of the particular colourings similar to proofs of the ordered  $KM_f$  and increasing the 'space' in the  $D$ -homogeneous set to solve the problem caused by allowing any linear order to determine  $\min_{\prec}$ -homogeneity. Fix  $s \in \mathbb{N}$ .

**Lemma 2 (lower bound for roots).** Let  $f : i \mapsto \sqrt[i]{i}$ , then:

$$uKM_f(R_c^2(m+4)) \geq A_{c+1}^s(m)$$

for all  $c, m \in \mathbb{N}$ .

*Proof.* Take  $k = R_c^2(m+4)$  and  $R = uKM_f(k)$ . Define a colouring  $C$  on  $R$  as follows for  $x < y$ :

$$C(x, y) = \begin{cases} 0 & \text{if } A_{c+1}^s(x) \leq y \\ l & \text{else} \end{cases}$$

where  $l$  is such that for the smallest  $p$  for which  $A_{p+1}^s(x) > y$  we have  $A_p^{s(l)}(x) \leq y < A_p^{s(l+1)}(x)$ .

Taking  $p$  for  $x, y$  as above, define colouring  $D$  of  $R$  for  $x < y$ :

$$D(x, y) = \begin{cases} 0 & \text{if } A_{c+1}^s(x) \leq y \\ p & \text{else} \end{cases}$$

Note that  $C$  is  $f$ -regressive (because  $A_p^{s(\lfloor f(x) \rfloor)}(x) = A_{p+1}^s(x)$ ). Let  $H \in P_k(R)$  with order  $\prec$  be  $\min_\prec$ -homogeneous for  $C$ , then by definition of  $k$  there exists  $Y \in P_{m+4}(H)$  which is  $D$ -homogeneous. Enumerate such a  $Y$  with a strictly  $<$ -increasing sequence  $Y = \{y_1, \dots, y_m, x, y, z, z'\}$ . Then we have the following cases for the relative  $\prec$ -ordering of  $x, y, z, z'$ :

1.  $x \prec y, x \prec z$

*Claim:*  $A_{c+1}^s(x) \leq y$ .

Assume for a contradiction that  $A_{c+1}^s(x) > y$ , then by definition of  $C$  we get  $C(x, y) = l \neq 0$ . Hence (by  $\min_\prec$ -homogeneity of  $H$ )  $C(x, z) = C(x, y) = l$ . By definition of  $D$  and  $D$ -homogeneity of  $Y$  we also get  $D(y, z) = D(x, y) = p \neq 0$ .

So the definition of  $C$  gives us:

$$A_p^{s(l)}(x) \leq y < A_p^{s(l+1)}(x)$$

and

$$A_p^{s(l)}(x) \leq z < A_p^{s(l+1)}(x),$$

that of  $D$  delivers:

$$A_p^s(y) \leq z.$$

Combining these inequalities, taking note that  $A_p^s$  is increasing, we get the contradiction:

$$z < A_p^{s(l+1)}(x) = A_p^s(A_p^{s(l)}(x)) \leq A_p^s(y) \leq z$$

2.  $z \prec x, z \prec y$

*Claim:*  $A_{c+1}^s(x) \leq z$ .

Assume  $A_{c+1}^s(x) > z$ , then by definition and  $\min_\prec$ -homogeneity of  $C$  we

have  $C(x, z) = C(y, z) = l \neq 0$ , by definition and homogeneity of  $D$  we get:  $D(x, y) = D(x, z) = p$ . This gives us inequalities:

$$A_p^{s(l)}(x) \leq z < A_p^{s(l+1)}(x),$$

$$A_p^{s(l)}(y) \leq z < A_p^{s(l+1)}(y)$$

and

$$A_p^s(x) \leq y.$$

Combining these we get:

$$z < A_p^{s(l+1)}(x) = A_p^{s(l)}(A_p(x)) \leq A_p^{s(l)}(y) \leq z$$

3.  $y \prec x, y \prec z$ , we distinguish two possibilities:

(a)  $y \prec z'$

*Claim:*  $A_{c+1}^s(y) \leq z$ .

Assume  $A_{c+1}^s(y) > z$ , then  $C(y, z) = C(y, z') = l \neq 0$  and  $D(z, z') = D(y, z') = p$ . So we have inequalities:

$$A_p^{s(l)}(y) \leq z < A_p^{s(l+1)}(y),$$

$$A_p^{s(l)}(y) \leq z' < A_p^{s(l+1)}(y)$$

and

$$A_p^s(z) \leq z'.$$

Combining these:

$$z' < A_p^s(A_p^{s(l)}(y)) \leq A_p^s(z) \leq z'$$

(b)  $z' \prec y$

*Claim:*  $A_{c+1}^s(x) \leq z'$ .

Assume  $A_{c+1}^s(x) > z'$ , then  $C(x, z') = C(y, z') = l \neq 0$  and  $D(x, y) = D(x, z') = p$ . So we have:

$$A_p^{s(l)}(x) \leq z' < A_p^{s(l+1)}(x),$$

$$A_p^{s(l)}(y) \leq z' < A_p^{s(l+1)}(y)$$

and

$$A_p^s(x) \leq y.$$

Combining these:

$$z' < A_p^{s(l)}(A_p^s(x)) \leq A_p^{s(l)}(y) \leq z'$$

Examining the cases above allows us to conclude  $A_{c+1}^s(x) \leq z'$ . But then:  $A_{c+1}^s(m) \leq A_{c+1}^s(y_m) \leq A_{c+1}^s(x) \leq z' \in Y \subseteq H \subseteq R$ . So we finally have:

$$A_{c+1}^s(m) \leq R$$

For this lower bound to result in Ackermannian functions  $uKM_f$  the  $A^s$  have to be Ackermannian as well:

**Lemma 3.**  $A_n(i) \leq A_{n+2s^2+1}^s(i)$  for any  $i \geq 4^s$ .

*Proof.* See [2], corollary 4.3.

**Theorem 3.**  $uKM_f$  is Ackermannian for  $f = f_s : i \mapsto \sqrt[s]{i}$  and for  $f : i \mapsto \sqrt[A^{-1}(i)]{i}$ .

*Proof.* For the first assertion combine lemmas 1, 2 and 3.

For the second we claim that

$$N(i) := uKM_f(R_{i+2i^2+1}^2(4^i + 3)) > A(i)$$

for all  $i$ . Assume for contradiction that  $N(i) \leq A(i)$  for some  $i$ . Then for  $l \leq N(i)$  we have  $A^{-1}(l) \leq i$ , so  $\sqrt[l]{l} \leq \sqrt[A^{-1}(i)]{l}$ . Hence:

$$\begin{aligned} uKM_f(R_{i+2i^2+1}^2(4^i + 3)) &\geq uKM_{f_i}(R_{i+2i^2+1}^2(4^i + 3)) \\ &\geq A_{i+2i^2+1}^i(4^i) \\ &> A(i) \end{aligned}$$

Where the first inequality is a consequence if the definition of  $uKM_f$ , the second of lemma 2 and the third of lemma 3. The resulting inequality contradicts with our assumption.

Now the claim with lemma 1 implies that  $uKM_f$  is Ackermannian.

## References

1. C. Calude.: Theories of computational complexity, Annals of Discrete Mathematics volume 35, North-Holland, Amsterdam (1988)
2. L. Carlucci, G. Lee, A. Weiermann: A Sharp Phase Transition for Gödel Incompleteness and Finite Combinatorics.  
<http://www.lix.polytechnique.fr/~leegy/Publi/kmjams.pdf>
3. P. Cholak, C. Jockusch, T. Slaman On the strength of Ramsey's theorem for pairs. *J. Symbolic Logic* 66 (2001), no. 1, 1–55.
4. P. Erdős, A. Hajnal, A. Máté, and R. Rado. *Combinatorial set theory: partition relations for cardinals*, volume 106 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1984.
5. P. Erdős and R. Rado. Combinatorial theorems on classifications of subsets of a given set. *Proc. London Math. Soc. (3)*, 2:417–439, 1952.
6. K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. *Monatshefte f. Math. u. Phys.*, 38:173–198, 1931.
7. R. L. Graham, B. L. Rothschild, and J. H. Spencer. *Ramsey theory*. Second edition. John Wiley & Sons Inc., New York, 1990.
8. A. Kanamori and K. McAlloon. On Gödel incompleteness and finite combinatorics. *Ann. Pure Appl. Logic*, 33(1):23–41, 1987.

9. J. Ketonen and R. Solovay. Rapidly growing Ramsey functions. *Ann. of Math.* (2), 113(2):267–314, 1981.
10. M. Kojman, S. Shelah: Regressive Ramsey Numbers Are Ackermannian, *Journal of Combinatorial Theory, Series A* 86, 177-181 (1999)
11. M. Kojman, G. Lee, E. Omri and A. Weiermann. Sharp Thresholds for the Phase Transition between Primitive Recursive and Ackermannian Ramsey Numbers. To appear in *Journal of Combinatorial Theory*.  
<http://www.math.bgu.ac.il/~kojman/Thresh.pdf>
12. J. Mileti, Partition theorems and computability theory, dissertation at UIUC (2004)
13. J. B. Paris. Some independence results for Peano arithmetic. *J. Symbolic Logic*, 43(4):725–731, 1978.
14. J. B. Paris and L. Harrington. A mathematical incompleteness in Peano arithmetic. In *J. Barwise, ed., Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 1133–1142. North-Holland, 1977.
15. R. Péter. *Recursive functions*. Third edition. Academic Press, New York, 1967.
16. F. P. Ramsey. On a problem of formal logic. *Proc. London Math. Soc.*, 30:264–285, 1930.
17. D. Richer: Unordered Canonical Ramsey Numbers. *Journal of Combinatorial Theory, Series B* 80, 172-177 (2000)
18. S.G. Simpson: Subsystems of Second Order Arithmetic.
19. A. Weiermann: 2005: Analytic combinatorics, proof-theoretic ordinals, and phase transitions for independence results. APAL 136, Issues 1-2 , 189-218.
20. A. Weiermann: Phasenübergänge in Logik und Kombinatorik. MDMV 13 (3) (2005), 152-156.
21. A. Weiermann: An extremely sharp phase transition threshold for the slow growing hierarchy. MSCS 16 (5) (2006), 925-46.
22. A. Weiermann: Phase transition thresholds for some natural subclasses of the recursive functions. Proceedings of CiE'06, LNCS 3988 (2006), 556-570.
23. A. Weiermann: 2007 Phase transition thresholds for some Friedman-style independence results. MLQ. 53 (1), (2007) 4-18.
24. A. Weiermann. A classification of rapidly growing Ramsey functions. *Proc. Amer. Math. Soc.*, 132(2):553–561, 2004.

# Almost Partial m-Reducibility

Katya Petrova and Boris Solon

Ivanovo State University of Chemistry and Technology

**Abstract.** New reducibility (the so-called apm-reducibility) of enumeration type which is weaker than pm-reducibility is introduced in this paper. Initial segments of the upper semilattice of apm-degrees are studied here.

The notations and terminology similar to those of the monograph [3] are used. Let  $\omega$  denote the set of positive integers;  $A, B, \dots, X, Y$  (with or without indices) are used to denote the subsets of  $\omega$  and  $\bar{A} = \omega - A$ . We will use the letters  $V$  and  $W$  as variables which range over a set of all c.e. sets. Given a partial function  $\alpha : \omega \rightarrow \omega$  let  $\text{dom}\alpha$ ,  $\text{ran}\alpha$  and  $\text{graph}\alpha = \{\langle x, \alpha(x) \rangle : x \in \text{dom}\alpha\}$  be the domain, the range and the graph of  $\alpha$  respectively. We will write  $\alpha(x) \downarrow$  if  $x \in \text{dom}\alpha$  and  $\alpha(x) \uparrow$  if  $x \notin \text{dom}\alpha$ . Let  $\alpha^{-1}(X) = \{x : x \in \text{dom}\alpha \wedge \alpha(x) \in X\}$ . We will use the letters  $\varphi$  and  $\psi$  as variables which range over a set of all p.c. functions.

Yu.L. Ershov introduced in [1] *partial m-reducibility* of sets:

$$\begin{aligned} A \leq_{pm} B &\iff (\exists \varphi)(\forall x)[x \in A \iff x \in \text{dom}\varphi \wedge \varphi(x) \in B] \iff \\ &\iff (\exists \varphi)[A = \varphi^{-1}(B)]. \end{aligned}$$

In this case we will talk that  $A$  is pm-reducible to  $B$  via  $\varphi$ .

In this paper we introduce a reducibility which slightly weakens pm-reducibility :

$$\begin{aligned} A \leq_{apm} B &\iff (\exists W)(\exists A_1)[A = W \cup A_1 \wedge A_1 \leq_{pm} B] \iff \\ &\iff (\exists W)(\exists \varphi)[A = W \cup \varphi^{-1}(B)]. \end{aligned}$$

In this case we will talk that  $A$  is *apm-reducible* to  $B$  via  $(W, \varphi)$ . These new ideas of defining the reducibility were proposed in 70-s by S.D.Zakharov.

It is clear that if  $A \leq_{pm} B$  via  $\varphi$  then  $A \leq_{apm} B$  via  $(\emptyset, \varphi)$ . It is easy to check that apm-reducibility is reflexive and transitive, hence it is a reducibility in ordinary sense. Also it is clear that  $A \leq_{apm} B \Rightarrow A \leq_e B$  for all  $A$  and  $B$ , i.e. apm-reducibility is a reducibility of enumeration type.

Let  $\deg_{apm}(A) = \{X : X \leq_{apm} A \wedge A \leq_{apm} X\}$  be *apm-degree* of  $A$  and  $\mathbf{D}_{apm}$  be the p.o. set of all apm-degrees.

**Theorem 1**  $\mathbf{D}_{apm}$  is an upper semilattice with least element  $\mathbf{0}_{apm} = \{W_t : t \in \omega\}$  in which the least upper bound of the apm-degrees  $\mathbf{a} = \deg_{apm}(A)$  and  $\mathbf{b} = \deg_{apm}(B)$  is  $\mathbf{a} \cup \mathbf{b} = \deg_{apm}(A \oplus B)$ .

**Proof.** Obviously that  $W \leq_{apm} A$  via  $(W, \phi)$  where  $\phi$  is p.c. function with  $\text{dom}\phi = \emptyset$ . Next let  $A \leq_{apm} C$  via  $(V_1, \psi_1)$  and  $B \leq_{apm} C$  via  $(V_2, \psi_2)$ , we set  $V = V_1 \cup V_2$  and  $\psi = \psi_1 \oplus \psi_2$ . Then it is clear that  $A \oplus B \leq_{apm} C$  via  $(V, \psi)$ .

The following theorem gives an example of two sets such that  $A \not\leq_{pm} B \wedge A \leq_{apm} B$ , i.e. pm-reducibility is stronger than apm-reducibility.

**Theorem 2** *There are the sets  $A$  and  $B$  such that  $A \not\leq_{pm} B$  and  $A \leq_{apm} B$ .*

**Proof.** At first we construct c.e. sets  $W$  and  $V$  such that

- (i)  $\overline{V}$  is infinite;
- (ii)  $\forall y[|\text{ran}\varphi_y| = \infty \Rightarrow (\exists x)[x \in W \cap \text{dom}\varphi_y \wedge \varphi_y(x) \in V]]$ .

*Step 0.* Set  $W = V = \emptyset$ .

*Step z+1.* Let  $z = \langle k, y \rangle$ . Compute  $k$  steps in enumerations for each of the  $y + 1$  c.e. sets  $\text{dom}\varphi_0, \dots, \text{dom}\varphi_y$ . For each  $i = 0, 1, \dots, y$  see whether

$$(\exists x)[x \in \text{dom}\varphi_i \wedge \varphi_i(x) > 2i]. \quad (1)$$

If (1) is true then let  $x^*$  be the least number  $x$  which satisfies (1). In this case we place  $x^*$  in  $W$  and  $\varphi_i(x^*)$  in  $V$ .

The end of the construction.

As a result we have

$$|\{0, 1, \dots, 2y\} \cap V| \leq y$$

for every  $y \in \omega$ . Hence the set  $\overline{V}$  is infinitive. The construction is such that all steps  $z + 1$  are computable, hence sets  $W$  and  $V$  are c.e. So the conditions (i) and (ii) are satisfied.

Now we construct a set  $B \subset \overline{V}$  such that  $W \cup \{\langle x, x \rangle : x \in B\}$  is not c.e. Let  $Z = \{z : W_z \supseteq W\}$ ,  $Z = \{z_0, z_1, \dots\}$  and  $\overline{V} = \{c_0, c_1, \dots\}$ . (Both enumerations are not c.e.)

For every  $i = 0, 1, 2, \dots$  we see whether

$$\langle c_i, c_i \rangle \in W_{z_i}. \quad (2)$$

If (2) is true then let  $j > i$  such that  $W_{z_j} = W_{z_i}$  then we place  $c_j$  in  $B$  and we do not place  $c_i$  in  $B$ . If (2) is not true then we place  $c_i$  in  $B$ . As a result we have  $W \cup \{\langle x, x \rangle : x \in B\} \neq W_z$  for all  $z \in \omega$ , i.e.  $W \cup \{\langle x, x \rangle : x \in B\}$  is not c.e. set.

Let  $\varphi(z)$  be a p.c. function such that  $\forall z[\varphi(z) \downarrow \iff \exists x[\langle x, x \rangle = z]]$  and  $\forall x[\varphi(\langle x, x \rangle) = x]$ . As  $\{\langle x, x \rangle : x \in B\} \leq_{pm} B$  via  $\varphi$  then  $W \cup \{\langle x, x \rangle : x \in B\} \leq_{apm} B$  via  $(W, \varphi)$ .

Prove that  $W \cup \{\langle x, x \rangle : x \in B\} \not\leq_{pm} B$ . We assume that

$$W \cup \{\langle x, x \rangle : x \in B\} \leq_{pm} B$$

then  $W \cup \{\langle x, x \rangle : x \in B\} = \varphi_y^{-1}(B)$  for some  $y \in \omega$ . If  $\text{ran}\varphi_y$  is infinite then

$$\exists x[x \in W \cap \text{dom}\varphi_y \wedge \varphi_y(x) \in V].$$

In this case the construction of  $B$  guarantees that

$$\exists x[x \in W \cap \text{dom}\varphi_y \wedge \varphi_y(x) \notin B]$$

what contradicts the premise.

If  $\text{ran}\varphi_y$  is finite then  $W \cup \{\langle x, x \rangle : x \in B\}$  is a c.e. set that contradicts the premise too.

Let  $A = W \cup \{\langle x, x \rangle : x \in B\}$ , then it is clear that  $A \not\leq_{pm} B$  and  $A \leq_{apm} B$  and the theorem is proved completely.

Very simple properties of amp-reducibility are given in the following

**Theorem 3** (i). If  $A$  differs from  $B$  on a finite set then  $A \equiv_{apm} B$ ;

(ii). If  $V$  is c.e. set then  $A \cup V \leq_{apm} A$  for any  $A$ ;

(iii). For any  $A$  and c.e.  $V$  if there is c.e.  $W$  such that  $A \subseteq W$  and  $W \cap V = \emptyset$  then  $A \equiv_{apm} A \cup V$ .

A set  $A$  is called *cohesive* [2] if  $A$  is infinite and  $A \cap V$  is finite or  $A \cap \bar{V}$  is finite for any c.e. set  $V$ . In other words an infinite set  $A$  is cohesive iff it cannot be divided into two infinite parts by a c.e. set.

Let  $(\mathbf{D}, \leq)$  be a partial ordering,  $\mathbf{a}, \mathbf{b} \in \mathbf{D}$  and  $\mathbf{b} < \mathbf{a}$ . The element  $\mathbf{a}$  is called **b-minimal** if

$$\forall \mathbf{x}[\mathbf{x} \in \mathbf{D} \wedge \mathbf{x} \leq \mathbf{a} \Rightarrow [\mathbf{x} \leq \mathbf{b} \vee \mathbf{a} \leq \mathbf{x}]].$$

Let  $\mathbf{0}$  be the least element in  $\mathbf{D}$ ;  $\mathbf{0}$ -minimal element  $\mathbf{a}$  is called *minimal*.

M.Rozinas proved that a pm-degree of cohesive set is a minimal element in  $\mathbf{D}_{pm}$ . The following theorem shows that this is true in  $\mathbf{D}_{apm}$ .

**Lemma 1** If  $A \leq_{apm} B$  via  $(V, \psi)$  then  $A \equiv_{apm} (B \cup V) \cap \text{ran}\psi$ .

**Proof.** Let  $A \leq_{apm} B$  via  $(V, \psi)$  then  $A = V \cup \psi^{-1}(B)$ . It is clear that  $\psi^{-1}(B) = \psi^{-1}(B \cap \text{ran}\psi)$  thus  $A = V \cup \psi^{-1}(B \cap \text{ran}\psi) = V \cup \psi^{-1}((B \cup V) \cap \text{ran}\psi)$ . Then  $A \leq_{apm} (B \cup V) \cap \text{ran}\psi$  via  $(V, \psi)$ .

Conversely, let  $W = \{\langle y, x \rangle : \langle x, y \rangle \in \text{graph}\psi\}$ . It is well known that there is p.c.  $\varphi$  such that  $\text{dom}\varphi = \langle W \rangle_1 = \text{ran}\psi$  and  $\text{graph}\varphi \subseteq W$ . Let  $V^{-1} = \varphi^{-1}(V)$ , now we will prove that

$$(V \cup B) \cap \text{ran}\psi = V^{-1} \cup \varphi^{-1}(A).$$

We have for any  $y \in \omega$

$$\begin{aligned} y \in (V \cup B) \cap \text{ran}\psi &\iff y \in V \cap \text{ran}\psi \vee y \in B \cap \text{ran}\psi \Rightarrow \\ &\Rightarrow \varphi(y) \downarrow \wedge \varphi(y) = x \in (V \cup A) \Rightarrow y \in V^{-1} \cup \varphi^{-1}(A) \end{aligned}$$

and

$$y \in V^{-1} \cup \varphi^{-1}(A) \Rightarrow y \in V^{-1} \vee y \in \varphi^{-1}(A).$$

We consider each case separately:

$$y \in V^{-1} \iff \varphi(y) \downarrow \wedge \varphi(y) = x \in V \Rightarrow y \in V \cap \text{ran}\psi \Rightarrow y \in (V \cup B) \cap \text{ran}\psi$$

and

$$y \in \varphi^{-1}(A) \iff \varphi(y) \downarrow \wedge \varphi(y) = x \in A \Rightarrow y \in B \cap \text{ran}\psi \Rightarrow y \in (V \cup B) \cap \text{ran}\psi.$$

So  $(V \cup B) \cap \text{ran}\psi = V^{-1} \cup \varphi^{-1}(A)$ , it implies  $(B \cup V) \cap \text{ran}\psi \leq_{apm} A$ . Now let's move to

**Theorem 4** *There are minimal elements in  $\mathbf{D}_{apm}$ .*

First we prove a lemma on cohesive sets.

**Lemma 2** *There is cohesive set  $B$  such that*

$$\forall n [|W_n| = \infty \iff |W_n \cap B| = \infty] \quad (3)$$

**Proof of the lemma.** Let  $B_{-1} = \omega$ , next by induction on  $n$  if  $|B_{n-1} \cap W_n| = \infty$  then set  $B_n = B_{n-1} \cap W_n$  and if  $|B_{n-1} \cap W_n| < \infty$  then set  $B_n = B_{n-1} \cap \overline{W_n}$ .

Let  $b_0 \in B_0, b_1 \in B_1, \dots, b_n \in B_n, \dots$  such that  $b_0 < b_1 < \dots < b_n < \dots$ , set  $B = \{b_i : i \in \omega\}$ . Prove that  $B$  is a cohesive set and satisfies (3).

Assume that  $B$  is not a cohesive set then  $|B \cap W_m| = \infty$  and  $|B \cap \overline{W_m}| = \infty$  for some  $m$ . According our construction  $B_m \subset W_m$  or  $B_m \subset \overline{W_m}$ . Since all but a finite number of members of  $B$  must lie in  $B_m$ , either  $|B \cap W_m| < \infty$  or  $|B \cap \overline{W_m}| < \infty$  that contradicts the premise.

Now prove that (3) is true. Let  $|W_n| = \infty$  and let  $m_0 < m_1 < \dots$  be an infinite sequence such that  $W_n = W_{m_i}$  for all  $i \in \omega$ . It is clear that  $b_{m_i} \in B \cap W_n$  for all  $i \in \omega$ , hence  $|B \cap W_n| = \infty$ . The lemma is proved.

**Proof of the theorem.** Let  $B$  be a cohesive set satisfying (3) and  $\mathbf{b} = \deg_{apm}(B)$ . It is sufficient to show that

$$\forall \mathbf{x} [\mathbf{x} \in \mathbf{D}_{apm} \wedge \mathbf{x} \leq \mathbf{b} \Rightarrow [\mathbf{x} = \mathbf{0}_{apm} \vee \mathbf{b} \leq \mathbf{x}]].$$

Let  $\mathbf{a} = \deg_{apm}(A)$  and  $A \leq_{apm} B$  via  $(V, \psi)$ , i.e.  $A = V \cup \psi^{-1}(B)$ . Lemma 1 implies  $A \equiv_{apm} (B \cup V) \cap \text{ran}\psi$ .

As  $B$  is cohesive then two alternative cases are possible:  $B \cap \text{ran}\psi$  is finite or  $B \cap \overline{\text{ran}\psi}$  is finite. Let  $B \cap \text{ran}\psi$  be finite then  $(B \cup V) \cap \text{ran}\psi$  is c.e., hence we have  $\mathbf{a} = \mathbf{0}_{apm}$ .

Let  $B \cap \overline{\text{ran}\psi}$  is finite. In this case as  $B$  is cohesive two alternative cases are possible:  $V \cap B$  is finite or  $\overline{V} \cap B$  is finite. Let  $V \cap B$  is finite then  $V$  is a finite set by (3). In this case we have

$$A \equiv_{apm} (B \cup V) \cap \text{ran}\psi = (B \cap \text{ran}\psi) \cup (V \cap \text{ran}\psi) \equiv_{apm} B \cap \text{ran}\psi \equiv_{apm} B$$

and so we have  $\mathbf{a} = \mathbf{b}$ .

If  $\overline{V} \cap B$  is finite then  $B \cup V$  differs from  $V$  on a finite set so in this case we have  $\mathbf{a} = \mathbf{0}_{apm}$ . The theorem is proved completely.

**Theorem 5** *If  $B_1, \dots, B_n, n \geq 1$  are cohesive sets satisfying (3) and their apm-degrees are different in pairs then the initial segment  $[\mathbf{0}_{apm}, \deg_{apm}(B_1 \oplus \dots \oplus B_n)] \subset \mathbf{D}_{apm}$  is isomorphic to Boolean algebra of subsets of a  $n$ -element set.*

**Proof.** Let  $B = B_1 \oplus \dots \oplus B_n$  and  $A \leq_{apm} B$ , we show that in this case  $A$  is c.e. or  $A \equiv_{apm} \bigcup_{i \in I} B_i^*$  where  $B_i^* = \{xn + i : x \in B_i\}$  for some  $I \subseteq \{1, \dots, n\}$ ,  $I \neq \emptyset$ .

Let  $A \leq_{apm} B$  via  $(W, \psi)$  and  $V = \text{ran}\psi$ . Let

$$I = \{i : 1 \leq i \leq n \wedge |V \cap B_i^*| = \infty\}.$$

If  $I = \emptyset$  then  $V \cap B_i^*$  is finite for all  $i = 1, \dots, n$ , thus  $V \cap B$  is finite and then  $A$  is a c.e. set.

Let  $I \neq \emptyset$ . Since  $B_i$  is a cohesive set satisfying (3) then  $B_i^*$  is a cohesive set satisfying (3) too, thus  $\bar{V} \cap B_i^*$  is finite for all  $i \in I$ . Then

$$|\bar{V} \cap (\cup_{i \in I} B_i^*)| = |\cup_{i \in I} B_i^* - V| < \infty.$$

In this case the set  $\cup_{i \in I} B_i^*$  differs from  $\cup_{i \in I} (B_i^* \cap V)$  on a finite set, thus

$$\bigcup_{i \in I} B_i^* \equiv_{apm} \bigcup_{i \in I} (B_i^* \cap V). \quad (4)$$

Let  $i \in \{1, \dots, n\} - I$  then the set  $B_i^* \cap V$  is finite, thus the set  $B \cap V = \cup_{i=1}^n (B_i^* \cap V)$  differs from  $\cup_{i \in I} (B_i^* \cap V)$  on a finite set. So

$$B \cap V \equiv_{apm} \cup_{i \in I} (B_i^* \cap V). \quad (5)$$

By Lemma 1  $A \leq_{apm} B$  via  $(W, \psi)$  implies

$$A \equiv_{apm} (B \cup W) \cap \text{ran } \psi = (B \cap V) \cup (W \cap V).$$

Taking into consideration the fact that the set  $V \cap W$  may be finite or infinite we conclude that  $A$  is c.e. or  $A \equiv_{apm} B \cap V$ . In the last case we have  $A \equiv_{apm} \cup_{i \in I} B_i^*$ , as every  $B_i^*$  satisfies (3).

Now we show that for any  $I_1 \subseteq \{1, \dots, n\}$  and  $I_2 \subseteq \{1, \dots, n\}$

$$I_1 \subseteq I_2 \iff \cup_{i \in I_1} B_i^* \leq_{apm} \cup_{i \in I_2} B_i^*.$$

If  $I_1 \subseteq I_2$  then  $\cup_{i \in I_1} B_i^* \leq_{apm} \cup_{i \in I_2} B_i^*$  via  $(\emptyset, \alpha)$  where  $\alpha(x) \downarrow \iff x \in \cup_{i \in I_1} \{nx + i : x \in \omega\}$  and  $x \in \text{dom } \alpha \Rightarrow \alpha(x) = x$ .

Let  $\cup_{i \in I_1} B_i^* \leq_{apm} \cup_{i \in I_2} B_i^*$ . We assume that  $I_1 \not\subseteq I_2$  and  $j \in I_1 - I_2$ . It is clear that  $\{j\} \subseteq I_1$  implies

$$B_j^* \leq_{apm} \cup_{i \in I_1} B_i^* \leq_{apm} \cup_{i \in I_2} B_i^*.$$

Let  $B_j^* \leq_{apm} \cup_{i \in I_2} B_i^*$  via  $(W^*, \varphi)$  i.e.  $B_j^* = W^* \cup (\cup_{i \in I_2} \varphi^{-1}(B_i^*))$ . Denote by  $V_i = \varphi^{-1}(\{nx + i : x \in \omega\})$  for all  $i \in \{1, \dots, n\}$ . As the  $B_j^*$  is a cohesive set and  $B_j^* \subseteq W^* \cup (\cup_{i \in I_2} V_i)$  then there is  $l \in I_2$  (note that  $l \neq j$ ) such that  $B_j^* \cap V_l$  is an infinite set. Otherwise  $B_j^* = W^* \cup D$  for some finite set  $D$  and then  $B_j^*$  is c.e. that contradicts the premise.

Let  $\varphi^*$  be a restriction of  $\varphi$  to  $V_l$ . Obviously  $B_j^* \cap V_l \leq_{apm} B_l^*$  via  $(W^* \cap V_l, \varphi^*)$ . As  $B_j^* \cap \bar{V}_l$  is a finite set then

$$B_j^* \equiv_{apm} B_j^* \cap V_l \leq_{apm} B_l^* \Rightarrow B_j \leq_{apm} B_l.$$

As the  $B_l^*$  is a cohesive set satisfying (3) then by Theorem 4 the set  $B_j^*$  is c.e. or  $B_l \leq_{apm} B_j$ . But the first is not possible as  $B_j$  is a cohesive set and

the second is not possible as  $B_j$  and  $B_l$  should be in a distinct apm-degrees by hypothesis of Theorem. So we get a contradiction that proves

$$\cup_{i \in I_1} B_i^* \leq_{apm} \cup_{i \in I_2} B_i^* \Rightarrow I_1 \subseteq I_2.$$

Let  $\Xi$  be the mapping from the set of all non-empty subsets of  $\{1, \dots, n\}$  into  $\mathbf{D}_{apm}$ , which is defined by

$$\Xi(I) = \deg_{apm}(\cup_{i \in I} B_i^*)$$

for all  $\emptyset \neq I \subseteq \{1, \dots, n\}$  and  $\Xi(\emptyset) = \mathbf{0}_{apm}$ . We proved above that  $\Xi$  is an order-preserving embedding Boolean algebra of subsets of  $\{1, \dots, n\}$  to the initial segment  $[\mathbf{0}_{apm}, \deg_{apm}(B_1 \oplus \dots \oplus B_n)] \subset \mathbf{D}_{apm}$ . The theorem is proved completely.

**Corollary 1** *For all  $n \geq 1$   $\mathbf{D}_{apm}$  contains a continuum of initial segments each of which is isomorphic to Boolean algebra of subsets of a  $n$ -element set.*

In conclusion we would like to announce another weak pm-reducibility:

$$A \leq_{wpm} B \iff (\exists k)(\exists \psi_1, \dots, \psi_k)[A = \psi_1^{-1}(B) \cup \dots \cup \psi_k^{-1}(B)].$$

In this case we will talk that  $A$  is *wpm-reducible* to  $B$  via  $(\psi_1, \dots, \psi_k)$ .

It is clear that  $\leq_{wpm}$  is reflexive and transitive, hence it is a reducibility in ordinary sense. It is also clear that  $A \leq_{apm} B \Rightarrow A \leq_{wpm} B$  and  $A \leq_{wpm} B \Rightarrow A \leq_e B$  (moreover  $A \leq_{wpm} B \Rightarrow A \leq_s B$ ) for all  $A$  and  $B$ , i.e. apm-reducibility is a reducibility of enumeration type. We are able to construct two sets  $A$  and  $B$  such that  $A \not\leq_{apm} B \wedge A \leq_{wpm} B$ .

We invite to study this wpm-reducibility as a reducibility of enumeration type which has a good intuitive background.

## References

1. Ershov Yu. L.: Numbering Theory. "Nauka". Moskou. 1977
2. Rogers H., Jr.: Theory of Recursive Functions and Effective Computability. McGraw-Hill. New York. 1967
3. Soare Robert I.: Recursively Enumerable Sets and Degrees. Springer-Verlag. Berlin, Heidelberg, New York, London. 1987

# Two-Dimensional Cellular Automata Transforms for a Novel Edge Detection

Yongri Piao<sup>1</sup>, \*Seok-Tae Kim<sup>1</sup>, Sung-Jin Cho<sup>2</sup>

<sup>1</sup>Department of Telematics, Pukyong National University,  
599-1, Daeyeon 3-Dong, Nam-Gu, Busan, Republic of Korea (608-737)  
pyr-bww@hanmail.net, setakim@pknu.ac.kr  
<sup>2</sup>Division of Mathematical Sciences, Pukyong National University,  
599-1, Daeyeon 3-Dong, Nam-Gu, Busan, Republic of Korea (608-737)  
sjcho@pknu.ac.kr

**Abstract.** In this paper, we propose a novel edge detection scheme using two-dimensional cellular automata transforms (CAT). Cellular Automata (CA) is discrete dynamical system whose function is completely specified in terms of local relation. First, we get the gateway values such as wolfram Rule, number of cells in lattice, number of cells per neighborhood, initial configuration and boundary configuration. Second, we use the gateway values to generate a dual-state, two-dimensional cellular automata and dual-coefficients basis function. Finally, we transform images into cellular automata domain according to the basis function. Then we use the basis function and cellular automata transform coefficients to extract the edge of the image. The experimental results verify that the proposed scheme is a new attempt to process image with cellular automata model.

## 1 Introduction

Sharp variations of image intensity could be described by edges, which convey important information in an image. Edge detection is a pivotal technique in pattern recognition, image procession, and computer vision.

One edge detection algorithm can not be applied to all of the images. Based on the complexity in edge detection, each algorithm has positive and negative performance. For instance, Smith et al. [1] proposed low-level feature extraction-SUSAN operator. It attempted to provide robust signature for every edge point. The method proposed by Smith et al, however, is strongly influenced by the presence of edges and corners. On the other hand, noise contamination is always a problem and edge detection in noisy environment can be treated as an optimal linear filter design problem [2-6]. The Canny [3] edge detector is more accurate, though more complex. It employs Gaussian first order differential equation which is able to reach the balance between robust noise and edge detection. Unfortunately, the Canny detector failed to verify whether discontinuity is contributed by noise or true edges. Unlike previous works, we present

---

\* Corresponding Author. Tel.: +82-51-629-6234; Fax: +82-51-629-6210

a novel edge detection method in the CAT. The essence of CAT is that we can always find CA rules (and its associated neighborhood, initial/boundary configuration, lattice arrangement etc) which will result in basis function and transform coefficients.

CA was originally introduced by Ulam [7] and Von Neumann [8]. Afterwards, the cellular automata theory was developed by Stephen Wolfram [9-10]. Now cellular automata have generated much interest because of their diverse functions and usefulness as a discrete model for many processes [11-13]. Along with the development of the digital image technology, some efforts are expended in using CA in image processing. C. L. Chang et al. [14] proposed an image edge detection using CA. They defined a new kind of neighborhood of the CA. Then a suitable local rule of the CA is designed. However, their method only uses the relationship of the neighborhood to detect edges in the spatial domain and is weak at noise.

In this paper, we propose a novel edge detection method using two-dimensional cellular automata transforms. After we get the gateway values (such as Wolfram Rule, the number of cells in lattice, the number of cells per neighborhood, the initial configuration, and the boundary configuration), we use them to generate a dual-state, two-dimensional cellular automata dual-coefficients basis function. Then the basis function and their associated transform coefficients transform an image into cellular automata domain components to extract the edge of the image. The experimental results verify that the proposed method is not only a new attempt to detect edge detection with the cellular automata model but also is proved to be very efficient.

## 2 Cellular Automata Basics

CA is a dynamical system in which space and time are discrete. The cells, which are arranged in the form of a regular lattice structure, have a finite number of states. These states are updated synchronously according to a specified local rule of interaction. Using a specified rule, the values are updated synchronously in discrete time steps for all cells[15].

In general, for a K-state, m-site neighborhood CA, there are  $K^{K^m}$  rules. For example, there are  $2^8$  rules for the two- state, 3-site neighborhood CA. The Wolfram Rule convention is to assign the integer R to rule generating the function F such that:

$$R = \sum_{n=0}^{K^m-1} C_n 2^n \quad (1)$$

where  $C_n$  is the Boolean value generated by the rule given the **n-th** configuration.

Consider a 3-site neighborhood dual-state one-dimensional CA. The quantity  $a_{it}$  represents the state of the i-th cell, at discrete time t, whose two neighbors are in the following states:  $a_{i-1t}$ ,  $a_{i+1t}$ . In general, we seek a rule that will be used to synchronously calculate the state  $a_{it+1}$  from the state of the cells in the neighborhood at the t-th time level. The cellular automaton evolution can be expressed in the form:

$$a_{it+1} = F(a_{i-1t}, a_{it}, a_{i+1t}) \quad (2)$$

where  $F$  is a Boolean function defining the rule.

The number of cellular automata rules can be astronomical even for a modest lattice space, neighborhood size, and CA state. Therefore, in order to develop practical applications, a system must be developed for addressing a subset of this infinitely large universe of CA rules. Consider, for example, a  $K$ -state,  $m$ -site neighborhood CA with  $m=2r+1$  points per neighborhood. We define the rule of evolution of a cellular automaton by using a vector of integers  $W_j$  ( $j = 0, 1, 2, 3, \dots, 2^m$ ) such that

$$a_{(r)(t+1)} = \left( \sum_{j=0}^{2^m-2} W_j \alpha_j + W_{2^m-1} \right)^{W_{2^m}} \bmod K$$

(3)

where  $0 \leq W_j < K$ ,  $r$  is a distance.  $\alpha_j$  are made up of the permutations of the states of the cells in the neighborhood. Hence, each set of  $W_j$  results in a given rule of evolution.

### 3 Cellular Automata Transforms

Given a process described by a function  $f$ , defined in physical space of lattice grid  $i$ , we seek basis function  $A$  and their associated transform coefficients  $c$ , defined in cellular automata space  $k$  [15]. We write

$$f_i = \sum_k c_k A_{ik} \quad (4)$$

Equation (4) represents a mapping of the process  $f$  (in the physical domain) in to  $c$  (in the cellular automata domain) using the building blocks  $A$  as transfer functions.

In a 2-D square space consisting of  $N \times N$  cells, the transform base  $A = A_{ijkl}$ , ( $i, j, k, l = 0, 1, \dots, N - 1$ ). For the data sequence  $f_{ij}$  ( $i, j = 0, 1, 2, \dots, N - 1$ ) we can write:

$$f_{ij} = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} c_{kl} A_{ijkl} \quad (i, j = 0, 1, \dots, N - 1) \quad (5)$$

in which  $c_{kl}$  are the transform coefficients.

There are two approaches for generating two-dimensional CA transform bases:

1. Using the evolving states derived from two-dimensional cellular spaces.  
Here,  $A_{ijkl}$  is calculated from  $a \equiv a_{ijt}$ , ( $i, j, t = 0, 1, \dots, N - 1$ ).
2. Calculating the canonical products of one-dimensional bases so that  $A_{ijkl} = A_{ik} A_{jl}$ .

## 4 Edge Detection Using CAT

In this section, we introduce the edge detection scheme using two-dimensional cellular automata transforms in details.

**Step 1:** We get the gateway values (wolfram Rule, number of cells in lattice, number of cells per neighborhood, initial configuration and boundary configuration etc.).

**Step 2:** We use the gateway values to generate a dual-state and two-dimensional cellular automata dual-coefficients basis function.

Consider, for example, a 2-state 8-node CA with 3points per neighborhood. In Table 1, we show the gateway values for generating a dual-state, two-dimensional CA dual –coefficient basis function.

**Table 1.** Gateway Values for 2-D dual-coefficient basis function

<b>Wolfram Rule Number</b>	<b>46</b>
<b>Number of Cells per Neighborhood</b>	<b>3</b>
<b>Number of Cells in Lattice</b>	<b>8</b>
<b>Initial Configuration</b>	<b>00001001</b>
<b>Boundary Configuration</b>	<b>Cyclic</b>
<b>Basis Function Type</b>	<b>2</b>

The cyclic boundary conditions imposed on the end sites are of the form equation (6):

$$a_{-1,k} = a_{N-1,k} \quad a_{N,k} = a_{0k} \quad (6)$$

Then a dual-coefficient cellular automata basis function is described by equation (7):

$$A_{ik} = 2a_{ik}a_{ki} - 1 \quad (7)$$

where  $a_{ik}$  is the state of the CA at the node i at time  $t=k$ . The states are obtained from N (N=8) cells evolved from a specific initial configuration for N time steps.

**Step 3:** The CA basis function is derived from the 1-D types in form equation (8):

$$A_{jkl} = A_{ik}A_{jl} \quad (8)$$

2-D basis functions are derived from the evolving one-dimensional CA as below:

$$A_{ijkl} = L_w \left\{ \left( a_{ik}a_{ki} + a_{jl}a_{lj} \right) \bmod L_w \right\} - (L_w - 1) \quad (9)$$

where  $L_w \geq 2$  is the number of states of the automaton.

**Step 4:** The two-dimensional cellular automata basis function of the edge detection rule transforms images into cellular automata domain. Then we use the basis function and cellular automata transform coefficients to extract the edge of the image.

In Figure 1 we show the steps for generating two-dimensional dual-coefficient basis function. This method produces many types of these basis functions which can detect edges. Of the basis functions generated, Figure 2 graphically shows the one using the gateway values from Table 1.  $A_{00kl}$  is the block at the extreme upper left corner. The top row represents  $0 \leq j < 8$ ;  $i=0$ . The left column is  $j=0$ ;  $0 \leq i < 8$ .  $A_{ij00}$  is the upper left corner of each block. The white rectangular dots represent 1 while the black dots are -1.

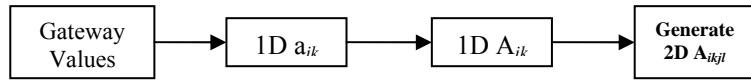


Figure 1. Process of 2D basis function

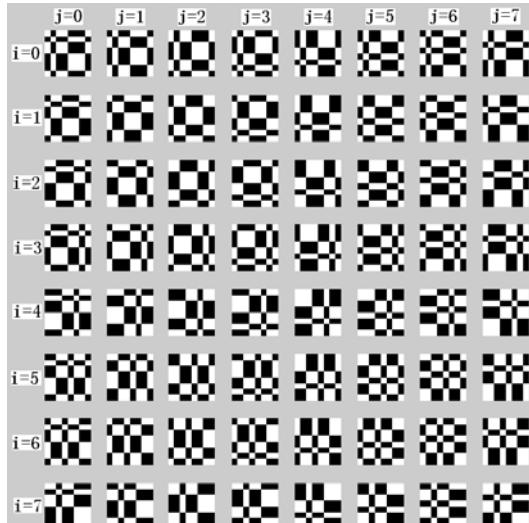


Figure 2. 2D basis function of Table 1 gateway values

## 5 Experimental Results

The performance of the proposed scheme is tested on various types of images. Here, the results are represented by three grayscale 8-bit images of size  $256 \times 256$  (Figure 3 (a)-(c)). Table 2 shows the wolfram rule and its initial configuration of the dual-coefficient CAT filters in detecting the edge of tested images. The experimental results consist of two parts. Part one is the performance of the proposed method on the edge detection of real images, including noise and non-noise images. Part two is the performance of different edge detectors by using a synthetic image.

### 5.1 Edge detection of clean and noisy images

The Sobel, Laplacian of Gaussian (LoG) and Canny detectors are the most common edge operators. Figure 3 (d)-(f) shows the extracted edge maps using 2D CAT filter (Figure 2). The edge maps of the test images obtained by the three detectors are shown in Figure 3 (g)-(o). Figure 3 (g)-(i), (j)-(l) and (m)-(o) show the extracted edge maps using Sobel, LoG and Canny detector in MATLAB, respectively. From the experimental results, we can find that the proposed method has good performance in the delicate region where the gray level is finely changing.

Figure 4 shows the proposed method compared with the Sobel, LoG and Canny detector under random noise. Figure 4 verifies that the Sobel, LoG and Canny edge detectors can detect the edges; though, noise pixels still exist in the extracted edge maps. The Sobel edge detector can remove the small speckles from the face and head. Using LoG and Canny edge detector to extract edge maps, it is difficult to find the true edges because of the noise. Even though the edges extracted by proposed edge detector include partial noise pixels, we can still easily find the true edges by using proposed CAT edge detector.

**Table 2.** Wolfram rule and initial configuration

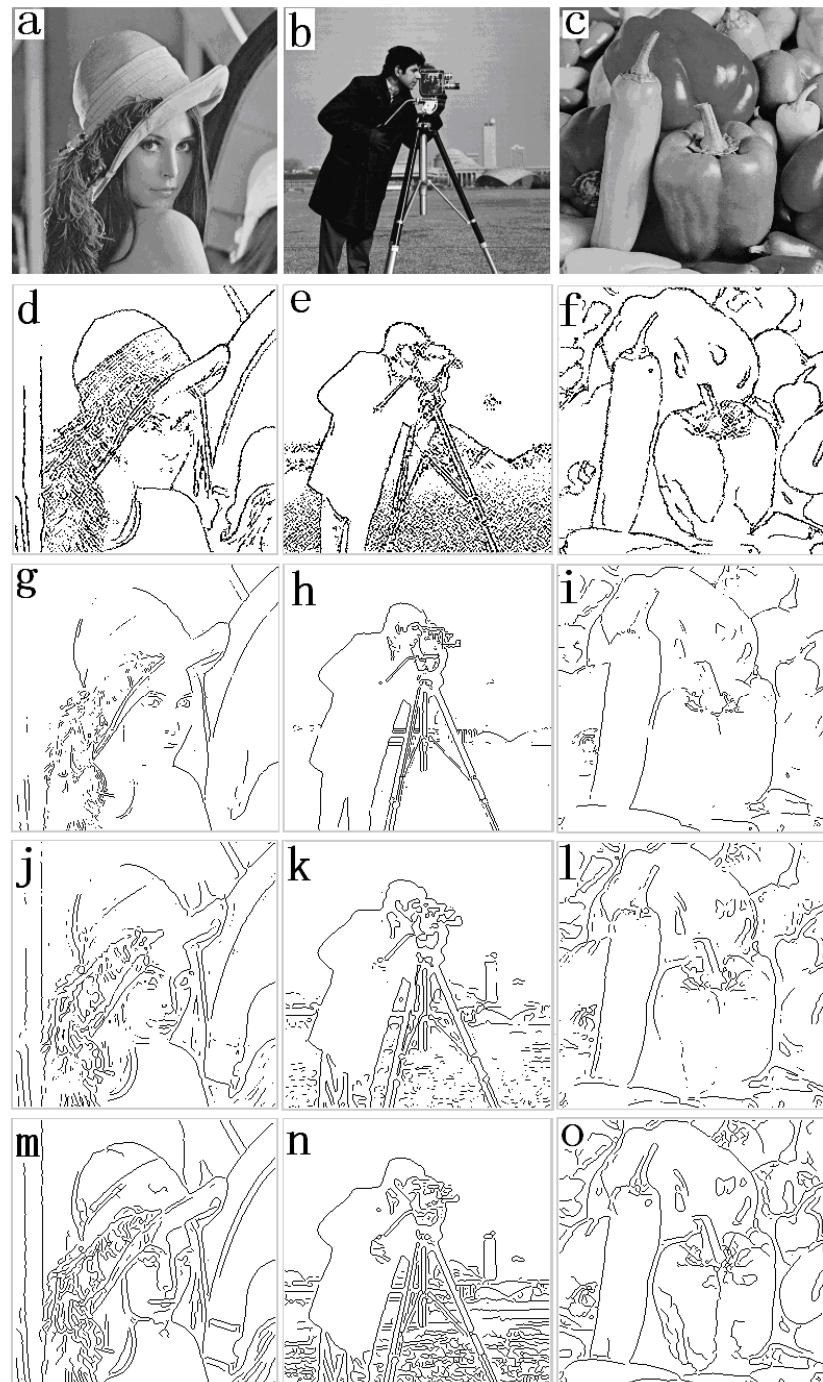
Filter	Wolfram rule	Initial configuration
A	46	18
B	155	137
C	174	33
D	209	28
E	231	109
F	245	232

### 5.2 Edge detection of a synthetic image

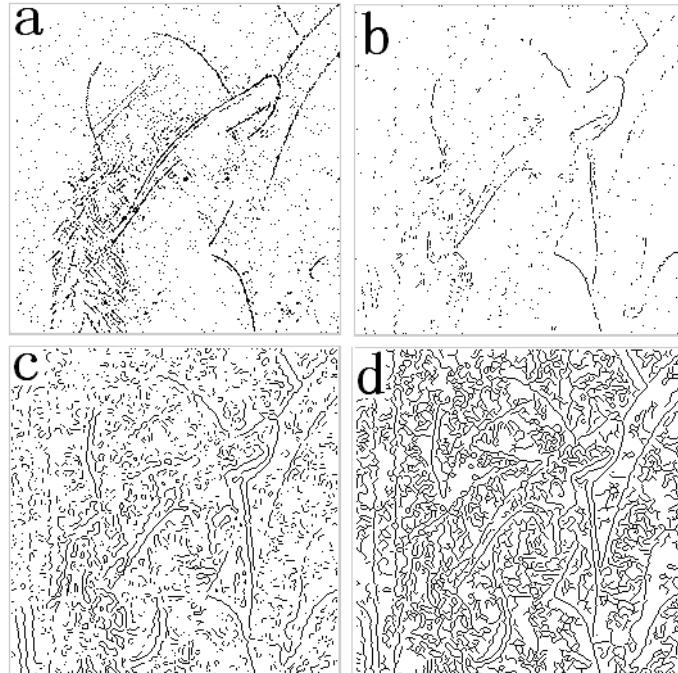
On the other hand, noise contamination is always a problem and edge detection in noisy environment can be treated as an optimal linear filter design problem. The performance of the proposed method comparing with other methods (Sobel, LoG and Canny) is tested on the synthetic image (Figure 5 (a)). That image is an 8-bit gray-scale image of size  $256 \times 256$ . The result of synthetic image added Gaussian noise as shown in Figure 5(b). The FOM (Figure of Merit) [16] is used to evaluate the performance of the edge detection methods. The FOM is defined as follows

$$F = \frac{1}{I_N} \sum_{i=1}^{I_A} \frac{1}{1 + \beta d_i^2}, \quad I_N = \max(I, I_A) \quad (10)$$

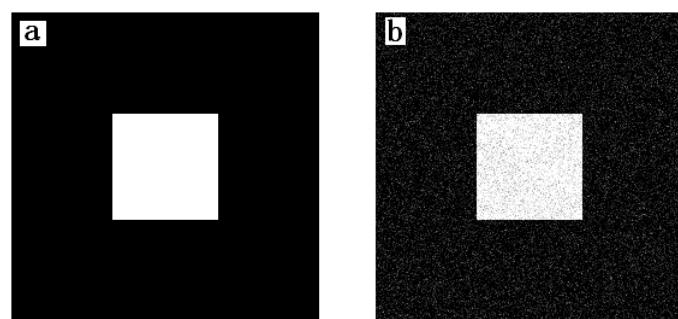
where  $I$  is original pixels of edge,  $I_A$  is detected pixels of edge,  $\beta$  ( $\beta=1/9$ ) is scaling parameter,  $d_i$  is distance of  $i$ -th detected pixel of edge between original pixel of edge.



**Figure 3.** (a)-(c) Test images (d)-(f) CAT detector (g)-(i) Sobel detector (j)-(l) LoG detector (m)-(o) Canny detector.



**Figure 4.** Extracted edge maps of Lena image with random noise (a) CAT detector (b) Sobel detector (c) LoG detector (d) Canny detector.



**Figure 5.** (a) Synthetic image (b) Synthetic image with Gaussian noise

**Table 3.** The proposed method compared with other methods

Detector	Power of Gaussian Noise				
	0.05	0.10	0.15	0.20	0.25
Sobel	0.8301	0.7059	0.3947	0.2706	0.2341
LoG	0.1776	0.1774	0.1768	0.1763	0.1753
Canny	0.3194	0.2945	0.2940	0.2778	0.2219
CAT	0.5320	0.2953	0.2117	0.1726	0.1520

Table 3 shows the proposed method compared with other methods. As shown in Table 3, the different gateway values generate two-dimensional CAT edge detection filters which are close to FoM results. Also, the performance of the proposed method was better than LoG and Canny methods in the region where the power of Gaussian noise was less than 0.15. Moreover, we found many basis functions which could detect edges. Furthermore, we expect them to be applied in various types of image processing in the future.

## 6 Conclusions

This paper presents a new edge detection scheme using two-dimensional cellular automata transforms. By using the gateway values (such as Wolfram Rule, the number of cells in lattice, the number of cells per neighborhood, the initial configuration, and the boundary configuration), we generated a dual-state, two-dimensional, and dual-coefficients cellular automata basis function. Then we transformed images into cellular automata domain with the basis function. Finally, we used the basis function and cellular automata transform coefficients to extract the edge of the image.

The experimental results verify that the proposed method is a new attempt to detect edge with cellular automata model as well as to maintain its high efficiency.

Although researchers continue to experiment on innovative edge detectors, they are still not able to define the ultimate suitable method for all conditions. However, with our proposed method, we can equally apply it to all images. We may collate many basis functions produced through the method and open possibilities for more developed edge detectors with novel features. Overall, our method will play as a clue to solving such inconvenience.

## References

1. S. Smith, M. Brady : SUSAN-A New Approach to Low Level Image Processing. *Int. J. Comput. Vision* 23 (1) (1997) 45-47
2. V. Torre, T. Poggio: On Edge Detection, *IEEE Trans. Pattern Anal. Mach. Intell. PAMI* 2 (1980) 147-163
3. J. Canny: A Computational Approach to Edge detection. *IEEE Trans. Pattern Anal. Mach. Intell. PAMI* 8 (1986) 679-698
4. T. D. Sanger : Optimal Unsupervised Learning in A Single-Layer Feedforward Neural Network. *Neural Networks* 2 (1989) 459-473
5. B. S. Manjunath, R. Chellappa : A Unified Approach to Boundary Perception : Edges, Textures and Illusory Contours. *IEEE Trans. Neural Network* 4 (1993) 96-108
6. S. Sarkar, K. L. Boyer : On Optimal infinite impulse response edge detection filters. *IEEE Trans. Pattern Anal. Mach. Intell. PAMI* 13 (1991) 1154-1171
7. Ulam, S : Some Ideas and Prospects in Biomathematics. *Annu. Rev. Biophys. Bioengin*, Vol. 1 (1963) 277-291
8. Von Neumann, J : Theory of Self-Reproducing Automata, University of Illinois Press, IL, (1966)
9. Wolfram S : Statistical Mechanics of Cellular Automata. *Rev. Mod. Phys.* 55 (1983) 601

10. Wolfram S : Computation Theory of Cellular Automata. *Commun. Math. Phys.* 96 (1984) 15-57
11. S. K. Lee, S. T. Kim and S. J. Cho: A Potts Automata algorithm for Noise Removal and Edge detection. *Journal of Korean institute of Communication Science*, Vol. 28-3C (2003) 327-335
12. S.J. Cho, U.S. Choi, H.D. Kim, Y.H. Hwang, J.G. Kim and S.H. Heo: New Synthesis of One-Dimensional 90/150 Linear Hybrid Group Cellular Automata. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 26(9) (2007) 1720-1724
13. S.J. Cho, U.S. Choi, H.D. Kim and Y.H. Hwang: Analysis of complemented CA derived from linear hybrid group CA. *Computers and Mathematics with Applications*, Vol. 53(1) (2007) 54-63
14. C. L. Chang, Y. J. Zhang, Y. Y. Gdong : CA for Edge detection of Images. *IEEE International Conference on Machine Learning and Cybernetics*, Shanghai (2004) 3830-3834.
15. Olu Lafe: *Cellular Automata Transforms: Theory and Application in Multimedia Compression, Encryption, and Modeling*. Kluwer Academic Publishers, Boston/ Dordrecht/London, (2000)
16. I. E. Abdou, W. K. Pratt : Quantitative design and evaluation of enhancement/thresholding edge detectors. *Proc. IEEE*, 69 (1979) 753-763.

# Computable Counter-examples to the Brouwer Fixed-point Theorem\*

Petrus H. Potgieter

Department of Decision Sciences, University of South Africa (Pretoria)  
PO Box 392, Unisarand, 0003, Republic of South Africa  
[php@member.ams.org](mailto:php@member.ams.org), [potgiph@unisa.ac.za](mailto:potgiph@unisa.ac.za), [www.potgieter.org](http://www.potgieter.org)

**Abstract.** This paper is an overview of results that show the Brouwer fixed-point theorem (BFPT) to be essentially non-constructive and non-computable. The main results, the counter-examples of Orevkov and Baigger, imply that there is no procedure for finding the fixed point in general by giving an example of a computable function which does not fix any computable point. Research in reverse mathematics has shown the BFPT to be equivalent to the weak König lemma in  $\text{RCA}_0$  (the system of recursive comprehension) and this result is illustrated by relating the weak König lemma directly to the Baigger example.

**Key words:** Computable analysis, Brouwer fixed-point theorem, weak König lemma

## 1 Introduction

We consider the Brouwer fixed-point theorem (BFPT) in the following form, where the standard unit interval is denoted by  $I = [0, 1]$ .

**Theorem 1 (Brouwer).** *Any continuous function  $f : I^2 \rightarrow I^2$  has a fixed point, i.e. there exists an  $x \in I^2$  such that  $f(x) = x$ .*

A computable real number is a number for which a Turing machine exists that, on input  $n$ , produces a rational approximation with error no more than  $2^{-n}$ . A computable point is a point all the coordinates of which are computable reals. The notation

$\mathbb{N}_0$  for the non-negative natural numbers;  
 $\mathbb{R}_c$  for the set of computable reals;  
 $I_c$  for  $I \cap \mathbb{R}_c$ ; and  
 $\delta X$  for the boundary of a set  $X$ , being  $\overline{X} \cap \overline{X^c}$

is also used. The two examples discussed use distinct definitions of a computable function of real variables.

---

\* This work was supported in part by a grant under the South African-Hungarian Science and Technology Agreement (National Research Foundation of South Africa UID: 62110) and in part by a research grant from the College of Economic and Management Sciences of the University of South Africa.

**Russian school** In the Russian school of Markov and others, a computable function maps computable reals to computable reals by a single algorithm for the function that translates an algorithm approximating the argument to an algorithm approximating the value of the functions. It need not be possible to extend a function that is computable in the Russian school to a continuous function on all of the reals. These functions are often called *Markov-computable*.

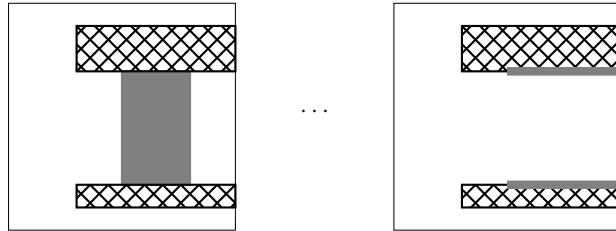
**Polish school** In the Polish school of Lacombe, Grzegorczyk, Pour-El and Richards, and others, a function is computable on a region if it maps every computable sequence of reals to a computable sequence of reals and it has a computable uniform modulus of continuity on the region [1].

## 2 Orevkov's example for the Russian school

One can construct a Markov-computable function  $f$  through a computable mapping of descriptions of computable points  $x \in I_c^2$  to descriptions of  $f(x) \in I_c^2$ , such that

$$f(x) \neq x \quad \forall x \in I_c^2.$$

That is, no computable point is a fixed point for  $f$ . Unfortunately the  $f$  which is constructed in this way, cannot be extended to a continuous function on  $I^2$ . This is the construction of [2], another instance of which can be found in [3].



**Fig. 1.** Basic contraction in the Orevkov counter-example

**Lemma 1.** Suppose  $A_k$  is a sequence of rectangles in  $I^2$  with computable vertices, disjoint interiors, and such that

- (i)  $\emptyset \neq \delta A_j \setminus \bigcup_{i < j} A_i$  for all  $j$ ;
- (ii) for each  $j$  there exists  $n > j$  such that  $\delta A_j \subset \left( \bigcup_{i \leq n} A_i \right)^\circ$ ; and
- (iii)  $I_c^2 \subseteq \bigcup_{i \geq 1} A_i$

then there exists a Markov computable  $g$ , mapping  $I_c^2$  to  $\delta I_c^2$  and fixing  $\delta I_c^2$ .

The conditions ensure that

- (i) rectangles  $A_j$ , when added, have some part of their boundary in  $I^2 \setminus \bigcup_{i < j} A_i$ ;
- (ii) each  $A_j$  is eventually closed off by new rectangles on all sides;
- (iii) all computable points lie in  $\bigcup_{i \geq 1} A_i$ .

The function  $f$  is obtained by composing  $g$  with a  $90^\circ$  rotation. It therefore remains only to prove the lemma and the existence of a sequence of rectangles which is as required. Suppose that  $g$  has been defined on  $\bigcup_{i < j} A_i$ . For

$\emptyset = \delta A_j \cap \bigcup_{i < j} A_i$ : let  $g$  on  $A_j$  consist of the simplest possible mapping to  $\delta I^2$ , that fixes  $\delta I^2$ ;

$\emptyset \neq \delta A_j \cap \bigcup_{i < j} A_i$ : we extend  $g$  to  $A_j$  by using (i)—if  $g$  has already been defined on the crosshatched set in Figure 1 then the definition can be extended to the solid gray set  $A_j$  by composing a contraction of the solid gray set in Figure 1 to

$$\left( \delta A_j \cap \bigcup_{i < j} A_i \right) \cup (\delta A_j \cap \delta I^2)$$

with the function  $g$  as it has already been defined on the crosshatched set. This is always possible because, by construction of the  $A_i$ , our  $A_j$  will always have at least two sides non-contiguous with  $\bigcup_{i < j} A_i$ , at least one of which will not coincide with  $\delta I^2$ .

So far only condition (i) has been used. Conditions (ii) and (iii) are necessary for showing that  $g$  is Markov-computable on  $I^2$ . Let a description of any  $x \in I_c^2$  be given. We can find a description of  $g(x)$  in the following way.

- Simultaneously, compute approximations of  $x$  using the given description and construct  $g$  on  $\bigcup_{i \leq n} A_i$  for  $n = 1, 2, \dots$
- Together, (ii) and (iii) imply that for some  $n$  we will be able to verify that

$$x \in \left( \bigcup_{i \leq n} A_i \right)^\circ$$

where the interior is with respect to the subset topology on  $I^2$ , of course.

- When such an  $n$  has been identified, we already know the definition of  $g$  for  $\bigcup_{i \leq n} A_i$  as well as the modulus of continuity of  $g$  on the same set. This is now used to describe  $g(x)$ .

It remains to be shown that a suitable sequence of rectangles  $(A_n)_{n \geq 1}$  exists. This follows from the next fact, assumed without proof for now<sup>1</sup>.

**Lemma 2 (see [4], for example).** *There exist computable sequences of rational numbers  $(a_n)$  and  $(b_n)$  in the interval  $I = [0, 1]$  such that the intervals  $J_n = [a_n, b_n]$  have the following properties.*

<sup>1</sup> Later we shall deduce the fact from the existence of a Kleene tree.

- (i) If  $n \neq m$  then  $|J_n \cap J_m| \leq 1$ .
- (ii) If  $a_n \neq 0$  then  $a_n \in \{b_0, b_1, \dots\}$  and if  $b_n \neq 1$  then  $b_n \in \{a_0, a_1, \dots\}$ .
- (iii)  $I_c \subsetneq \bigcup_n J_n$ , i.e. the  $J_n$  cover the computable reals in  $I = [0, 1]$ .

Now, let  $(A_n)_{n \geq 1}$  be any computable enumeration of the  $J_k \times J_\ell$ . This completes the proof of the lemma, and the example.

### 3 Baigger's example for the Polish school

Let  $a$  be any non-computable point in  $I^2$ . Consider the function  $f$  which moves each point half-way to  $a$ ,

$$f(x) = x + \frac{1}{2}(a - x)$$

and has a single fixed point, namely  $a$  itself. The function  $f$  is continuous and defined on all of  $I^2$  and has no computable fixed point. Nevertheless, this is not really interesting since

- the fixed point  $a$  has no reasonable description—since it is itself not computable; and therefore
- the function  $f$  has no reasonable description—it is not computable in any sense.

One would like to see a function which is computable, defined (and therefore continuous) on all of  $I^2$  and yet avoids fixing any of the computable points  $I_c^2$ . The following example, having appeared in [5] and in [3], modifies the construction of Orevkov to produce a computable  $f$  defined on all of  $I^2$  having no computable fixed point. One uses the intervals  $J_n = [a_n, b_n]$  of Orevkov's example and sets

$$C_n = \bigcup_{k, \ell \leq n} J_k \times J_\ell$$

after which one defines  $f$  progressively, using the sets  $C_n$ . The points

$$t_n = (v_n, v_n)$$

where

$$v_n = \min_{x \in I} \{x \mid (x, x) \notin C_n\}$$

are used as “target point” at each stage of the construction, as in Figure 2. Note that

$$v = \lim_{n \rightarrow \infty} v_n$$

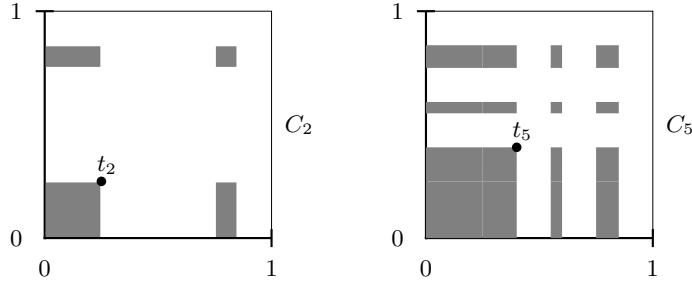
is not a computable number and  $(v, v)$  will be one of the fixed points of  $f$ .

**Definition 1.** For any  $W \subseteq I^2$  we define

$$W^{\blacksquare\varepsilon} = \{x \in W \mid d(x, \delta W \setminus \delta I^2) \geq \varepsilon\}$$

and

$$W^{\square\varepsilon} = \{x \in W \mid d(x, \delta W \setminus \delta I^2) = \varepsilon\}.$$



**Fig. 2.** The “target points”  $t_n$

One can define  $f_n$  such that

1.  $f_n$  moves every point in the *interior* of  $C_n^{\blacksquare 2^{-n}}$  but is the identity outside the set, and is computable;
2.  $f_{n+1}$  agrees with  $f_n$  on  $C_n^{\blacksquare 2^{-n} \cdot \frac{3}{2}}$  and therefore
3.  $f = \lim_{n \rightarrow \infty} f_n$  is computable.

Every computable point eventually lies in some

$$C_n^{\blacksquare 2^{-n} \cdot \frac{3}{2}} \subset \left( C_n^{\blacksquare 2^{-n}} \right)^\circ$$

and is therefore moved by  $f$ . Clearly  $f(I^2) \subseteq I^2$  and  $f$  will be as required. In fact,  $f$  has no fixed point in

$$\bigcup_n C_n = \bigcup_{k, \ell \geq 1} J_k \times J_\ell.$$

Also,  $f$  has no isolated fixed point—its fixed points all occur on horizontal and vertical lines spanning the height and breadth of the unit square. Further details of the construction appear in Appendix A. The construction cannot be applied in the one-dimensional case because it is impossible to effect a change of direction by continuous rotation.

#### 4 BFPT and the König lemma

In reverse mathematics it is known that in  $\text{RCA}_0$ , the system of recursive comprehension and  $\Sigma_1^0$ -induction, the weak König lemma,  $\text{WKL}_0$ , is equivalent to the Brouwer FPT [6].

**Lemma 3 (WKL<sub>0</sub>, König).** *Every infinite binary tree has an infinite branch.*

The König lemma does not have a direct computable counterpart.

**Theorem 2 (Kleene [7]).** *There exists an infinite binary tree, all the computable paths of which are finite.*

The relation of the Kleene tree to the Baigger counterexample is reviewed in this section. The discussion is informal and attempts only to give the essential ideas that have been revealed by the approach of reverse mathematics. In  $\text{RCA}_0$ , the weak König lemma  $\text{WKL}_0$  has been shown to be equivalent to a number of other results in elementary analysis, such as the fact that any continuous function on a compact interval is also uniformly continuous [8].  $\text{WKL}_0$  and  $\text{RCA}_0$  can, furthermore, be used to prove Gödel's incompleteness theorem for a countable language [9].

#### 4.1 From Baigger $f$ to Kleene tree

Let  $f$  be a computable function, as in the Baigger example, mapping  $I^2$  to itself—with no computable fixed point. The following auxiliary result will be used to construct the Kleene tree.

**Lemma 4.** *Let a computable  $g : I^2 \rightarrow [0, 1]$  be given. Then there exists a Turing-computable  $h : \mathbb{N}_0^9 \rightarrow \mathbb{N}_0^2$  such that for any  $(n_1, n_2, \dots, n_8, k)$  with*

$$0 \leq \frac{n_1}{n_2} \leq \frac{n_3}{n_4} \leq 1 \quad \text{and} \quad 0 \leq \frac{n_5}{n_6} \leq \frac{n_7}{n_8} \leq 1$$

we have  $h : (n_1, n_2, \dots, n_8, k) \mapsto (m_1, m_2)$  with  $m_1 \leq m_2$  where

$$\frac{m_1}{m_2} \leq \min g \left( \left[ \frac{n_1}{n_2}, \frac{n_3}{n_4} \right] \times \left[ \frac{n_5}{n_6}, \frac{n_7}{n_8} \right] \right) \leq \frac{m_1}{m_2} + \frac{1}{k}.$$

Let  $g = \|f(x) - x\|$  and let  $h$  be as in the lemma. Note that  $g(x) = 0$  if and only if  $x$  is a fixed point of  $f$ . We shall use only the essential consequences that

- $g(x) > 0$  for all computable  $x$ ; and
- there exists a (non-computable)  $x_0$  such that  $g(x_0) = 0$ .

As usual,  $\{0, 1\}^*$  denotes the set of finite binary sequences and  $ab$  is the concatenation of  $a$  and  $b$ .

**Definition 2.** *A binary tree is a function  $t : \{0, 1\}^* \rightarrow \{0, 1\}$  such that*

$$t(ab) = 0 \quad \text{for all } b \quad \text{whenever} \quad t(a) = 0.$$

*An infinite branch of a tree  $t$  is an infinite binary sequence, on all of which finite initial segments  $t$  takes the values 1.*

The tree is *computable* whenever the function  $t$  is Turing-computable and a *computable branch* is a computable binary sequence which is an infinite branch. Define the Kleene tree as follows. Let

$$t(i_1 \dots i_n) = \prod_{m=1}^n s(i_1 \dots i_m)$$

where  $s$  is a function taking values in  $\{0, 1\}$ . This definition of  $t$  ensures that  $t$  is in fact a tree and if  $s$  is computable,  $t$  will be a computable tree. The function  $s$  will use  $h$  to estimate whether  $g$  gets close to zero on a specific square and if  $g$  has been bounded away from zero on the square, that branch of the tree will terminate.

Define  $s : \{0, 1\}^* \rightarrow \{0, 1\}$  for all sequences  $i_1 j_1 \dots i_n j_n$  of even length by

$$s(i_1 j_1 \dots i_n j_n) = \chi_{\{0\}} \left( \frac{m_1}{m_2} \right)$$

where

$$(m_1, m_2) = h(i_1 \dots i_n, 2^n, i_1 \dots i_n + 1, 2^n, j_1 \dots j_n, 2^n, j_1 \dots j_n + 1, 2^n, n)$$

and binary strings have been interpreted as the natural numbers which they represent. Let  $s$  take the value 1 on sequences of odd length.

The tree  $t$  defined in this way is obviously computable. It remains to show that  $t$  is

- infinite; and
- has no infinite computable branch.

Let  $x_0$  be any point where  $g(x_0) = 0$ . Then there exist infinite sequences  $(i_n)$  and  $(j_n)$  such that

$$x_0 \in \left[ \frac{i_1 \dots i_n}{2^n}, \frac{j_1 \dots j_n}{2^n} \right] \times \left[ \frac{i_1 \dots i_n + 1}{2^n}, \frac{j_1 \dots j_n + 1}{2^n} \right] \quad \text{for all } n$$

and therefore, for all  $n$ ,  $s(i_1 j_1 \dots i_n j_n) = 1$  and so  $t(i_1 j_1 \dots i_n j_n) = 1$  which proves the existence of an infinite branch, hence that the tree  $t$  is infinite.

Suppose that  $t$  had an infinite computable branch. The branch would correspond to a decreasing chain of closed squares, the intersection of which would be non-empty. Let  $x_1$  be a point in the intersection. Since, by construction of the tree,  $g(x_1) \leq \frac{1}{n}$  for all  $n$ ,  $g(x_1) = 0$  and hence  $x_1$  would be a fixed point of  $f$ . However, by the construction—the branch being computable—the point  $x_1$  would also be computable, contradicting the fact that  $f$  has not computable fixed point. Therefore the tree  $t$  has no infinite computable branch.

## 4.2 From Kleene tree to Baigger $f$

Suppose we are given a computable tree  $t$  with no infinite computable branch. This tree can be used to construct a sequence of closed intervals with a computable sequence of end-points, covering all the computable real numbers in the unit interval and for which the corresponding open intervals are pair-wise disjoint.

Using the computable function  $t$ , one can enumerate all of the maximal finite branches of the tree. Say,

$$b(n) = b_1(n) \dots b_{\lambda(n)}(n)$$

and set

$$\begin{aligned} J_{n,1} &= \left[ \frac{b_1(n) \dots b_{\lambda(n)}}{2^{\lambda(n)}}, \frac{b_1(n) \dots b_{\lambda(n)} + \frac{1}{2}}{2^{\lambda(n)}} \right] \\ J_{n,m} &= \left[ \frac{b_1(n) \dots b_{\lambda(n)} + 2^{-m+1}}{2^{\lambda(n)}}, \frac{b_1(n) \dots b_{\lambda(n)} + 2^{-m}}{2^{\lambda(n)}} \right] \quad \text{for } m \geq 2. \end{aligned}$$

It remains to show that the union of the intervals  $J_{n,m}$  covers *all* the computable points  $I_c$  but not all of the unit interval  $I$ . It is easy to see that

- for every computable  $x \in I_c$  there exists a computable binary sequence  $(x_n)$  such that

$$\frac{x_1 \dots x_n}{2^n} \leq x < \frac{x_1 \dots x_n + 1}{2^n} \quad \text{for all } n$$

and since  $t$  has no infinite computable branch  $t(x_1 \dots x_\ell) = 0$  for some least  $\ell$ , in which case  $x \in \cup_m J_{n,m}$  where  $b(n) = x_1 \dots x_\ell$ ;

- if  $(x_n)$  is an infinite branch of  $t$  then, since it is not computable, for all  $w$  we have  $x_1 x_2 \dots \neq w1111\dots$  and therefore

$$\lim_n \frac{x_1 \dots x_n + 1}{2^n} \notin \bigcup_m J_{\ell,m}$$

for every  $\ell$ .

The Baigger example  $f$  can now be constructed using the intervals  $J_{n,m}$  and by that construction one obtains a computable  $f$  with no computable fixed point, as required.

## 5 Conclusion

The existence of the Kleene tree can quite easily be derived from the impossibility of ensuring the existence of a computable fixed point for a computable function (in both Russian and Polish senses), in two dimensions (or higher). The ingenuous constructions of Orevkov and Baigger provide a way of defining a computable function with no computable fixed point from the set of intervals derived from the Kleene tree, in a constructive manner. This correspondence is, perhaps, more attractive for the “working mathematician” than the elegant derivation of the result in reverse mathematics. In one dimension, any computable  $f : I \rightarrow I$  does have a computable point  $x \in I_c$  such that  $f(x) = x$ , which can be seen by fairly straight-forward reduction *ad absurdum* from the assumption that this is not the case.

## References

1. Pour-El, M.B., Richards, J.I.: Computability in analysis and physics. Perspectives in Mathematical Logic. Springer-Verlag, Berlin (1989)

2. Orevkov, V.P.: A constructive map of the square into itself, which moves every constructive point. *Dokl. Akad. Nauk SSSR* **152** (1963) 55–58
3. Wong, K.C., Richter, M.K.: Non-computability of competitive equilibrium. *Economic Theory* **14**(1) (1999) 1–27
4. Miller, J.S.: Degrees of unsolvability of continuous functions. *J. Symbolic Logic* **69**(2) (2004) 555–584
5. Baigert, G.: Die Nichtkonstruktivitat des Brouwerschen Fixpunktsatzes. *Arch. Math. Logik Grundlag.* **25**(3-4) (1985) 183–188
6. Shioji, N., Tanaka, K.: Fixed point theory in weak second-order arithmetic. *Ann. Pure Appl. Logic* **47**(2) (1990) 167–188
7. Kleene, S.C.: Recursive functions and intuitionistic mathematics, Providence, R. I., Amer. Math. Soc. (1952) 679–685
8. Simpson, S.G.: Which set existence axioms are needed to prove the cauchy/peano theorem for ordinary differential equations? *The Journal of Symbolic Logic* **49** (1984) 783–802
9. Simpson, S.G.: Subsystems of second order arithmetic. Perspectives in Mathematical Logic. Springer-Verlag, Berlin (1999)

## Appendix A: details of the construction in Section 3

The constructions should guarantee that at each stage, the function  $f_n$  moves every point of

$$D_n = \left( C_n^{\blacksquare 2^{-n}} \setminus C_n^{\blacksquare 2^{-n} \cdot \frac{5}{4}} \right)^\circ$$

in the direction of  $t_n$  by an amount proportional to its distance to  $C_n^{\square 2^{-n}}$ . The construction of  $f_1$  with this property is trivial. We proceed to construct  $f_{n+1}$  from  $f_n$ .

- (i) Extend and modify  $f_n$  to  $C_{n+1}^{\blacksquare 2^{-n}}$  so that every point  $x$  of

$$\left( C_{n+1}^{\blacksquare 2^{-n}} \setminus C_{n+1}^{\blacksquare 2^{-n} \cdot \frac{5}{4}} \right)^\circ$$

is moved in the direction of  $t_n$  by an amount proportional to  $d(x, C_{n+1}^{\square 2^{-n}})$ .

- (ii) Modify the resulting function so that each point in

$$C_{n+1}^{\blacksquare 2^{-n}} \setminus C_{n+1}^{\blacksquare 2^{-n} \cdot \frac{9}{8}}$$

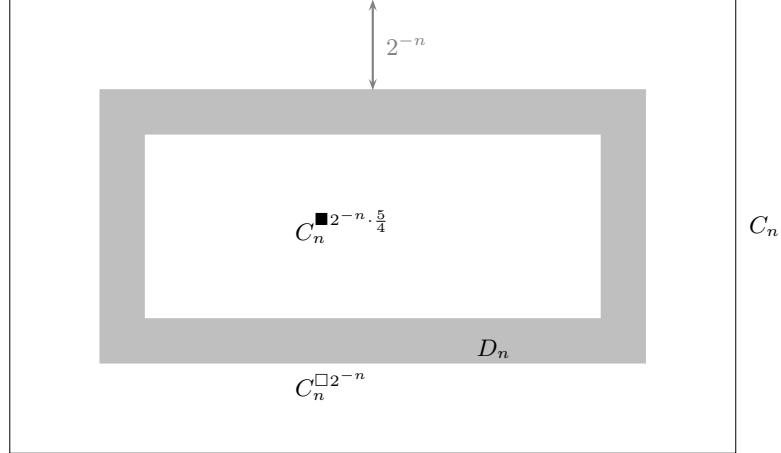
is mapped a non-negative amount proportional to its distance to  $C_{n+1}^{\square 2^{-(n+1)}}$  in the direction of  $t_n$ .

- (iii) By rotation of the direction of the mapping, extend the function to

$$C_{n+1}^{\blacksquare 2^{-(n+1)}}$$

such that every point  $x$  of

$$D_{n+1} = \left( C_{n+1}^{\blacksquare 2^{-(n+1)}} \setminus C_{n+1}^{\blacksquare 2^{-(n+1)} \cdot \frac{5}{4}} \right)^\circ$$

**Fig. 3.** Sets used in the construction

is mapped in the direction of  $t_{n+1}$  by an amount proportional to

$$d\left(x, C_{n+1}^{\square 2^{-(n+1)}}\right).$$

The final step is the only one in which we use the fact that we are working in two dimensions as this step requires the continuous (computable) rotation of a vector in the direction of  $t_n$  to a vector in the direction of  $t_{n+1}$ .

A construction is given explicitly in [5] but it should be clear from the preceding that it can be done in many different ways. The important part of the proof is that the construction is, at each stage, extended at the boundary to “look right” from the outside. This ensures that, eventually every point is in fact moved towards one of a sequence of points that converge to the non-computable fixed point  $(v, v)$  on the diagonal. The Baigger construction is a somewhat delicate construction of a function that is in fact computable but that—somehow—mimics a simple mapping of every point in  $I^2$  in the direction of  $(v, v)$ .

# Polynomial Iterations over Finite Fields

Mihai Prunescu<sup>1;2</sup>

<sup>1</sup> Brain Products, Freiburg, Germany

<sup>2</sup> Institute of Mathematics “Simion Stoilow” of the Romanian Academy  
Bucharest, Romania  
`mihai.prunescu@math.uni-freiburg.de`

**Abstract.** Consider the following natural algorithm: given a finite field  $\mathbb{F}$  and a polynomial  $f \in \mathbb{F}[x, y, z]$  one produces the double sequence  $(a_{i,j})$  defined by  $a_{0,j} = a_{i,0} = 1$  und  $a_{i,j} = f(a_{i,j-1}, a_{i-1,j-1}, a_{i-1,j})$ . If the polynomials  $f$  are linear, self-similarity arises. On the other hand, the class of double sequences  $(a_{i,j})$  generated by symmetric polynomials  $f(x, z)$  over arbitrary finite fields is Turing complete.

## 1 Introduction

The research reported here has a strong experimental back-ground. Let  $\mathbb{F}$  be a finite field and  $f \in \mathbb{F}[x, y, z]$  some polynomial. Iterating  $f$  one gets a recurrent double sequence  $(a_{i,j})$  defined by  $a_{0,j} = 1$ ,  $a_{i,0} = 1$  and:

$$a_{i,j} = f(a_{i,j-1}, a_{i-1,j-1}, a_{i-1,j}).$$

If one fixes a correspondence between the elements of  $\mathbb{F}$  and a set of colours, one can draw an image corresponding to the initial matrix  $(a_{i,j})_{0 \leq i,j < n}$  of the recurrent double sequence. Such images will be sometimes called carpets. The starting point of this research was a conjecture of Lakhtakia and Passoja [3] telling that the polynomial  $f(x, y, z) = x + y + z$  generate self-similar carpets over the prime fields  $\mathbb{F}_p$ .

In [5] the author proved that all polynomials  $f(x, y, z) = x + my + z$  produce self-similar (fractal) images over arbitrary finite fields and classified the occurring symmetries. The proofs uses elements of algebra and number theory, but also applies modern algebraic algorithms as those by Wilf and Zeilberger. In [6] the author proved that for arbitrary polynomials in only two variables  $f(x, z)$  the recurrent double sequences interpret instances of the Halting Problems and have undecidable properties, as for exemple the property of being ultimately zero.

The goal of this extended abstract is to present all definitions and results, but only (at most) sketches of proofs. It should be seen as a complementary text to [5] and [6] and has also the goal to emphasize their contrast.

## 2 The linear case

**Definition 1.** Let  $\mathbb{F}_q$  be an arbitrary finite field and fix an element  $m \in \mathbb{F}_q$ . The matrices occurring in this section are always indexed from 0 and have elements

in  $\mathbb{F}_q$ , if not otherwise specified. Let the prime  $p$  be the characteristic of the finite field,  $q = p^k$  for some  $k$ . Let  $M_d = (a_{i,j})$  be the  $p^d \times p^d$  matrix constructed following the recurrence  $a_{i,0} = a_{0,j} = 1$  and  $a_{i,j} = a_{i-1,j} + m \cdot a_{i-1,j-1} + a_{i,j-1}$ . The matrix  $M_1$  shall be denoted by  $F(p, m)$  and called **fundamental block**.

**Definition 2.** The black and white image  $I^d$  is defined as follows: one tiles the compact square  $[0, 1] \times [0, 1]$  in  $p^d \times p^d$  many equal squares  $S_{i,j}$ , and excludes the interior of  $S_{i,j}$  if and only if  $a_{i,j} = 0$ .

**Definition 3.** The self-similar set in question shall be  $I = \lim I^d$ . The name of the variable  $d$  is chosen to mean the **depth** of the recursive approximation of  $I$ . The limit operator can be understood in the sense of the Hausdorff metric for compact subsets of  $\mathbb{R}^2$ .

## 2.1 The recurrent function

**Definition 4.** Let  $K$  be an arbitrary field and the element  $m \in K$  be fixed. We consider the function  $f : \mathbb{N} \times \mathbb{N} \rightarrow K$  recursively defined by the conditions  $f(n, 0) = f(0, k) = 1$  and:

$$f(n, k) = f(n, k - 1) + m \cdot f(n - 1, k - 1) + f(n - 1, k)$$

for  $n, k \geq 1$ .

**Lemma 1.** *The function  $f$  is symmetric and satisfies:*

$$f(n, k) = \sum_{a=0}^{\min(n,k)} m^a \binom{n}{a} \binom{n+k-a}{k-a}.$$

*Proof.* The symmetry follows from the symmetry of the recurrence formula and of the initial conditions. To compute  $f$ , use the method of generating functions, see [12]. Define the generating function  $A_n(x) = \sum_{k \geq 0} f(n, k)x^k$ . It follows:

$$A_{n+1}(x) = A_n(x) + xA_{n+1}(x) + mxA_n(x).$$

This recurrence have the solution:

$$A_n(x) = \left(\frac{1}{1-x}\right)^{n+1} (1+mx)^n.$$

Using that  $(1+mx)^n = \sum_{k \geq 0} \binom{n}{k} m^k x^k$  and that  $\left(\frac{1}{1-x}\right)^{n+1} = \sum_{k \geq 0} \binom{n+k}{k} x^k$ , one gets the Lemma.

## 2.2 Tensor powers and the automorphism of Frobenius

In this section we prove some properties of the fundamental block  $F(p, m) \in \mathcal{M}_{p \times p}(\mathbb{F}_p)$ . Recall the notation  $F(p, m) = (a_{i,j})$  with  $i$  and  $j = 0, \dots, p-1$ .

**Lemma 2.** *The last column and the last row of  $F(p, m)$  are exactly:*

$$1, -m, (-m)^2, \dots, (-m)^{p-1}.$$

*This works also for  $m = 0$ .*

*Proof.* Take  $k \leq n = p-1$  and work over  $\mathbb{F}_q$ . For  $a < k$  the term:

$$t(a, p-1, k) = m^a \binom{p-1}{a} \binom{p-1+k-a}{k-a} = m^a \binom{p-1}{a} \cdot p \cdots = 0,$$

so all these terms do not contribute in  $\mathbb{F}_q$ . For the last term one has:

$$t(k, k, p-1) = m^k \frac{(p-1) \cdots (p-k)}{k!} = m^k (-1)^k \frac{k!}{k!} = (-m)^k.$$

**Definition 5.** Let  $R$  be some commutative ring and  $A = (a_{i,j}) \in \mathcal{M}_{s \times t}(R)$ ,  $B \in \mathcal{M}_{u \times v}(R)$  two matrices. Then the **tensor product**  $A \otimes B$  is a matrix in  $\mathcal{M}_{su \times tv}(R)$  having the block-representation  $(a_{i,j}B)$ . If  $A_1, A_2, \dots, A_n$  are arbitrary matrices, we denote the tensor term:

$$((\dots ((A_1 \otimes A_2) \otimes A_3) \dots) \otimes A_{n-1}) \otimes A_n.$$

by:

$$A_1 \otimes A_2 \otimes \cdots \otimes A_{n-1} \otimes A_n.$$

For all  $n \geq 1$  we define the **tensor power**  $A^{\otimes n}$  of  $A$  inductively by:  $A^{\otimes 1} = A$  and  $A^{\otimes(n+1)} = A^{\otimes n} \otimes A$ .

*Remark 1.* (Principle of Substitution) For some  $n \geq 2$  consider a matrix  $A \in \mathcal{M}_{n \times n}(\{0, 1\})$  containing at least one zero and at least two ones. Let  $I^d$  be the black and white image associated to  $A^{\otimes d}$ . Then  $I^d$  is the  $d$ -th step in the transfinite construction of a non-trivial self-similar set  $I = \lim I^d$ .

**Definition 6.** The automorphismus of Frobenius  $\varphi : \mathbb{F}_q \rightarrow \mathbb{F}_q$  is defined by  $\varphi(x) = x^p$ . This automorphism generates the Galois group  $G(\mathbb{F}_q/\mathbb{F}_p)$ .

**Lemma 3.** *Let  $F = F(p, m)$  be a fundamental block for some  $m \in \mathbb{F}_q$ . Consider the matrix in construction:*

$$\begin{matrix} \alpha F & \beta F \\ \gamma F & \end{matrix}.$$

*with  $\alpha, \beta, \gamma \in \mathbb{F}_q$ . By application of the recurrent rule one gets:*

$$\begin{matrix} \alpha F & \beta F \\ \gamma F & \delta F \end{matrix}$$

*with  $\delta = \varphi(m)\alpha + \beta + \gamma$ .*

**Definition 7.** For a matrix  $A = (a_{i,j})$  over  $\mathbb{F}_q$ , let  $\varphi(A)$  be the matrix  $(\varphi(a_{i,j}))$ .

**Theorem 1.** Recall that  $M_d$  is the  $p^d \times p^d$  matrix computed by the recurrent rule over the finite field  $\mathbb{F}_q$  and  $F = F(p, m) = M_1$  is the fundamental block. Then for all  $d \geq 1$ :

$$M_d = \varphi^{d-1}(F) \otimes \varphi^{d-2}(F) \otimes \cdots \otimes \varphi(F) \otimes F.$$

*Proof.* The proof works by induction and is a immediate application of the Lemma 3.

**Corollary 1.** For all finite fields  $\mathbb{F}_q$  and all  $m \in \mathbb{F}_q$ , if the fundamental block  $F(p, m)$  contains at least a zero, the black and white image  $I^d$  of  $M_d$  is the  $d$ -th step in the transfinite construction of a non-trivial self-similar set  $I$ .

*Proof.* Immediate application of the Principle of Substitution.

**Lemma 4.** If  $m \in \mathbb{F}_p$ , the fundamental block  $F(p, m)$  contains zeros if and only if  $m \neq -1$ . In this situation it contains in the row  $i = 1$  exactly one zero:

$$a_{1,k} = 0 \Leftrightarrow \mathbb{F}_p \models k = -(m+1)^{-1}.$$

Note: in general there are many other zeros in the fundamental block.

*Proof.* The element  $a_{1,k} = km + (k+1) = k(m+1) + 1$  which is zero only for  $k = -(m+1)^{-1}$ , existing for all  $m \neq -1$  in  $\mathbb{F}_p$ . Every such  $k$  has a representative between 1 and  $p-1$  inclusively. If  $m = -1$  the matrix  $F(p, -1)$  contains only the repeated element 1.

Now from Remark 1 and from the Lemma 1 the main result follows:

**Theorem 2.** For all primes  $p$  and all  $m \in \mathbb{F}_p \setminus \{-1\}$  the black and white image  $I^d$  of  $M_d$  is the  $d$ -th step in the transfinite construction of a non-trivial self-similar set  $I$ . For  $m = -1$  the set  $I$  is the full square. The Pascal Triangle modulo  $p$  (got for  $m = 0$ ) and the Passoja-Lakhtakia Carpets (got for odd  $q = p$  and  $m = 1$ ) are non-trivial self-similar sets.

The following example shows the step  $M_2$  for  $p = 3$  and  $m = 1$ , a step in the construction for the celebrated Sierpinski Carpet, used in [11]. The zeros are not displayed.

$$\begin{array}{cccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ 1 & 1 & 1 & & & -1 & -1 & -1 \\ 1 & -1 & & & & -1 & & 1 \\ 1 & -1 & 1 & & & -1 & 1 & -1 \\ 1 & 1 & 1 & -1 & -1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 1 & & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & 1 \end{array}$$

### 2.3 Multiplicative inverse means mirroring

For studying the groups of symmetries of the black and white image  $I$  is enough to understand the symmetries for the fundamental block  $F(p, m)$ . All groups of symmetries we are looking for are subgroups of the dihedral group of symmetries  $D_8$  of the square. We start with the least symmetric case, the case of Pascal's Triangle:

**Lemma 5.** *If  $m = 0$  the group of symmetries consists of two elements: the identity and the reflection through the first diagonal.*

*Proof.* In  $F(p, 0)$  for  $0 \leq i, j \leq p - 1$ :

$$a_{i,j} = 0 \leftrightarrow p \mid f(i, j) = \binom{i+j}{i} \leftrightarrow i + j \geq p.$$

So exactly the elements strictly below the second diagonal are 0.

**Definition 8.** For a matrix  $A$  we define the mirrored image  $\Sigma A$  using the definition of a matrix as a family of column-vectors. If  $A = (\mathbf{a}_1, \dots, \mathbf{a}_n)$  then  $\Sigma A = (\mathbf{a}_n, \dots, \mathbf{a}_1)$ .

**Definition 9.** For  $m \neq 0$  we define the operator  $\mathcal{O}$  acting over the fundamental block  $F(p, m)$  in the following way:

For  $i = 0$  to  $p - 1$ , one divides the row  $i$  by  $(-m)^i$ .

The result is denoted by  $\mathcal{OF}(p, m)$ .

**Lemma 6.** *For all finite fields  $\mathbb{F}_q$  and for all  $m \in \mathbb{F}_q \setminus \{0\}$  the following identity holds:*

$$\mathcal{OF}(p, m) = \Sigma F(p, m^{-1}).$$

*Proof.* The Lemma follows from the following claims: (1) The first row and the last column of  $\mathcal{OF}(p, m)$  consist only of ones. (2) For every connected  $2 \times 2$  sub-block of  $\mathcal{OF}(p, m)$ :

$$\begin{matrix} A & B \\ C & D \end{matrix}$$

is true that  $C = m^{-1}B + A + D$ . The first claim follows from Lemma 2 and from the definition of the operator  $\mathcal{O}$ : one divides exactly with the elements of the last column. We prove the second claim. Let  $(a, b | c, d)$  be the corresponding elements in  $F(p, m)$ . They fulfill the equality:

$$d = ma + b + c.$$

Using the definition of  $\mathcal{OF}(p, m)$ , we see that:

$$A = \mu a, \quad B = \mu b,$$

$$C = (-m)^{-1}\mu c, \quad D = (-m)^{-1}\mu d,$$

where  $\mu = (-m)^i$  for some  $i$ . It follows that:

$$\begin{aligned} C &= (-m)^{-1}\mu c = (-m)^{-1}\mu(d - ma - b) = \\ &= (-m)^{-1}\mu d + \mu a - (-m)^{-1}\mu b = D + A + m^{-1}B. \end{aligned}$$

**Lemma 7.** *The following statements follow directly from Lemma 6:*

1. *For all  $0 \leq i, j \leq p-1$ :*

$$a'_{i,p-1-j} = a_{i,j}(-m)^{-i},$$

$$a_{i,j}(-m)^{-i} = a_{p-1-j,p-1-i}(-m)^{j+1-p}.$$

2. *If  $m \in \mathbb{F}_q \setminus \{0\}$  then:*

$$\delta F(p, m) = \delta \Sigma F(p, m^{-1}) = \Sigma \delta F(p, m^{-1}).$$

*Moreover, the matrix  $\delta F(p, m)$  allows two diagonal symmetries; and so all its tensor powers.*

3. *Given  $m \in \mathbb{F}_q \setminus \{0\}$  fixed, some matrix  $M_d$  contains zeros if and only if  $M_1 = F(p, m)$  contains zeros. If this takes places, then*

$$\deg(m/\mathbb{F}_p) \leq \frac{p-1}{2}.$$

The last condition occurring here is quite strong and implies that there cannot be too much elements  $m$  generating non-trivial self-similar sets in arbitrary finite fields. Look at the case  $\mathbb{F}_{19^2} = \mathbb{F}_{361}$  seen as  $\mathbb{F}_{19}[x]$  where  $x^2 + 1 = 0$ . Encode the element  $ax + b$  in the natural number  $19a + b$ . I do not mention both  $m$  and  $m^{-1}$  because they produce mirrored carpets. Also, if  $m$  has been already mentioned, I don't mention its Frobenius  $m^{19}$ , because it produces the same carpet. So, up to Frobenius and multiplicative inverse, one has non-trivial self-similar carpets over  $\mathbb{F}_{361}$  if and only if  $m$  is equal with one of the following 29 elements: 0, 1, 2, 3, 4, 6, 7, 8, 9, 14, 19, 21, 35, 47, 52, 53, 56, 63, 69, 76, 78, 88, 92, 102, 130, 136, 137, 148, 168. Values of  $m \in \mathbb{F}_{361}$  which are not itself, inverses of, or Frobenius of elements in this list generate however interesting coloured images. Over the prime fields  $\mathbb{F}_p$  the situation looks better:

## 2.4 $\mathbb{F}_p$ as a field of self-similar carpets

**Theorem 3.** *Let  $p$  be a prime and  $m \in \mathbb{F}_p$ . Exactly one of the following situations arrises:*

1.  *$m = 0$ . In this case  $I$  is a self-similar Pascal Triangle,  $I$  is only symmetric through the first diagonal, and the group of symmetries of  $I$  is isomorphic with  $S_2$ .*

2.  $m = \pm 1$ . In this case  $I$  is a full square (for  $m = -1$ ) or a nontrivial self-similar set (for  $m = 1$ ) and the group of symmetries of  $I$  is the full dihedral group  $D_8$  of the square.
3.  $p \geq 5$ ,  $m \in \mathbb{F}_p \setminus \{-1, 0, 1\}$ . In this case  $I$  is a non-trivial self-similar set and the group of symmetries of  $I$  is generated by the reflexions through the diagonals of the square. This group is isomorphic with Klein's group  $K_4$ .

*Proof.* Let now  $m \in \mathbb{F}_p \setminus \{0\}$ , let  $K$  be the group generated by the symmetries through the both diagonals (isomorphic with Klein's group  $K_4$ ) and let  $G$  be the group of symmetries of  $I$ . From Lemma 7 it follows that  $K \leq G \leq D_8$ . If  $m = -1$  then  $I$  is the full square and trivially  $G = D_8$ . If  $m = 1$  than it follows from Lemma 7 that:

$$\delta F(p, 1) = \Sigma \delta F(p, 1),$$

because  $1^{-1} = 1$  so  $G$  is strictly bigger than  $K$  which already has 4 elements, hence  $G = D_8$ .

The converse is easy to prove using Lemma 4.

## 2.5 The special case $m \in \{-2, -2^{-1}\}$ : Diagonal Carpets

For  $m = -2$  one has  $a_{1,1} = 0$ . Mirror-symmetric: for  $m = -2^{-1}$  one has  $a_{1,p-2} = 0$ . In fact, in these cases, all the elements of odd index on the corresponding diagonal are zero!

**Definition 10.** Call **first odd diagonal** (respectively **second odd diagonal**) the following set of indexes:

$$D^+ = \{(i, i) \mid 0 < i < p - 1 \wedge 2 \nmid i\}.$$

$$D^- = \{(i, j) \mid i + j = p - 1 \wedge 2 \nmid i\}.$$

**Theorem 4.** Let  $p \geq 5$  be a prime. Following statements hold:  $D^+$  consists of zeros of  $F(p, -2)$  and  $D^-$  consists of zeros of  $F(p, -2^{-1})$ . Moreover, the elements of even index on the respective diagonals are  $\neq 0$ .

*Proof.* We prove that for  $m = -2 \in \mathbb{Z}$  the recurrent function  $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{Z}$  defined in the second section has the property  $f(2s+1, 2s+1) = 0$  for all  $k \in \mathbb{N}$ . This follows from the following identity:

$$\sum_{a=0}^n (-2)^a \binom{n}{a} \binom{2n-a}{n-a} = \begin{cases} (-1)^s \binom{2s}{s}, & \text{if } n = 2s, \\ 0, & \text{if } n = 2s+1. \end{cases}$$

This identity can be proved with Zeilberger's Algorithm, see [7] and [1]. In fact, after running the software from [1], one gets the recurrent formula:

$$4(n+1)S(n) + (n+2)S(n+2) = 0,$$

where  $S(n)$  is the sum on the left side of the equality. Starting with  $S(0) = 1$  and  $S(1) = 0$  one gets the result.  $D^-$  follows from the case  $m = -2$  and the dualism from Lemma 6. Note that the corresponding values of  $f(n, k)$  are no more 0 in  $\mathbb{Z}$  but become 0 if projected in  $\mathbb{F}_p$ .

## 2.6 The special case $m = 1$ : Cross-carpets

The only one fully symmetric non-trivial case (where  $p$  is odd and  $m = 1$ ) is worth for a closer look. This is exactly the case of the spectacular Passoja-Lakhtakia Carpets, described in [3]. It is worth to notice that the infinite symmetric matrix  $(f(n, k))$  for  $m = 1$  is known as the double sequence of the Delanoy Numbers.

**Definition 11.** Let us call  $N = \{(i, j) \mid a_{i,j} = 0\}$  the set of **zeros** of the fundamental cell  $F(p, 1)$ . The set:

$$C = \left\{ \left( \frac{p-1}{2}, i \right); \left( i, \frac{p-1}{2} \right) \mid 0 \leq i \leq p-1 \wedge 2 \nmid i \right\}$$

shall be called the **Cross**, and  $S = N \setminus C$  the set of **sporadic zeros**. We call the elements of the cross **regular zeros**.

**Corollary 2.** If  $p$  is an odd prime, the fundamental block  $F(p, 1)$  has the following properties:

1. If  $0 \leq n, k \leq p-1$  then  $a_{p-1,k} = (-1)^k$  and  $a_{n,k} = (-1)^n a_{n,p-1-k}$ .
2. The Cross  $C$  consists of zeros of  $F(p, 1)$ .

*Proof.* This follows from Lemma 2. combined with Lemma 6.

The primes 3, 5, 7, 11, 19 have only regular zeros in  $F(p, 1)$ . 13 is the first odd prime with sporadic zeros, followed by 17. By all other primes tryed out by the author (from 23 to 599) there are lots of sporadic zeros in the fundamental block  $F(p, 1)$ .

As final remark: the general diagonally symmetric linear polynomial  $ax + by + az$  produce over  $\mathbb{F}_p$  carpets consisting of overlappings of several periodic motives and one self-similar carpet of type  $x + my + z$ . However, the complete characterization of their behavior is an open problem.

## 3 General polynomials

The situation for the general polynomials is no more algebraic or combinatoric anymore. In fact according to the property of interpolation over finite fields, all functions are polynomial; and commutative binary operations can always be represented by symmetric polynomials in two variables.

**Definition 12.** Consider an arbitrary finite algebra  $\mathfrak{A} = (A, f, 0, 1)$  where  $f : A \times A \rightarrow A$  is a binary operation and 0, 1 are two constants. We call 1 start symbol. The recurrent double sequence associated to  $\mathfrak{A}$  is a function  $a : \mathbb{N} \times \mathbb{N} \rightarrow A$  defined as follows:

$$a(i, j) = \begin{cases} 1 & \text{if } i = 0 \vee j = 0, \\ f(a(i, j-1), a(i-1, j)) & \text{if } i > 0 \wedge j > 0. \end{cases}$$

If  $f$  is commutative, the recurrent double sequence is symmetric:  $a(i, j) = a(j, i)$ .

**Definition 13.** The recurrent double sequence  $a(i, j)$  is said to be ultimately zero if in  $\mathfrak{A}$  holds:

$$\exists N \in \mathbb{N} \ \forall i, j \in \mathbb{N} \ i > 0 \wedge j > 0 \wedge i + j > N \implies a(i, j) = 0.$$

**Theorem 5.** *It is undecidable if the recurrent double sequence defined by an algebra  $\mathfrak{A}$  is ultimately zero. This question remains undecidable if is restricted to the class of commutative finite algebras.*

**Definition 14.** An instance of the Halting Problem is a pair  $(M, w)$  where  $M = (\Sigma, Q, q_0, q_s, \bar{b}, \delta)$  is a Turing machine and  $w \in \Sigma^*$  is an input for  $M$ . Here the tape of  $M$  is infinite in both directions,  $\Sigma$  is the alphabet of  $M$ ,  $Q$  is  $M$ 's set of states,  $q_0$  and  $q_s$  are the start state and respectively the stop state,  $\bar{b} \in \Sigma$  is the blank symbol, and  $\delta : \Sigma \times Q \rightarrow \Sigma \times Q \times \{R, L, S\}$  is the transition function.

**Lemma 8.** *To every instance  $(M, w)$  of the Halting Problem one can algorithmically associate a finite algebra  $\mathfrak{A} = (A, f, 0, 1)$  such that  $\mathfrak{A} \in Z$  if and only if for input  $w$  the machine  $M$  stops and after stopping the tape is cleared.*

**Lemma 9.** *To every instance  $(M, w)$  of the Halting Problem one can algorithmically associate a commutative finite algebra  $\mathfrak{A} = (A, f, 0, 1)$  such that  $\mathfrak{A} \in CZ$  if and only if for input  $w$ : (the machine  $M$  stops with cleared tape without having done any step in the negative side of the tape) or (the machine  $M$  makes at least one step in the negative side of its tape and the first time when  $M$  makes such a step the tape of  $M$  is cleared).*

The Lemma 9 together with the Theorem of Rice implies in fact alone the Theorem 5, but is more natural to start considering the easy Lemma 8. Both Lemmas are proved by interpreting temporally successive tape configurations of the Turing machine  $M$  in diagonals of the recurrent double sequence, given by  $(a_{i,j})$  with  $i + j = k$  and  $k$  constant. To prove the Lemma 8 one needs two alternating types of diagonals. The diagonals of type 0 encode the tape configurations. The diagonals of type 1 are needed only to transmit the information between two diagonals of type 0:

$$\begin{array}{ccc} & b & \\ \delta & (\delta, b) & \\ a (a, \delta) & c & \end{array}$$

This idea of proof is also used in the more difficult situation of the Lemma 9. There we cannot use ordered pairs of symbols because of the commutativity of the algebra to construct.

**Definition 15.** Let  $\Gamma \neq \emptyset$  be a set and  $\equiv$  be the partition of  $\Gamma \times \Gamma$  consisting of the following sets: for all  $a \in \Gamma$  the singleton sets  $\{(a, a)\}$  and for all  $a, b \in \Gamma$  with  $a \neq b$  the two-element sets  $\{(a, b), (b, a)\}$ . Then  $\equiv$  is an equivalence relation over  $\Gamma$ . Consider the set of equivalence classes:

$$\Gamma \cdot \Gamma = (\Gamma \times \Gamma) / \equiv$$

which is the set of unordered pairs of elements of  $\Gamma$ . We denote the equivalence class of  $(a, b)$  with  $[a, b]$  and call this the unordered pair of  $a$  and  $b$ .

For the proof of Lemma 9, the alphabet of  $\Sigma \cup \Sigma \times Q$  is extended by disjoint union with two copies of itself and all letters  $c$  are encoded in special words  $cc'c''$ . For the simulation of the Halting Problem in recurrent double sequences, one needs eight types of successive diagonals. Only the diagonals of type zero simulates the Turing tape. The alphabets used for the diagonals of type  $i > 0$  is always  $\Gamma_i = \Gamma_{i-1} \cdot \Gamma_{i-1}$  with  $\Gamma_0 = \Sigma \cup \Sigma \times Q$  together with its two disjoint copies. The only thing to prove is that one can symmetrically define the function  $f$  over diagonals of type seven such that one gets the successor Turing machine configuration on the next diagonal, which is indeed of type zero.

## References

1. **Wolfram Köpf:** *Hypergeometric Summation. An Algorithmic Approach to Summation and Special Function Identities*. Vieweg, Braunschweig/Wiesbaden, 1998. <http://www.mathematik.uni-kassel.de/koepf/hyper.html>
2. **Benoit B. Mandelbrot:** *The fractal geometry of nature*. W. H. Freeman and Company, San Francisco, 1977, 1982.
3. **Dann E. Passoja, Akhlesh Lakhtakia:** *Carpets and rugs: an exercise in numbers*. Leonardo, 25, 1, 1992, 69 - 71.
4. **Dann E. Passoja, Akhlesh Lakhtakia:** *Variations on a Persian theme*. Journal of Recreational Mathematics, 24, 1, 1 - 5, 1992.
5. **Mihai Prunescu:** *Self-similar carpets over finite fields*. Submitted.
6. **Mihai Prunescu:** *An undecidable property of recurrent double sequences*. To appear in The Notre Dame Journal of Formal Logic, 2008.
7. **Marko Petkovsek, Herbert Wilf and Doron Zeilberger:** *A = B*. A K Peters. Ltd, 1997. <http://www.cis.upenn.edu/wilf/AeqB.html>.
8. **Gordon H. Rice:** *Classes of recursively enumerable sets and their decision problems*. Transactions of the American Mathematical Society, 74, 358 - 366, 1953.
9. **N. J. Rose:** *The Pascal triangle and Sierpinski's tree*. Mathematical Calendar, Releigh, N. C, Rome Press, 1981.
10. **Marjorie Senechal:** *Quasicrystals and Geometry*. Cambridge University Press, 1995.
11. **Waclaw Sierpinski:** *Sur une courbe cantorienne qui contient une image biunique et continue de toute courbe donne*. C. R. Acad. Sci, Paris, Sr. 162, 629, 1916.
12. **Herbert S. Wilf:** *Generatingfunctionology*. Academic Press, 1990, 1994.
13. **Stephen J. Willson:** *Cellular automata can generate fractals*. Discrete Applied Mathematics, 8, 1984, 91 - 99.

# Simulations of Quantum Turing Machines by Quantum Multi-Counter Machines<sup>\*</sup>

Daowen Qiu

Department of Computer Science, Zhongshan University, Guangzhou 510275, China  
{E-mail:issqdw@mail.sysu.edu.cn(D.Qiu)}

**Abstract.** We establish a kind of quantum multi-stack machines and quantum multi-counter machines, and use them to simulate quantum Turing machines. The major technical contributions are stated as follows: (i) We define *quantum multi-stack machines* (abbr. QMSMs) by generalizing a kind of *quantum pushdown automata* (abbr. QPDAs), and the *well-formedness* (abbr. W-F) conditions for characterizing the unitary evolution of the QMSMs are presented. (ii) By means of QMSMs we define *quantum multi-counter machines* (abbr. QMCMs) whose state transition functions are different from the *quantum counter automata* (abbr. QCAs) in the literature. (iii) To simulate *quantum Turing machines* (abbr. QTMs), we show that any given QMCM allowed to count with  $\pm n$  for  $n > 1$  can be simulated by another QMCM that counts with  $0, \pm 1$  only. (iv) We demonstrate the efficient simulations of QTMs in terms of QMSMs, and show that QTMs can be simulated by QMCMs as well.

**Key words:** Quantum Computation; Quantum Turing Machines; Quantum Multi-Counter Machines; Quantum Multi-Stack Machines.

## 1 Introduction

### 1.1 Motivation and purpose

Quantum computing is an intriguing and promising research field, which touches on quantum physics, computer science, and mathematics [8]. To a certain extent, this intensive attention given by the research community originated from Shor's findings of quantum algorithms for factoring large integers in polynomial time [17] and Grover's algorithm of searching in a database of size  $n$  with only  $O(\sqrt{n})$  accesses [7] which could also be sped up on a quantum computer.

Let us briefly recall the work of pioneers in this area. In 1980, Benioff [1] first considered that the computing devices in terms of the principles of quantum mechanics could be at least as powerful as classical computers. Then Feynman

---

\* This work was supported by the National Natural Science Foundation under Grant 90303024 and Grant 60573006, the Research Foundation for the Doctorial Program of Higher School of Ministry of Education under Grant 20050558015, and Program for New Century Excellent Talents in University (NCET) of China.

[4] pointed out that there appears to be no efficient way of simulating a quantum mechanical system on a classical computer, and suggested that a computer based on quantum physical principles might be able to carry out the simulation efficiently. In 1985 Deutsch [3] re-examined the Church-Turing Principle and defined *quantum Turing machines* (abbr. QTMs).

Quantum computation from the complexity theoretical viewpoint was studied systematically by Bernstein and Vazirani [2] and they described an efficient universal QTM that can simulate a large class of QTMs. Notably, in 1993 Yao [19] demonstrated the equivalence between QTMs and quantum circuits. More exactly, Yao [19] showed that for any given QTM, there exists a quantum Boolean circuit  $(n, t)$ -simulating this QTM with polynomial time slowdown, where  $n$  denotes the length of input strings, and  $t$  is the number of move steps before the machine stops.

In the theory of classical computation [10], both 2-stack machines, as a generalization of pushdown automata, and 2-counter machines can efficiently simulate Turing machines [12, 5, 10]. However, as far as the author is aware, the simulations of QTMs in terms of QMSMs and QMCMs still have not been considered. Since Turing machines, circuits, multi-stack machines, and multi-counter machines are equivalent in classical computation, we naturally hope to clarify their computing power in quantum computers. Therefore, our focuses in this article are to introduce QMSMs and QMCMs that are somewhat different from the *quantum counter automata* (abbr. QCAs) in the literature [11, 18], and particularly, to simulate QTMs by virtue of these two quantum computing devices.

Indeed, in quantum computing devices, the unitarity of evolution operators is generally characterized by the W-F conditions of the local transition function of the quantum models under consideration. Bernstein and Vazirani [2] gave the W-F conditions for the QTMs whose read/write heads are not allowed to be stationary in each move. In QTMs whose read/write heads are allowed to be stationary (called *generalized QTMs*, as in [2]), the first sufficient conditions for preserving the unitarity of time evolution were given by Hirvensalo [9], and then Ozawa and Nishimura [14] further presented the W-F conditions for the general QTMs. For the details, see ([8], p. 173). Also, Yamakami [20] gave the simple W-F conditions for multiple-tape stationary-head-move QTMs. Golovkins [6] defined a kind of QPDAs and gave the corresponding W-F conditions; Yamasaki *et. al.* [18] defined quantum 2-counter automata and presented the corresponding W-F conditions, as well.

We see that those aforementioned W-F conditions given by these authors for corresponding quantum computing devices are quite complicated. Therefore, based on the QPDAs proposed in [16] where QPDAs in [16] and [13] are shown to be equivalent, we would like to define QMSMs that generalize the QPDAs in [16], and further define QMCMs. Also, we will give the W-F conditions for these defined devices. Notably, these W-F conditions are more succinct than those mentioned above. In particular, motivated by Yao's work [19] concerning the  $(n, t)$ -simulations of QTMs by quantum circuits, we will use QMSMs to  $(n, t)$ -simulate QTMs, where  $n$  denotes that the length of input strings are not beyond

$n$ , and  $t$  represents that the number of move steps of QTMs (time complexity) is not bigger than  $t$  for those input strings.

## 1.2 Main results

According to the above analysis, we state the main contributions in this article. In Section 2, we define QMSMs by generalizing QPDAs in [16] from one-stack to multi-stack and present the corresponding W-F conditions (Theorem 1) for the quantum devices.

In Section 3, by means of QMSMs we define QMCMs that are somewhat different from the QCAs by Kravtsev [11] and Yamasaki *et al.* [18]; also, the W-F conditions (Theorem 2) are given for the defined QMCMs. It is worth indicating that the state transition functions in QCAs defined by Kravtsev [11] and Yamasaki *et al.* [18] have local property, since they are defined on  $Q \times \{0, 1\} \times (\Sigma \cup \{\#, \$\}) \times Q \times \{0, 1\}$ , but their W-F conditions are quite complicated, while in QMCMs defined in this article, the state transition functions are on  $Q \times \mathbf{N}^k \times (\Sigma \cup \{\#, \$\}) \times Q \times \mathbf{N}^k$ , and consequently, the corresponding W-F conditions are more succinct (see Theorem 2). To simulate QTMs, we deal with a number of properties regarding simulations between QMCMs with different counters and different counts (Lemmas 1 and 2). We show that QMCMs allowed to count with  $0, \pm 1, \pm 2, \dots, \pm n$  can be simulated by QMCMs that are able to count with  $0, \pm 1$  only but need more counters.

In particular, in Section 4, we present the simulations of QTMs in terms of QMCMs with polynomial time slowdown. More specifically, we prove that for any QTM  $M_1$ , there exists QMCM  $M_2$   $(n, t)$ -simulating  $M_1$ , where  $n$  denotes the length of input strings not bigger than  $n$ , and  $t$  represents that the number of move steps of QTMs is not bigger than  $t$  for those input strings. Also, we show that QMCMs can be simulated by QMSMs with the same time complexity, and by this result it then follows the efficient simulations of QTMs by QMSMs.

Due to the limit space, all proofs in the article are omitted and are referred to [15].

## 2 Quantum multi-stack machines

Here we will define quantum  $k$ -stack machines by generalizing the QPDAs in [16] from one stack to  $k$  stacks.

**Definition 1.** A quasi-quantum two-stack machine is defined as  $M = (Q, \Sigma, \Gamma, \delta, Z_0, q_0, q_a, q_r)$  where  $Q$  is the set of states,  $\Sigma$  is the input alphabet,  $\Gamma$  is the stack alphabet,  $Z_0 \in \Gamma$  denotes the bottom symbol that is not allowed to be popped,  $q_0 \in Q$  is the initial state, and  $q_a, q_r \in Q$  are respectively the accepting and rejecting states, and transition function  $\delta$  is defined as follows:

$$\delta : Q \times \Gamma^* \times \Gamma^* \times (\Sigma \cup \{\#, \$\}) \times Q \times \Gamma^* \times \Gamma^* \rightarrow \mathbf{C}$$

where  $\Gamma^*$  denotes the set of all strings over  $\Gamma$ , and  $\delta(q, \gamma_1, \gamma_2, \sigma, q', \gamma'_1, \gamma'_2) \neq 0$  if and only if (i)  $\gamma_1 = \gamma'_1$  or  $X\gamma_1 = \gamma'_1$  or  $\gamma_1 = X\gamma'_1$  for some  $X \in \Gamma \setminus \{Z_0\}$ ; and (ii)

$\gamma_2 = \gamma'_2$  or  $Y\gamma_2 = \gamma'_2$  or  $\gamma_2 = Y\gamma'_2$  for some  $Y \in \Gamma \setminus \{Z_0\}$ . In addition, if  $\gamma_1 = Z_0$ , then  $\gamma_1 = Z_0$  or  $\gamma_1 = ZZ_0$  for some  $Z \in \Gamma$  with  $Z \neq Z_0$ ; similar restriction is imposed on  $\gamma_2$ .

A configuration of the machine is described by  $|q\rangle|\gamma_1\rangle|\gamma_2\rangle$ , where  $q$  is the current control state,  $\gamma_1$  and  $\gamma_2$  represent the current strings of stack symbols in two stacks, respectively, and the leftmost symbol of  $\gamma_i$  represents the top stack symbol of stack  $i$  for  $i = 1, 2$ . Therefore, the rightmost symbol of  $\gamma_i$  is  $Z_0$ . We denote by  $C_M$  the set of all configurations of  $M$ , that is,

$$C_M = \{|q\rangle|\gamma_1 Z_0\rangle|\gamma_2 Z_0\rangle : q \in Q, \gamma_i \in \Gamma^* \setminus Z_0, i = 1, 2\}.$$

Let  $H_X$  represent the Hilbert space whose orthonormal basis is the set  $X$ , that is,  $H_X = l_2(X)$ . Therefore,  $H_Q \otimes H_{\Gamma^*} \otimes H_{\Gamma^*}$  is a Hilbert space whose orthonormal basis is  $C_M$ , that is,  $l_2(C_M) = H_Q \otimes H_{\Gamma^*} \otimes H_{\Gamma^*}$ . In addition, we assume that there are endmarkers  $\#$  and  $\$$  representing the leftmost and rightmost symbols for any input string  $x \in \Sigma^*$ , respectively. Therefore, any input string  $x \in \Sigma^*$  is put on the input tape in the form of  $\#x\$$ , and the read head of  $M$  begins with  $\#$  and ends after reading  $\$$ .

For any  $\sigma \in \Sigma \cup \{\#, \$\}$  we defined the time evolution operators  $U_\sigma$  and  $U'_\sigma$  from  $H_Q \otimes H_{\Gamma^*} \otimes H_{\Gamma^*}$  to  $H_Q \otimes H_{\Gamma^*} \otimes H_{\Gamma^*}$  as follows:

$$U_\sigma(|q\rangle|\gamma_1\rangle|\gamma_2\rangle) = \sum_{q', \gamma'_1, \gamma'_2} \delta(q, \gamma_1, \gamma_2, \sigma, q', \gamma'_1, \gamma'_2) |q'\rangle|\gamma'_1\rangle|\gamma'_2\rangle, \quad (1)$$

$$U'_\sigma(|q\rangle|\gamma_1\rangle|\gamma_2\rangle) = \sum_{q', \gamma'_1, \gamma'_2} \delta^*(q', \gamma'_1, \gamma'_2, \sigma, q, \gamma_1, \gamma_2) |q'\rangle|\gamma'_1\rangle|\gamma'_2\rangle, \quad (2)$$

where  $\delta^*$  denotes the conjugate complex number  $\delta$ . By linearity  $U_\sigma$  and  $U'_\sigma$  can be extended to  $H_Q \otimes H_{\Gamma^*} \otimes H_{\Gamma^*}$ .

**Remark 1.**  $U'_\sigma$  is the adjoint operator of  $U_\sigma$ . Indeed, for any  $(q_i, \gamma_{i1}, \gamma_{i2}) \in Q \times \Gamma^* \times \Gamma^*$ , by means of Eqs. (1,2) we have

$$\begin{aligned} & \langle U_\sigma|q_1\rangle|\gamma_{11}\rangle|\gamma_{12}\rangle, U_\sigma|q_2\rangle|\gamma_{21}\rangle|\gamma_{22}\rangle \rangle \\ &= \sum_{q, \gamma_1, \gamma_2} \delta(q_1, \gamma_{11}, \gamma_{12}, \sigma, q, \gamma_1, \gamma_2) \times \delta^*(q_2, \gamma_{21}, \gamma_{22}, \sigma, q, \gamma_1, \gamma_2) \\ &= \left\langle |q_1\rangle|\gamma_{11}\rangle|\gamma_{12}\rangle, U'_\sigma U_\sigma|q_2\rangle|\gamma_{21}\rangle|\gamma_{22}\rangle \right\rangle. \end{aligned} \quad (3)$$

**Definition 2.** Let  $M$  be a quasi-quantum two-stack machine with input alphabet  $\Sigma$ . If  $U_\sigma$  is unitary for any  $\sigma \in \Sigma \cup \{\#, \$\}$ , then  $M$  is called a quantum two-stack machine.

Now we give the well-formedness conditions for justifying the unitarity of  $U_\sigma$  for any  $\sigma \in \Sigma \cup \{\#, \$\}$ .

**Theorem 1.** Let  $M$  be a quasi-quantum two-stack machine with input alphabet  $\Sigma$ . Then for any  $\sigma \in \Sigma \cup \{\#, \$\}$ , linear operator  $U_\sigma$  is unitary if and only if  $\delta$  satisfies the following well-formedness conditions:

(I) For any  $\sigma \in \Sigma \cup \{\#, \$\}$ ,

$$\begin{aligned} & \sum_{q', \gamma'_1, \gamma'_2} \delta(q_1, \gamma_{11}, \gamma_{12}, \sigma, q', \gamma'_1, \gamma'_2) \times \delta^*(q_2, \gamma_{21}, \gamma_{22}, \sigma, q', \gamma'_1, \gamma'_2) \\ &= \begin{cases} 1, & \text{if } (q_1, \gamma_{11}, \gamma_{12}) = (q_2, \gamma_{21}, \gamma_{22}), \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \quad (4)$$

(II) For any  $\sigma \in \Sigma \cup \{\#, \$\}$ ,

$$\begin{aligned} & \sum_{q', \gamma'_1, \gamma'_2} \delta(q', \gamma'_1, \gamma'_2, \sigma, q_1, \gamma_{11}, \gamma_{12}) \times \delta^*(q', \gamma'_1, \gamma'_2, \sigma, q_2, \gamma_{21}, \gamma_{22}) \\ &= \begin{cases} 1, & \text{if } (q_1, \gamma_{11}, \gamma_{12}) = (q_2, \gamma_{21}, \gamma_{22}), \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \quad (5)$$

### 3 Quantum multi-counter machines

As stated above, QCAs were first considered by Kravtsev [11], and further developed by Yamasaki et al. [18]. In this section, we introduce a different definition of quantum  $k$ -counter machines.

**Definition 3.** A quasi-quantum  $k$ -counter machine is defined as  $M = (Q, \Sigma, \delta, q_0, q_a, q_r)$  where  $Q$  is a set of states with initial state  $q_0 \in Q$  and states  $q_a, q_r \in Q$  representing accepting and rejecting states, respectively,  $\Sigma$  is an input alphabet, and transition function  $\delta$  is a mapping from  $Q \times \mathbf{N}^k \times (\Sigma \cup \{\#, \$\}) \times Q \times \mathbf{N}^k$  to  $\mathbf{C}$ , where  $\mathbf{N}$  denotes the set of all nonnegative integer and  $\#, \$$  represent two endmarkers that begins with  $\#$  and ends with  $\$$ , and  $\delta$  satisfies that

$$\delta(q, n_1, n_2, \dots, n_k, \sigma, q', n'_1, n'_2, \dots, n'_k) \neq 0 \quad (6)$$

only if  $|n_i - n'_i| \leq 1$  for  $i = 1, 2, \dots, k$ . Furthermore, let  $|q\rangle|n_1\rangle|n_2\rangle\dots|n_k\rangle$  represent a configuration of  $M$ , where  $q \in Q$ ,  $n_i \in \mathbf{N}$  for  $i = 1, 2, \dots$ , and let the set  $C_M = \{|q\rangle|n_1\rangle|n_2\rangle\dots|n_k\rangle : q \in Q, n_i \in \mathbf{N}, i = 1, 2, \dots, k\}$  be an orthonormal basis for the space  $H_{C_M} = l_2(C_M)$ . For any  $\sigma \in \Sigma$ , linear operator  $V_\sigma$  on  $H_{C_M}$  is defined as follows:

$$\begin{aligned} & V_\sigma|q\rangle|n_1\rangle|n_2\rangle\dots|n_k\rangle \\ &= \sum_{q', n'_1, n'_2, \dots, n'_k} \delta(q, n_1, n_2, \dots, n_k, \sigma, q', n'_1, n'_2, \dots, n'_k) |q'\rangle|n'_1\rangle|n'_2\rangle\dots|n'_k\rangle \end{aligned} \quad (7)$$

and  $V_\sigma$  is extended to  $H_{C_M}$  by linearity.

**Definition 4.** We say that the quasi-quantum counter machine  $M = (Q, \Sigma, \delta, q_0, q_a, q_r)$  defined above is a *quantum  $k$ -counter machine*, if  $V_\sigma$  is unitary for any  $\sigma \in (\Sigma \cup \{\#, \$\})$ . Also, we define linear operator  $V'_\sigma$  on  $H_{C_M}$  as follows:

$$\begin{aligned} & V'_\sigma|q\rangle|n_1\rangle|n_2\rangle\dots|n_k\rangle \\ &= \sum_{q', n'_1, n'_2, \dots, n'_k} \delta^*(q', n'_1, n'_2, \dots, n'_k, \sigma, q, n_1, n_2, \dots, n_k) |q'\rangle|n'_1\rangle|n'_2\rangle\dots|n'_k\rangle. \end{aligned} \quad (8)$$

**Remark 2.** Clearly  $V'_\sigma$  is an adjoint operator of  $V_\sigma$ , which can be checked in terms of the process of Remark 1, and the details are therefore omitted here.

Now we give the W-F conditions for characterizing the unitarity of  $V_\sigma$ . Without loss of generality, we deal with the case of  $k = 2$ .

**Theorem 2.** Let  $M$  be a quasi-quantum two-counter machine with input alphabet  $\Sigma$ . Then for any  $\sigma \in \Sigma \cup \{\#, \$\}$ ,  $V_\sigma$  defined as Eq. (7) is unitary if and only if  $\delta$  satisfies the following W-F conditions:

(I) For any  $\sigma \in \Sigma \cup \{\#, \$\}$ ,

$$\begin{aligned} & \sum_{p,n'_1,n'_2} \delta(q_1, n_{11}, n_{12}, \sigma, p, n'_1, n'_2) \times \delta^*(q_2, n_{21}, n_{22}, \sigma, p, n'_1, n'_2) \\ &= \begin{cases} 1, & \text{if } (q_1, n_{11}, n_{12}) = (q_2, n_{21}, n_{22}), \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \quad (9)$$

(II) For any  $\sigma \in \Sigma \cup \{\#, \$\}$ ,

$$\begin{aligned} & \sum_{p,n'_1,n'_2} \delta(p, n'_1, n'_2, \sigma, q_1, n_{11}, n_{12}) \times \delta^*(p, n'_1, n'_2, \sigma, q_2, n_{21}, n_{22}) \\ &= \begin{cases} 1, & \text{if } (q_1, n_{11}, n_{12}) = (q_2, n_{21}, n_{22}), \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \quad (10)$$

In order to simulate QTMs by QMSMs, we need some related lemmas and definitions.

**Definition 5.** A quasi-quantum  $k$ -counter machine  $M = (Q, \Sigma, \delta, q_0, q_a, q_r)$  is called to *count with*  $\pm r$  for  $r \geq 1$ , if its  $k$ 's counters are allowed to change with numbers  $0, \pm 1$ , or  $\pm r$  at each step. In this case, if  $|n_i - n'_i| \leq 1$  or  $|n_i - n'_i| = r$  for  $i = 1, 2, \dots, k$ , then  $\delta(q, n_1, n_2, \dots, n_k, \sigma, q', n'_1, n'_2, \dots, n'_k) \neq 0$  may hold; otherwise it is 0. We say that the quasi-quantum  $k$ -counter machine  $M$  is quantum if for any  $\sigma \in \Sigma \cup \{\#, \$\}$ ,  $V_\sigma$  is a unitary operator on  $l_2(C_M)$ , where

$$C_M = \{|q\rangle |n_1\rangle |n_2\rangle \dots |n_k\rangle : q \in Q, n_i \in \mathbf{N}, i = 1, 2, \dots, k\}.$$

It is ready to obtain that Theorem 2 also holds for quantum  $k$ -counter machines with count  $\pm r$  for  $r \geq 1$ .

**Theorem 3.** Let  $M = (Q, \Sigma, \delta, q_0, q_a, q_r)$  be a quasi-quantum  $k$ -counter machine that is allowed to count with a certain  $\pm r$  for  $r \geq 1$ . Then for any  $\sigma \in \Sigma \cup \{\#, \$\}$ ,  $V_\sigma$  defined as Eq. (7) is unitary if and only if  $\delta$  satisfies Eqs. (9,10).

**Definition 6.** Let  $M_1$  and  $M_2$  be quantum  $k_1$ -counter machine  $M_1$  and quantum  $k_2$ -counter machine  $M_2$ , respectively, and,  $M_1$  and  $M_2$  have the same input alphabet  $\Sigma$ . For any  $\sigma \in \Sigma \cup \{\#, \$\}$ ,  $V_\sigma^{(1)}$  and  $V_\sigma^{(2)}$  defined as Eq. (7) represent the evolution operators in  $M_1$  and  $M_2$ , respectively. We say that  $M_1$  can simulate  $M_2$ , if for any string  $\sigma_1 \sigma_2 \dots \sigma_n \in \Sigma^*$ ,

$$\sum_{i_1, i_2, \dots, i_{k_1} \geq 0} \left| \langle i_{k_1} | \dots \langle i_1 | \langle q_a^{(1)} | V_\$^{(1)} V_{\sigma_n}^{(1)} V_{\sigma_{n-1}}^{(1)} \dots V_{\sigma_1}^{(1)} V_\#^{(1)} | q_0^{(1)} \rangle | 0 \rangle \dots | 0 \rangle \right|^2$$

$$= \sum_{j_1, j_2, \dots, j_{k_1} \geq 0} \left| \langle j_{k_1} | \dots \langle j_1 | \langle q_a^{(2)} | V_{\$}^{(2)} V_{\sigma_n}^{(2)} V_{\sigma_{n-1}}^{(2)} \dots V_{\sigma_1}^{(2)} V_{\#}^{(2)} | q_0^{(2)} \rangle | 0 \rangle \dots | 0 \rangle \right|^2 \quad (11)$$

where  $q_0^{(i)}$  and  $q_a^{(i)}$  denote the initial and accepting states of  $M_i$ , respectively,  $i = 1, 2$ .

For convenience, for any quantum  $k$ -counter machine  $M = (Q, \Sigma, \delta, q_0, q_a, q_r)$ , we define the accepting probability  $P_{accept}^M(\sigma_1 \sigma_2 \dots \sigma_n)$  for inputting  $\sigma_1 \sigma_2 \dots \sigma_n$  as:

$$\begin{aligned} P_{accept}^M(\sigma_1 \sigma_2 \dots \sigma_n) \\ = \sum_{i_1, i_2, \dots, i_k \geq 0} \left| \langle i_k | \dots \langle i_1 | \langle q_a | V_{\$}^M V_{\sigma_n}^M V_{\sigma_{n-1}}^M \dots V_{\sigma_1}^M V_{\#}^M | q_0 \rangle | 0 \rangle \dots | 0 \rangle \right|^2, \end{aligned} \quad (12)$$

where  $V_\sigma^M$  is unitary operator on  $l_2(C_M)$  for any  $\sigma \in \Sigma \cup \{\#, \$\}$ .

**Lemma 1.** For any quantum  $k$ -counter machine  $M_1$  that is allowed to count with  $\pm r$  for  $r \geq 1$ , there exists quantum  $2k$ -counter machine  $M_2$  simulating  $M_1$  with the same time complexity, where  $M_2$  is allowed to count with  $0, \pm 1$ , and  $\pm(r - 1)$ .

**Lemma 2.** For any quantum  $k$ -counter machine  $M_1$  that is allowed to count with  $0, \pm 1, \pm 2, \dots, \pm r$ , then there exists a quantum  $kr$ -counter machine  $M_2$  simulating  $M_1$  with the same time complexity, where  $M_2$  is allowed to count with  $0, \pm 1$  only.

## 4 Simulations of quantum Turing machines

To simulate QTMs in terms of QMCMs, we give the definition of QTMs in terms of Bernstein and Vazirani [2], in which the read-write head will move either to the right or to the left at each step. Indeed, generalized QTMs can also be simulated by QMCMs, but the discussion regarding unitarity is much more complicated. For the sake of simplicity, we here consider the former QTMs.

**Definition 7.** A QTM is defined by  $M = (\Sigma, Q, \delta, B, q_0, q_a, q_r)$ , where  $\Sigma$  is a finite input alphabet,  $B$  is an identified blank symbol,  $Q$  is a finite set of states with an identified initial state  $q_0$  and final state  $q_a, q_r \neq q_0$ , where  $q_a$  and  $q_r$  represent accepting and rejecting states, respectively, and the quantum transition function  $\delta$  is defined as

$$\delta : Q \times \Sigma \times \Sigma \times Q \times \{L, R\} \rightarrow \mathbf{C}.$$

The QTM has a two-way infinite tape of cells indexed by  $\mathbf{Z}$  and a single read-write tape head that moves along the tape. A configuration of this machine is described by the form  $|q\rangle|\tau\rangle|i\rangle$ , where  $q$  denotes the current state,  $\tau \in \Sigma^{\mathbf{Z}}$  describes the tape symbols, and  $i \in \mathbf{Z}$  represents the current position of tape head. Naturally, a configuration containing initial or final state is called an initial or final configuration. Let  $C_M$  denote the set of all configurations in  $M$ , and therefore  $H_{C_M} = l_2(C_M)$ , that is a Hilbert space whose orthonormal basis can

be equivalently viewed as  $C_M$ . Then the evolution operator  $U_M$  on  $l_2(C_M)$  can be defined in terms of  $\delta$ : for any configuration  $|c\rangle \in C_M$ ,

$$U_M|c\rangle = \sum_{|c'\rangle \in C_M} a(c, c')|c'\rangle, \quad (13)$$

where  $a(c, c')$  is the amplitude of configuration  $|c\rangle$  evolving into  $|c'\rangle$  in terms of the transition function  $\delta$ .  $U_M$  is a unitary operator on  $l_2(C_M)$ .

As in [2], we define that QTM halts with running time  $T$  on input  $x$  if after the  $T$ 's step moves beginning with its initial configuration, the superposition contains only final configurations, and at any time less than  $T$  the superposition contains no final configuration. Therefore, we assume that the QTM satisfies this requirement.

**Definition 8.** For nonnegative integer  $n, T$ , let  $M_1 = (Q_1, \Sigma_1, \delta_1, B_1, q_{10}, q_{1a}, q_{1r})$  be a quantum Turing machine with initial state  $q_{10}$ , and let  $M_2$  be a quantum  $k$ -counter machine with initial state  $q_{20}$  and the same input alphabet  $\Sigma_1$  as  $M_1$ . We say that  $M_2$  ( $n, T$ )-simulates quantum Turing machine  $M_1$  with *polynomial time*  $O(n, T)$  *slowdown*, if there exist some tape symbols added in  $M_2$ , say  $B_2, B_3, \dots, B_m$  such that for any input  $x = \sigma_1\sigma_2\dots\sigma_l \in \Sigma_2^*$  ( $l \leq n$ ), if the computation of  $M_1$  ends with  $t$  steps ( $t \leq T$ ), then there is nonnegative integers  $k_{l_1}, k_{l_2}, \dots, k_{l_{m_1}}$  and  $k_{s_1}, k_{s_2}, \dots, k_{s_{m_2}}$  that are related to  $l$  and  $t$ , satisfying  $\sum_{i=1}^{m_1} k_{l_i} + \sum_{i=1}^{m_2} k_{s_i} \leq O(n, T)$ , and

$$P_a^{M_1}(x) = P_a^{M_2}(x), \quad (14)$$

where

$$P_a^{M_1}(x) = \sum_{-T \leq i \leq T, \tau \in \Sigma^{[-T, T]}} |\langle i | \langle \tau | \langle q_{1a} | U_{M_1}^t | q_{10} \rangle | \tau_0 \rangle | 0 \rangle|^2 \quad (15)$$

where  $\tau_0$  is defined as:  $\tau_0(j) = \begin{cases} \sigma_{j+1}, & \text{if } j \in [0, l-1] \mathbf{Z}, \\ B_1, & j \in [-T, -1] \mathbf{Z} \cup [l, T] \mathbf{Z}, \end{cases}$  and

$$\begin{aligned} P_a^{M_2}(x) = \sum_{n_1, n_2, \dots, n_k \geq 0} & \left| \langle n_k | \dots \langle n_1 | \langle q_{2a} | V_{\$} V_{B_{s_{m_2}}}^{k_{s_{m_2}}} \dots V_{B_{s_1}}^{k_{s_1}} \right. \\ & \left. V_{\sigma_l} V_{\sigma_{l-1}} \dots V_{\sigma_1} V_{B_{l_{m_1}}}^{k_{l_{m_1}}} \dots V_{B_{l_1}}^{k_{l_1}} V_{\#} | q_0 \rangle | 0 \rangle \dots | 0 \rangle \right|^2. \end{aligned} \quad (16)$$

One of the main result is as follows:

**Theorem 4.** For any QTM  $M_1 = (Q_1, \Sigma_1, \delta_1, B_1, q_{10}, q_{1a}, q_{1r})$  with initial state  $q_{10}$  and accepting and rejecting states  $q_{1a}, q_{1r}$ , and for any nonnegative integer  $n, t$  with  $n \leq t + 1$ , there exists a quantum  $(2t+2)$ -counter machine  $M_2$  that  $(n, t)$ -simulates  $M_1$  with most slowdown  $O(n + t)$ .

QTMs can be also  $(n, t)$ -simulated by quantum multi-stack machine, since quantum  $k$ -counter machine can be simulated by quantum multi-stack machine in terms of the following Theorem 5.

**Definition 9.** We say that quantum  $k$ -stack machine  $M_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2, Z_0, q_{20}, q_{2a}, q_{2r})$  simulates quantum  $k$ -counter machine  $M_1 = (Q_1, \Sigma_1, \delta_1, q_{10}, q_{1a}, q_{1r})$

that has the same input alphabet  $\Sigma_1 = \Sigma_2$  with the same time complexity in the sense of any efficient overhead, if for any input string  $x = \sigma_1\sigma_2\dots\sigma_n \in \Sigma_1^*$ , we have

$$P_{accept}^{M_1}(x) = P_{accept}^{M_2}(x) \quad (17)$$

where

$$\begin{aligned} & P_{accept}^{M_1}(x) \\ &= \sum_{\gamma_1, \gamma_2, \dots, \gamma_k} |\langle \gamma_k | \langle \gamma_{k-1} | \dots \langle \gamma_1 | \langle q_{1a} | U_{\$} U_{\sigma_n} \dots U_{\sigma_1} U_{\#} | q_{10} \rangle | 0 \rangle \dots | 0 \rangle|^2. \end{aligned} \quad (18)$$

Also, the  $(n, t)$ -simulations of QTMs in terms of quantum  $k$ -stack machine can be similarly defined as Definition 8, and we leave out the details here.

**Theorem 5.** For any given quantum  $k$ -counter machine  $M_1 = (Q_1, \Sigma_1, \delta_1, q_{10}, q_{1a}, q_{1r})$ , there exists quantum  $k$ -stack machine  $M_2$  that simulates  $M_1$  with the same time complexity.

**Corollary 1.** For any  $n, t \in \mathbb{N}$ , and any QTM  $M_1$ , there exists QMSM  $M_2$  that simulates  $M_1$  with slowdown  $O(n + t)$ .

## 5 Concluding remarks

The unitary evolution of quantum physics requires that quantum computation should be necessarily time reversible (unitary). This makes some simulations between quantum computing devices quite complicated. Indeed, the unitarity is reflected by the W-F conditions. The W-F conditions for these QMSMs and QMCMs defined in this paper are more succinct than the W-F conditions for QCAs introduced by Yamasaki *et al.* [18], but we note that the transition functions in our quantum devices employ the whole property of the symbols in the stacks or counters at each move. An issue worthy of further consideration is to give also succinct W-F conditions but yet more local transition functions for characterizing the unitarity of these QMSMs and QMCMs defined in this paper. Moreover, the relationships between QMCMs in the paper and QCAs by Yamasaki *et al.* [18] still need to be further clarified. Finally, how to improve the  $(n, t)$ -simulations of QTMs by QMCMs and QMSMs towards more general simulations and how to decrease the number of counters of QMCMs for simulating QTMs are also worth studying.

The unitarity of the above models of computation leads to the complicated W-F conditions [14, 18], while, in this article, the W-F conditions are simpler but, the transition functions lose local property. Therefore, defining the above models by means of *measurement-based quantum computation* [21] may be one of the feasible ways to solve this problem, since it is still physically allowed.

## Acknowledgment

The author would like to thank the four anonymous reviewers for invaluable comments and suggestions.

## References

1. P. Benioff, The computer as a physical system: a microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines, *J. Statist. Phys.* 22 (1980) 563-591.
2. E. Bernstein and U. Vazirani, Quantum complexity theory, *SIAM J. Comput.* 26 (1997) 1411-1473.
3. D. Deutsh, Quantum theory, the Church-Turing principle and the universal quantum computer, *Proc. Roy. Soc. London A* 400 (1985) 97-117.
4. R.P. Feynman, Simulating physics with computers, *Internat. J. Theoret. Phys.* 21 (1982) 467-488.
5. P.C. Fischer, Turing machine with restricted memory access, *Information and Control* 9 (4) (1966) 364-379.
6. M. Golovkins, Quantum Pushdown Automata, in: Proc. 27th Conf. on Current Trends in Theory and Practice of Informatics, Milovy, Lecture Notes in Computer Science, Vol. 1963, Springer-Verlag, Berlin, 2000, pp. 336-346.
7. L. Grover, A fast quantum mechanical algorithms for database search, in: Proc. 28th Annual ACM Symp. Theory of Computing, Philadelphia, Pennsylvania, 1996, pp. 212-219.
8. J. Gruska, *Quantum Computing*, McGraw-Hill, London, 1999.
9. M. Hirvensalo, On quantum computation. PhD thesis, Turku Center for Computer Science, 1997.
10. J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, New York, 1979.
11. M. Kravtsev, Quantum finite one-counter automata, in: SOFSEM'99, Lecture Notes in Computer Science, Vol.1725, Springer-Verlag, Berlin, 1999, pp.431-440.
12. M.L. Minsky, Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing machines, *Annals of Mathematics* 74 (3) (1961) 437-455.
13. C. Moore and J.P. Crutchfield, Quantum automata and quantum grammars, *Theoret. Comput. Sci.* 237 (2000) 275-306. Also quant-ph/9707031, 1997.
14. M. Ozawa, H. Nishimura, Local transition functions of quantum Turing machines, quant-ph/9811069, 1998.
15. D.W. Qiu, Simulations of quantum Turing machines by quantum multi-stack machines, quant-ph/0501176, 2005.
16. D.W. Qiu and M.S. Ying, Characterization of quantum automata, *Theoret. Comput. Sci.* 312 (2004) 479-489.
17. P.W. Shor, Algorithm for quantum computation: discrete logarithms and factoring, in: Proc. 37th IEEE Annu. Symp. on Foundations of Computer science, 1994, pp. 124-134.
18. T. Yamasaki, H. Kobayashi, H. Imai, Quantum versus deterministic counter automata, *Theoret. Comput. Sci.* 334 (2005) 275-297.
19. A.C. Yao, Quantum circuit complexity, in: Proc. 34th IEEE Symp. on Foundations of Computer science, 1993, pp. 352-361.
20. T. Yamakami, A Foundation of Programming a Multi-Tape Quantum Turing Machine, in: MFCS'99, Lecture Notes in Computer Science, Vol.1672, Springer-Verlag, Berlin, 1999, pp. 430-441.
21. R. Jozsa, An introduction to measurement based quantum computation, quant-ph/0508124, 2005.

# Optimal Proof Systems and Complete Languages (Extended Abstract)

Zenon Sadowski

Institute of Mathematics, University of Białystok  
15-267 Białystok, ul. Akademicka 2, Poland  
[sadowski@math.uwb.edu.pl](mailto:sadowski@math.uwb.edu.pl)

**Abstract.** We investigate the connection between optimal propositional proof systems and complete languages for promise classes. We prove that an optimal propositional proof system exists if and only if there exists a propositional proof system in which every promise class with the test set in co-NP is representable. Additionally, we prove that there exists a complete language for UP if and only if there exists a propositional proof system such that UP is representable in it. UP is the standard promise class with the test set in co-NP.

**Key words:** Optimal proof systems, promise classes, complete languages

## 1 Introduction

Although there are many different formal systems for proving propositional tautologies in logic textbooks, they all fall under the concept of an abstract propositional proof system (a proof system for *TAUT*) introduced by S. Cook and R. Reckhow [5]. In order to compare the relative strength of different proof systems for *TAUT* we use the notion of simulation [10] and the notion of p-simulation [5]. A proof system for *TAUT* is optimal (p-optimal) if and only if it simulates (p-simulates) any other proof system for *TAUT*. The still unresolved problem of the existence of an optimal (p-optimal) proof system for *TAUT* was posed by J. Krajíček and P. Pudlák [10] in 1989.

The notion of p-simulation between proof systems for *TAUT* is similar to the notion of reducibility between languages. Analogously, the notion of a p-optimal proof system for *TAUT* should correspond to the notion of a complete language.

Informally, a class of languages is a promise class if the languages in this class are accepted by nondeterministic polynomial-time clocked Turing machines which obey special conditions (promises). Common promise classes are **UP**, **NP**  $\cap$  **co-NP**, and **BPP**. It is still open whether there exist complete languages for these classes. The reason lies in the undecidability of the problem of whether a given nondeterministic polynomial-time Turing machine indeed obeys the promise of any of these classes. Moreover, there exist relativizations for which these classes do not have complete languages (see [8]).

Recently, O. Beyersdorff [3] introduced the notion of a disjoint **NP**-pair representable in a given propositional proof system  $f$ . The disjointness of such a pair is expressible by a sequence of propositional tautologies with short  $f$ -proofs. He also considered the complexity class of all disjoint **NP**-pairs representable in a proof system  $f$ . In this paper we extend these notions to any promise class with a propositionally expressible promise. It results in the notion of a language representable in a given proof system  $f$  and in the notion of a promise class representable in  $f$ . O. Beyersdorff proved [2] that the class of all disjoint **NP**-pairs has a complete pair if and only if there exists a proof system for  $TAUT$  in which every disjoint **NP**-pair is representable. We prove the analogous theorem for the class **UP**. Namely, **UP** has a complete language if and only if there exists a proof system for  $TAUT$  such that **UP** is p-representable in it.

It turns out that there is a close connection between optimal proof systems and complete languages for promise classes, namely, the existence of optimal proof systems implies the existence of complete languages for various promise classes (see [9]). Let us mention two exemplary results of this type. A. Razborov [14] observed that the existence of an optimal proof system suffices to guarantee the existence of complete disjoint **NP**-pairs. J. Messner and J. Torán showed [12] that a complete language for **UP** exists in case there is a p-optimal proof system for  $TAUT$ . The converses of these implications probably do not hold [6] and in this paper we address the question of just why it is so.

It seems that the promise that a Turing machine computes a proof system for  $TAUT$ , or more precisely it produces only propositional tautologies, is the hardest one among those promises which are propositionally expressible. Therefore, the sufficient condition for the existence of an optimal proof system should be as strong as the existence of a complete language for every promise class with a propositionally expressible promise. At present, this intuition is only supported by the result of J. Messner [11] which states that a p-optimal proof system for  $TAUT$  exists if and only if every promise function class with a test set polynomial-time reducible to  $TAUT$  has a complete function (see also [9]). The analogous theorem in the setting of promise language classes instead of promise function classes is missing. The main result from this paper, that the existence of an optimal proof system for  $TAUT$  is equivalent to the existence of a proof system for  $TAUT$  in which any promise class with the test set in  $\text{co-NP}$  is representable, may be treated as the first step in this direction.

## 2 Preliminaries

We assume some familiarity with basic complexity theory and refer the reader to [1] and [13] for standard notions and for definitions of complexity classes appearing in the paper. The class of all disjoint pairs  $(A, B)$  of **NP**-languages is denoted by **DisNP**.

The symbol  $\Sigma$  denotes a certain fixed finite alphabet throughout the paper. The set of all strings over  $\Sigma$  is denoted by  $\Sigma^*$ . For a string  $x$ ,  $|x|$  denotes the length of  $x$ .

Given two languages  $L_1$  and  $L_2$  ( $L_1, L_2 \subseteq \Sigma^*$ ), we say that  $L_1$  is polynomial-time many-one reducible to  $L_2$  if and only if there exists a polynomial-time computable function  $f : \Sigma^* \rightarrow \Sigma^*$  such that  $x \in L_1$  if and only if  $f(x) \in L_2$  holds for any  $x \in \Sigma^*$ .

We use Turing machines (acceptors and transducers) as our basic computational model. We will not distinguish between a machine and its code. For a Turing machine  $M$  the symbol  $L(M)$  denotes the language accepted by  $M$ .

We consider deterministic and nondeterministic polynomial-time clocked Turing machines ( $PTM$  and  $NPTM$  for short) with uniformly attached standard clocks that stop their computations in polynomial time (see [1]). We impose some restrictions on our encoding of these machines. From the code of any polynomial-time clocked Turing machine we can easily detect (in polynomial time) the polynomial  $p_N$  which is its polynomial-time bound.

Let  $D_1, D_2, D_3, \dots$  and  $N_1, N_2, N_3, \dots$  be respectively standard enumerations of all deterministic and nondeterministic polynomial-time clocked Turing machines. For any class of languages  $\mathbf{C}$ , we say that  $\mathbf{C}$  has an uniform enumeration if and only if there exists a recursively enumerable list of nondeterministic polynomial-time clocked Turing machines  $N_{i_1}, N_{i_2}, N_{i_3}, \dots$  such that  $\{L(N_{i_k}) : k \geq 1\} = \mathbf{C}$ .

We consider only languages over the alphabet  $\Sigma$  (this means that, for example, boolean formulas have to be suitably encoded). The symbol  $TAUT$  denotes the set (of encodings) of all propositional tautologies over a fixed adequate set of connectives. Finally,  $\langle ., \dots, . \rangle$  denotes some standard polynomial-time computable tupling function.

### 3 Propositional proof systems

The concept of an abstract propositional proof system, subsuming all propositional proof systems used in practice, was introduced by S. Cook and R. Reckhow [5] in the following way:

**Definition 1.** *A proof system for  $TAUT$  is a polynomial-time computable function  $f : \Sigma^* \xrightarrow{\text{onto}} TAUT$ .*

A string  $w$  such that  $f(w) = \alpha$  we call an  $f$ -proof of a formula  $\alpha$ . We write  $f \vdash^* \alpha_n$  if and only if  $\{\alpha_n : n \geq 1\}$  is a sequence of tautologies with polynomial-size  $f$ -proofs. A polynomially bounded proof system for  $TAUT$  (which allows short proofs to all tautologies) exists if and only if  $\mathbf{NP}=\mathbf{co-NP}$  (see [5]).

Proof systems are compared according to their strength using the notion of simulation and the presumably stronger notion of p-simulation.

**Definition 2.** *(Krajíček, Pudlák) Let  $h, h'$  be two proof systems for  $TAUT$ . We say that  $h$  simulates  $h'$  if there exists a polynomial  $p$  such that for any  $x \in TAUT$ , if  $x$  has a proof of length  $n$  in  $h'$ , then  $x$  has a proof of length  $\leq p(n)$  in  $h$ .*

**Definition 3.** (Cook, Reckhow) Let  $h, h'$  be two proof systems for TAUT. We say that  $h$  p-simulates  $h'$  if there exists a polynomial-time computable function  $\gamma : \Sigma^* \rightarrow \Sigma^*$  such that for every  $x \in \text{TAUT}$  and every  $w \in \Sigma^*$ , if  $w$  is a proof of  $x$  in  $h'$ , then  $\gamma(w)$  is a proof of  $x$  in  $h$ .

In other words,  $\gamma$  translates  $h'$ -proofs into  $h$ -proofs of the same formula.

The notions of an optimal proof system for TAUT and a p-optimal proof system for TAUT were introduced by J. Krajíček and P. Pudlák [10].

**Definition 4.** A proof system for TAUT is optimal (p-optimal) if and only if it simulates (p-simulates) any proof system for TAUT.

We will study the problem of the existence of an optimal proof system and the problem of the existence of a p-optimal proof system from computational-complexity perspective.

#### 4 Promise classes representable in a proof system

A nondeterministic polynomial-time clocked Turing machine which is the computational model of a given promise (semantic) class should obey the special condition, called the promise of the class. It can be illustrated by an example of the class **UP**. We call a nondeterministic Turing machine categorical or unambiguous if it has the following property: for any input  $x$  there is at most one accepting computation. We define  $\mathbf{UP} = \{L(N_i) : N_i \text{ is categorical}\}$ .

Let  $T$  be any formal theory whose language contains the language of arithmetic. We say that  $T$  is "reasonable" if and only if  $T$  is sound (that is, in  $T$  we can prove only true theorems) and the set of all theorems of  $T$  is recursively enumerable. Let  $N$  be any NPTM. The notation  $T \vdash "N \text{ is categorical}"$  means that the first order formula expressing the categoricity of  $N$  is provable in  $T$ . We say that **UP** is representable in  $T$  if and only if for any  $A \in \mathbf{UP}$  there exists an NPTM  $N$  such that  $T \vdash "N \text{ is categorical}"$ . J. Hartmanis and L. Hemachandra [8] proved that **UP** has a complete language if and only if it has a uniform enumeration (see also [4]). It follows from Naming Lemma [7] that, the existence of a uniform enumeration of **UP** is equivalent to the existence of a "reasonable" theory  $T$  such that **UP** is representable in  $T$  (see also [8]). Therefore, the problem of the existence of a complete language for **UP** can be characterized in terms of a uniform representability of **UP** in a first order arithmetic theory  $T$ .

In this section we show that this problem can be also characterized in terms of a nonuniform representability of **UP** in a propositional proof system. In this case the promise of the class is expressed as the sequence of propositional tautologies with short proofs. We begin with the introduction of the necessary machinery.

Following J. Messner's approach [11], we define promise classes in a very general way. A promise  $R$  is described as a binary predicate on the Cartesian product of the set of all NPTMs and the set of all strings, i. e.,  $R(N, x)$  means that  $N$  obeys a promise  $R$  on input  $x$ . An NPTM  $N$  is called an  $R$ -machine if and only if  $N$  obeys  $R$  on any input  $x \in \Sigma^*$ . For a given promise predicate  $R$

we define the class of languages  $C_R = \{L(N) : N \text{ is an } R\text{-machine}\}$  and call it the promise class generated by  $R$ .

**Definition 5.** (Messner) A class of languages  $C$  is called a promise class if and only if  $C = C_R$  for some promise predicate  $R$ .

The following notion of the test set for a promise class  $C_R$  serves as a tool for estimating the complexity of the promise  $R$  of this class.

**Definition 6.** By the test set for a promise class  $C_R$  we mean the set  $T_R = \{\langle N, 0^n, 0^{p_N(n)} \rangle : n \text{ is a natural number, } N \text{ is an NPTM such that } R(N, x) \text{ holds for any } x \text{ such that } |x| = n\}$

The notion of the test set for  $C_R$  corresponds to the notion of the generic and length-only dependent test set from [9].

We are especially interested in the situation when the fact that a given NPTM is an  $R$ -machine can be expressed propositionally, as a sequence of propositional tautologies. It can be done only when the promise  $R$  has an appropriate complexity.

To any NPTM  $N$  we will assign the set  $D_R^N = \{\alpha_1^N, \alpha_2^N, \alpha_3^N, \dots\}$  of propositional formulas such that  $\alpha_n^N$  is a propositional tautology if and only if  $R(N, x)$  holds for any  $x$  such that  $|x| = n$ . So, for any NPTM  $N$  it holds:  $N$  is an  $R$ -machine if and only if  $D_R^N \subset \text{TAUT}$ .

It should be possible to construct the formulas  $\alpha_n^N$  for different sets  $D_R^N$ , corresponding to different NPTMs, easily and in an uniform manner. This leads to the following definitions:

**Definition 7.** By a propositional description of a promise  $R$  we mean a set  $D_R = \{\alpha_n^N : N \text{ is an NPTM, } n \text{ is a natural number}\}$  of propositional formulas fulfilling conditions (1) – (3):

(1) Adequacy:

$\alpha_n^N$  is a propositional tautology if and only if  $R(N, x)$  holds for any  $x$  such that  $|x| = n$ .

(2) Uniform constructibility:

There exists a polynomial time computable function  $f$  such that for any NPTM  $N$  and for any  $n$  natural

$$f(\langle N, 0^n, 0^{p_N(n)} \rangle) = \alpha_n^N$$

(3) Local recognizability:

For any fixed NPTM  $N$ , the set  $D_R^N = \{\alpha_1^N, \alpha_2^N, \alpha_3^N, \dots\}$  is in  $\mathbf{P}$ .

**Definition 8.** We say that a promise  $R$  is propositionally expressible if and only if there exists a propositional description of  $R$ .

The next lemma will be needed in Section 5 in the proof of the main result of the paper.

**Lemma 1.** *Any promise  $R$  such that the class  $C_R$  possesses the test set  $T_R$  in  $\text{co-NP}$  is propositionally expressible.*

Let  $R$  be a propositionally expressible promise and let  $D_R = \{\alpha_n^N : N \text{ is an NPTM, } n \text{ is a natural number}\}$  be its propositional description. Let  $h$  be a proof system for  $\text{TAUT}$ .

**Definition 9.** *A language  $A$  is weakly  $D_R$ -representable in  $h$  if and only if there exists an NPTM  $K$  such that conditions (1) – (2) are fulfilled:*

- (1)  $L(K) = A$
- (2)  $h \vdash^* \alpha_n^K$

**Definition 10.** *A language  $A$  is strongly  $D_R$ -representable in  $h$  if and only if there exists an NPTM  $K$  such that conditions (1) – (2) are fulfilled:*

- (1)  $L(K) = A$
- (2) *There exists a polynomial time algorithm that on input  $0^n$  produces an  $h$ -proof of  $\alpha_n^K$ , for any  $n$  natural.*

Finally, we have the following definitions:

**Definition 11.** *A promise class  $C_R$  is representable in  $h$  if and only if there exists a propositional description  $D_R$  of  $R$  such that any language  $A \in C_R$  is weakly  $D_R$ -representable in  $h$ .*

**Definition 12.** *A promise class  $C_R$  is p-representable in  $h$  if and only if there exists a propositional description  $D_R$  of  $R$  such that any language  $A \in C_R$  is strongly  $D_R$ -representable in  $h$ .*

Now we present the above mentioned characterization of the problem of the existence of a complete language for **UP**.

**Theorem 1.** *There exists a complete language for **UP** if and only if there exists a propositional proof system  $h$  such that **UP** is p-representable in  $h$ .*

*Proof. (Sketch)* (i)  $\rightarrow$  (ii) The existence of the desired propositional proof system  $h$  follows from the existence of a uniform enumeration of the class **UP**.

(ii)  $\rightarrow$  (i) Assume that there exists a propositional proof system  $h$  such that **UP** is p-representable in it. There exists a propositional description  $D_R = \{\alpha_n^N : N \text{ is an NPTM, } n \text{ is a natural number}\}$  of the promise of **UP** such that any language  $A \in \text{UP}$  is strongly  $D_R$ -representable in  $h$ .

The following language  $L$  is the desired **UP** complete language.

$$L = \{\langle N, x, \alpha_{|x|}^N, \text{Proof}, 0^{p_N(|x|)} \rangle : x \in L(N)\}$$

where  $N$  is an NPTM,  $p_N$  is the polynomial that bounds the running time of  $N$ ,  $x$  is a string,  $\alpha_{|x|}^N$  is a propositional formula from the propositional description of the promise of **UP**,  $\text{Proof}$  is an  $h$ -proof of  $\alpha_{|x|}^N$ ,  $0^{p_N(|x|)}$  is the sequence of zeros (padding).

## 5 Main results

J. Messner [11] considered the family of all proof systems for  $TAUT$  as a promise function class. He proved that a p-optimal proof system exists if and only if any promise function class with a test set polynomial-time reducible to  $TAUT$  possesses a complete function. Our intention was to find an analogous theorem in the setting of promise language classes. In our results from this section we characterize the existence of optimal proof systems in terms of a nonuniform presentability of promise classes in a proof system. For most promise classes, having a complete language and a uniform enumeration (a uniform representation in an arithmetic theory) are equivalent. Similarly, it seems that for promise classes a nonuniform p-presentability in a proof system is close to the possession of a complete language.

In our characterization of the problem of the existence of optimal and p-optimal proof systems the families of all easy and all  $\mathbf{NP}$ -easy subsets of  $TAUT$  play a very important role.

**Definition 13.** *By an easy subset of  $TAUT$  we mean a set  $A$  such that  $A \subset TAUT$  and  $A \in \mathbf{P}$ .*

**Definition 14.** *By an  $\mathbf{NP}$ -easy subset of  $TAUT$  we mean a set  $A$  such that  $A \subset TAUT$  and  $A \in \mathbf{NP}$ .*

The next lemma shows that for every easy subset of  $TAUT$  there exists a proof system in which tautologies from this set have short and easily constructible proofs.

**Lemma 2.** *(Messner, Torán [12]) If  $A$  is an easy subset of  $TAUT$  then there exists a proof system  $f : \Sigma^* \xrightarrow{\text{onto}} TAUT$  and a polynomial-time computable function  $t$  that on input  $\alpha$  produces  $f$ -proof of  $\alpha$ , for any tautology  $\alpha$  in  $A$ . That is for every  $\alpha \in A$ ,  $f(t(\alpha)) = \alpha$ .*

Let us proceed to the main results of the paper.

**Theorem 2.** *Statements (i) - (iii) are equivalent:*

- (i) *There exists an optimal propositional proof system.*
- (ii) *There exists a propositional proof system in which any promise class with the test set in co- $\mathbf{NP}$  is representable.*
- (iii) *There exists a propositional prof system in which the class of all  $\mathbf{NP}$ -easy subsets of  $TAUT$  is representable.*

The previous result can be translated to the deterministic case in the following way:

**Theorem 3.** *Statements (i) - (iii) are equivalent:*

- (i) *There exists a p-optimal propositional proof system.*
- (ii) *There exists a propositional proof system in which any promise class with the test set in co- $\mathbf{NP}$  is p-representable.*

(iii) *There exists a propositional prof system in which the class of all easy subsets of TAUT is p-representable.*

We proved [15] that there exists a p-optimal proof system for TAUT if and only if the class of all easy subsets of TAUT is uniformly enumerable. It can be otherwise stated thus: there exists a p-optimal proof system for TAUT if and only if there exists a "reasonable" arithmetic theory  $T$  such that the class of all easy subsets of TAUT is representable in it (see [7]). In this paper we replaced representability in an arithmetic theory by representability in a proof system, in the characterization of the existence of a p-optimal proof system in terms of easy subsets of TAUT. It is typical for proof complexity that an arithmetic theory coincides with a proof system for TAUT and the latter is a nonuniform version of the former.

## References

1. J. Balcazar, J. Díaz and J. Gabarró, Structural Complexity I (Springer-Verlag, Berlin, 1995).
2. O. Beyersdorff, Tuples of disjoint NP-sets, Technical Report TR 05-123, Electronic Colloquium on Computational Complexity, 2005.
3. O. Beyersdorff, Disjoint NP-Pairs and Propositional Proof systems, PhD Thesis, Humboldt-Universität zu Berlin, July 2006.
4. H. Buhrman, S. Fenner, L. Fortnow and D. van Melkebeek, Optimal proof systems and sparse sets, Proc. Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science 1770, Springer - Verlag, Berlin, 2000.
5. S. Cook and R. Reckhow, The relative efficiency of propositional proof systems, Journal of Symbolic Logic 44 (1979) 36-50.
6. C. Glasser, A. L. Selman, S. Sengupta and L. Zhang, Disjoint NP-pairs, SIAM Journal on Computing, 33(6), (2004) 1369 –1416.
7. J. Hartmanis, Independence Results about Context-Free Languages and Lower Bounds, Technical Report, TR 84-606, Department of Computer Science Cornell University, May 1984.
8. J. Hartmanis and L. Hemachandra, Complexity classes without machines: On complete languages for UP, Theoretical Computer Science 58(1988) 129-142.
9. J. Köbler, J. Messner and J. Torán, Optimal proof systems imply complete sets for promise classes, Information and Computation 184 (2003) 71 – 92.
10. J. Krajíček and P. Pudlák, Propositional proof systems, the consistency of first order theories and the complexity of computations, Journal of Symbolic Logic 54 (1989) 1063-1079.
11. J. Messner, On the simulation order of proof systems, PhD Thesis, Universität Ulm, December 2000.
12. J. Messner and J. Torán, Optimal proof systems for Propositional Logic and complete sets, in: Proc. 15th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science 1373 (Springer,Berlin,1998) 477-487.
13. C. Papadimitriou, Computational Complexity, (Addison - Wesley, 1994).
14. A. Razborov, On provably disjoint NP-pairs, Technical Report 94-006, Electronic Colloquium on Computational Complexity, 1994.
15. Z. Sadowski, On an optimal propositional proof system and the structure of easy subsets of TAUT, Theoretical Computer Science, 288 (1), 2002, 181 – 193.

# On the Complexity of Computing Winning Strategies for Finite Poset Games

Michael Soltys and Craig Wilson

Department of Computing and Software  
McMaster University  
1280 Main Street West, Hamilton, Ontario, L8S 4K1, Canada  
[soltys@mcmaster.ca](mailto:soltys@mcmaster.ca)

**Abstract.** We study the complexity of computing winning strategies for poset games; these are two player games on finite partially ordered sets. Given a poset game, we consider the corresponding language of those “board configurations” from which the first player has a winning strategy. While it is reasonably clear that such a language is in PSPACE (as the answer can be determined by evaluating a quantified boolean formula), we give a simple and direct proof of this fact by reducing general poset games to the game of geography. We also show how to reason about poset games in Skelley’s theory  $\mathbf{W}_1^1$  for PSPACE reasoning—we show that this theory can formalize the “strategy stealing argument” and show that the first player has a winning strategy in the poset game Chomp. More than anything, this paper is an invitation to consider the problem whether poset games are PSPACE complete, and/or show under what conditions computing a winning strategy can be done efficiently.

**Key words:** Finite games, posets, PSPACE, proof complexity.

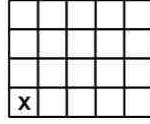
## 1 Introduction

A partially ordered set (a *poset*) is a set  $U$  together with an ordering relation  $\preceq$  on its elements, where  $\preceq$  is a subset of  $U \times U$ . The relation  $\preceq$  must satisfy the following conditions: (1) if  $a \preceq b$ , then  $b \not\preceq a$  (*anti-symmetry*), and (2) if  $a \preceq b$  and  $b \preceq c$ , then  $a \preceq c$  (*transitivity*). Not all elements are necessarily comparable, in that there may be elements  $a, b$  such that  $a \neq b$ , where  $a \not\preceq b$  and  $b \not\preceq a$ . When two elements are incomparable, we write  $a \parallel b$ .

Given a poset  $(U, \preceq)$ , a *poset game*  $(A, \preceq)$  on  $(U, \preceq)$  is played as follows: at first  $A := U$ . Then, two players take turns making moves. On each move, a player picks an element  $x \in A$ , and removes all the elements  $y \in A$  such that  $x \preceq y$ . The player who is unable to move because  $A = \emptyset$  loses.

An example of a poset game is Chomp, first detailed by Gale in [1]. A game of Chomp is played on a “chocolate bar” divided into individual squares, with the bottom-left square being “poisoned”. This is usually represented by a grid of  $m$  rows and  $n$  columns, with the poisoned square residing at position  $(1, 1)$  (see Fig. 1). Two players take turns breaking off pieces of the chocolate by selecting a

square  $(i, j)$  from among the remaining squares and deleting all the squares  $(k, l)$  such that  $i \leq k$  and  $j \leq l$ . The game ends when a player selects the poisoned square. The player who does so loses. It is easy to show, by a “strategy-stealing argument”, that player 1 always wins ([1]). To transform Chomp to a poset



**Fig. 1.** An example  $4 \times 5$  Chomp grid

game we delete the lower-left square, so now the player who ends up without any chocolate to chomp loses. Let  $\text{Bar}_{m,n} := \{(i, j) | 1 \leq i \leq m, 1 \leq j \leq n\} - \{(1, 1)\}$ , and  $(i, j) \preceq (k, l)$  if and only if  $(i \leq k \wedge j < l) \vee (i < k \wedge j \leq l)$ . So, for any  $m, n$  we have the poset game  $(\text{Bar}_{m,n}, \preceq)$ .

Recall that PSPACE is the class of languages decidable in polynomial space in the length of the input on a Turing machine. A language  $L$  is PSPACE complete if it is in PSPACE, and for every language  $L'$  in PSPACE, there exists a polynomial time function  $f : \Sigma^* \rightarrow \Sigma^*$ , such that  $x \in L' \iff f(x) \in L$ . See [2] or [3] for more background on PSPACE.

## 2 From Posets to Geography

In this section we present a polynomial time reduction from poset games to Geography. In order to study the complexity of the reduction, we assume that the posets are finite (i.e.,  $U$  is finite). However, the reduction itself still works on infinite posets, and yields infinite instances of the game of Geography.

The game of *Geography* is played as follows: a directed graph is given (it does not have to be acyclic), and a starting node  $s$  is specified. The first player selects  $s$ , then the second player selects an outgoing edge of  $s$ , and takes that edge to another node. The two players then traverse the graph, alternatively selecting outgoing edges of the current node. The player who is forced to revisit a node, or has no outgoing edges to select from, loses. It is a well known fact that Geography is PSPACE-complete (see [2] or [3]).

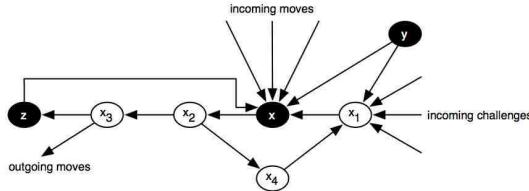
Let  $\text{POSETGAMES} = \{\langle (U, \preceq) \rangle : \text{player 1 has a winning strategy}\}$  and let  $\text{GEOGRAPHY} = \{\langle (G, s) \rangle : \text{player 1 has a winning strategy}\}$ . Here  $\langle (U, \preceq) \rangle$  and  $\langle (G, s) \rangle$  are the encodings of  $(U, \preceq)$  and  $(G, s)$ , respectively. Note that  $s$  is the specified starting node for the game of Geography on the graph  $G$ , so the game starts by player 1 selecting node  $s$ , then player 2 selects an edge out of  $s$  to a new node  $n$ . This is in contrast to poset games, where any element  $x \in U$  can be selected on the first turn.

In the following paragraphs we will show how to transform  $(U, \preceq)$  into  $(G = (V, E), s)$ . More precisely, in what follows we describe a function  $f$  such that  $\langle\langle U, \preceq \rangle\rangle \xrightarrow{f} \langle\langle G = (V, E), s \rangle\rangle$ , where  $\preceq$  is described as a list of pairs. Note that such a list has length at most  $O(|U|^2 \log(|U|))$ , and  $f$  can be computed in logarithmic space in  $|\langle\langle U, \preceq \rangle\rangle|$ .

For every element  $x$  of  $U$  we have a node in  $V$ , and for every pair  $x, y \in U$  such that  $x \preceq y$ , we put the directed edge  $(y, x)$  in  $E$ . For every pair  $x, z \in U$  where  $x||z$ , we put  $(x, z)$  and  $(z, x)$  in  $E$ . This is not enough to simulate the poset game on  $G$ , however, because we have to keep track of all the elements of  $U$  that have been eliminated, that is, we need to keep track of  $A$ .

We are going to accomplish keeping track of  $A$  as follows: every time player 2 makes a move to a comparable node, player 1 can challenge that move. What is this challenge? It is player 1's claim that player 2 moved to a node  $y$  such that  $x \preceq y$  and node  $x$  has already been visited. As well, we must ensure that challenging a legal move, or even one's own move yields no benefit to the challenger.

To be able to do these challenges, we are going to add four different auxiliary nodes  $\{x_1, x_2, x_3, x_4\}$  to each node  $x \in V$ , as represented in Fig. 2. The edges are placed as follows: for each outgoing edge of  $x$  to a node  $z$  (comparable or incomparable to  $x$ ), we have the edge  $(x_3, z)$ . For each incoming edge from a comparable node  $y$  such that  $x \preceq y$ , we add the edge  $(y, x_1)$ . Finally, we have the edges  $(x_1, x), (x_2, x), (x_2, x_3), (x_2, x_4)$ , and  $(x_4, x_1)$ .



**Fig. 2.** Reduction gadget: each  $x \in U$  is represented by five nodes in  $G$ .

We repeat this process for every  $x \in U$ . Finally, we add two more nodes,  $s, s'$ , with an edge from  $s$  to  $s'$ , and edges from  $s'$  to every other non-auxiliary node. These nodes are required because in a poset game  $(U, \preceq)$  we can select any  $x \in U$  to start the game, whereas in Geography the first player must select node  $s$  on their first turn. To emulate the free choice of the poset game, we have  $s$  as the initial node, which means player 2 must move to  $s'$ , then player 1 can select any non-auxiliary node in  $V$ . It is clear that this simple construction can be carried out in polynomial time (in fact, logarithmic space).

It is obvious that the Geography game on  $G$  mirrors a poset game on  $(U, \preceq)$  if it is played correctly, in the sense that we never go to a node that corresponds to an element in  $U$  that was removed at a previous step. As mentioned previously,

the auxiliary nodes enable players to challenge each other's moves. We examine below what occurs during these challenges.

**Legitimate Challenge:** Here player 2 moves to a node  $y$  where  $x \preceq y$  and node  $x$  was visited in the past. This means that player 2 moves to a  $y \notin A$ . Then player 1 challenges player 2 as follows: he moves from  $y$  to the challenging node  $x_1$  of  $x$ . Player 2 is now forced to revisit  $x$ , and loses. This ensures that players can only move to legal nodes.

**False Challenge:** What if a player challenges a legal move? That is, if player 2 moved to node  $y$  which was in  $A$ ? Then the following happens: player 1 challenges player 2 by moving to node  $x_1$ , and now player 2 moves to  $x$  and nothing happens as  $x$  has not been visited before. Then player 1 is forced to move to  $x_2$ , and then player 2 moves to  $x_4$ , and player 1 is now forced to move to  $x_1$ , and thereby revisit a node, and player 1 loses. This not only shows that challenges to legal nodes yield no benefit, but also challenges to incomparable nodes.

**Self-Challenge:** The final type of challenge occurs when a player challenges their own move. Suppose player 2 arrives at node  $x$  without a challenge, i.e., directly through one of the incoming edges, then player 1 moves to  $x_2$ . Now player 2 has a choice to move to  $x_4$ , as if player 2 were going to do a counter-challenge. If indeed player 2 moves to  $x_4$ , then player 1 moves to  $x_1$ , and player 2 is forced to revisit node  $x$ , and player 2 loses. Thus, no gains are made if a player challenges their own move.

Therefore, if player 2 arrives at node  $x$  directly (without a challenge) he must continue (after the move of player 1) from  $x_2$  to  $x_3$ , and then player 1 is free to select an outgoing edge from  $x_3$  to another node  $z$ . The above shows that there is a (correct) polynomial time reduction from POSETGAMES to GEOGRAPHY. As GEOGRAPHY itself is in PSPACE, this gives us that POSETGAMES  $\in$  PSPACE as well. Of course, this was expected as the existence of a winning strategy for a given “board configuration” can be expressed as the satisfiability of a quantified Boolean formula. On the other hand, we have given a simple and direct proof by reducing to GEOGRAPHY. The interesting (open) question is whether POSETGAMES is PSPACE-complete, and to characterize the poset games for which a winning strategy can be computed efficiently<sup>1</sup>.

### 3 $\mathbf{W}_1^1$ proves the existence of a strategy for Chomp

In this section we show how to extract a PSPACE algorithm for computing a winning strategy for Chomp from a  $\mathbf{W}_1^1$ -proof<sup>2</sup> of the existence of a winning

---

<sup>1</sup> For example, consider a chomp configuration in the shape of an “L”. That is, we have the first column and last row, with their intersection containing the poisoned square. Then as long as the two arms of the “L” have a different length, the first player has the following (simple) winning strategy: the first player chomps the longer arm to be the same length as the shorter, and then copies symmetrically the moves of the second player on the opposite arm, to force the second player to be left with the poisoned square.

<sup>2</sup> The system  $\mathbf{W}_1^1$  was introduced in [4].

strategy<sup>3</sup>. We hope that a program consisting in formalizing proofs of existence of winning strategies in weaker and weaker fragments of Bounded Arithmetic can yield better algorithms (i.e., algorithms of lower complexity) for computing winning strategies for poset games.

$\mathbf{W}_1^1$  is a logical theory that captures PSPACE reasoning. It extends  $\mathbf{V}^1$  (which captures polynomial time, the class P) presented by Cook and Nguyen in [5], but  $\mathbf{W}_1^1$  is designed to reason over sets of strings, which themselves encode sets of sets of elements.  $\mathbf{W}_1^1$  is a three-sorted (“third-order”) predicate calculus with free and bound variables of three sorts (the bound variables are given in parenthesis):  $a, b, c, \dots (x, y, z, \dots)$  for the first sort, intended to denote natural numbers encoded in unary,  $A, B, C, \dots (X, Y, Z, \dots)$  for the second sort, intended to denote finite binary strings, and  $\mathbb{A}, \mathbb{B}, \mathbb{C}, \dots (\mathbb{X}, \mathbb{Y}, \mathbb{Z}, \dots)$  for the third sort, intended to denote sets of strings, and in particular functions from strings to strings.

The third order language we use is  $\mathcal{L}_A^3 = [0, 1, +, \cdot, | \cdot |_2, \in_2, \in_3, \leq, =]$  where 0, 1 are numbers,  $+, \cdot$  is addition and multiplication of numbers,  $| \cdot |_2$  denotes length of strings (note that size of third sort objects is not there),  $i \in_2 X$  is equivalent to  $X(i)$  (i.e., it is true if the  $i$ -th bit is turned on),  $X \in_3 \mathbb{X}$  denotes the string  $X$  in  $\mathbb{X}$  (we may omit the subscript 2 or 3 if it is clear from the context which type it is), and  $\leq, =$  are used to compare numbers (there is no equality symbol for the second and third sort; equality for these objects can be defined in  $\mathcal{L}_A^3$ ). We are interested in sets of pairs defining functions, and so we use the notation  $\mathbb{X}(X) = Y$ , which formally denotes  $\langle X, Y \rangle \in \mathbb{X}$ . Our functions are going to be finite, thus they shall be defined for  $X, Y$  such that  $|X|, |Y| \leq n$  for some number term  $n$ . Of course, the intention is that we shall have a “strategy function”  $S(C_1) = C_2$  which on configuration  $C_1$  produces a configuration  $C_2$ .

**Table 1.** The set **2-BASIC**

<b>B1.</b> $x + 1 \neq 0$	<b>B7.</b> $(x \leq y \wedge y \leq x) \rightarrow x = y$
<b>B2.</b> $(x + 1 = y + 1) \rightarrow x = y$	<b>B8.</b> $x \text{ lex } + y$
<b>B3.</b> $x + 0 = x$	<b>B9.</b> $0 \leq x$
<b>B4.</b> $x + (y + 1) = (x + y) + 1$	<b>B10.</b> $x \leq y \vee y \leq x$
<b>B5.</b> $x \times 0 = 0$	<b>B11.</b> $x \leq y \leftrightarrow x < y + 1$
<b>B6.</b> $x \times (y + 1) = (x \times y) + x$	<b>B12.</b> $x \neq 0 \rightarrow \exists y \leq x (y + 1 = x)$
<b>L1.</b> $X(y) \rightarrow y <  X $	<b>L2.</b> $y + 1 =  X  \rightarrow X(y)$
<b>SE.</b> $[ X  =  Y  \wedge \forall i <  X  (X(i) \leftrightarrow Y(i))] \rightarrow X = Y$	

The theory  $\mathbf{W}_1^1$  consists of axiom B1–B12 and L1, L2 (listed in table 1, from [5]) as well as two comprehension axioms (or rather axiom schemes) given below. We let  $\Sigma_i^{\mathbb{P}}$  be the set of formulas over  $\mathcal{L}_A^3$  containing formulas with arbitrarily

<sup>3</sup> In fact, our method works for any poset game to which we can apply the “strategy steering argument,” namely to any poset game with a supremum.

many bounded first and second-order quantifiers, and exactly  $i$  alternations of third-order quantifiers. The purpose of the comprehension axioms is to allow the definition of new strings and functions (from strings to strings):

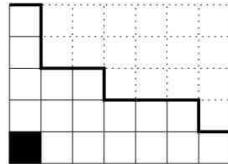
$$\begin{aligned} (\exists Y \leq t(\bar{x}, \bar{X})) (\forall z \leq s(\bar{x}, \bar{X})) [\phi(\bar{x}, \bar{X}, \bar{\bar{X}}, z) \leftrightarrow Y(z)] & \quad \Sigma_0^{\mathbb{B}}\text{-2COMP} \\ (\exists \mathbb{Y}) (\forall Z \leq s(\bar{x}, \bar{X})) [\phi(\bar{x}, \bar{X}, \bar{\bar{X}}, Z) \leftrightarrow \mathbb{Y}(Z)] & \quad \Sigma_0^{\mathbb{B}}\text{-3COMP} \end{aligned}$$

In each of these schemes  $\phi \in \Sigma_0^{\mathbb{B}}$  is subject to the restriction that neither  $Y$  nor  $\mathbb{Y}$  (as appropriate) occurs free in  $\phi$ .  $\mathbb{Y}(Z)$  abbreviates  $Z \in_3 \mathbb{Y}$ , and similarly for  $Y(z)$ .

Following [4, theorem 4] we know that if we can show that  $\mathbf{W}_1^1$  proves the existence of a winning strategy for the first player, where the formula asserting the existence of this strategy is in a proper syntactic form (i.e., it is a  $\Sigma_1^{\mathbb{B}}$ -formula), then we can conclude directly that the strategy can be computed in PSPACE. This is what we do next.

Any configuration of chomp on an  $n \times m$  board can be represented with a binary string  $X \in \{0, 1\}^{n+m}$ . Following a suggestion of [6], a configuration can be represented with a string of 0s and 1s of length  $(n + m)$ , where there are exactly  $n$  1s and  $m$  0s. In this scheme, a board configuration can be seen as a path from the upper-left corner to the lower-right corner (the poisoned square is in the lower-left corner), where we are only allowed to move right or down.

Reading the string left to right, every time we see a 0 we move right, and every time we see a 1 we move down. The original configuration would be 000000011111, and the final configuration would be the same but with all the 1s moved to the left. Whoever moves the game into the final configuration loses. A possible intermediate configuration is given by Fig. 3. Each move is a “chomp”,



**Fig. 3.** A chomp board; it is represented by 011001000101.

which consists of picking a square, from among the surviving squares (i.e., to the left of the thick line in the picture above), and chomping off all the squares to the right and up. This means, that each move of chomp consists in moving one or more 1s one or more positions to the left, without ever overtaking the 1s in front.

All this can be expressed easily in  $\mathbf{W}_1^1$ . We are going to provide a formula  $\Phi(X, n, m)$  which will assert that  $X$  is a valid chomp game on a  $n \times m$  board. Here  $X$  is a long string, consisting of  $n \times m$  many segments (the longest possible

chomp game, where each player takes just one square at a time) of length  $(n+m)$  each. We denote the  $i$ -th segment by  $X^{[i]}$  (all valid notation in the language of  $\mathbf{W}_1^1$ ).  $\Phi$  is the conjunction of three formulas:  $\phi_{\text{init}}$ ,  $\phi_{\text{final}}$ , and  $\phi_{\text{move}}$ .  $\phi_{\text{init}}$  asserts that  $X^{[1]}$  is the initial configuration, i.e.

$$\phi_{\text{init}}(X^{[1]}, n, m) = \forall i \leq (n+m)((i > m) \rightarrow X^{[1]}(i) = 1)$$

$\phi_{\text{final}}$  asserts that  $X^{[n \times m]}$  is the final configuration:

$$\phi_{\text{final}}(X^{[n \times m]}, n, m) = \forall i \leq (n+m)((i > n) \rightarrow X^{[n \times m]}(i) = 0)$$

$\phi_{\text{move}}$  asserts that each segment of  $X$  can be obtained from one legal move on the previous segment:

$$\phi_{\text{move}}(X, n, m) = \forall i < (n \times m)(X^{[i]} \text{ "yields" } X^{[i+1]})$$

In order to define the “yields” portion of  $\phi_{\text{move}}$ , we first take the high-level approach of determining what square was played between two consecutive configurations  $X^{[i]}$  and  $X^{[i+1]}$ , and then express the conditions with  $\mathbf{W}_1^1$  formulas. Note that for the remainder of this paper we will “invert” our coordinate system for Chomp configurations, as now the “origin”  $(1, 1)$  is the top-right square, while  $(m, n)$  is the bottom-right square.

First, we notice that in our string representation of configurations, a 0 occurring to the left of at least a single 1 corresponds to a column of playable squares, the height of which is the number of 1s to the right of the 0. Thus, a square  $(j, k)$  can be played on  $X^{[i]}$  if and only if in reading  $X^{[i]}$  from left to right, the  $k^{\text{th}}$  0 is encountered before the  $j^{\text{th}}$  1. To express this formally, we introduce two functions  $F_0$  and  $F_1$ .  $F_0$  is defined as

$$F_0(a, b, X) = c \leftrightarrow \text{numones}(1, X(a : c)) = b$$

or, the position in the string  $X$  where, starting from (and including) position  $a$ , there are  $b$  0s.  $F_1$  is defined analogously for  $b$  1s:

$$F_1(a, b, X) = c \leftrightarrow |X(a : c)| - \text{numones}(1, X(a : c)) = b$$

Note that we define  $F_0(a, 0, X)$  and  $F_1(a, 0, X)$  to be 0 for any  $a, X$ . Also, if we reach the end of  $X$  before encountering  $b$  0s or 1s, we define  $F_0(a, b, X)$  and  $F_1(a, b, X)$  to be  $|X|$ . We now restate the conditions for the existence of playable square as a formula using these functions. This formula is the first of many sub-formulas which will comprise the final “yields” formula. Each of these sub-formulas will be marked with a label  $\psi_i$ .

**Lemma 1** *A square  $(j, k)$  can be played on a configuration  $X$  if and only if*

$$F_0(1, k, X) < F_1(1, j, X) \tag{\psi_1}$$

*Proof.* We will argue by contradiction. Suppose square  $(j, k)$  exists on configuration  $X$ , but  $F_0(1, k, X) > F_1(1, j, X)$  - note that we cannot have  $F_0(1, k, X) = F_1(1, j, X)$  as a position in  $X^{[i]}$  can't simultaneously be 0 and 1. This means that the  $k^{\text{th}}$  0 occurs after that  $j^{\text{th}}$  1, which in turn means that row  $j$  terminated before column  $k$ . Thus, there are no squares beyond column  $k - 1$  in row  $j$ , so square  $(i, j)$  does not exist in  $X$ , and cannot be played.  $\square$

Now that we can identify playable squares, we need to determine the configurations which result from playing these squares. In general, playing a square  $(j, k)$  shifts at least a single 1 behind the 0 corresponding to the move, to delimit the row being shortened or eliminated. If a move affects multiple rows, then multiple 1s will be shifted over. In order to figure out the substring of  $X$  which is affected by playing a square  $(j, k)$ , we calculate two values  $p$  and  $q$  which mark the beginning and ending of the substring (inclusive):

$$p = F_0(1, k - 1, X^{[i]}) + 1 \quad (\psi_2)$$

$$q = F_1(p, j, X^{[i]}) \quad (\psi_3)$$

Intuitively,  $(p - 1)$  marks the location of the  $(k - 1)^{\text{th}}$  0, so  $p$  forms the left boundary of the substring.  $q$  is the position of  $X^{[i]}$  in which we have seen  $j$  1s starting from  $p$ , so it designates the right boundary. Finally, to arrive at the new configuration  $X^{[i+1]}$  we replace  $X^{[i]}(p : q)$  with  $1^j 0^{(q-p-k+1)}$ , i.e., the number of 1s corresponding to the number of rows affected, then the number of 0s indicating the new difference in row lengths. In order to express this new substring as a formula we break it down into several sub-formulas. First, positions that are outside the range of  $p$  and  $q$  remain unchanged:

$$(r < p) \rightarrow (X^{[i+1]}(r) = X^{[i]}(r)) \quad (\psi_4)$$

$$(r > q) \rightarrow (X^{[i+1]}(r) = X^{[i]}(r)) \quad (\psi_5)$$

Next, for positions between  $p$  and  $q$  we assign values based on their placement relative to the 0 associated with the move. Positions before the 0 hold 1s affected by the move, while positions after contain 0s.

$$(r < p + j) \rightarrow (X^{[i+1]}(r) = 1) \quad (\psi_6)$$

$$(r \geq p + j) \rightarrow (X^{[i+1]}(r) = 0) \quad (\psi_7)$$

$$(p \leq r \leq q) \rightarrow (\psi_6 \wedge \psi_7) \quad (\psi_8)$$

We now put formulas  $\psi_4$ ,  $\psi_5$  and  $\psi_8$  together to create one formula describing the changes between configurations  $X^{[i]}$  and  $X^{[i+1]}$ :

$$(\forall r \leq |X|)(\psi_4 \wedge \psi_5 \wedge \psi_8) \quad (\psi_9)$$

Finally, we combine formulas  $\psi_2$  and  $\psi_3$  with existential quantification to describe the existence of valid  $p$  and  $q$ :

$$(\exists p < |X^{[i]}|)(\exists q \leq |X^{[i]}|)(\psi_2 \wedge \psi_3) \quad (\psi_{10})$$

Thus, if we take the formula  $\psi_9$  for describing legal changes between configurations and combine it with formula  $\psi_1$  for the existence of a playable square, as well as  $\psi_{10}$  for the existence of values for  $p$  and  $q$ , we have the following formula for “yields”:

$$(\exists j \leq \text{NumOnes})(\exists k \leq \text{NumZeros}) [\psi_1 \wedge \psi_{10} \wedge \psi_9]$$

where

$$\begin{aligned} \text{NumOnes} &= |X^{[i]}| - \text{numones}(1, X^{[i]}) \\ \text{NumZeroes} &= \text{numones}(1, X^{[i]}) \end{aligned}$$

Having completed the formula  $\Phi(X, n, m)$  we can now restate the existence of a winning strategy for the first player in  $\mathbf{W}_1^1$ .

A winning strategy in chomp is just a function  $\mathbb{S} : X \rightarrow Y$ , which maps configurations to configurations, so that whoever plays by  $\mathbb{S}$ , wins. It is well known that the first player has a winning strategy. How can this be stated? As follows:  $\forall Y$ , such that  $Y$  has  $nm/2$  many segments of length  $(n+m)$  each, there exists an  $X$  as above, such that every other segment of  $X$  is  $Y$ , and  $X^{[i+1]} = \mathbb{S}(Y^{[i]})$ , and  $\Phi(X, n, m)$ , and also the first time the final configuration occurs, it is in an even segment (so second player loses).

We want to say that given any configuration  $C$  as an initial configuration, either player 1 or player 2 has a winning strategy. This can be stated as follows:

$$\forall C \exists \mathbb{S} [\text{Win}_{P_1}(\mathbb{S}, C) \vee \text{Win}_{P_2}(\mathbb{S}, C)], \quad (1)$$

where  $\text{Win}_{P_i}(\mathbb{S}, C)$  ( $i \in \{0, 1\}$ ), and  $\bar{i}$  will denote the value in the set  $\{0, 1\} - \{i\}$ ) asserts that if player  $i$  plays by strategy  $\mathbb{S}$ , then player  $i$  wins. This can be easily state as follows:  $\forall Y$  (where  $Y$  is a sequence of moves of player  $\bar{i}$ ), if it is player  $i$ 's move on configuration  $C$ , and player  $i$  plays  $C' = \mathbb{S}(C)$ , then player  $\bar{i}$  will end up with the poisoned square. Note that (1) is a  $\Sigma_1^{\mathbb{B}}$  formula since the “ $\forall C$ ” is bounded by the size of the chomp grid—we omit the bound for clarity, and  $\text{Win}_{P_i}$  is a  $\Sigma_0^{\mathbb{B}}$  formula.

**Theorem 1.**  $\mathbf{W}_1^1 \vdash \forall C \exists \mathbb{S} [\text{Win}_{P_1}(\mathbb{S}, C) \vee \text{Win}_{P_2}(\mathbb{S}, C)]$ .

*Proof.* We prove it by induction on  $|C|$ , where we let  $|C|$  is the number of squares in configuration  $C$  (in particular, if  $C$  consists of the poisoned square only, then  $|C| = 1$ ). The basis case is simple: if  $|C| = 1$ , then  $P_1$  loses, so in particular  $\exists \mathbb{S} \text{Win}_{P_2}(\mathbb{S}, C)$  holds (and it does not matter what  $\mathbb{S}$  is, since  $P_2$  will not use it anyways).

For the induction step, suppose that the claim holds for all  $C$  such that  $|C| \leq n$ . Consider some  $C'$  such that  $|C'| = n + 1$ . We want to show that

$$\exists \mathbb{S} [\text{Win}_{P_1}(\mathbb{S}, C') \vee \text{Win}_{P_2}(\mathbb{S}, C')]. \quad (2)$$

Let  $C''$  be the result of  $P_1$  making a move; since  $P_1$  must select some square, it follows that  $|C''| < |C'|$ , and so we can apply the induction hypothesis to

it; in other words,  $\exists \text{SWin}_{P_1}(\mathbb{S}, C'') \vee \exists \text{SWin}_{P_2}(\mathbb{S}, C'')$  (we used the fact here that  $\exists x(\alpha \vee \beta)$  is equivalent to  $\exists x\alpha \vee \exists x\beta$ ). If no matter what first move  $P_1$  makes (to obtain  $C''$ ) it is always the case that  $\exists \text{SWin}_{P_2}(\mathbb{S}, C'')$ , then we can conclude that  $\exists \text{SWin}_{P_2}(\mathbb{S}, C')$ . If, on the other hand, for some move of  $P_1$  it is the case that  $\exists \text{SWin}_{P_1}(\mathbb{S}, C'')$ , then define (using comprehension)  $\mathbb{S}'$  to be the same as  $\mathbb{S}$ , except  $\mathbb{S}'(C') = C''$ , and we can conclude that  $\text{Win}_{P_1}(\mathbb{S}', C')$ , and so  $\exists \text{SWin}_{P_1}(\mathbb{S}, C')$ . Therefore, in either case we obtain (2).  $\square$

Once we know that  $\mathbf{W}_1^1$  proves (1), we can prove that  $P_1$  has a winning strategy for the full rectangle.

**Theorem 2.**  $\mathbf{W}_1^1 \vdash "C \text{ is full rectangle"} \rightarrow \exists \text{SWin}_{P_1}(\mathbb{S}, C)$ .

*Proof.* Suppose that  $C$  is a full rectangle (an  $r \times c$  chomp grid). We know that either  $P_1$  or  $P_2$  has a winning strategy. Suppose that it is  $P_2$ ; so  $P_2$  (playing by some  $\mathbb{S}$ ) will win no matter what first move  $P_1$  makes. So let  $P_1$  select the top-right square (i.e., the square  $(r, c)$ ), and let  $C'$  be the resulting configuration (i.e.,  $C'$  consists of all squares but it has a dent in the top-right square).

Now  $P_2$  makes a move according to  $\mathbb{S}$ , i.e.,  $P_2$  plays to obtain  $C'' = \mathbb{S}(C')$ . Now observe that whoever starts playing from configuration  $C''$  loses, i.e.,  $P_1$  loses on  $C''$ , (by our assumptions). Also observe that  $P_1$  could have played to obtain  $C''$  directly, since no matter what  $C''$  is, it must contain the top-right corner, as it is the supremum of the grid. Thus, by taking the strategy  $\mathbb{S}'$  to be  $\mathbb{S}(C) = C''$  and otherwise  $\mathbb{S}' = \mathbb{S}$ ,  $P_1$  can win (note that we have used comprehension to define  $\mathbb{S}'$  from  $\mathbb{S}$ ). Contradiction; so  $P_2$  cannot have a winning strategy, and so  $P_1$  must have a winning strategy.  $\square$

Using the Witnessing Theorem for  $\mathbf{W}_1^1$  ([4]) we can conclude from theorem 2 that the winning strategy  $\mathbb{S}$  can be computed in PSPACE (which, of course, we know from the previous section—and the fact that there is a simple way of computing the strategy by querying repeatedly whether one exists). The interesting question is: can we prove the same in a weaker theory than  $\mathbf{W}_1^1$ ; in particular, is  $\mathbf{V}^1$  too much to ask for?

## References

1. Gale, D.: A Curious Nim-Type Game. *The American Mathematical Monthly* **81**(8) (October 1974) 876–879
2. Papadimitriou, C.H.: Computational Complexity. Addison Wesley Longman (1995)
3. Sipser, M.: Introduction to the Theory of Computation. Second edn. Thomson Course Technology (2006)
4. Skelley, A.: A Third-Order Bounded Arithmetic Theory for PSPACE. In: CSL. (2004) 340–354
5. Cook, S., Nguyen, P.: Foundations of Proof Complexity: Bounded Arithmetic and Propositional Translations. <http://www.cs.toronto.edu/~sacook/csc2429h/book>. Last checked April 29, 2008 (2006)
6. Herman, G.: Private communication (2006)

# A Statistical Mechanical Interpretation of Algorithmic Information Theory

Kohtaro Tadaki

Research and Development Initiative, Chuo University  
1-13-27 Kasuga, Bunkyo-ku, Tokyo 112-8551, Japan.  
[tadaki@kc.chuo-u.ac.jp](mailto:tadaki@kc.chuo-u.ac.jp)

**Abstract.** We develop a statistical mechanical interpretation of algorithmic information theory by introducing the notion of thermodynamic quantities, such as free energy, energy, statistical mechanical entropy, and specific heat, into algorithmic information theory. We investigate the properties of these quantities by means of program-size complexity from the point of view of algorithmic randomness. It is then discovered that, in the interpretation, the temperature plays a role as the compression rate of the values of all these thermodynamic quantities, which include the temperature itself. Reflecting this self-referential nature of the compression rate of the temperature, we obtain fixed point theorems on compression rate.

**Key words:** algorithmic information theory, algorithmic randomness, Chaitin's  $\Omega$ , compression rate, fixed point theorem, statistical mechanics, temperature

## 1 Introduction

Algorithmic information theory is a framework to apply information-theoretic and probabilistic ideas to recursive function theory. One of the primary concepts of algorithmic information theory is the *program-size complexity* (or *Kolmogorov complexity*)  $H(s)$  of a finite binary string  $s$ , which is defined as the length of the shortest binary program for the universal self-delimiting Turing machine  $U$  to output  $s$ . By the definition,  $H(s)$  can be thought of as the information content of the individual finite binary string  $s$ . In fact, algorithmic information theory has precisely the formal properties of classical information theory (see Chaitin [3]). The concept of program-size complexity plays a crucial role in characterizing the randomness of a finite or infinite binary string. In [3] Chaitin introduced the halting probability  $\Omega$  as an example of random infinite binary string. His  $\Omega$  is defined as the probability that the universal self-delimiting Turing machine  $U$  halts, and plays a central role in the metamathematical development of algorithmic information theory. The first  $n$  bits of the base-two expansion of  $\Omega$  solves the halting problem for a program of size not greater than  $n$ . By this property, the base-two expansion of  $\Omega$  is shown to be a random infinite binary string.

In [7, 8] we generalized Chaitin's halting probability  $\Omega$  to  $\Omega^D$  by

$$\Omega^D = \sum_{p \in \text{dom } U} 2^{-\frac{|p|}{D}}, \quad (1)$$

so that the degree of randomness of  $\Omega^D$  can be controlled by a real number  $D$  with  $0 < D \leq 1$ . Here,  $\text{dom } U$  denotes the set of all programs  $p$  for  $U$ . As  $D$  becomes larger, the degree of randomness of  $\Omega^D$  increases. When  $D = 1$ ,  $\Omega^D$  becomes a random real number, i.e.,  $\Omega^1 = \Omega$ . The properties of  $\Omega^D$  and its relations to self-similar sets were studied in Tadaki [7, 8].

Recently, Calude and Stay [2] pointed out a formal correspondence between  $\Omega^D$  and a partition function in statistical mechanics. In statistical mechanics, the partition function  $Z(T)$  at temperature  $T$  is defined by

$$Z(T) = \sum_{x \in X} e^{-\frac{E_x}{kT}},$$

where  $X$  is a complete set of energy eigenstates of a statistical mechanical system and  $E_x$  is the energy of an energy eigenstate  $x$ . The constant  $k$  is called the Boltzmann Constant. The partition function  $Z(T)$  is of particular importance in equilibrium statistical mechanics. This is because all the thermodynamic quantities of the system can be expressed by using the partition function  $Z(T)$ , and the knowledge of  $Z(T)$  is sufficient to understand all the macroscopic properties of the system. Calude and Stay [2] pointed out, in essence, that the partition function  $Z(T)$  has the same form as  $\Omega^D$  by performing the following replacements in  $Z(T)$ :

### Replacements 1

- (i) Replace the complete set  $X$  of energy eigenstates  $x$  by the set  $\text{dom } U$  of all programs  $p$  for  $U$ .
- (ii) Replace the energy  $E_x$  of an energy eigenstate  $x$  by the length  $|p|$  of a program  $p$ .
- (iii) Set the Boltzmann Constant  $k$  to  $1/\ln 2$ , where the  $\ln$  denotes the natural logarithm.  $\square$

In this paper, inspired by their suggestion above, we develop a statistical mechanical interpretation of algorithmic information theory, where  $\Omega^D$  appears as a partition function.

Generally speaking, in order to give a statistical mechanical interpretation to a framework which looks unrelated to statistical mechanics at first glance, it is important to identify a microcanonical ensemble in the framework. Once we can do so, we can easily develop an equilibrium statistical mechanics on the framework according to the theoretical development of normal equilibrium statistical mechanics. Here, the microcanonical ensemble is a certain sort of uniform probability distribution. In fact, in the work [9] we developed a statistical mechanical interpretation of the noiseless source coding scheme in information theory by identifying a microcanonical ensemble in the scheme. Then, in [9] the notions in statistical mechanics such as statistical mechanical entropy, temperature, and thermal equilibrium are translated into the context of noiseless source coding.

Thus, in order to develop a statistical mechanical interpretation of algorithmic information theory, it is appropriate to identify a microcanonical ensemble in the framework of the theory. Note, however, that algorithmic information

theory is not a physical theory but a purely mathematical theory. Therefore, in order to obtain significant results for the development of algorithmic information theory itself, we have to develop a statistical mechanical interpretation of algorithmic information theory in a mathematically rigorous manner, unlike in normal statistical mechanics in physics where arguments are not necessarily mathematically rigorous. A fully rigorous mathematical treatment of statistical mechanics is already developed (see Ruelle [6]). At present, however, it would not as yet seem to be an easy task to merge algorithmic information theory with this mathematical treatment in a satisfactory manner.

On the other hand, if we do not stick to the mathematical strictness of an argument and make an argument on the same level of mathematical strictness as statistical mechanics in physics, we can develop a statistical mechanical interpretation of algorithmic information theory while realizing a perfect correspondence to normal statistical mechanics. In the physical argument, we can identify a microcanonical ensemble in algorithmic information theory in a similar manner to [9], based on the probability measure which gives Chaitin's  $\Omega$  the meaning of the halting probability actually.<sup>1</sup> In consequence, for example, the statistical mechanical meaning of  $\Omega^D$  is clarified.

In this paper, we develop a statistical mechanical interpretation of algorithmic information theory in a different way from the physical argument mentioned above.<sup>2</sup> We introduce the notion of thermodynamic quantities into algorithmic information theory based on Replacements 1 above.

After the preliminary section on the mathematical notion needed in this paper, in Section 3 we introduce the notion of the thermodynamic quantities at any given fixed temperature  $T$ , such as partition function, free energy, energy, statistical mechanical entropy, and specific heat, into algorithmic information theory by performing Replacements 1 for the corresponding thermodynamic quantities in statistical mechanics. These thermodynamic quantities in algorithmic information theory are real numbers which depend only on the temperature  $T$ . We prove that if the temperature  $T$  is a computable real number with  $0 < T < 1$  then, for each of these thermodynamic quantities, the compression rate by the program-size complexity  $H$  is equal to  $T$ . Thus, the temperature  $T$  plays a role as the compression rate of the thermodynamic quantities in this statistical mechanical interpretation of algorithmic information theory.

Among all thermodynamic quantities in thermodynamics, one of the most typical thermodynamic quantities is temperature itself. Thus, based on the results of Section 3, the following question naturally arises: Can the compression rate of the temperature  $T$  be equal to the temperature itself in the statistical mechanical interpretation of algorithmic information theory? This question is rather self-referential. However, in Section 4 we answer it affirmatively by prov-

---

<sup>1</sup> Due to the 10-page limit, we omit the detail of the physical argument in this paper. It will be included in a full version of this paper, and is also available in Section 6 of an extended and electronic version of this paper at URL: <http://arxiv.org/abs/0801.4194v1>

<sup>2</sup> We make an argument in a fully mathematically rigorous manner in this paper.

ing Theorem 9. One consequence of Theorem 9 has the following form: For every  $T \in (0, 1)$ , if  $\Omega^T = \sum_{p \in \text{dom } U} 2^{-\frac{|p|}{T}}$  is a computable real number, then

$$\lim_{n \rightarrow \infty} \frac{H(T_n)}{n} = T,$$

where  $T_n$  is the first  $n$  bits of the base-two expansion of  $T$ . This is just a fixed point theorem on compression rate, which reflects the self-referential nature of the question.

The works [7, 8] on  $\Omega^D$  might be regarded as an elaboration of the technique used by Chaitin [3] to prove that  $\Omega$  is random. The results of this paper may be regarded as further elaborations of the technique. Due to the 10-page limit, we omit most proofs. A full paper describing the details of the proofs is in preparation.<sup>3</sup>

## 2 Preliminaries

We start with some notation about numbers and strings which will be used in this paper.  $\mathbb{N} = \{0, 1, 2, 3, \dots\}$  is the set of natural numbers, and  $\mathbb{N}^+$  is the set of positive integers.  $\mathbb{Q}$  is the set of rational numbers, and  $\mathbb{R}$  is the set of real numbers.  $\{0, 1\}^* = \{\lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, \dots\}$  is the set of finite binary strings, where  $\lambda$  denotes the *empty string*. For any  $s \in \{0, 1\}^*$ ,  $|s|$  is the *length* of  $s$ . A subset  $S$  of  $\{0, 1\}^*$  is called a *prefix-free set* if no string in  $S$  is a prefix of another string in  $S$ .  $\{0, 1\}^\infty$  is the set of infinite binary strings, where an infinite binary string is infinite to the right but finite to the left. For any  $\alpha \in \{0, 1\}^\infty$  and any  $n \in \mathbb{N}^+$ ,  $\alpha_n$  is the prefix of  $\alpha$  of length  $n$ . For any partial function  $f$ , the domain of definition of  $f$  is denoted by  $\text{dom } f$ . We write “r.e.” instead of “recursively enumerable.”

Normally,  $o(n)$  denotes any function  $f: \mathbb{N}^+ \rightarrow \mathbb{R}$  such that  $\lim_{n \rightarrow \infty} f(n)/n = 0$ . On the other hand,  $O(1)$  denotes any function  $g: \mathbb{N}^+ \rightarrow \mathbb{R}$  such that there is  $C \in \mathbb{R}$  with the property that  $|g(n)| \leq C$  for all  $n \in \mathbb{N}^+$ .

Let  $T$  be an arbitrary real number.  $T \bmod 1$  denotes  $T - \lfloor T \rfloor$ , where  $\lfloor T \rfloor$  is the greatest integer less than or equal to  $T$ . Hence,  $T \bmod 1 \in [0, 1)$ . We identify a real number  $T$  with the infinite binary string  $\alpha$  such that  $0.\alpha$  is the base-two expansion of  $T \bmod 1$  with infinitely many zeros. Thus,  $T_n$  denotes the first  $n$  bits of the base-two expansion of  $T \bmod 1$  with infinitely many zeros.

We say that a real number  $T$  is *computable* if there exists a total recursive function  $f: \mathbb{N}^+ \rightarrow \mathbb{Q}$  such that  $|T - f(n)| < 1/n$  for all  $n \in \mathbb{N}^+$ . We say that  $T$  is *right-computable* if there exists a total recursive function  $g: \mathbb{N}^+ \rightarrow \mathbb{Q}$  such that  $T \leq g(n)$  for all  $n \in \mathbb{N}^+$  and  $\lim_{n \rightarrow \infty} g(n) = T$ . We say that  $T$  is *left-computable* if  $-T$  is right-computable. It is then easy to see that, for any  $T \in \mathbb{R}$ ,  $T$  is computable if and only if  $T$  is both right-computable and left-computable. See e.g. Weihrauch [12] for the detail of the treatment of the computability of real numbers and real functions on a discrete set.

---

<sup>3</sup> The details of the proofs are also available in an extended and electronic version of this paper at URL: <http://arxiv.org/abs/0801.4194v1>

## 2.1 Algorithmic information theory

In the following we concisely review some definitions and results of algorithmic information theory [3, 4]. A *computer* is a partial recursive function  $C: \{0, 1\}^* \rightarrow \{0, 1\}^*$  such that  $\text{dom } C$  is a prefix-free set. For each computer  $C$  and each  $s \in \{0, 1\}^*$ ,  $H_C(s)$  is defined by  $H_C(s) = \min \{ |p| \mid p \in \{0, 1\}^* \& C(p) = s \}$ . A computer  $U$  is said to be *optimal* if for each computer  $C$  there exists a constant  $\text{sim}(C)$  with the following property; if  $C(p)$  is defined, then there is a  $p'$  for which  $U(p') = C(p)$  and  $|p'| \leq |p| + \text{sim}(C)$ . It is easy to see that there exists an optimal computer. Note that the class of optimal computers equals to the class of functions which are computed by *universal self-delimiting Turing machines* (see Chaitin [3] for the detail). We choose a particular optimal computer  $U$  as the standard one for use, and define  $H(s)$  as  $H_U(s)$ , which is referred to as the *program-size complexity* of  $s$  or the *Kolmogorov complexity* of  $s$ .

*Chaitin's halting probability*  $\Omega$  is defined by

$$\Omega = \sum_{p \in \text{dom } U} 2^{-|p|}.$$

For any  $\alpha \in \{0, 1\}^\infty$ , we say that  $\alpha$  is *weakly Chaitin random* if there exists  $c \in \mathbb{N}$  such that  $n - c \leq H(\alpha_n)$  for all  $n \in \mathbb{N}^+$  [3, 4]. Then Chaitin [3] showed that  $\Omega$  is weakly Chaitin random. For any  $\alpha \in \{0, 1\}^\infty$ , we say that  $\alpha$  is *Chaitin random* if  $\lim_{n \rightarrow \infty} H(\alpha_n) - n = \infty$  [3, 4]. It is then shown that, for any  $\alpha \in \{0, 1\}^\infty$ ,  $\alpha$  is weakly Chaitin random if and only if  $\alpha$  is Chaitin random (see Chaitin [4] for the proof and historical detail). Thus  $\Omega$  is Chaitin random.

In the works [7, 8], we generalized the notion of the randomness of an infinite binary string so that the degree of the randomness can be characterized by a real number  $D$  with  $0 < D \leq 1$  as follows.

**Definition 1 (weak Chaitin  $D$ -randomness and  $D$ -compressibility).** Let  $D \in \mathbb{R}$  with  $D \geq 0$ , and let  $\alpha \in \{0, 1\}^\infty$ . We say that  $\alpha$  is *weakly Chaitin  $D$ -random* if there exists  $c \in \mathbb{N}$  such that  $Dn - c \leq H(\alpha_n)$  for all  $n \in \mathbb{N}^+$ . We say that  $\alpha$  is  *$D$ -compressible* if  $H(\alpha_n) \leq Dn + o(n)$ , which is equivalent to  $\lim_{n \rightarrow \infty} H(\alpha_n)/n \leq D$ .  $\square$

In the case of  $D = 1$ , the weak Chaitin  $D$ -randomness results in the weak Chaitin randomness. For any  $D \in [0, 1]$  and any  $\alpha \in \{0, 1\}^\infty$ , if  $\alpha$  is weakly Chaitin  $D$ -random and  $D$ -compressible, then

$$\lim_{n \rightarrow \infty} \frac{H(\alpha_n)}{n} = D. \quad (2)$$

Hereafter the left-hand side of (2) is referred to as the *compression rate* of an infinite binary string  $\alpha$  in general. Note, however, that (2) does not necessarily imply that  $\alpha$  is weakly Chaitin  $D$ -random.

In the works [7, 8], we generalized Chaitin's halting probability  $\Omega$  to  $\Omega^D$  by (1) for any real number  $D > 0$ . Thus,  $\Omega = \Omega^1$ . If  $0 < D \leq 1$ , then  $\Omega^D$  converges and  $0 < \Omega^D < 1$ , since  $\Omega^D \leq \Omega < 1$ .

**Theorem 2 (Tadaki [7, 8]).** Let  $D \in \mathbb{R}$ .

- (i) If  $0 < D \leq 1$  and  $D$  is computable, then  $\Omega^D$  is weakly Chaitin  $D$ -random and  $D$ -compressible.
- (ii) If  $1 < D$ , then  $\Omega^D$  diverges to  $\infty$ .  $\square$

**Definition 2 (Chaitin  $D$ -randomness, Tadaki [7, 8]).** Let  $D \in \mathbb{R}$  with  $D \geq 0$ , and let  $\alpha \in \{0, 1\}^\infty$ . We say that  $\alpha$  is Chaitin  $D$ -random if  $\lim_{n \rightarrow \infty} H(\alpha_n) - Dn = \infty$ .  $\square$

In the case of  $D = 1$ , the Chaitin  $D$ -randomness results in the Chaitin randomness. Obviously, for any  $D \in [0, 1]$  and any  $\alpha \in \{0, 1\}^\infty$ , if  $\alpha$  is Chaitin  $D$ -random, then  $\alpha$  is weakly Chaitin  $D$ -random. However, in 2005 Reimann and Stephan [5] showed that, in the case of  $D < 1$ , the converse does not necessarily hold. This contrasts with the equivalence between the weakly Chaitin randomness and the Chaitin randomness, each of which corresponds to the case of  $D = 1$ .

For each real numbers  $Q > 0$  and  $D > 0$ , we define  $W(Q, D)$  by

$$W(Q, D) = \sum_{p \in \text{dom } U} |p|^Q 2^{-\frac{|p|}{D}}.$$

As the first result of this paper, we can show the following theorem.

**Theorem 3.** Let  $Q$  and  $D$  be positive real numbers.

- (i) If  $Q$  and  $D$  are computable and  $0 < D < 1$ , then  $W(Q, D)$  converges to a left-computable real number which is Chaitin  $D$ -random and  $D$ -compressible.
- (ii) If  $1 \leq D$ , then  $W(Q, D)$  diverges to  $\infty$ .  $\square$

Thus, we see that the weak Chaitin  $D$ -randomness in Theorem 2 is replaced by the Chaitin  $D$ -randomness in Theorem 3 in exchange for the divergence at  $D = 1$ .

### 3 Temperature as a compression rate

In this section we introduce the notion of thermodynamic quantities such as partition function, free energy, energy, statistical mechanical entropy, and specific heat, into algorithmic information theory by performing Replacements 1 for the corresponding thermodynamic quantities in statistical mechanics.<sup>4</sup> We investigate their convergence and the degree of randomness. For that purpose, we first choose a particular enumeration  $q_1, q_2, q_3, \dots$  of the countably infinite set  $\text{dom } U$  as the standard one for use throughout this section.<sup>5</sup>

---

<sup>4</sup> For the thermodynamic quantities in statistical mechanics, see Chapter 16 of [1] and Chapter 2 of [11]. To be precise, the partition function is not a thermodynamic quantity but a statistical mechanical quantity.

<sup>5</sup> The enumeration  $\{q_i\}$  is quite arbitrary and therefore we do not, ever, require  $\{q_i\}$  to be a recursive enumeration of  $\text{dom } U$ .

In statistical mechanics, the partition function  $Z_{\text{sm}}(T)$  at temperature  $T$  is given by

$$Z_{\text{sm}}(T) = \sum_{x \in X} e^{-\frac{E_x}{kT}}, \quad (3)$$

Motivated by the formula (3) and taking into account Replacements 1, we introduce the notion of partition function into algorithmic information theory as follows.

**Definition 3 (partition function).** *For each  $n \in \mathbb{N}^+$  and each real number  $T > 0$ , we define  $Z_n(T)$  by*

$$Z_n(T) = \sum_{i=1}^n 2^{-\frac{|q_i|}{T}}.$$

*Then, the partition function  $Z(T)$  is defined by  $Z(T) = \lim_{n \rightarrow \infty} Z_n(T)$ , for each  $T > 0$ .*  $\square$

Since  $Z(T) = \Omega^T$ , we restate Theorem 2 as in the following form.

**Theorem 4 (Tadaki [7, 8]).** *Let  $T \in \mathbb{R}$ .*

- (i) *If  $0 < T \leq 1$  and  $T$  is computable, then  $Z(T)$  converges to a left-computable real number which is weakly Chaitin  $T$ -random and  $T$ -compressible.*
- (ii) *If  $1 < T$ , then  $Z(T)$  diverges to  $\infty$ .*  $\square$

In statistical mechanics, the free energy  $F_{\text{sm}}(T)$  at temperature  $T$  is given by

$$F_{\text{sm}}(T) = -kT \ln Z_{\text{sm}}(T), \quad (4)$$

where  $Z_{\text{sm}}(T)$  is given by (3). Motivated by the formula (4) and taking into account Replacements 1, we introduce the notion of free energy into algorithmic information theory as follows.

**Definition 4 (free energy).** *For each  $n \in \mathbb{N}^+$  and each real number  $T > 0$ , we define  $F_n(T)$  by  $F_n(T) = -T \log_2 Z_n(T)$ . Then, for each  $T > 0$ , the free energy  $F(T)$  is defined by  $F(T) = \lim_{n \rightarrow \infty} F_n(T)$ .*  $\square$

**Theorem 5.** *Let  $T \in \mathbb{R}$ .*

- (i) *If  $0 < T \leq 1$  and  $T$  is computable, then  $F(T)$  converges to a right-computable real number which is weakly Chaitin  $T$ -random and  $T$ -compressible.*
- (ii) *If  $1 < T$ , then  $F(T)$  diverges to  $-\infty$ .*  $\square$

In statistical mechanics, the energy  $E_{\text{sm}}(T)$  at temperature  $T$  is given by

$$E_{\text{sm}}(T) = \frac{1}{Z_{\text{sm}}(T)} \sum_{x \in X} E_x e^{-\frac{E_x}{kT}}, \quad (5)$$

where  $Z_{\text{sm}}(T)$  is given by (3). Motivated by the formula (5) and taking into account Replacements 1, we introduce the notion of energy into algorithmic information theory as follows.

**Definition 5 (energy).** For each  $n \in \mathbb{N}^+$  and each real number  $T > 0$ , we define  $E_n(T)$  by

$$E_n(T) = \frac{1}{Z_n(T)} \sum_{i=1}^n |q_i| 2^{-\frac{|q_i|}{T}}.$$

Then, for each  $T > 0$ , the energy  $E(T)$  is defined by  $E(T) = \lim_{n \rightarrow \infty} E_n(T)$ .  $\square$

**Theorem 6.** Let  $T \in \mathbb{R}$ .

- (i) If  $0 < T < 1$  and  $T$  is computable, then  $E(T)$  converges to a left-computable real number which is Chaitin  $T$ -random and  $T$ -compressible.
- (ii) If  $1 \leq T$ , then  $E(T)$  diverges to  $\infty$ .  $\square$

In statistical mechanics, the entropy  $S_{\text{sm}}(T)$  at temperature  $T$  is given by

$$S_{\text{sm}}(T) = \frac{1}{T} E_{\text{sm}}(T) + k \ln Z_{\text{sm}}(T), \quad (6)$$

where  $Z_{\text{sm}}(T)$  and  $E_{\text{sm}}(T)$  are given by (3) and (5), respectively. Motivated by the formula (6) and taking into account Replacements 1, we introduce the notion of statistical mechanical entropy into algorithmic information theory as follows.

**Definition 6 (statistical mechanical entropy).** For each  $n \in \mathbb{N}^+$  and each real number  $T > 0$ , we define  $S_n(T)$  by  $S_n(T) = \frac{1}{T} E_n(T) + \log_2 Z_n(T)$ . Then, for each  $T > 0$ , the statistical mechanical entropy  $S(T)$  is defined by  $S(T) = \lim_{n \rightarrow \infty} S_n(T)$ .  $\square$

**Theorem 7.** Let  $T \in \mathbb{R}$ .

- (i) If  $0 < T < 1$  and  $T$  is computable, then  $S(T)$  converges to a left-computable real number which is Chaitin  $T$ -random and  $T$ -compressible.
- (ii) If  $1 \leq T$ , then  $S(T)$  diverges to  $\infty$ .  $\square$

Finally, in statistical mechanics, the specific heat  $C_{\text{sm}}(T)$  at temperature  $T$  is given by

$$C_{\text{sm}}(T) = \frac{d}{dT} E_{\text{sm}}(T), \quad (7)$$

where  $E_{\text{sm}}(T)$  is given by (5). Motivated by this formula (7), we introduce the notion of specific heat into algorithmic information theory as follows.

**Definition 7 (specific heat).** For each  $n \in \mathbb{N}^+$  and each real number  $T > 0$ , we define  $C_n(T)$  by  $C_n(T) = E'_n(T)$ , where  $E'_n(T)$  is the derived function of  $E_n(T)$ . Then, for each  $T > 0$ , the specific heat  $C(T)$  is defined by  $C(T) = \lim_{n \rightarrow \infty} C_n(T)$ .  $\square$

**Theorem 8.** Let  $T \in \mathbb{R}$ .

- (i) If  $0 < T < 1$  and  $T$  is computable, then  $C(T)$  converges to a left-computable real number which is Chaitin  $T$ -random and  $T$ -compressible, and moreover  $C(T) = E'(T)$  where  $E'(T)$  is the derived function of  $E(T)$ .
- (ii) If  $T = 1$ , then  $C(T)$  diverges to  $\infty$ .  $\square$

Thus, the theorems in this section show that the temperature  $T$  plays a role as the compression rate for all the thermodynamic quantities introduced into algorithmic information theory in this section.

These theorems also show that the values of the thermodynamic quantities: partition function, free energy, energy, and statistical mechanical entropy diverge in the case of  $T > 1$ . This phenomenon might be regarded as some sort of phase transition in statistical mechanics.<sup>6</sup>

#### 4 Fixed point theorems on compression rate

In this section, we show the following theorem and its variant.

**Theorem 9 (fixed point theorem on compression rate).** *For every  $T \in (0, 1)$ , if  $Z(T)$  is a computable real number, then the following hold:*

- (i)  *$T$  is right-computable and not left-computable.*
- (ii)  *$T$  is weakly Chaitin  $T$ -random and  $T$ -compressible.*
- (iii)  $\lim_{n \rightarrow \infty} H(T_n)/n = T$ .

□

Theorem 9 follows immediately from the following three theorems.

**Theorem 10.** *For every  $T \in (0, 1)$ , if  $Z(T)$  is a right-computable real number, then  $T$  is weakly Chaitin  $T$ -random.* □

**Theorem 11.** *For every  $T \in (0, 1)$ , if  $Z(T)$  is a right-computable real number, then  $T$  is also a right-computable real number.* □

**Theorem 12.** *For every  $T \in (0, 1)$ , if  $Z(T)$  is a left-computable real number and  $T$  is a right-computable real number, then  $T$  is  $T$ -compressible.* □

Theorem 9 is just a fixed point theorem on compression rate, where the computability of the value  $Z(T)$  gives a sufficient condition for a real number  $T \in (0, 1)$  to be a fixed point on compression rate. Note that  $Z(T)$  is a strictly increasing continuous function on  $(0, 1)$ . In fact, Tadaki [7, 8] showed that  $Z(T)$  is a function of class  $C^\infty$  on  $(0, 1)$ . Thus, since the set of all computable real numbers is dense in  $\mathbb{R}$ , we have the following for this sufficient condition.

**Theorem 13.** *The set  $\{T \in (0, 1) \mid Z(T) \text{ is computable}\}$  is dense in  $[0, 1]$ .* □

We thus have the following corollary of Theorem 9.

**Corollary 1.** *The set  $\{T \in (0, 1) \mid \lim_{n \rightarrow \infty} H(T_n)/n = T\}$  is dense in  $[0, 1]$ .* □

From the point of view of the statistical mechanical interpretation introduced in the previous section, Theorem 9 shows that the compression rate of temperature is equal to the temperature itself. Thus, Theorem 9 further confirms the role of temperature as the compression rate, which is observed in the previous section.

In a similar manner to the proof of Theorem 9, we can prove another version of a fixed point theorem on compression rate as follows. Here, the weak Chaitin  $T$ -randomness is replaced by the Chaitin  $T$ -randomness.

---

<sup>6</sup> It is still open whether  $C(T)$  diverges or not in the case of  $T > 1$ .

**Theorem 14 (fixed point theorem on compression rate II).** *Let  $Q$  be a computable real number with  $Q > 0$ . For every  $T \in (0, 1)$ , if  $W(Q, T)$  is a computable real number, then the following hold:*

- (i)  *$T$  is right-computable and not left-computable.*
- (ii)  *$T$  is Chaitin  $T$ -random and  $T$ -compressible.*

□

For the sufficient condition of Theorem 14, in a similar manner to the case of Theorem 9, we can show that, for every  $Q > 0$ , the set  $\{T \in (0, 1) \mid W(Q, T) \text{ is computable}\}$  is dense in  $[0, 1]$ .

## Acknowledgments

This work was supported both by KAKENHI, Grant-in-Aid for Scientific Research (C) (20540134) and by SCOPE (Strategic Information and Communications R&D Promotion Programme) from the Ministry of Internal Affairs and Communications of Japan.

## References

1. H. B. Callen, *Thermodynamics and an Introduction to Thermostatistics*, 2nd ed. John Wiley & Sons, Inc., Singapore, 1985.
2. C. S. Calude and M. A. Stay, “Natural halting probabilities, partial randomness, and zeta functions,” *Inform. and Comput.*, vol. 204, pp. 1718–1739, 2006.
3. G. J. Chaitin, “A theory of program size formally identical to information theory,” *J. Assoc. Comput. Mach.*, vol. 22, pp. 329–340, 1975.
4. G. J. Chaitin, *Algorithmic Information Theory*. Cambridge University Press, Cambridge, 1987.
5. J. Reimann and F. Stephan, On hierarchies of randomness tests. Proceedings of the 9th Asian Logic Conference, World Scientific Publishing, August 16–19, 2005, Novosibirsk, Russia.
6. D. Ruelle, *Statistical Mechanics, Rigorous Results*, 3rd ed. Imperial College Press and World Scientific Publishing Co. Pte. Ltd., Singapore, 1999.
7. K. Tadaki, Algorithmic information theory and fractal sets. Proceedings of 1999 Workshop on Information-Based Induction Sciences (IBIS’99), pp. 105–110, August 26–27, 1999, Shizuoka, Japan. In Japanese.
8. K. Tadaki, “A generalization of Chaitin’s halting probability  $\Omega$  and halting self-similar sets,” *Hokkaido Math. J.*, vol. 31, pp. 219–253, 2002. Electronic Version Available: <http://arxiv.org/abs/nlin/0212001v1>
9. K. Tadaki, A statistical mechanical interpretation of instantaneous codes. Proceedings of 2007 IEEE International Symposium on Information Theory (ISIT2007), pp. 1906–1910, June 24–29, 2007, Nice, France.
10. K. Tadaki, The Tsallis entropy and the Shannon entropy of a universal probability. To appear in the Proceedings of 2008 IEEE International Symposium on Information Theory (ISIT2008), July 6–11, 2008, Toronto, Canada.
11. M. Toda, R. Kubo, and N. Saitô, *Statistical Physics I. Equilibrium Statistical Mechanics*, 2nd ed. Springer, Berlin, 1992.
12. K. Weihrauch, *Computable Analysis*. Springer-Verlag, Berlin, 2000.

# Solving Tripartite Matching by Interval-valued Computation in Polynomial Time<sup>\*</sup>

Ákos Tajti and Benedek Nagy<sup>\*\*</sup>

Faculty of Informatics, University of Debrecen  
Hungary H-4010 Debrecen, P.O. Box 12  
[akos.tajti@gmail.com](mailto:akos.tajti@gmail.com), [nbenedek@inf.unideb.hu](mailto:nbenedek@inf.unideb.hu)

**Abstract.** New computing paradigms are usually legitimated by showing their computing power. Hard, usually NP-complete problems are shown to be solved in efficient way. One of the well known NP-complete problem is Tripartite Matching. In this paper this problem is solved by a polynomial Interval-valued computation.

**Key words:** new computing paradigms, interval-valued computing, NP-complete problems

## 1 Introduction

There are several problems shown to be computationally hard. Karp presented several computational problems that are proved to be NP-complete ([Karp 1972]). These intractable problems cannot be solved efficiently by the traditional computing way (deterministic Turing machines or similar devices) unless P=NP. The problem P=NP is one of the most important challenges of the (Theoretical) Computer Science. The fact that there is no known method to solve efficiently these problems by traditional computations leads to the phenomenon of developing several new paradigms of computation. Usually in these new paradigms one or more of the basic properties of the classical computing methodology are dropped. The traditional computation is sequential, uses discrete time-steps, the amount of data that can be processed in a step is finitely limited, etc. ([Tanenbaum 1984]) DNA and membrane computing, inspired by molecular biology, drop the sequential mode of computation. In his famous paper ([Adleman 1994]) Adleman showed a method to solve the NP-complete Hamiltonian Path problem in polynomial time by DNA computation. Several various methods are known to solve various NP-complete problems efficiently by membrane computing ([Paun 2002]), as well. The ways of parallelism of these (bio)computations are essentially different ([Loos–Nagy 2007]).

Interval-valued computation is another computing paradigm initiated by B. Nagy in [Nagy 2005b,Nagy 2005c,Nagy–Vályi 2008] based on an interval-valued

---

<sup>\*</sup> The work is partly supported by the Öveges programme of NKTH, Hungary.

<sup>\*\*</sup> Corresponding author

fuzzy logic ([Nagy 2005a]) instead of the binary logic of traditional architectures ([Tanenbaum 1984]). In this paradigm the amount of data that can be processed at a step has not restriction. The paradigm keeps some features of the traditional computations and of classical computers. Classical computers work on finite sequences of bits, called bytes or memory cells. Instead of finite sequences of bits the interval-valued computations work on specific subsets of the interval  $[0, 1]$ , more specifically, on finite unions of  $[]$ -type subintervals. In this way it is a straightforward extension of the classical paradigm. It also can be considered as a 1-dimensional version of the optical computing [Woods–Naughton 2005].

In this paper, after the formulation of the problem of tripartite matching (Section 2) and a formulation of the computing model (Section 3), a method is shown that solves the problem in the frame of the model in an efficient way (Section 4). An example is also presented (Section 5), finally Conclusions close the paper.

## 2 Formulation of the Problem

Now the formal definition of the proposed problem is presented (based on [Papadimitriou 1994]).

### **Definition 1. ( The Tripartite Matching problem )**

*Let  $B = \{b_1, \dots, b_m\}$ ,  $G = \{g_1, \dots, g_m\}$  and  $H = \{h_1, \dots, h_m\}$  be three disjoint sets such that  $|B| = |G| = |H| = m$  ( $m > 1$ ) and let  $T \subseteq B \times G \times H$  be a relation. The task is to find a relation  $U \subseteq T$  containing each element of  $B$ ,  $G$  and  $H$  exactly once. In other words no two distinct triplets in  $T$  contain the same  $b_i \in B$ ,  $g_j \in G$  and  $h_k \in H$ , moreover every element  $b_i \in B$ ,  $g_j \in G$ , and  $h_k \in H$  appears in one triplet of  $T$ .*

One generalization of the problem is the *simple d-partite matching* problem. An instance of this problem consists of the sets  $A_1, A_2, \dots, A_d$  ( $d \geq 3$ ), and the  $T \subseteq A_1 \times A_2 \times \dots \times A_d$  relation. The task is (similarly to Tripartite Matching) to find a relation  $U \subseteq T$  containing each element of the sets  $A_i$  ( $i = 1, \dots, d$ ) exactly once. Further generalizing the problem we get the *d-partite matching* problem in which a weight  $c_a$  is associated with every *d-tuple* and the task is to find a matching with the minimal cost.

Our approach can be extended to solve these more general problems but in the remaining of this paper we are only concerned with the original Tripartite Matching problem.

The d-partite matching problem is also important in – mainly biological – applications. In [Singh–Xu–Berger 2008] the authors used the problem in an algorithm for global alignment of multiple protein-protein inter-action (PPI) networks. This is the first known algorithm about the problem. A new computational method is invented for recognition of binding patterns common to a set of protein structures in [Shatsky–Shulman–Peleg–Nussinov–Wolfson 2005a]. The d-partite matching is also used in that algorithm. By the same authors a method

of recognition of a set of common physico-chemical properties is presented in [Shatsky–Shulman–Peleg–Nussinov–Wolfson 2005b].

In 1972 Karp proved that the Tripartite Matching problem is NP-complete ([Karp 1972], see also [Papadimitriou 1994]). Since there are not known efficient algorithms to solve this problem in traditional computer, in our approach a new computing paradigm will be used. In the next section we briefly describe the interval-valued computing.

### 3 A Brief Description of Interval-valued Computations

In this section we briefly describe the model. For full description we refer to [Nagy 2005b] and [Nagy–Vályi 2008].

First we present an inductive definition of the *interval-values*.

Base of induction:

Every interval of the form  $[a, b]$  with  $0 \leq a < b \leq 1$  is an *atomic interval*.

The unit interval  $[0, 1]$  is an atomic interval. The interval-values can be considered as sets of real numbers (points) of the unit interval  $[0, 1]$ .

The atomic intervals are interval-values.

Induction steps:

If  $A$  and  $B$  are interval-values, then  $A \cup B$  and  $A \setminus B$  are interval-values (the operations union and difference can be applied on interval-values as sets).

Every interval-value can be obtained by finitely many applications of the inductive steps from some atomic interval-values.

In this way, the interval-values are finite unions of atomic intervals. The empty interval-value is also allowed since it can be obtained as the difference of any atomic-interval and the unit interval.

The *characteristic function* of the interval-values  $A$  is a function  $[0, 1] \rightarrow \{0, 1\}$  giving value 1 at exactly those points of the unit interval which belong to  $A$ .

The operations defined on these interval-values are motivated by operations of the traditional computers. There are logical and non-logical operators.

The logical operators are naturally extended for interval-values. They work on the values of the characteristic functions. Table 1 shows the interval operators corresponding to some well known logical operators. All Boolean operators can be defined and, as it was shown in [Nagy 2005b], the set of interval-values are closed under these operations. Usually the operations negation, disjunction, conjunction and implication are defined. By the expressiveness of the operators using each other, it is sufficient to define only nand or the Sheffer stroke. These operators are well-known, they have theoretical importance, the number of defined operators on interval-values can be reduced by using only one of them as logical operator.

Besides in this paper we need mostly the logical operators, we mention here that some non-logical operators such as shift and product are also defined in

**Table 1.** Definition of the logical operators on interval-values  
( $A$  and  $B$  are interval-values)

Operator	Negation	Disjunction	Conjunction	Implication
Sign	$\neg A$	$A \vee B$	$A \wedge B$	$A \supset B$
Value	$[0, 1] \setminus A$	$A \cup B$	$A \cap B$	$\neg A \cup B$

[Nagy 2005b] and used also in [Nagy–Vályi 2008]. For the completeness of this paper we recall the definition of product:

Let  $A = \bigcup_{i=1}^k [a_{i_1}, a_{i_2})$  and  $B = \bigcup_{j=1}^l [b_{j_1}, b_{j_2})$  be two interval values with  $k$  and  $l$  atomic intervals, respectively. The value of  $C = A * B$  is defined as follows: the number of its atomic intervals is  $k \cdot l$ . Their indices are given in the form  $(i, j)$  from  $(1, 1)$  to  $(k, l)$ . Then the atomic interval with index  $(i, j)$  is  $[a_{i_1} + b_{j_1}(a_{i_2} - a_{i_1}), a_{i_1} + b_{j_2}(a_{i_2} - a_{i_1})]$ .

An interval-valued computation is a deterministic sequence of operator applications starting by some atomic intervals. The operands of the operators can be any initially defined atomic interval and any previously computed interval-value. In decision problems one may check if the interval-value obtained by the computation is the empty interval-value. Note, that since the negation is allowed operation, it has the same efficiency than the checking whether the result is the unit interval  $[0, 1]$ . The complexity is measured by the length of the computation (i.e., the length of the computational sequence: the sum of the initial atomic intervals and operator applications). We note here, that the interval-valued computations proved to be very powerful using only 1 initially defined atomic interval, the  $[0, \frac{1}{2}]$ , for details see [Nagy–Vályi 2008].

## 4 Theory: Solution in Polynomial Time

NP-completeness means that we cannot solve the problem in polynomial time by computers with traditional architecture, unless P=NP. At the same time, the paradigm of interval-valued computations makes it possible to execute the steps of the algorithms in a parallel way. More precisely, the real power comes from the fact that each logical operation is performed (in parallel) on every interval of the interval-value, independent of the number of these intervals. Due to this property of this computing architecture we can construct an algorithm to solve the Tripartite Matching problem in polynomial time using the inner parallelism.

For this reason, we must build an equivalent specification of the problem using interval-values instead of triplets. To do this, we will engage the theory of logic: we will transform the elements of set theory to elements of logic. We know the solution of the SAT problem on interval computers ([Nagy 2005b]) and so, if we reduce our problem to the SAT problem, then we will be able to solve it efficiently.

#### 4.1 Reduction to the SAT problem

We will present a method of constructing a logical formula for each instance of the Tripartite Matching problem. These logical formulas are satisfiable if and only if the original problem instance is solvable; moreover if a solution exists, we can use these formulas to construct a solution.

We define the  $X_{bgh}$  Boolean variable as the following:

$$X_{bgh} = \begin{cases} \text{true}, & \text{if the } (b, g, h) \text{ triplet is in the set } U \\ \text{false}, & \text{otherwise,} \end{cases}$$

where  $b \in B$ ,  $g \in G$  and  $h \in H$ . This assignment can be done efficiently. (In a cubic complexity of  $m$ .)

Next, using Definition 1, the following logical formulas are constructed using these variables:

$$E_B = \begin{cases} \bigwedge_{b \in B} \left( \bigvee_{\substack{g, h \\ (b, g, h) \in T}} \left( X_{bgh} \wedge \bigwedge_{\substack{g', h' \\ (b, g', h') \in T \\ g \neq g' \wedge h \neq h'}} \neg X_{bg'h'} \right) \right), & \text{if every } b \in B \text{ appears in a triplet of } T; \\ \text{false,} & \text{otherwise.} \end{cases}$$

$$E_G = \begin{cases} \bigwedge_{g \in G} \left( \bigvee_{\substack{b, h \\ (b, g, h) \in T}} \left( X_{bgh} \wedge \bigwedge_{\substack{b', h' \\ (b', g, h') \in T \\ b \neq b' \wedge h \neq h'}} \neg X_{b'gh'} \right) \right), & \text{if every } g \in G \text{ appears in a triplet of } T; \\ \text{false,} & \text{otherwise.} \end{cases}$$

$$E_H = \begin{cases} \bigwedge_{h \in H} \left( \bigvee_{\substack{b, g \\ (b, g, h) \in T}} \left( X_{bgh} \wedge \bigwedge_{\substack{b', g' \\ (b', g', h) \in T \\ b \neq b' \wedge g \neq g'}} \neg X_{bg'h'} \right) \right), & \text{if every } h \in H \text{ appears in a triplet of } T; \\ \text{false,} & \text{otherwise.} \end{cases}$$

The signs of conjunction and disjunction can be used as  $n$ -ary operators. Since both the conjunction and disjunction are associative operations, their  $n$ -ary forms are well-defined. For this reason it is allowed to use these operations as  $\wedge$  and  $\vee$  in the way we used above. These formulas also can be built in a polynomial complexity.

The interpretation of the above formulas is as follows:

1.  $E_B$  is *true* if and only if there are no two distinct triplets in the set  $U$  such that the two triplets are equal in their first elements and  $b \in B$  is included in a triplet.

To show this let  $b \in B$ ,  $g_1, g_2 \in G$  and  $h_1, h_2 \in H$  such that  $g_1 \neq g_2$  or  $h_1 \neq h_2$ . Suppose we have  $(b, g_1, h_1), (b, g_2, h_2) \in U$ , where  $U$  is a solution of the problem. In each of the  $m$  direct subformulas of  $E_B$  there is exactly one Boolean variable which is not negative (i.e., not prefixed by the operator  $\neg$ ). So, if both  $(b, g_1, h_1)$  and  $(b, g_2, h_2)$  are in  $U$ , then both  $X_{bg_1h_1}$  and  $X_{bg_2h_2}$  are true and so the whole subformula is false, thus  $E_B$  is false. (The presented atomic conjunctions are true if and only if exactly one of the used  $X_{bgh}$ 's has value *true*.)

On the other side, since  $E_B$  is a conjunction, it can only be false if one of its conjunctive subformulas is false or there exists a  $b \in B$  such that it doesn't appear in any triplets of  $T$ . In the first case, from the definition of  $E_B$  it follows that such a subformula is false if and only if there exists a  $b \in B$  such that  $(b, g', h')$  and  $(b, g'', h'')$  ( $g', g'' \in G$ ,  $h', h'' \in H$ ,  $g' \neq g''$  or  $h' \neq h''$ ) are in  $U$ , that is, the problem instance has no solution.

2.  $E_G$  is *true* if and only if there are no two triplets in the set  $U$  such that the two triplets are equal in their second elements and every  $g \in G$  is included in a triplet.

The correctness of this statement can be seen similarly as above.

3.  $E_H$  is *true* if and only if there are no two triplets in the set  $U$  such that the two triplets are equal in their third elements and every  $h \in H$  is included in a triplet.

This statement can be proved in the same way as in the previous cases.

If the above formulas are satisfiable in the same evaluation, then the actual problem instance has a solution. That is, the logical formula assigned to the instance is the following:

$$E_B \wedge E_G \wedge E_H \tag{1}$$

If the formula (1) is satisfiable, then the problem is solvable and a solution can be extracted from the satisfying evaluation.

The formula can be built in complexity  $O(m^5)$ .

## 4.2 Constructing a solution

The solution can be divided into two phases. In the first phase we construct a logical formula and check if it is satisfiable. If it is satisfiable, then in the next phase we can construct a solution.

**Solution to SAT** The solution of the SAT problem using interval-valued computations can be found in [Nagy 2005b]. We recall the main steps.

Let  $n$  be the number of the Boolean variables in the formula. Let the interval-value assigned to the  $i$ -th variable be given in the following form

$$A_i = \bigcup_{j=0}^{2^{i-1}-1} \left[ \frac{2j}{2^i}, \frac{2j+1}{2^i} \right), \quad 1 \leq i \leq n.$$

These values can be computed in a linear way (with respect to number of variables). One can easily construct the above interval-values, for instance, using the product operator:

$$A_1 = \left[ 0, \frac{1}{2} \right) \quad \text{and} \quad A_{i+1} = (A_i * A_1) \vee (\neg A_i * A_1).$$

After constructing the  $n$  interval-values the logical expression (the given formula) must be evaluated using these interval-values and the definition of the Boolean operators on the interval-values. This is also a linear computations (with respect to the length of the formula). If the result interval-value is non-empty, then the formula is satisfiable. Every non-empty interval-value contains a point  $x \in [0, 1]$ . One can find a satisfying evaluation by evaluating the logical variables according to the characteristic function of their interval-values at  $x$ .

**Constructing the solution to Tripartite Matching** Through the description of the algorithm we are referring to the notion of characteristic functions of interval-values.

If the actual problem instance is solvable, then the following algorithm builds a solution:

1. let  $U$  be the empty set.
2. let  $x$  be a point of the result interval-value (which satisfies all the formulas) where its characteristic function is *true*.
3. for all  $X_{bgh}$ : if the characteristic function of the interval assigned to  $X_{bgh}$  on  $x$  is *true* (i.e.,  $c_{bgh}(x) = \text{true}$ , where  $c_{bgh}$  is the characteristic function of the appropriate interval value) then  $U = U \cup (b, g, h)$ ,  $b \in B$ ,  $g \in G$  and  $h \in H$ .

The set  $U$  is a solution of the Tripartite Matching problem instance.

## 5 Example

Let the sets be the following:

$$B = \{b_1, b_2\}, \quad G = \{g_1, g_2\} \text{ and } H = \{h_1, h_2\},$$

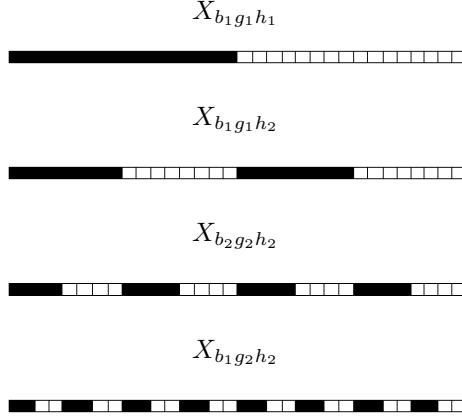
and define the relation  $T$  on these sets as follows:

$$T \subseteq B \times G \times H$$

and

$$T = \{(b_1, g_1, h_1), (b_1, g_1, h_2), (b_2, g_2, h_2), (b_1, g_2, h_2), (b_2, g_1, h_2)\}$$

Now we define five Boolean variables,  $X_{b_1g_1h_1}$ ,  $X_{b_1g_1h_2}$ ,  $X_{b_2g_2h_2}$ ,  $X_{b_1g_2h_2}$  and  $X_{b_2g_1h_2}$ , one for each triplet in  $T$ . For each variable we construct an appropriate interval-value. These values are the following:



and

$$X_{b_2g_1h_2}$$



We build the logical formulas:

$$\begin{aligned} E_B &= ((X_{b_1g_1h_1} \wedge \neg X_{b_1g_1h_2} \wedge \neg X_{b_1g_2h_2}) \vee \\ &\quad \vee (\neg X_{b_1g_1h_1} \wedge X_{b_1g_1h_2} \wedge \neg X_{b_1g_2h_2}) \vee \\ &\quad \vee (\neg X_{b_1g_1h_1} \wedge \neg X_{b_1g_1h_2} \wedge X_{b_1g_2h_2})) \wedge \\ &\quad \wedge ((X_{b_2g_2h_2} \wedge \neg X_{b_2g_1h_2}) \vee \\ &\quad \vee (\neg X_{b_2g_2h_2} \wedge X_{b_2g_1h_2})) \\ E_G &= ((X_{b_1g_1h_1} \wedge \neg X_{b_1g_1h_2} \wedge \neg X_{b_2g_1h_2}) \vee \\ &\quad \vee (\neg X_{b_1g_1h_1} \wedge X_{b_1g_1h_2} \wedge \neg X_{b_2g_1h_2}) \vee \\ &\quad \vee (\neg X_{b_1g_1h_1} \wedge \neg X_{b_1g_1h_2} \wedge X_{b_2g_1h_2})) \wedge \\ &\quad \wedge ((X_{b_2g_2h_2} \wedge \neg X_{b_1g_2h_2}) \vee \\ &\quad \vee (\neg X_{b_2g_2h_2} \wedge X_{b_1g_2h_2})) \\ E_H &= X_{b_1g_1h_1} \wedge \\ &\quad \wedge ((X_{b_1g_1h_2} \wedge \neg X_{b_1g_2h_2} \wedge \neg X_{b_2g_2h_2} \wedge \neg X_{b_2g_1h_2}) \vee \\ &\quad \vee (\neg X_{b_1g_1h_2} \wedge X_{b_1g_2h_2} \wedge \neg X_{b_2g_2h_2} \wedge \neg X_{b_2g_1h_2}) \vee \\ &\quad \vee (\neg X_{b_1g_1h_2} \wedge \neg X_{b_1g_2h_2} \wedge X_{b_2g_2h_2} \wedge \neg X_{b_2g_1h_2}) \vee \\ &\quad \vee (\neg X_{b_1g_1h_2} \wedge \neg X_{b_1g_2h_2} \wedge \neg X_{b_2g_2h_2} \wedge X_{b_2g_1h_2})) \end{aligned}$$

Based on the interval-values assigned to the Boolean variables, we calculate the interval-values corresponding to the formulas  $E_B$ ,  $E_G$  and  $E_H$ .

For example, to calculate the interval-value of  $E_B$  we calculate

$$(X_{b_1g_1h_1} \wedge \neg X_{b_1g_1h_2} \wedge \neg X_{b_1g_2h_2})$$

then compute

$$\neg X_{b_1g_1h_1} \wedge X_{b_1g_1h_2} \wedge \neg X_{b_1g_2h_2}$$

and calculate

$$\neg X_{b_1g_1h_1} \wedge \neg X_{b_1g_1h_2} \wedge X_{b_1g_2h_2}$$

Now, if we apply disjunction on the above values, then we get first part of  $E_B$ :

Then, as above, we compute the second part of the formula  $E_B$ :

and applying conjunction on these two subformulas,  $E_B$  is obtained:

Similarly, the computation of  $E_G$  will result:

and for  $E_H$  we get:

Now, by computing  $E_B \wedge E_G \wedge E_H$  the result is:

As we can see the formula is satisfiable. Thus, there exists a set  $U \subseteq T$  with the properties mentioned in Definition 1. Evaluating the variables with the result interval-value will give us the following:

$$X_{b_1g_1h_1} = \text{true}, \quad X_{b_2g_2h_2} = \text{true}, \quad X_{b_1g_1h_2} = \text{false}, \quad X_{b_1g_2h_2} = \text{false}$$

$$\text{and } X_{b_2g_1h_2} = \text{false}$$

Therefore the set  $U$  is the following:

$$U = \{(b_1, g_1, h_1), (b_2, g_2, h_2)\}$$

One can easily check that  $U$  is a solution of the problem.

## 6 Conclusions

It was known ([Nagy 2005b]) that SAT can be solved in a linear interval-valued computation. Since the computing model is based on logic, it was natural to solve problems with strong relation to logic. In this paper another NP-complete problem was considered; the Tripartite Matching proved to be solvable by an algorithm of a polynomial length in the frame of interval-valued computation. This paper also proves that not only logical problems can be solved efficiently

by the interval-valued computations. The method presented here is easily extendible to solve the simple  $d$ -partite matching problem. The solution will also be polynomial for any value of  $d \in \mathbb{N}$ :  $O(m^{2d-1})$ .

We note here that by the method presented in [Jackson–Sheridan 2005] the logical formulas can be transformed to conjunctive normal form in complexity  $O(m^3)$ , in this way polynomial solution exists to the Tripartite Matching problem in other computing paradigms in which the SAT of formulas of conjunctive normal form is solvable in polynomial time.

## References

- [Adleman 1994] Adleman, L.: Molecular Computation of Solutions To Combinatorial Problems, *Science* 266 (1994) 1021–1024.
- [Jackson–Sheridan 2005] Jackson, P. and Sheridan, D.: Clause Form Conversions for Boolean Circuits. in: *SAT 2004*, LNCS 3542, Springer, (2005) pp. 183–198.
- [Karp 1972] Karp, R.: Complexity of Computer Computations. PlenumPress, New-York, (1972).
- [Loos–Nagy 2007] Loos, R. and Nagy, B.: Parallelism in DNA and Membrane Computing, in: “Computability in Europe 2007: Computation and Logic in the Real World”, (S. B. Cooper, T. F. Kent, B. Löwe, A. Sorbi eds.), Siena, Italy, pp. 283–287.
- [Nagy 2005a] Nagy, B.: A general fuzzy logic using intervals, in: Proceedings of the 6th International Symposium of Hungarian Researchers on Computational Intelligence, Budapest, Hungary, (2005), pp. 613–624.
- [Nagy 2005b] Nagy, B.: An Interval-valued Computing Device, in: “Computability in Europe 2005: New Computational Paradigms”, (S. B. Cooper, B. Löwe, L. Torenvliet eds.), ILLC Publications X-2005-01, Amsterdam, pp. 166–177.
- [Nagy 2005c] Nagy, B.: Új elvű számítógépek. (New computational paradigms) Lecture Notes, University of Debrecen, Hungary, (2005).
- [Nagy–Vályi 2008] Nagy, B. and Vályi, S.: Interval-valued computations and their connection with **PSPACE**, *Theoretical Computer Science* 394 (2008) 208–222.
- [Papadimitriou 1994] Papadimitriou, C. H.: Computational Complexity, Addison-Wesley, (1994).
- [Paun 2002] Paun, Gh.: Membrane Computing. An Introduction, Springer-Verlag, Berlin (2002).
- [Shatsky–Shulman–Peleg–Nussinov–Wolfson 2005a] Shatsky, M.; Shulman–Peleg, A.; Nussinov, R. and Wolfson, H.J.: Recognition of Binding Patterns Common to a Set of Protein Structures, in: *Research in Computational Molecular Biology*, LNCS 3500, Springer, (2005) pp. 440–455.
- [Shatsky–Shulman–Peleg–Nussinov–Wolfson 2005b] Shatsky, M.; Shulman–Peleg, A.; Nussinov, R. and Wolfson, H.J.: MAPPIS: Multiple 3D Alignment of Protein-Protein Interfaces, in: *Computational Life Sciences*, LNCS 3695, Springer, (2005) pp. 91–103.
- [Singh–Xu–Berger 2008] Singh, R.; Xu, J. and Berger, B.: Global Alignment of Multiple Protein Interaction Networks, in: *Proceedings 13th Pacific Symposium on Biocomputing* (2008) pp. 303–314.
- [Tanenbaum 1984] Tanenbaum, A. S.: Structured Computer Organization, Prentice-Hall, 1984.
- [Woods–Naughton 2005] Woods, D. and Naughton, T.: An optical model of computation, *Theoretical Computer Science* 334 (2005) 227–258.

# Probabilistic Machines vs. Relativized Computation

Hayato Takahashi<sup>1,2\*</sup> and Kazuyuki Aihara<sup>1,2\*\*</sup>

<sup>1</sup> Aihara complexity modelling project, ERATO, JST,

<sup>2</sup> Institute of Industrial Science, University of Tokyo  
4-6-1 Komaba, Meguro-ku, Tokyo 153-8505, Japan.

**Abstract.** Computational power of probabilistic machines (Turing machines with random input) is studied. In particular, we extend the classical result of K. de Leeuw et al. (1956) to various distributions then, apply our result to analyze the computational power of analog machines.

**Key words:** probabilistic machine, relativized computation, analog computation, ergodic process

## 1 Introduction

There are two kinds of computational models of noisy environment: Turing machine with random input [1] and machine models that allow the transition of states to be stochastic, e.g., probabilistic (analog) automata [2, 3] and computable-stochastic machine [1]. In this paper, we call the former model (Turing machines with random input) as *probabilistic machine*. The difference of these two models is that the former is a model of Turing machine with external random input and the transition of states is deterministic; on the other hand the latter model does not have external random input but allows the transition of states to be stochastic (not deterministic). In [2], it is shown that a probabilistic automaton (the number of states is finite) is equivalent to a finite automaton under the condition isolated cut-off. In [1], a machine is called computable-stochastic if it has countably many states and the probability of the transition is computable, then it is shown that the class that is computable by computable-stochastic machines with positive probability is equivalent to that of Turing machines. Thus under these conditions, the former model includes the latter model.

In this paper, we study the former models (probabilistic machines). In [1], they showed that if the distribution of random input is Bernoulli with parameter  $p$ , then a set is enumerable with positive probability from random inputs iff it is enumerable with oracle  $p$ . In Section 1,2, we extend the above result to various distributions and show some examples and counter-examples. We also study the class that probabilistic machines can compute with probability one

---

\* Present address: The Institute of Statistical mathematics, 4-6-7 Minami-Azabu, Minato-ku, Tokyo 106-8569, Japan. [takahasi@ism.ac.jp](mailto:takahasi@ism.ac.jp)

\*\* E-mail: [aihara@sat.t.u-tokyo.ac.jp](mailto:aihara@sat.t.u-tokyo.ac.jp)

and show an interesting example (Sturmian sequence). In Section 3, we propose analog machine models with noise and analyze their computational power. In particular, we show that it is possible to increase its computational power by simple feedback.

Let  $\mathbb{N}$  and  $\mathbb{Q}$  be the set of natural numbers and the set of rational numbers, respectively. Let  $A$  be a finite alphabet; and let  $A^*$  and  $A^\infty$  be the set of finite strings and the set of infinite sequences of  $A$ , respectively. For  $x, y \in A^*$ ,  $xy$  is the concatenation of  $x$  and  $y$ , and  $x \sqsubseteq y$  if  $x$  is a prefix of  $y$ . For  $x \in A^*$ ,  $|x|$  is the length of  $x$  and  $\bar{x} := 0^{|x|}1x$ . Let  $q : \mathbb{Q} \rightarrow A^*$  be a computable bijection. Throughout the paper,  $\xi$  and lowercase bold letters (e.g.  $\mathbf{p}$ ) denote elements of  $A^\infty$ .

We say that  $S$  is  $\xi$ -enumerable if  $S$  is recursively enumerable (r.e.) relative to the set  $\{(i, \xi_i) | i \in \mathbb{N}\}$ , where  $\xi = \xi_1 \xi_2 \dots, \forall i \xi_i \in A$ , and  $\xi'$  is called  $\xi$ -computable if  $\{(i, \xi'_i) | i \in \mathbb{N}\}$  is  $\xi$ -enumerable. Let  $M \subseteq N \times A^*$  be a r.e. set. For  $x \in A^*$ , we write  $M(x) := \{n | y \sqsubseteq x, (n, y) \in M\}$  and  $M(\xi) := \cup_{x \sqsubseteq \xi} M(x)$ . Then  $S$  is  $\xi$ -enumerable iff there is a r.e.  $M$  such that  $S = M(\xi)$ . We write the class of  $\xi$ -enumerable sets as  $\mathcal{M}(\xi)$ , i.e.,

$$\mathcal{M}(\xi) = \{S \subseteq \mathbb{N} | \exists M, S = M(\xi)\}.$$

Throughout the paper,  $M$  denotes a r.e. set of  $N \times A^*$ .

Let  $P$  be a probability on  $(\mathcal{B}, A^\infty)$ , where  $\mathcal{B}$  is the Borel-field generated by the cylinder sets  $\Delta(x) = \{x\xi | \xi \in A^\infty\}$ ,  $x \in A^*$ . For  $x \in A^*$ , let  $P(x) := P(\Delta(x))$ , then  $\forall x \in A^* P(x) = \sum_{y \in A} P(xy)$ .  $P$  is called computable relative to  $\xi$  ( $\xi$ -computable) if there is a  $\xi$ -computable total function  $p : A^* \times \mathbb{N} \rightarrow \mathbb{Q}$  such that

$$\forall x, k, |P(x) - p(x, k)| < 1/k. \quad (1)$$

Let  $\mathbf{p} := \overline{q(p(n(1)))} \overline{q(p(n(2)))} \dots \in A^\infty$ , where  $n : \mathbb{N} \rightarrow A^* \times \mathbb{N}$  is a computable bijection. Then the graph of  $p$  is  $\mathbf{p}$ -enumerable and vice versa, so that a set  $S$  is r.e. relative to  $p$  iff it is  $\mathbf{p}$ -enumerable.  $\mathbf{p}$  is called *approximation* of  $P$  if  $p$  satisfies (1). Note that 1) an approximation of  $P$  is not unique and 2)  $P$  is computable relative to its approximation. Let  $A_P$  be the set of approximations of  $P$  and

$$\mathcal{M}(A_P) := \cap_{\mathbf{p} \in A_P} \mathcal{M}(\mathbf{p}).$$

Let  $M^{-1}(S) := \{\xi \in A^\infty | M(\xi) = S\}$  for  $S \subseteq \mathbb{N}$ . Note that  $M^{-1}(S)$  is measurable for all  $S$ .  $S$  is called  $P$ -enumerable if there is a r.e.  $M$  such that  $P(M^{-1}(S)) > 0$ <sup>3</sup>. We write the class of  $P$ -enumerable sets as

$$\mathcal{M}(P) := \{S \subseteq \mathbb{N} | \exists M P(M^{-1}(S)) > 0\}.$$

$P$  is called *effectively estimated* if there are computable functions  $e : A \times A^* \rightarrow \mathbb{Q}$  and  $r : \mathbb{N} \times \mathbb{N} \times A^* \rightarrow \mathbb{Q}$  such that

$$\forall x \in A^* \forall k \in \mathbb{N}, P(|P(x) - e(x, X_1, X_2, \dots, X_n)| > 1/k) < r(n, k, x) \quad (2)$$

and  $r(n, k, x) \rightarrow 0$  as  $n \rightarrow \infty$  for each  $k, x$ . The following is the theorem of K. de Leeuw et al.[1].

---

<sup>3</sup> In [1], it is called strongly enumerable.

**Theorem 1 (K. de Leeuw et al.[1]).** *Let  $P$  be a probability on  $A^\infty$ .*

- a)  $\mathcal{M}(P) \subseteq \mathcal{M}(A_P)$ .
- b) *If  $P$  is effectively estimated then,  $\mathcal{M}(P) = \mathcal{M}(A_P)$ .*

Note that  $\mathcal{M}(P)$  always includes the class of r.e. sets (ignore the random input), so that part a) implies that if  $P$  is computable,  $\mathcal{M}(P)$  is the class of r.e. sets.

For example, let  $P$  be the Bernoulli process with parameter  $p$ , i.e.,

$$P(x_1 x_2 \cdots x_n) = p^{\sum x_i} (1-p)^{n-\sum x_i}, \quad (3)$$

where  $p \in [0, 1]$ . Let  $p = 0.p_1 p_2 \cdots$  be the representation of base  $A$  and  $\mathbf{p} := p_1 p_2 \cdots \in A^\infty$ . Then  $P$  is  $\mathbf{p}$ -computable. Since  $p = P(1)$ , we see that for any approximation  $\mathbf{p}'$  of  $P$ ,  $\mathbf{p}$  is  $\mathbf{p}'$ -computable. Conversely, from (3), there is an approximation  $\mathbf{p}'$  such that  $\mathbf{p}'$  is  $\mathbf{p}$ -computable. Therefore  $\mathcal{M}(A_P) = \mathcal{M}(\mathbf{p})$ . Since  $P$  is effectively estimated, we have  $\mathcal{M}(P) = \mathcal{M}(A_P) = \mathcal{M}(\mathbf{p})$ .

In general, the converse of a) of Theorem 1 does not hold. For example, let  $p \in [0, 1]$  be a non-computable real and  $P(100\cdots) = p$  and  $P(00\cdots) = 1-p$ , where  $100\cdots$  consists of 0s following 1 and  $00\cdots$  consists of 0s. Then possible sequences generated by  $P$  are  $100\cdots$  and  $00\cdots$ , so that  $\mathcal{M}(P)$  coincides with the class of r.e. sets. On the other hand  $A_P$  consists of approximations of non-computable  $p$ , we see  $\mathcal{M}(P) \neq \mathcal{M}(A_P)$ . In the same way, if  $P(11\cdots) = p$  and  $P(00\cdots) = 1-p$  and  $p$  is not computable, then  $\mathcal{M}(P) \neq \mathcal{M}(A_P)$ . Note that the first example is not stationary and the second one is stationary but not ergodic.

*Example 1.* If  $P$  is ergodic, by ergodic theorem, there is a function  $r$  such that  $r(n, k, x) \rightarrow 0$  as  $n \rightarrow \infty$  for each  $k, x$  and

$$\forall x \in A^* \forall k \in \mathbb{N}, P(|P(x) - \sum_{i=1}^{n-|x|+1} I_{X_i^{i+|x|-1}=x}/n| > 1/k) < r(n, k, x), \quad (4)$$

where  $I$  is the indicator function. If  $r$  is computable, then  $P$  is effectively estimated and  $\mathcal{M}(P) = \mathcal{M}(A_P)$ . For example, in case of ergodic Markov processes,  $r$  decreases with exponential rate so that it is computable. However it is possible that  $r$  is not computable. Indeed, in pp.171 [4], it is shown that for any given decreasing function  $r$ , there is an ergodic process that satisfies

$$\forall n, P(|P(1) - \sum_{i=1}^n I_{X_i=1}/n| \geq 1/2) > r(n).$$

In particular if  $r$  is chosen such that  $r$  decreases to 0 asymptotically slower than any computable function then  $P$  is not effectively estimated from sample mean. For a similar example for a stationary process (not ergodic), see [5]. Note that this fact does not imply that  $\mathcal{M}(P) \neq \mathcal{M}(A_P)$  for that  $P$ . The authors of the paper do not know whether  $\mathcal{M}(P) = \mathcal{M}(A_P)$  for all ergodic processes or not. We study the weakened form of the equivalence in the next section.

In order to study the converse of a) of Theorem 1 for arbitrary distributions, we introduce some notations. Let  $x_i^j := x_i x_{i+1} \cdots x_j$  for  $i \leq j$ , and  $f : \mathbb{N} \rightarrow \mathbb{N}$ .  $P'$  is called independent blocking process of  $(P, f)$  if

$$P'(x_1^n) = P(x_1^{f(1)}) P(x_{1+f(1)}^{f(1)+f(2)}) \cdots P(x_{1+f(1)+\cdots+f(k)}^n) \quad (5)$$

for  $x_1^n \in A^*$ , where  $k$  is the number such that  $1 + \sum_i^k f(i) \leq n \leq \sum_i^{k+1} f(i)$ .  $P'$  is a probability on  $A^\infty$ .

**Theorem 2.** *Let  $P$  be a probability on  $A^\infty$  and  $f$  be an unbounded computable function. Let  $P'$  be the independent blocking process of  $(P, f)$ . Then  $\mathcal{M}(P') = \mathcal{M}(A_{P'}) = \mathcal{M}(A_P)$ .*

## 2 Ergodic process

First we study a weakened form of Theorem 1 for ergodic process. Let

$$\hat{r}(n, k, x) := P(|P(x) - \sum_{i=1}^{n-|x|+1} I_{X_i^{i+|x|-1}=x}/n| > 1/k), \quad (6)$$

for  $x \in A^*$ ,  $k \in \mathbb{N}$ . Let  $r : \mathbb{N} \times \mathbb{N} \times A^* \times \mathbb{N} \rightarrow \mathbb{Q}$  be a total function such that  $\forall n, k, x, t, |r(n, k, x, t) - \hat{r}(n, k, x)| < 1/t$ . Let  $m : \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N} \times A^* \times \mathbb{N}$  be a computable bijection, and set  $\mathbf{r} := q(r(m(1))) \overline{q(r(m(2)))} \cdots$ . Then the graph of  $r$  is  $\mathbf{r}$ -enumerable and vice versa.  $\mathbf{r}$  is called approximation of  $\hat{r}$ . Let  $M_{\mathbf{r}} \subseteq \mathbb{N} \times A^*$  be  $\mathbf{r}$ -enumerable. Let  $A_{\hat{r}}$  be the set of approximations of  $\hat{r}$  and

$$\mathcal{M}(P \cup A_{\hat{r}}) := \cap_{\mathbf{r} \in A_{\hat{r}}} \{S \subseteq \mathbb{N} \mid \exists M_{\mathbf{r}} \ P(M_{\mathbf{r}}^{-1}(S)) > 0\}.$$

Roughly speaking,  $\mathcal{M}(P \cup A_{\hat{r}})$  is the class of subsets of natural numbers that is computable with positive probability from  $\hat{r}$  and random inputs.

**Corollary 1.** *If  $P$  is ergodic,  $\mathcal{M}(P \cup A_{\hat{r}}) = \mathcal{M}(A_P)$ .*

Next, we study the class of subsets of natural numbers that probabilistic machines can compute with probability one. Let

$$\mathcal{M}_1(P) := \{S \subseteq \mathbb{N} \mid \exists M \ P(M^{-1}(S)) = 1\},$$

$$S_P := \{x \mid P(x) > 0\},$$

and  $\mathcal{M}(S_P)$  be the class of r.e. sets relative to  $S_P$ .

**Theorem 3.** *If  $P$  is ergodic,  $\mathcal{M}_1(P) = \mathcal{M}(S_P)$ .*

*Remark 1.* Let  $\xi'$  be the shifted sequence of  $\xi$ , i.e.,  $\xi'_i = \xi_{i+1}$ .  $M$  is called shift invariant if  $M(\xi) = M(\xi')$  for almost all  $\xi$  with respect to  $P$ . Then for a shift invariant  $M$ , the set  $\{\xi \mid M(\xi) = S\}$  is shift invariant with respect to  $P$ , so that if  $P$  is ergodic, we have  $P(M^{-1}(S)) > 0 \Leftrightarrow P(M^{-1}(S)) = 1$ , and  $\mathcal{M}_1(P) = \{S \mid \exists \text{ shift invariant } M, P(M^{-1}(S)) > 0\}$ . Therefore if 1)  $S_P$  is a r.e. set, 2)  $\mathcal{M}(P) = \mathcal{M}(A_P)$ , and 3)  $P$  is not computable, then  $M$  that computes an approximation of  $P$  with positive probability cannot be shift invariant.

For example if  $P$  is a finite state Markov process then  $S_P$  is a r.e. set, and hence  $\mathcal{M}_1(P)$  coincide with the class of r.e. sets. However, there is an example such that  $M_1(P)$  is beyond the class of r.e. sets.

*Example 2 (Sturmian sequence).* Let  $x_{n+1} = x_n + \theta \bmod 1$  for  $\theta \in [0, 1]$  and  $y_n := 0$  if  $x_n \in [0, \theta)$  else 1. If the initial distribution of  $x_0$  is uniform then  $y_0, y_1, \dots$  is an ergodic process. Let  $P_\theta$  be such the process. Then  $P_\theta$  is computable iff  $\theta$  is computable. In [6, 7], it is shown that there are a computable function  $e$  and a computable decreasing (to 0) function  $r$  such that  $|e(x) - \theta| \leq r(|x|)$ , for all  $x$  generated by  $P_\theta$ . Therefore we have  $\mathcal{M}(P_\theta) = \mathcal{M}(A_{P_\theta}) = \mathcal{M}_1(P_\theta) = \mathcal{M}(S_{P_\theta})$ .

### 3 Analog machines with noise

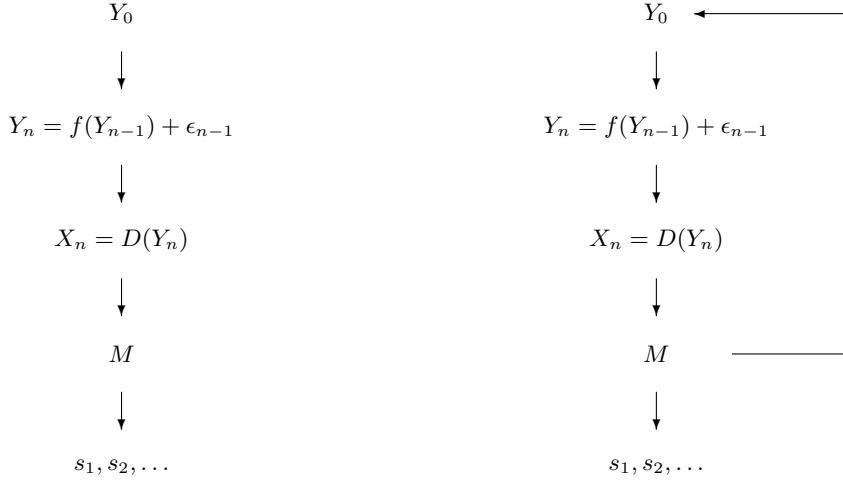
Certain kind of analog machines with noise can be modeled as follows (Model A):

$$\begin{aligned} f : [0, 1] &\rightarrow [0, 1], \quad Y_n := f(Y_{n-1}) + \epsilon_{n-1} \\ D : \mathbb{R} &\rightarrow A, \quad X_n := D(Y_n) \\ &\cdots X_2 X_1 \rightarrow M \rightarrow s_1 s_2 \cdots, \end{aligned}$$

where  $\{\epsilon_n\}$  are i.i.d. random variables (noise) and  $S = \{s_1, s_2, \dots\}$  is an output of the machine. Then  $X_1, X_2, \dots$  are stochastic process on  $A^\infty$ , and if the distributions of  $Y_0$  and  $\{\epsilon_n\}$  are given, the distribution of  $X_1, X_2, \dots$  is determined. This machine is a combination of a dynamical system with noise and a Turing machine, and it is considered to be a probabilistic machine, see Fig. 1 and Remark 2. Thus the results of the previous section can be applied to this machine. For example, if 1.  $f : [0, 1] \rightarrow [0, 1]$  is a polynomial with rational coefficient (e.g. logistic map), 2. distribution of  $Y_0$  is computable, 3. distributions of  $\{\epsilon_n\}$  are i.i.d. and computable, and 4.  $D(x) = 0$  if  $x \in [0, a)$  and  $D(x) = 1$  if  $x \in [a, 1]$ , where  $a$  is a rational number, then the distribution of  $X_1, X_2, \dots$  is computable. In such a case, Model A is equivalent to a Turing machine, which follows from the first statement of Theorem 1.

In general, let  $P$  be the probability that  $X_1, X_2, \dots$  obey and  $A_P$  be the set of approximations of  $P$ . Then  $\mathcal{M}(P) \subseteq \mathcal{M}(A_P)$ . However, unless  $P$  is effectively estimated, in general, the converse does not hold. In order to increase the computational power of Model A when  $\mathcal{M}(P) \neq \mathcal{M}(A_P)$ , let us consider the following transformation (Model B): Let  $f : \mathbb{N} \rightarrow \mathbb{N}$  be an unbounded computable function. If  $n = f(k) + 1$ ,  $k = 1, 2, \dots$ , then  $Y_n$  in Model A is reset i.e.,  $Y_n$  is drawn according to the initial distribution and it is independent from the past sequence, see Fig. 1. Model B corresponds to the independent blocking process  $P'$  of  $(P, f)$ , and we have  $\mathcal{M}(P') = \mathcal{M}(A_P)$  from Theorem 2.

*Remark 2.* Physically realizable analog machines always contain noise (uncertainty of initial value, noise effect of transition, and so on), and some of them are modeled by discrete-time dynamical systems with noise, see [8].  $\{Y_n\}$  (and

**Fig. 1.** machine models

$\{X_n\}$ ) in Fig. 1 is a discrete-time dynamical system with noise and it is considered to be an analog machine model with noise. Thus our model in Fig. 1 is a combination of noisy analog machine and Turing machine. In this paper, we are mainly interested in the distribution of  $\{X_n\}$ , and do not explore the dynamics of analog machines. For example, analog neural nets might be modeled by a transformation of a higher dimensional space, however it is not essential in this paper.

## 4 Proof

We need Lebesgue density theorem.

**Lemma 1 (Lebesgue).** *Let  $D \subseteq A^\infty$  be measurable and  $I_D$  be the characteristic function of  $D$ . Let  $f(x) := P(I_D \cap \Delta(x))/P(\Delta(x))$ . Then  $\lim_{x \rightarrow \xi} f(x) = I_D$  for almost all  $\xi$  with respect to  $P$ .*

Proof of Theorem 1)

a: Assume that  $S$  is  $P$ -enumerable. Then there is a r.e.  $M$  such that  $P(\{\xi \in A^\infty | M(\xi) = S\}) > 0$ . From Lemma 1, we have

$$\forall \epsilon > 0 \exists x \in A^* P(\{x\xi | M(x\xi) = S\})/P(x) > 1 - \epsilon.$$

Let  $x$  be a finite string that satisfies the above inequality for  $\epsilon = 1/2$ . We see that

$$s \in S \Leftrightarrow P(\{\Delta(xy) | s \in M(xy), y \in A^*\})/P(\Delta(x)) > 1/2.$$

Since  $\{(s, y) | s \in M(y)\}$  is recursively enumerable and  $P$  is computable relative to  $\mathbf{p}$ ,  $S$  is enumerable from the finite string  $x$  and  $\mathbf{p}$ .

b: By assumption,  $P$  is effectively estimated and (2) holds. Let  $e$  and  $r$  are computable functions in (2). Let  $N(x, k)$  be the least integer such that  $r(N(x, k), k, x) < 6/(\pi n^{-1}(x, k))^2$ , where  $n : \mathbb{N} \rightarrow A^* \times \mathbb{N}$  is a computable bijection. Since  $r$  is computable and  $r(n, k, x) \rightarrow 0$  as  $n \rightarrow \infty$  for each  $k, x$ , we see that  $N$  is computable for all  $x, k$ . Let

$$A(x, k) := \{\Delta(y) | |P(x) - e(x, y)| \leq 1/k, |y| = N(x, k), y \in A^*\},$$

where  $|y|$  is the length of  $y$ . Then  $P(A(x, k)^c) < r(N(x, k), k, x)$ , where  $A^c$  is the complement of  $A$ . Since  $\sum_{x, k} 6/(\pi n^{-1}(x, k))^2 = 1$ , we have

$P(\cap_{x, k} A(x, k)) \geq 1 - \sum_{x, k} P(A(x, k)^c) > 1 - \sum_{x, k} 6/(\pi n^{-1}(x, k))^2 = 0$ . Let  $p_{x, k} := e(x, X_1, X_2, \dots, X_{N(x, k)})$  for all  $x, k$  and  $\mathbf{p} := \overline{q(p_{n(1)})} \overline{q(p_{n(2)})} \dots$ . Then,  $\mathbf{p}$  is an approximation of  $P$  with positive probability and it is  $P$ -enumerable. Thus, if  $S \in \mathcal{M}(A_P)$  then  $S \in \mathcal{M}(P)$ .  $\square$

Proof of Theorem 2) First we show that  $\mathcal{M}(A_P) \subseteq \mathcal{M}(P')$ . In order to show this, it is enough to show that  $P$  is effectively estimated from random sampling according to  $P'$ . Let  $X_1, X_2, \dots$  be random variables according to  $P'$  and  $X_i^j := X_i \cdots X_j$  for  $i \leq j$ . Then blocks of random variables  $X_1^{f(1)}, X_{f(1)+1}^{f(1)+f(2)}, \dots$  are independent. Thus we see that  $P$  is effectively estimated from  $X_1^\infty$ . Formally let  $N(l) := \{k | f(k) \geq l\}$ . Since  $f$  is an unbounded computable function,  $N(l)$  is computable and infinite for all  $l$ . For each fixed  $l$ ,  $Y_k := X_{1+f(1)+\dots+f(k-1)+l}^{f(1)+\dots+f(k-1)+l}, k \in N(l)$  are independent, where  $f(0) = 0$ . Let  $N(l) = \{l_1, l_2, \dots\}$ ,  $l_1 < l_2 < \dots$  and  $e(x, X_1, X_2, \dots, X_n) := \sum_{i=1}^m I_{Y_{l_i}=x}/m$  for  $f(1) + \dots + f(l_m - 1) + l \leq n \leq f(1) + \dots + f(l_{m+1})$ . Then for  $x \in A^*, |x| = l$ , we have  $E(I_{Y_k=x}) = P(x)$ , where expectation is average with respect to  $P'$ , and from Chebyshev inequality, we have  $P'(|e(x, X_1, X_2, \dots, X_n) - P(x)| > 1/k) \leq k^2/4m$ . Since  $m$  is computable from  $n$  and  $m \rightarrow \infty$  as  $n \rightarrow \infty$ , we see that  $P$  is effectively estimated from random sampling according to  $P'$ . Thus we have  $\mathcal{M}(A_P) \subseteq \mathcal{M}(P')$ .  $\mathcal{M}(P') \subseteq \mathcal{M}(A_{P'})$  follows from Theorem 1. Since  $f$  is computable, by (5), we see that for any approximation  $\mathbf{p}$  of  $P$  there is an approximation  $\mathbf{p}'$  of  $P'$  such that  $\mathbf{p}'$  is  $\mathbf{p}$ -computable, and hence  $\mathcal{M}(A_{P'}) \subseteq \mathcal{M}(A_P)$ .  $\square$

Proof of Corollary 1) From (6), we see that  $\mathcal{M}(A_{\hat{r}}) \subseteq \mathcal{M}(A_P)$ . From Theorem 1 (a), we have  $\mathcal{M}(P \cup A_{\hat{r}}) \subseteq \mathcal{M}(A_P)$ . Since approximation of  $\hat{r}$  is given, as in the same way of proof of Theorem 1 (b), we can show the converse inclusion.  $\square$

Proof of Theorem 3) Let  $S_\xi$  be the set of strings that appear in  $\xi$ , i.e.,  $S_\xi = \{x | \exists y \exists \xi' yx\xi' = \xi\}$ . Since  $P$  is ergodic, by ergodic theorem, we have  $S_\xi = \{x | P(x) > 0\}$  almost surely. Thus  $\mathcal{M}_1(P) \supseteq \mathcal{M}(S_P)$ . Conversely if  $P(M^{-1}(S)) = 1$  then  $s \in S \Leftrightarrow \exists x, P(x) > 0, s \in M(x)$ . Thus we have  $\mathcal{M}_1(P) \subseteq \mathcal{M}(S_P)$ .  $\square$

## References

1. de Leeuw, K., Moore, E.F., Shannon, C.E., Shapiro, N.: Computability by probabilistic machines. In Shannon, C.E., McCarthy, J., eds.: Automata Studies. Princeton Univ. Press (1956) 183–212

2. Rabin, M.O.: Probabilistic automata. *Information and Control* **6** (1963) 230–245
3. Ben-Hur, A., Roitershtein, A., Siegelmann, H.T.: On probabilistic analog automata. *Theoret. Comput. Sci.* **320** (2004) 449–464
4. Shields, P.: The ergodic theory of discrete sample paths. AMS (1996)
5. V'yugin, V.V.: Ergodic theorems for individual random sequences. *Theoret. Comput. Sci.* **207** (1998) 343–361
6. Takahashi, H., Aihara, K.: Algorithmic analysis of irrational rotations in a single neuron model. *J. Complexity* **19** (2003) 132–152
7. Kamae, T., Takahashi, H.: Statistical problems related to irrational rotations. *Ann. Inst. Statist. Math.* **58** (2006) 573–593
8. Maass, W., Orponen, P.: On the effect of analog noise in discrete-time analog computations. *Neural Computation* **10** (1998) 1071–1095

# Quantum Query Algorithms for AND and OR Boolean Functions \*

Alina Vasilieva

Institute of Mathematics and Computer Science, University of Latvia  
Raina bulvaris 29, Riga, LV-1459, Latvia  
[alina.vasilieva@gmail.com](mailto:alina.vasilieva@gmail.com)

**Abstract.** Quantum algorithms can be analyzed in a query model to compute Boolean functions where input is given in a black box and the aim is to compute function value for arbitrary input using as few queries as possible. We concentrate on quantum query algorithm designing tasks in this paper. The main aim of the research was to find new efficient algorithms and develop general algorithm designing techniques. In this article we propose quantum query algorithm constructing methods. Given algorithms for the set of sub-functions, our methods use them to build a more complex one, based on algorithms described before. Methods are applicable to input algorithms with specific properties and preserve acceptable error probability and number of queries. Methods offer constructions for computing AND and OR kinds of Boolean functions.

## 1 Introduction

Let  $f(x_1, x_2, \dots, x_n) : \{0, 1\}^n \rightarrow \{0, 1\}$  be a Boolean function. We have studied the query model, where a black box contains the input  $(x_1, x_2, \dots, x_n)$  and can be accessed by questioning  $x_i$  values. The goal is to compute the value of the function. The complexity of a query algorithm is measured by number of questions it asks. The classical version of this model is known as *decision trees* [1]. Quantum algorithms can solve certain problems faster than classical algorithms. The best-known exact quantum algorithm was designed for XOR function with  $n/2$  questions vs.  $n$  questions required by classical algorithm [2, 3]. No analogous efficient algorithms exist for AND and OR Boolean functions.

Quantum query model differs from quantum circuit model [2, 5] and algorithm construction techniques for this model are less developed. The problem of quantum query algorithm construction is not that easy. Although there is a large amount of lower and upper bound estimations of quantum query algorithm complexity [2, 6, 7], examples of non-trivial and original quantum query algorithms are very few. Moreover, there is no special technique described to build a quantum query algorithm for an arbitrary function with complexity defined in advance.

In our work we have tried to develop general constructions and approaches for computing Boolean functions efficiently in a quantum query settings.

---

\* Research supported by the European Social Fund

## 2 Notation and Definitions

Let  $f(x_1, x_2, \dots, x_n) : \{0, 1\}^n \rightarrow \{0, 1\}$  be a Boolean function. We use  $\oplus$  to denote XOR operation. We use abbreviation QQA for "quantum query algorithm".

### 2.1 Quantum Computing

We apply the basic model of quantum computing. For more details see textbooks by Gruska [4] and Nielsen and Chuang [5]. An  $n$ -dimensional quantum pure state is a unit vector in a Hilbert space. Let  $|0\rangle, |1\rangle, \dots, |n-1\rangle$  be an orthonormal basis for  $\mathbb{C}^n$ . Then, any state can be expressed as  $|\psi\rangle = \sum_{i=0}^{n-1} a_i |i\rangle$  for some  $a_i \in \mathbb{C}$ . Since the norm of  $|\psi\rangle$  is 1, we have  $\sum_{i=0}^{n-1} |a_i|^2 = 1$ . States  $|0\rangle, |1\rangle, \dots, |n-1\rangle$  are called *basis states*. Any state of the form  $\sum_{i=0}^{n-1} a_i |i\rangle$  is called a *superposition* of basis states. The coefficient  $a_i$  is called an *amplitude* of  $|i\rangle$ . The state of a system can be changed by applying *unitary transformation*. Unitary transformation  $U$  is a linear transformation on  $\mathbb{C}^n$  that maps vector of unit norm to vector of unit norm. The simplest case of quantum *measurement* is used in our model. It is the full measurement in the computation basis. Performing this measurement on a state  $|\psi\rangle = a_0 |0\rangle + \dots + a_{n-1} |n-1\rangle$  gives the outcome  $i$  with probability  $|a_i|^2$ . The measurement changes the state of the system to  $|i\rangle$  and destroys the original state.

### 2.2 Quantum Query Model

Query algorithm is the model for computing Boolean functions. A black box contains the input  $(x_1, x_2, \dots, x_n)$  and can be accessed by questioning  $x_i$  values. Algorithm must be able to determine the value of a function correctly for arbitrary input. The complexity of the algorithm is measured by the number of queries to the black box. We consider computing Boolean functions in the quantum query model. For more details, see the survey by Ambainis [6] and textbooks by Gruska [4] and de Wolf [2]. A quantum computation with  $T$  queries is a sequence of unitary transformations:

$$U_0 \rightarrow Q_0 \rightarrow U_1 \rightarrow Q_1 \rightarrow \dots \rightarrow U_{T-1} \rightarrow Q_{T-1} \rightarrow U_T.$$

$U_i$ 's can be arbitrary unitary transformations that do not depend on the input bits.  $Q_i$ 's are query transformations. Computation starts in the state  $|\bar{0}\rangle$ . Then we apply  $U_0, Q_0, \dots, Q_{T-1}, U_T$  and measure the final state.

We use the following definition of query transformation: if input is a state  $|\psi\rangle = \sum_i a_i |i\rangle$ , then the output is  $|\phi\rangle = \sum_i (-1)^{x_k} a_i |i\rangle$ , where we can arbitrarily choose a variable assignment  $x_k$  for each basis state  $|i\rangle$ .

Each quantum basis state corresponds to the algorithm output. We assign a value of a function to each output. The probability of obtaining result  $j$  after executing an algorithm on input  $X$  equals the sum of squares of all amplitudes, which corresponds to outputs with value  $j$ .

Very convenient way of QQA representation is a graphical picture and we will use this style when describing our algorithms.

### 2.3 Query Algorithm Complexity

The complexity of a query algorithm is based on the number of questions it uses to determine the value of a function on worst-case input.

The *deterministic complexity* of a function  $f$ , denoted by  $D(f)$ , is the maximum number of questions that must be asked on any input by a deterministic algorithm for  $f$  [1].

The *sensitivity* of  $f$  on input  $(x_1, x_2, \dots, x_n)$  is the number of variables  $x_i$  with the following property:  $f(x_1, \dots, x_i, \dots, x_n) \neq f(x_1, \dots, 1 - x_i, \dots, x_n)$ . The sensitivity of  $f$  is the maximum sensitivity of all possible inputs. It has been proved that  $D(f) \geq s(f)$  [1].

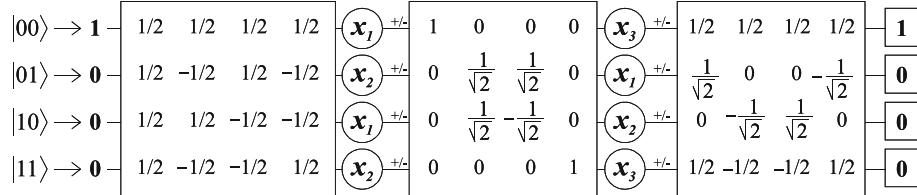
A quantum query algorithm *computes*  $f$  *exactly* if the output equals  $f(x)$  with a probability 1, for all  $x \in \{0, 1\}^n$ . Complexity is denoted by  $Q_E(f)$  [1].

A quantum query algorithm *computes*  $f$  *with bounded-error* if the output equals  $f(x)$  with probability  $p > 1/2$ , for all  $x \in \{0, 1\}^n$ . Complexity is denoted by  $Q_p(f)$  [1].

## 3 Basic Exact Quantum Query Algorithms

In this paper we will describe quantum query algorithm constructing methods, which use existing algorithms to construct more complex examples. To demonstrate methods we need at least few basic algorithms. The following exact QQAs were presented in [9] and we will use them as a base.

**Algorithm 1.** Exact QQA with 2 queries is presented in Fig. 1.



**Fig. 1.** Exact QQA for  $EQUALITY_3$  with two queries

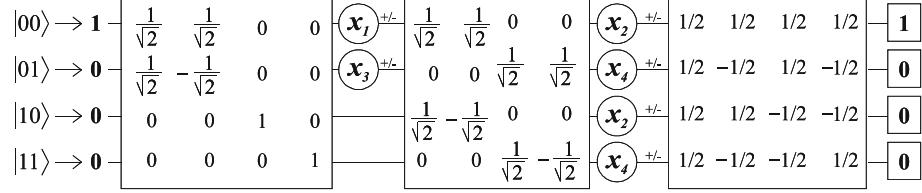
**Boolean function:**  $EQUALITY_3(X) = \neg(x_1 \oplus x_2) \wedge \neg(x_2 \oplus x_3)$ .

**Deterministic complexity:**  $D(EQUALITY_3) = 3$ , by sensitivity on any accepting input.

To make reader more familiar with QQA model we show the computation for  $X = 111$  (*bra* notation is used for convenience):

$$\begin{aligned} \langle \psi | &= (1, 0, 0, 0) U_0 Q_0 U_1 Q_1 U_2 = \left( \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2} \right) Q_0 U_1 Q_1 U_2 = \\ &= \left( -\frac{1}{2}, -\frac{1}{2}, -\frac{1}{2}, -\frac{1}{2} \right) U_1 Q_1 U_2 = \left( -\frac{1}{2}, -\frac{1}{\sqrt{2}}, 0, -\frac{1}{2} \right) Q_1 U_2 = \left( \frac{1}{2}, \frac{1}{\sqrt{2}}, 0, \frac{1}{2} \right) U_2 = \\ &= (\mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}) \Rightarrow [ACCEPT] \end{aligned}$$

**Algorithm 2.** Exact QQA for  $PAIR\_EQUALITY_4$  is presented in Fig. 2.



**Fig. 2.** Exact QQA for  $PAIR\_EQUALITY_4$  with two queries

**Boolean function:**  $PAIR\_EQUALITY_4(X) = \neg(x_1 \oplus x_2) \wedge \neg(x_3 \oplus x_4)$ .

**Deterministic complexity:**  $D(PAIR\_EQUALITY_4) = 4$ , by sensitivity on accepting input.

### 3.1 Exact Quantum Query Algorithm Classification

Two described exact QQAs have useful specific properties. They are designed in such a way that the final amplitude distribution satisfies certain condition on any possible input. To describe these properties we introduce QQA classification and define the following algorithm classes.

**Class 1.** *Exact QQA belongs to Class 1 IFF on any input system state before a measurement is such that for exactly one amplitude  $\alpha_i$  it is true that  $|\alpha_i|^2 = 1$ . For other amplitudes it is true that  $|\alpha_j|^2 = 0$ , for  $\forall j \neq i$ .*

Algorithm 1 and Algorithm 2 both belong to *Class 1*.

**Class 2+.** *Exact QQA belongs to Class 2+ IFF there is exactly one accepting basis state and on any input for its amplitude  $\alpha \in \mathbb{C}$  only two values are possible before the final measurement: either  $\alpha = 0$  or  $\alpha = 1$ .*

Algorithm 1 belongs to *Class 2+*.

**Class 2-.** *Exact QQA belongs to Class 2- IFF there is exactly one accepting basis state and on any input for its amplitude  $\alpha \in \mathbb{C}$  only two values are possible before the final measurement: either  $\alpha = 0$  or  $\alpha = -1$ .*

**Lemma 1.** *It is possible to transform algorithm that belongs to Class 2- to algorithm that belongs to Class 2+ by applying additional unitary transformation.*

*Proof.* Let's assume that we have QQA that belongs to *Class 2-* and  $k$  is the number of accepting output. To transform algorithm to *Class 2+* instance apply the following quantum gate:

$$U = (u_{ij}) = \begin{cases} 0, & \text{if } i \neq j \\ 1, & \text{if } i = j \neq k \\ -1, & \text{if } i = j = k \end{cases}$$

**Class 3.** *Exact QQA belongs to Class 3 IFF it belongs to Class 1; and there is exactly one accepting basis state; and on any input accepting state amplitude value before measurement is  $\alpha \in \{-1, 0, 1\}$ .*

Algorithm 1 and Algorithm 2 both belong to *Class 3*.

### 3.2 Algorithm Transformation Methods

QQA transformation methods were introduced in [9]. These methods are useful for enlarging a set of exactly computable functions. By applying transformation methods to Algorithm 1 and Algorithm 2 it is possible to obtain two sets:

- $QAlg3$  - which consists of exact QQAs that compute 3-variable Boolean functions using 2 queries.  $|QAlg3| = 8$ .
- $QAlg4$  - which consists of exact QQAs that compute 4-variable Boolean functions using 2 queries.  $|QAlg4| = 24$ .

$QAlg3$  contains two algorithms that belong to *Class 2+*, two algorithms that belong to *Class 2-* and four algorithms that belong to *Class 3*.  $QAlg4$  contains 12 algorithms that belong to *Class 3*. All these algorithms can be used as a base for constructions methods described in the next section.

We will use these results when calculating total number of algorithms that can be constructed by methods presented in this paper.

## 4 Algorithm Constructing Methods

In this section we will present two quantum query algorithm constructing methods. Each method requires explicitly specified exact QQAs on input, but as a result a bounded-error QQA for more complex function is constructed. The most important behavior is that methods do not increase overall algorithm complexity. No additional queries are necessary to compute complex function; input algorithms just have to be combined in a specific way. Methods maintain low quantum query complexity for complex function in comparison to increased deterministic complexity, thus enlarging a gap between classical and quantum complexities of an algorithm.

We offer a general constructions for computing AND and OR kinds of Boolean functions.

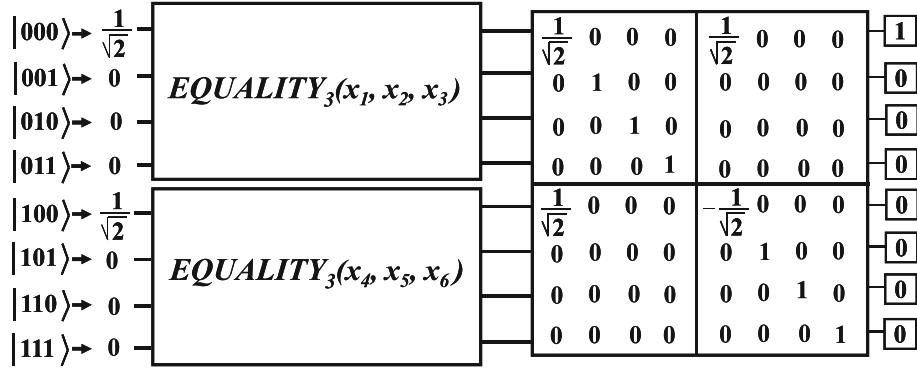
### 4.1 Bounded-error QQA for 6-Variable Function

We consider composite Boolean function, where two instances of  $EQUALITY_3$  are joined with logical AND operation:

$$EQUALITY_3^{\wedge 2}(x_1, \dots, x_6) = (\neg(x_1 \oplus x_2) \wedge \neg(x_2 \oplus x_3)) \wedge (\neg(x_4 \oplus x_5) \wedge \neg(x_5 \oplus x_6))$$

**Deterministic complexity.**  $D(EQ-LITY_3^{\wedge 2})=6$ , by sensitivity on  $X=111111$ .

**Algorithm 3.** Our approach in designing an algorithm for  $EQUALITY_3^{\wedge 2}$  is to employ quantum parallelism and superposition principle. We execute algorithm pattern defined by original algorithm for  $EQUALITY_3$  in parallel for both blocks of  $EQUALITY_3^{\wedge 2}$  variables. Finally, we apply additional quantum gate to correlate amplitude distribution. Algorithm flow is depicted in Fig. 3.



**Fig. 3.** Bounded-error QQA for  $EQUALITY_3^{\wedge 2}$

**Quantum complexity.** Algorithm 3 computes  $EQUALITY_3^{\wedge 2}$  using 2 queries with correct answer probability  $p = 3/4$ :  $Q_{3/4}(EQUALITY_3^{\wedge 2}) = 2$ .

#### 4.2 First Constructing Method - AND( $f_1, f_2$ )

It is possible to generalize approach demonstrated in the previous section. It is evident that complex algorithm behaviour is based on a structure of sub-algorithms. However, it does not depend on internal structure, but just on a properties of final amplitude distributions. Therefore, described construction is not limited to use algorithm for  $EQUALITY_3$  as a base, but can be applied to a particular class of exact QQAs. Generalized method is described in Table 1.

#### 4.3 Bounded-error QQA for 8-Variable Function

Next step is to realize similar approach for OR operation. This time we take Algorithm 2 for  $PAIR\_EQUALITY_4$  function as a base. We consider composite Boolean function, where two instances of  $PAIR\_EQUALITY_4$  are joined with OR operation:

$$PAIR\_EQLTY_4^\vee(X) = (\neg(x_1 \oplus x_2) \wedge \neg(x_3 \oplus x_4)) \vee (\neg(x_5 \oplus x_6) \wedge \neg(x_7 \oplus x_8))$$

We succeeded in constructing quantum algorithm for  $PAIR\_EQUALITY_4^\vee$  and this time algorithm structure is more complex than in AND operation case. However, this structure allows us to formulate generalized version of method, which will be applicable to QQAs that belong to *Class 3*.

**Table 1.** First Constructing Method - AND( $f_1, f_2$ )

<b>Input.</b> Two exact QQAs $Alg1$ and $Alg2$ that belong to <i>Class 2+</i> or <i>Class 2-</i> and compute Boolean functions $f_1(X_1)$ and $f_2(X_2)$ .
<b>Constructing steps.</b>
1. If any algorithm belongs to <i>Class 2-</i> , transform it to <i>Class 2+</i> algorithm as described in Lemma 1.
2. If $Alg1$ and $Alg2$ utilize quantum systems of different size, extend the smallest one with auxiliary space to obtain equal number of amplitudes. We denote the dimension of obtained Hilbert spaces with $m$ .
3. For new algorithm utilize a quantum system with $2m$ amplitudes.
4. Combine unitary transformations and queries of $Alg1$ and $Alg2$ in the following way: $U_i = \begin{pmatrix} U_i^1 & O \\ O & U_i^2 \end{pmatrix}$ , where $O$ 's are $m \times m$ zero-matrices, $U_i^1$ and $U_i^2$ are either unitary transformations or query transformations of $Alg1$ and $Alg2$ .
5. Start computation from the state: $\langle\psi  = (\frac{1}{\sqrt{2}}, 0, \dots, 0, \frac{1}{\sqrt{2}}, 0, \dots, 0)$ .
6. Before the final measurement apply additional unitary gate. Let us denote the positions of accepting outputs of $Alg1$ and $Alg2$ by $acc_1$ and $acc_2$ . Then the final gate is defined as follows:
$U = (u_{ij}) = \begin{cases} 1, & \text{if } (i = j) \text{ AND } (i \neq acc_1) \text{ AND } (i \neq (m + acc_2)) \\ 1/\sqrt{2}, & \text{if } (i = j = acc_1) \\ 1/\sqrt{2}, & \text{if } ((i = acc_1) \text{ AND } (j = (m + acc_2))) \\ & \text{OR } ((i = (m + acc_2)) \text{ AND } (j = acc_1)) \\ -1/\sqrt{2}, & \text{if } (i = j = (m + acc_2)) \\ 0, & \text{otherwise} \end{cases}$
7. Define as accepting output exactly one basis state $ acc_1\rangle$ .
<b>Output.</b> A bounded-error QQA $A$ for computing a function $F(X) = f_1(X_1) \wedge f_2(X_2)$ with probability $p = 3/4$ and complexity $Q_{3/4}(A) = \max(Q_E(Alg1), Q_E(Alg2))$ .

**Algorithm 4.** This time we use 4 qubit quantum system, so in total there are 16 amplitudes. First, we execute *PAIR-EQUALITY*<sub>4</sub> algorithm pattern in parallel on first 8 amplitudes, and then apply two additional quantum gates.

First gate  $U_{SWAP}$  swaps state amplitudes in the following way:  $2^{nd} \leftrightarrow 5^{th}$  and  $6^{th} \leftrightarrow 9^{th}$ .

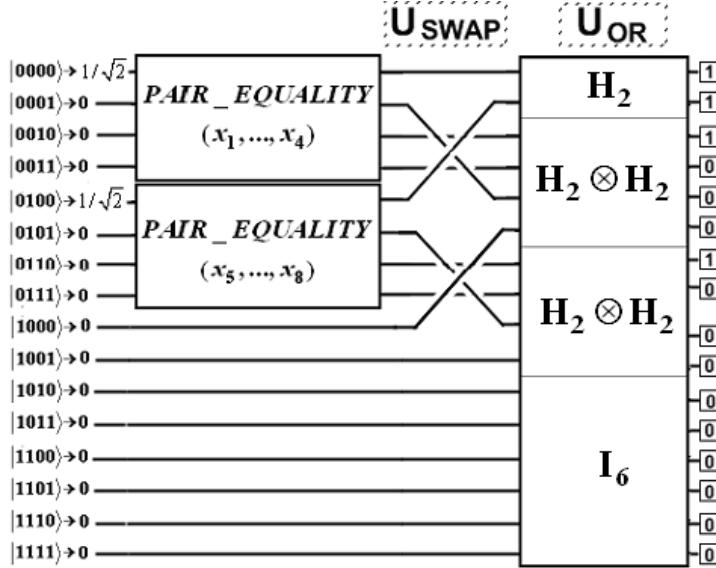
$$U_{SWAP} = (u_{ij}) = \begin{cases} 1, & \text{if } (i = 2 \& j = 5) \text{ OR } (i = 5 \& j = 2) \\ 1, & \text{if } (i = 6 \& j = 9) \text{ OR } (i = 9 \& j = 6) \\ 1, & \text{if } (i = j) \& (i \notin \{2, 5, 6, 9\}) \\ 0, & \text{otherwise} \end{cases}$$

Second gate  $U_{OR}$  is defined as follows:

$$U_{OR} = \begin{pmatrix} H_2 & O_{2 \times 4} & O_{2 \times 4} & O_{2 \times 6} \\ O_{4 \times 2} & H_2 \otimes H_2 & O_{4 \times 4} & O_{4 \times 6} \\ O_{4 \times 2} & O_{4 \times 4} & H_2 \otimes H_2 & O_{4 \times 6} \\ O_{6 \times 2} & O_{6 \times 4} & O_{6 \times 4} & I_6 \end{pmatrix},$$

where  $H_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ ,  $I_6$  is  $6 \times 6$  identity matrix;  $O_{i \times j}$  are zero matrices.

Complete algorithm structure is presented in Fig. 4.



**Fig. 4.** Bounded-error QQA for  $PAIR\_EQUALITY_4^\vee$

**Quantum complexity.** Algorithm 4 computes  $PAIR\_EQUALITY_4^\vee$  using 2 queries with probability  $p = 5/8$ :  $Q_{5/8}(PAIR\_EQUALITY_4^\vee) = 2$ .

#### 4.4 Second Constructing Method - OR( $f_1, f_2$ )

In this section we generalize approach for computing composite Boolean functions matching  $OR(f_1, f_2)$  pattern. The next lemma will be useful during method application.

**Lemma 2.** *For any QQA on any computation step it is possible to swap amplitude values in arbitrary order by applying specific quantum gate.*

*Proof.* Assume we need to swap amplitude values according to permutation  $\sigma = \begin{pmatrix} \alpha_1 \alpha_2 \dots \alpha_n \\ \beta_1 \beta_2 \dots \beta_n \end{pmatrix}$ . Then we can define quantum gate  $U_{SWAP} = (u_{ij})$  elements as:

- $\forall k \in \{1..n\} : u_{\alpha_k \beta_k} = 1$ ;
- $u_{ij} = 0$ , in all other cases.

Now we are ready to formulate a method for computing  $OR(f_1, f_2)$  kind of functions. For simplicity we consider only input algorithms, which employ 2 qubit system. However, approach can be generalized for quantum systems of arbitrary size.

**Table 2.** Second Constructing Method - OR( $f_1, f_2$ )

<b>Input.</b> Two exact QQAs $Alg1$ and $Alg2$ that belong to <i>Class 3</i> , employ 2 qubit quantum systems and compute Boolean functions $f_1(X_1)$ and $f_2(X_2)$ .
<b>Constructing steps.</b>
1. Use 4 qubit quantum system for a new algorithm, in total $2^4 = 16$ basis states.
2. Start from the state: $\langle \psi   = \left( \frac{1}{\sqrt{2}}, 0, 0, 0, \frac{1}{\sqrt{2}}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \right)$ .
3. Combine $Alg1$ and $Alg2$ unitary and query transformations in the following way: $U_i = \begin{pmatrix} U_i^1 & O_{4 \times 4} & O_{4 \times 8} \\ O_{4 \times 4} & U_i^2 & O_{4 \times 8} \\ O_{8 \times 4} & O_{8 \times 4} & I_8 \end{pmatrix}$ , where $I_8$ is $8 \times 8$ identity matrix; $O_{i \times j}$ - zero matrices.
4. Apply amplitude swapping gate $U_{SWAP}$ , which was defined in the proof of Lemma 2, to arrange amplitudes in the following order: – 1 <sup>st</sup> amplitude $\leftrightarrow$ accepting amplitude of the first sub-algorithm; – 2 <sup>nd</sup> amplitude $\leftrightarrow$ accepting amplitude of the second sub-algorithm; – 3 <sup>rd</sup> , 4 <sup>th</sup> , 5 <sup>th</sup> amplitudes $\leftrightarrow$ rejecting amplitudes of the first sub-algorithm; – 7 <sup>th</sup> , 8 <sup>th</sup> , 9 <sup>th</sup> amplitudes $\leftrightarrow$ rejecting amplitudes of the second sub-algorithm.
5. Apply the last quantum gate $U_{OR}$ that was precisely defined in previous section.
6. Assign function values to an algorithm according to Fig. 4.
<b>Output.</b> A bounded-error QQA $A$ for computing a function $F(X) = f_1(X_1) \vee f_2(X_2)$ with probability $p = 5/8$ and complexity $Q_{5/8}(A) = \max(Q_E(Alg1), Q_E(Alg2))$ .

## 5 Results of Applying Methods

We have applied constructing methods to the basic exact algorithms from sets  $QAlg3$  and  $QAlg4$  (see Sect. 3.2). In total we obtained 272 bounded-error QQAs. Each algorithm computes different Boolean function and uses only 2 queries.

We have demonstrated, that by employing quantum computation features it is possible to calculate composite functions by combining algorithms for sub-functions, without additional queries. It is doubtful that similar effect can be achieved in a classical model.

The important point is that invention of each brand-new exact QQA with required properties will at once significantly increase a set of efficiently computable functions.

**Table 3.** Results of transformation and constructing methods application

Basic exact quantum algorithms				
Set	Size	Number of function arguments	Queries	Probability
$QAlg3$	8	3	2	1
$QAlg4$	24	4	2	1
Constructed sets of algorithms				
$QAlg\_AND$	16	6	2	3/4
$QAlg\_OR$	256	6,7,8	2	5/8
<b>Total:</b>	<b>272</b>			

## 6 Conclusion

In this work we consider quantum query algorithm constructing problems. Main goal of research is to develop a framework for building efficient ad-hoc quantum algorithms for arbitrary Boolean functions. We have tried to develop general approaches for designing algorithms for computing Boolean functions defined by logical formula. In this paper we describe general algorithm constructions for computing AND and OR kinds of Boolean functions.

Suggested approaches allow building bounded-error quantum algorithms for complex functions based on already known algorithms. Significant behavior is that overall algorithm complexity does not increase. Additional queries are not required to compute composite function. However, error probability is the cost for efficient computing.

Further work in that direction is to improve general quantum query algorithm constructing techniques. Regarding already existing methods we would like to decrease existing limitations, such as restrictions on input basic algorithms. Another goal is to increase correct answer probability. From the other side, we need new methods to compute Boolean functions of different logical structure, to raise overall framework flexibility. Ultimate goal is to invent new efficient quantum algorithms that exceed already known separation from classical algorithms.

## 7 Acknowledgments

This research is supported by the European Social Fund.

## References

- [1] H. Buhrman and R. de Wolf: Complexity Measures and Decision Tree Complexity: A Survey. *Theoretical Computer Science*, v. 288(1): 2143 (2002).
- [2] R. de Wolf: Quantum Computing and Communication Complexity. University of Amsterdam (2001).
- [3] R. Cleve, A. Ekert, C. Macchiavello, et al. Quantum Algorithms Revisited. *Proceedings of the Royal Society, London*, A454 (1998).
- [4] J. Gruska: Quantum Computing. McGraw-Hill (1999).
- [5] M. Nielsen, I. Chuang: Quantum Computation and Quantum Information. Cambridge University Press (2000).
- [6] A. Ambainis: Quantum query algorithms and lower bounds (survey article). *Proceedings of FOTFS III*.
- [7] A. Ambainis and R. de Wolf: Average-case quantum query complexity. *Journal of Physics A* 34, pp 67416754 (2001).
- [8] A. Ambainis. Polynomial degree vs. quantum query complexity. *Journal of Computer and System Sciences* 72, pp. 220238 (2006).
- [9] A. Dubrovska, "Quantum Query Algorithms for Certain Functions and General Algorithm Construction Techniques," *Quantum Information and Computation V*, Proc. of SPIE Vol. 6573 (SPIE, Bellingham, WA, 2007) Article 65730F.

# Space Complexity in Ordinal Turing Machines

Joost Winter

## 1 Introduction

In [Ko], Peter Koepke introduced a model for transfinite computation that goes beyond the earlier infinite time Turing machine model described by Joel Hamkins and Andy Lewis in [HaLe]. Unlike ITTMs, which use a number of tapes of size  $\omega$ , these ordinal Turing machines have a class-sized tape of length  $\text{Ord}$ . Also unlike ITTMs, these machines also give rise to an intuitive notion of space complexity, allowing us to define a variety of space complexity classes.

As it turns out, these machines are strictly more powerful than the earlier Hamkins-Kidder model of ITTMs: it has been shown in [Lö] that the Hamkins-Kidder weak halting problem  $h$  is decidable by one of these Ordinal machines.

In this article, we will look at some issues of space complexity for these machines. Although in principle, here it is possible to look at classes of decidable ordinals (which need not even be sets!), in this article, the time and space complexity classes will be defined as containing only sets of reals (where ‘reals’ are taken to be subsets of  $\mathbb{N}$ ) decidable within a certain time and space complexity.

## 2 Definitions

Ordinal Turing machines can be defined as follows:

**Definition 2.1.** *An ordinal Turing machine, or OTM, is defined by a tuple of the form  $T = (Q, \delta, q_s, q_f)$ , where  $Q$  is a set of internal states,  $\delta$  is a transition function from  $(Q \setminus \{q_f\}) \times \{0, 1\}^3$  to  $Q \times \{0, 1\}^3 \times \{L, R\}$  and  $q_s, q_f \in Q$  are two special states called the initial and halting state, such that  $q_s \neq q_f$  holds.*

In the next definition, we will recursively define three operations: (1)  $q_\alpha^T(x)$  denotes the state a machine  $T$  is in at stage  $\alpha$ , having started from the input  $x$ ; (2)  $h_\alpha^T(x)$  denotes the position of the head at stage  $\alpha$ , having started from the input  $x$ ; and (3)  $c_{i,\alpha}^T(x)$ , where  $i \in \{1, 2, 3\}$ , denotes the content of tape  $i$  at stage  $\alpha$ , having started from the input  $x$ . Also, we will define another operation,  $c_{i,n,\alpha}^T(x)$ , which will be defined as 1 if  $n \in c_{i,\alpha}^T(x)$ , and 0 otherwise.

Of the three tapes we have, the first one will be called, and play the role of **input tape**, and the second one will be used as **output tape**. We will regularly use the term **the snapshot at  $\alpha$**  to refer to the tuple of values  $(q_\alpha^T(x), h_\alpha^T(x), c_{1,\alpha}^T(x), c_{2,\alpha}^T(x), c_{3,\alpha}^T(x))$ .

**Definition 2.2.** *For any ordinal Turing machine  $T$  and any input  $x \in \mathbb{R}$ , we define the operations  $q_\alpha^T(x)$ ,  $h_\alpha^T(x)$ ,  $c_{i,\alpha}^T(x)$  as follows for all ordinals  $\alpha$ , using transfinite recursion:*

- If  $\alpha = 0$ , we set  $q_\alpha^T(x) = q_s$ ,  $h_\alpha^T(x) = 0$ ,  $c_{1,\alpha}^T(x) = x$ , and  $c_{i,\alpha}^T(x) = \emptyset$  for  $i \in \{2, 3\}$ .
- If there is some ordinal  $\beta < \alpha$  such that  $q_\beta^T(x) = q_f$ , then  $q_\alpha^T(x)$ ,  $h_\alpha^T(x)$ , and  $c_{i,\alpha}^T(x)$  for  $i \in \{1, 2, 3\}$  are undefined.
- Otherwise, if  $\alpha$  is a successor ordinal  $\beta + 1$ , and  $h_\beta^T(x) = h$ , we look at the value of the transition function

$$(q', (a_1, a_2, a_3), \Delta) = \delta(q_\beta^T(x), (c_{1,h,\beta}^T(x), c_{2,h,\beta}^T(x), c_{3,h,\beta}^T(x)))$$

and set:

- $q_\alpha^T(x) = q'$
- $h_\alpha^T(x) = h + 1$  if  $\Delta = R$ ;  $h_\alpha^T(x) = h - 1$  if  $\Delta = L$  and  $h$  is a successor ordinal; and  $h_\alpha^T(x) = 0$  if  $\Delta = L$  and  $h$  is either 0 or a limit ordinal.
- For each  $i \in \{1, 2, 3\}$ ,  $c_{i,\alpha}^T(x)$  is defined as  $c_{i,\beta}^T(x) \setminus \{h\}$  if  $a_i = 0$ , and as  $c_{i,\beta(x)}^T \cup \{h\}$  if  $a_i = 1$ .
- Otherwise, if  $\alpha$  is a limit ordinal, we set  $q_\alpha^T(x) = \limsup\{q_\beta^T(x) : \beta < \alpha\}$ ,  $h_\alpha^T(x) = \liminf\{h_\beta^T(x) : \beta < \alpha\}$ , and for each  $i \in \{1, 2, 3\}$ ,  $c_{i,\alpha}^T(x) = \{\gamma \in \text{Ord} : \limsup_{\alpha < \beta} c_{i,\gamma,\alpha}^T(x) = 1\}$ .

As a result of defining the position of the head at limit stages as the  $\liminf$  of the earlier head positions, it follows directly that the head position may, at some point, be an infinite ordinal. This, in turn implies that infinite ordinals may be written to, or again deleted from the tapes: as a result, for an arbitrary ordinal  $\alpha$ , any  $x \in \mathbb{R}$ , and any  $i \in \{1, 2, 3\}$ ,  $c_{i,\alpha}^T(x)$  need not be a real anymore, but can in principle be any set of ordinals. In principle, we could also remove the restriction that the input of a function be a real, and replace it with the weaker requirement that it be a set of ordinals. However, in this article we will not do this, and focus only on computations by ordinal Turing machines starting from reals.

Now we have defined the ordinal Turing machines and their computation, we continue by defining the complexity classes **P** and **PSPACE**<sup>1</sup>. The **P** classes for ITMs were originally defined in [Sc], and the **P** and **PSPACE** classes for OTMs both were originally defined in [Lö].

**Definition 2.3.** If  $T$  is a machine that eventually reaches the halting state  $q_f$ , and  $\alpha$  is the smallest ordinal such that  $q_\alpha^T(x) = q_f$ , then we say that **time**( $x, T$ ) =  $\alpha$ .

**Definition 2.4.** For any ordinal Turing machine  $T$ , we say that  $T$  is a **time f machine** if, for all  $x \in \mathbb{R}$ , we have **time**( $x, T$ )  $\leq f(x)$ . For any ordinal  $\xi$ , we say that  $T$  is a **time  $\xi$  machine** if  $T$  is a time  $f$  machine for the constant function  $f$  such that  $f(x) = \xi$  for all  $x \in \mathbb{R}$ .

---

<sup>1</sup> It should be noted that is very little if anything at all polynomial about these classes: the **P** notation has been introduced in [Sc] originally, and later on the **PSPACE** was introduced in [Lö]. The fact that these notions have stuck around might be regrettable, but is true nonetheless. The present author has decided to stick with the familiar, but strange, notation in this paper.

Using these definitions, we can now define the classes  $\mathbf{P}_f^K$  and  $\mathbf{P}_\alpha^K$  for functions  $f$  and ordinals  $\alpha$ . Note that a set of reals  $A$  is decidable, if there is a OTM that terminates on all inputs  $r \in \mathbb{R}$ , and which outputs 1 on exactly the reals  $r \in A$ . Also, following [Lö], we will use the notations  $\mathbf{P}^{\text{HK}}$  and  $\mathbf{PSPACE}^{\text{HK}}$  for the corresponding classes for Hamkins-Kidder style infinite Time Turing machines.

**Definition 2.5.** *For any function  $f$ , we let  $\mathbf{P}_f^K$  denote the class of all sets of reals that are decidable by a time  $f$  machine. For any ordinal  $\xi$ , we let  $\mathbf{P}_\xi^K$  denote the class of all sets of reals that are decidable by a time  $\eta$  machine for some  $\eta < \xi$ .<sup>2</sup> Likewise, we use the similar notations  $\mathbf{P}_f^{\text{HK}}$  and  $\mathbf{P}_\xi^{\text{HK}}$  for the corresponding complexity classes for ITTMs.*

For the space complexity classes, we can likewise proceed by simply defining  $\mathbf{space}(x, T)$  as the largest  $\alpha$  that occurs as the position of the head at some stage of the computation:

**Definition 2.6.** *If  $T$  is a machine that, at one point reaches the halting state  $q_f$ , and  $\mathbf{time}(x, T) = \alpha$ , we define  $\mathbf{space}(x, T) = \sup\{h_\beta^T(x) : \beta \leq \alpha\}$ .*

**Definition 2.7.** *For any OTM  $T$ , we say that  $T$  is a **space  $f$  machine** if, for all  $x \in \mathbb{R}$ , we have  $\mathbf{space}(x, T) \leq f(x)$ . For any ordinal  $\xi$ , we say that  $T$  is a **space  $\xi$  machine** if  $T$  is a space  $f$  machine for the constant function  $f$  such that  $f(x) = \xi$  for all  $x \in \mathbb{R}$ .*

**Definition 2.8.** *For any function  $f$ , we let  $\mathbf{PSPACE}_f^K$  denote the class of all sets of reals that are decidable by a space  $f$  machine. For any ordinal  $\xi$ , we let  $\mathbf{PSPACE}_\xi^K$  denote the class of all sets of reals that are decidable by a space  $\eta$  machine for some  $\eta < \xi$ .*

An immediate result of these definitions (see [Lö]), is that a computation of a ordinal Turing machine can never take more space than it can take time:

**Proposition 2.9.** *For all  $f$  and all  $\alpha$ , we have  $\mathbf{P}_f^K \subseteq \mathbf{PSPACE}_f^K$  and  $\mathbf{P}_\alpha^K \subseteq \mathbf{PSPACE}_\alpha^K$  respectively.*

We will now introduce the notions of clockable, writable, eventually writable, and accidentally writable ordinals. All these notions were originally defined in [HaLe], from page 581 onward. Note that the these notions refer to writability by ITTMs, rather than by OTMs!

**Definition 2.10.** *We call an ordinal  $\alpha$   **$x$ -clockable** if there is an ITTM that terminates at time  $\alpha$  starting from input  $x$ . An ordinal  $\alpha$  is  **$x$ -writable** if there is an ITTM that terminates with a code for  $\alpha$  on the output tape, starting from input  $x$ . An ordinal  $\alpha$  is **eventually  $x$ -writable** if there is an ITTM that, on*

<sup>2</sup> The  $<$  here and in Definition 2.8 is correct, and should not be a  $\leq$ . A justification for this is found in the fact that using  $<$  instead of  $\leq$  here allows for a larger variety of classes—a broader discussion on the exact nature of these definitions can be found in Appendix A of [Wi].

input  $x$ , at one point writes a code for  $\alpha$  on the out tape, never to change it afterwards, without necessarily terminating. An ordinal  $\alpha$  is **accidentally  $x$ -writable** if there is an ITTM that, on input  $x$ , writes  $\alpha$  on any of the tapes at some stage of the computation.

With **clockable**, **writable**, etc., we mean  $\emptyset$ -clockable, -writable, etc.

We call the supremum of  $x$ -writable ordinals  $\lambda^x$ , the supremum of  $x$ -clockable ordinals  $\gamma^x$ , the supremum of eventually  $x$ -writable ordinals  $\zeta^x$ , and the supremum of accidentally  $x$ -writable ordinals  $\Sigma^x$ .

It turns out that the notions of clockability, writability, eventually and accidentally writability are related in the following way:

**Proposition 2.11.** *For all reals  $x$ ,  $\gamma^x = \lambda^x$ , and  $\lambda^x < \zeta^x < \Sigma^x$ .*

*Proof.* See [We, Corollary 2.1], and for a corrected and more detailed version, [Wi, Proposition 3.16].

### 3 Variants on the ordinal Turing machine architecture

It turns out that the exact number of tapes used by an OTM is generally irrelevant, and that, as long as there is just one head used for these tapes, OTMs with more tapes can be simulated by OTMs with less tapes, and vice versa.

**Proposition 3.1.** *For any  $n \in \mathbb{N}$ , with  $n > 2$ , a set  $A \subseteq \mathbb{R}$  is decidable by an OTM with  $n$  tapes if and only it is decidable by a ordinal Turing machine with 2 tapes.*

*Proof.* Because we only are considering sets of reals here, we only have to deal with inputs that are real numbers. Assume that  $A$  is decidable by a ordinal Turing machine  $T$  with  $n$  tapes. We can now construct a ordinal Turing machine  $T'$  with 2 tapes as follows:

- $T'$  starts out by stretching the input such that, if  $x$  is the input, after the stretching operation we find  $\{(n-1) \cdot k : k \in x\}$  on the first tape. We can use a signal state to recognize when we are done with this operation, and ensure this signal state only reads 1s at successor ordinals; at stage  $\omega$  we will find ourselves with the head at position  $\omega$ , in this signal state, and read a 0, and move left so that the head position will again be 0.
- After this, we simulate the original  $n$  tape machine on our 2 tape machine, using the second tape to simulate the output tape, and the first tape to simulate the  $n-1$  other tapes. We make sure that we have a fixed number  $k$  such that each step by  $T$  is simulated by  $k$  steps of  $T'$ , and moreover ensure that, for any ordinal  $\alpha$  such that  $\alpha = k \cdot \beta$  for some  $k$ ,  $h_\alpha^T(x) = (n-1) \cdot \eta$  for some  $\eta$ .

Because the only tape being simulated on the second tape is the original second tape, and because first cell of the second tape of  $T$  corresponds to the first cell of the second tape of  $T'$ , it follows that  $T$  and  $T'$  will, on the same input values, always give the same output values.

This gives us the following corollary:

**Corollary 3.2.** *For any  $m, n \geq 2$ , a set  $A$  is decidable by a ordinal Turing machine with  $m$  tapes if and only if it is decidable by a ordinal Turing machine with  $n$  tapes.*

Furthermore, it has been conjectured that this is also true in the case where  $n = 1$ .

#### 4 Time complexity for ordinal Turing machines

We can prove a number of theorems about time complexity for ordinal Turing machines: to start with, it turns out that for many recursive ordinals  $\alpha$ , a ordinal Turing machine can compute exactly the same sets in time  $\alpha$  that an ITTM machine can. The first theorem is a modification of [Lö, Proposition 4].

It should be noted that, with sufficiently closed  $\alpha$  and/or  $f$ ,  $\mathbf{P}_\alpha^{\text{HK}} \subseteq \mathbf{P}_\alpha^K$  and  $\mathbf{P}_f^{\text{HK}} \subseteq \mathbf{P}_f^K$  always follow. To simulate a Hamkins-Kidder machine by a Koepke machine, we only need to take care of finding a way to recognize when we are at a limit stage of the computation (which can be achieved through the use of flags on an auxiliary tape), then move left, and move to a special limit state. This ‘moving left’ might take at most  $\omega$  steps, and hence, the inequality holds for any ordinal  $\alpha$  such that  $\omega \times \alpha = \omega$ .

**Proposition 4.1.** *For any recursive ordinal  $\alpha$  larger than  $\omega^2$ , where  $\alpha$  is the successor of a limit ordinal  $\beta$ , and  $\omega \times \beta = \beta$ , we have  $\mathbf{P}_\alpha^{\text{HK}} = \mathbf{P}_\alpha^K$ .*

*Proof.* Assume that  $A \in \mathbf{P}_\alpha^K$ . This means that  $A$  is decidable by a time  $\eta$  machine for some  $\eta < \alpha$ , which, because  $\alpha = \beta + 1$ , here can be restated as:  $A$  is decidable by a time  $\beta$  OTM. Then  $A$  is, by Proposition 2.9, decidable by a time  $\beta$  OTM that uses  $\beta$  or less cells. Because  $\beta$  is a recursive ordinal, we can let an infinite time Turing machine write a code for  $\beta$  on the tape in  $\omega$  steps. After that, we can simulate the algorithm of the ordinal Turing machine, which needs  $\beta$  cells, on our  $\omega$ -sized tape. Because every step of the simulated computation will be simulated by a finite number of steps of the simulating computation, and because the simulated computation takes less than  $\beta$  steps (the halting state cannot be reached at  $\beta$  itself, because  $\beta$  is a limit ordinal), the simulating computation will also take less than  $\beta$  steps. At limit stages, we still need to find which the liminf of the visited tape positions, which might take  $\omega$  steps: hence, a single step might be simulated by  $\omega$  steps in the simulation.

Because  $\beta \geq \omega^2$ , the complete algorithm will take less than  $\omega + (\omega \times \beta) = \beta$  steps in total.

This equivalence between the time complexity classes of OTMs and ITTMs also holds at the level  $\lambda$ , the supremum of all writable ordinals:

**Proposition 4.2.**  $\mathbf{P}_\lambda^{\text{HK}} = \mathbf{P}_\lambda^K$ .

*Proof.* Assume that  $A$  is decidable by a time  $\alpha$  OTM  $T$ , where  $\alpha < \lambda$  (or, in other words, where  $\alpha$  is a writable ordinal). This machine  $T$  is also a space  $\alpha$  machine, so the computation uses at most  $\alpha$  cells. Say,  $\alpha$  is writable by an ITTM in time  $\beta$ : this implies that  $\beta$  is clockable and hence also writable. After writing a code for  $\alpha$  on the tape in  $\beta$  steps, we can continue by simulating the OTM algorithm by simulating a tape of size  $\alpha$ , in time less than  $\alpha$ . Thus, we can decide  $A$  with an ITTM in time  $\beta + \alpha$ , which is again a writable ordinal. Hence,  $A \in \mathbf{P}_\lambda^{\text{HK}}$ .

It turns out, that in fact, an equality of the above type still holds up to the level  $\Sigma^x$ , the supremum of all ordinals accidentally writable on input  $x$ . Note here that  $\mathbf{P}_{\Sigma^x}^K$  is taken to mean  $\mathbf{P}_f^K$  for the function  $f$  such that  $f(x) = \Sigma^x$  for all  $x$ .

**Proposition 4.3.**  $\mathbf{P}_{\Sigma^x}^K = \mathbf{P}_{\lambda^x}^{\text{HK}}$ .

*Proof.* Assume that a set  $A$  is in  $\mathbf{P}_{\Sigma^x}^K$ . Then it follows from Proposition 2.9 that  $A$  is in  $\mathbf{PSPACE}_{\Sigma^x}^K$ , and we know that there is an accidentally  $x$ -writable ordinal  $\alpha$  such that, if we can write  $\alpha$  on the tape, we can compute  $A$  on an ITTM, simulating an  $\alpha$ -sized fragment of the tape of the OTM within space  $\omega$ , based upon the just-found coding of  $\alpha$ .

The tactic now is to construct an ITTM, that simulates all other ITTMs on input  $x$ , and in doing so, performs an algorithm that, one by one, writes all reals that are accidentally writable from  $x$  on one of the scratch tapes. We can be sure that every real that is accidentally writable from  $x$  will be written on one of the scratch tapes at some point; and, each time a real is written on the scratch tape this way, we check if it codes an ordinal. If it does, we continue simulating the computation using this ordinal as tape length: eventually we will either run out of space, in which case we continue the algorithm writing accidentally writable reals on the tape, or it will turn out that we have sufficient space, in which case the computation will finish.

This computation will eventually halt, because a real coding an ordinal equal to or larger than  $\alpha$  will be written on the tape at some point and, as all halting computations will halt before  $\lambda^x$ , we know that the computation will halt before  $\lambda^x$ . So, it follows that the set  $A$  is in  $\mathbf{P}_{\lambda^x}^{\text{HK}}$ .

This gives us the following corollary:

**Corollary 4.4.**  $\mathbf{P}_{\Sigma^x}^K = \mathbf{Dec}^{\text{HK}}$

*Proof.* This follows directly from  $\mathbf{Dec}^{\text{HK}} = \mathbf{P}_{\lambda^x}^{\text{HK}}$  and  $\mathbf{P}_{\Sigma^x}^K = \mathbf{P}_{\lambda^x}^{\text{HK}}$ .

## 5 Space complexity for OTMs

Now we will take a quick look at space complexity issues for Koepke-type Ordinal machines.

**Proposition 5.1.**  $\mathbf{Dec}^{\text{HK}} \subseteq \mathbf{PSPACE}_{\omega+2}^K$

*Proof.* Assume that  $A \in \mathbf{Dec}^{\text{HK}}$ . We now can simulate an algorithm deciding  $A$  on an OTM with tape length  $\omega + 1$  by moving left towards position 0 at limit states, and besides this, simply performing the algorithm used by the ITTM.

**Proposition 5.2.**  $\mathbf{PSPACE}_{\Sigma^x}^K \subseteq \mathbf{Dec}^{\text{HK}}$

*Proof.* In the proof Proposition 4.3, it was shown that  $\mathbf{PSPACE}_{\Sigma^x}^K \subseteq \mathbf{P}_{\lambda^x}^{\text{HK}}$ . In combination with the fact that  $\mathbf{P}_{\lambda^x}^{\text{HK}} = \mathbf{Dec}^{\text{HK}}$ , this gives the desired result.

These two results combined give an interesting result: together they imply that it does not really matter how much extra space you use on an OTM, as long as it does not go beyond  $\Sigma^x$ . In particular, we have:

**Corollary 5.3.** *For any ordinal  $\alpha$ , such that  $\alpha \geq \omega + 1$  and  $\alpha \leq \Sigma + 1$ , and for any function  $f$  such that we have  $f(x) \geq \omega$  and  $f(x) \leq \Sigma^x$  for all  $x$ , we have respectively that  $\mathbf{PSPACE}_\alpha^K = \mathbf{Dec}^{\text{HK}}$  or  $\mathbf{PSPACE}_f^K = \mathbf{Dec}^{\text{HK}}$ .*

So, it turns out that, even with space  $\Sigma^x$  at our disposal, we are unable to compute any functions that an ITTM cannot compute. Moreover, we can now establish a relationship between the  $\mathbf{PSPACE}^K$ -classes and the  $\mathbf{P}^K$ -classes in a few cases.

**Proposition 5.4.** *For any ordinal  $\alpha$ , such that  $\alpha \geq \omega + 1$  and  $\alpha \leq \lambda$ , we have that  $\mathbf{P}_\alpha^K \subsetneq \mathbf{PSPACE}_\alpha^K$ .*

*Proof.* On one side, we know from Corollary 5.3 that  $\mathbf{PSPACE}_\alpha^K = \mathbf{Dec}^{\text{HK}}$ , and on the other side, we know that  $\mathbf{P}_\alpha^K \subsetneq \mathbf{Dec}^{\text{HK}}$ .

**Proposition 5.5.** *For functions  $f$  such that, for all  $x$ ,  $\Sigma^x > f(x) > \lambda^x$ , we have  $\mathbf{PSPACE}_f^K = \mathbf{P}_f^K = \mathbf{Dec}^{\text{HK}}$ .*

*Proof.* On one side, from Corollary 5.3 it follows that  $\mathbf{PSPACE}_f^K = \mathbf{Dec}^{\text{HK}}$ . On the other side, we know that  $\mathbf{P}_f^K \supseteq \mathbf{P}_{\lambda^x}^K$ , and from Proposition 4.3 we know that  $\mathbf{P}_{\lambda^x}^K = \mathbf{P}_{\lambda^x}^{\text{HK}}$ . From the fact that  $\mathbf{P}_{\lambda^x}^{\text{HK}} = \mathbf{Dec}^{\text{HK}}$ , it thus follows that  $\mathbf{P}_f^K \supseteq \mathbf{Dec}^{\text{HK}}$ . So we know that  $\mathbf{PSPACE}_f^K = \mathbf{Dec}^{\text{HK}}$ , as well as that  $\mathbf{P}_f^K \supseteq \mathbf{Dec}^{\text{HK}}$ . Finally, from Proposition 2.9 we know that  $\mathbf{P}_f^K \subseteq \mathbf{PSPACE}_f^K$ , so that we must have  $\mathbf{P}_f^K = \mathbf{Dec}^{\text{HK}}$ , completing the proof.

However, we will shortly show that Koepke machines are in fact more powerful than ITTMs, even when it comes to computing sets of reals. To see this, we will let ourselves be guided by the following question: are there any countable functions or ordinals, such that there are functions computable by an OTM, bounded in space by respectively that ordinal or function, that are not decidable by any ITTM? It turns out that this, indeed is the case. With Proposition 5.6, we can show that the weak halting problem  $h$  is indeed in  $\mathbf{PSPACE}_{\Sigma+2}^K$ :

**Proposition 5.6.** *For all reals  $x$ , and for any computation by an ITTM  $T$  from  $x$  that does not halt by time  $\Sigma^x$ , the snapshot at time  $\zeta^x$  is the same as that at time  $\Sigma^x$ .*

*Proof.* See [Wi, Proposition 3.16] for a modified version of an earlier proof by Philip Welch.

**Proposition 5.7.**  $h \in \text{PSPACE}_{\Sigma+2}^K$ .

*Proof.* We will sketch a possible way to compute  $h$  in  $\text{PSPACE}_{\Sigma+2}^K$  space. First we will simply check if the input codes a natural number  $n$ ; if it does not, we simply output 0.

If it does, we know that either the computation of ITTM  $n$  on input  $n$  will finish before time  $\lambda$ , or, as a result of Proposition 5.6 the contents of the tape at stage  $\zeta$  will be identical to that at stage  $\Sigma$ .

Now, we will simulate the (non-halting) computation of  $h$  on input  $n$ , but, using a scratch tape, we will keep track of all the  $\omega$ -sized tapes at any stage. If we start a new step in the computation, we will first check if the configuration at this step is equal to one at an earlier step. If this is the case, we halt. Because the tape of the simulated computation has size  $\omega$ , at any time  $\alpha$ , we will have used at most  $\omega \cdot \alpha$  tape. This leaves us with the following two possibilities:

- The ITTM  $n$ , on input  $n$ , finishes before time  $\lambda$ . In this case,  $n \in h$ , and we output 1, and we have used less than  $\omega \cdot \lambda < \Sigma$  space.
- The ITTM  $n$ , on input  $n$ , never finishes, and the content of stage  $\Sigma$  will be equal to that at time  $\zeta$  due to Proposition 5.6. We will realize this during the checking stage at step  $\Sigma$ , and at that point the used part of the tape has size  $\Sigma$ .

In either case, we will use at most  $\Sigma$  space during the computation, and thus we have obtained a computation of  $h$  in  $\text{PSPACE}_{\Sigma+2}^K$  space.

As a result, up to the level of  $\Sigma^x$ , OTMs bounded in space up to level  $\Sigma^x$  cannot compute any sets that ITTMs cannot compute. However, just above the level  $\Sigma$  this changes: the weak halting problem  $h$  is contained in  $\text{PSPACE}_{\Sigma+2}^K$ .

## 6 Relating the space classes to the ITTM degrees

The above results caused a few questions to be raised (initially by Benedikt Löwe and Peter Koepke) about other variations on the infinite time Turing machine architecture, with different space constraints. One of these questions was whether the supremum of ordinals writable by OTMs that use at most  $\Sigma + 2$  space is equal to the set  $\lambda^h$ , that is, the supremum of ordinals writable from the weak ITTM halting problem  $h$ . In this section a negative answer to this question will be provided. We will furthermore extend this question a bit further, and consider whether some connection can be made between OTMs with certain space constraints, and the ITTM degrees.

**Definition 6.1.** *We let  $\lambda_\kappa$  denote the supremum of ordinals (on input 0) writable by an OTM that, on any real input, uses at most  $\kappa$  space. In particular, in this article we will be concerned with OTM computations constrained in length by  $\Sigma + 2$ .*

Following [HaLe, p. 588], we define the weak jump  $A^\nabla$  of a set  $A \in \mathbb{R}$  as follows:

**Definition 6.2.** *We define*

$$A^\nabla = A \oplus h^A = A \oplus \{p : \phi_p^A(0) \downarrow\}$$

We let  $A^{\nabla^{(\alpha)}}$  denote the  $\alpha$ th iteration of the weak operation. For successor ordinals this simply means that  $A^{\nabla^{(\alpha+1)}} = A^{\nabla^{(\alpha)}\nabla}$ . For limit ordinals  $\alpha$ , in [HaLe]  $A^{\nabla^{(\alpha)}}$  is defined as  $A \oplus w_\alpha$ , where  $w_\alpha = \bigoplus_{\beta < \alpha} w_\beta$  (here, for ordinals  $\beta < \alpha$ ,  $w_\beta$  is the set such that  $A^{\nabla^{(\beta)}} = A \oplus w_\beta$ ), using some code  $z$  for  $\alpha$  to organize the information.

It is immediate from the definition that  $0^\nabla = 0 \oplus h$ , which is computable from  $h$  and vice versa.

**Proposition 6.3.** *If  $x$  is eventually writable, and  $y$  is eventually  $x$ -writable, then  $y$  is eventually writable.*

*Proof.* We perform the algorithm that eventually writes  $x$ . In parallel, we perform the algorithm that writes  $y$  from  $x$  on another set of tapes, using the tape on which  $x$  will eventually be written as the input. During each step of the first computation, we perform a single step of the second computation if  $x$  remains unchanged during that step; and we restart, again copying the input from the tape on which  $x$  will eventually be output, if  $x$  has changed during that step.

As  $x$  will be eventually written, from some point onward the second computation will never be restarted, and continue forever, and eventually give  $y$  as output. This is the desired algorithm which eventually writes  $y$ .

**Proposition 6.4.** *For all eventually writable ordinals  $\alpha$ , we have  $\zeta = \zeta^{0^{\nabla^{(\alpha)}}}$ , and hence  $\zeta > \lambda^{0^{\nabla^{(\alpha)}}}$ .*

*Proof.* By the Eventual Jump Theorem, [HaLe, Theorem 6.14], if  $z$  is an eventually writable real, and  $\alpha$  is an eventually writable ordinal, then the iterated jump  $z^{\nabla^{(\alpha)}}$  is also writable. Applying this theorem to iterated jumps of 0, we obtain that  $0^{\nabla^{(\alpha)}}$  is eventually writable for every eventually writable ordinal  $\alpha$ .

Assume for some ordinal  $\alpha$  and some eventually writable ordinal  $\beta$ , that  $\alpha < \zeta^{0^{\nabla^{(\beta)}}}$ . This means that some code for  $\alpha$  is eventually writable from  $0^{\nabla^{(\beta)}}$ , which is itself eventually writable.

But now, by Proposition 1,  $\alpha$  itself is eventually writable, and hence  $\alpha < \zeta$ . So we have established that  $\zeta^{0^{\nabla^{(\alpha)}}} \leq \zeta$ . The other direction,  $\zeta \leq \zeta^{0^{\nabla^{(\alpha)}}}$  is trivial. By [Wi, Proposition 3.14], which gives us the inequality of  $\lambda^x$  and  $\zeta^x$  for all  $x$ , the result  $\zeta > \lambda^{0^{\nabla^{(\alpha)}}}$  now follows.

**Proposition 6.5.**  $\zeta < \lambda_{\Sigma+2}$

*Proof.* We can compute  $\zeta$  by a machine with tape length  $\Sigma+2$  the following way: for every natural number  $n$ , we can, using the construction in the proof of [Wi, Proposition 6.21] decide if the computation by machine  $n$  on input 0 finishes, and moreover, we are able to decide if machine  $n$  on input 0 will eventually write some ordinal. The latter can be done by inspecting the repeating part once we have recognized it: if it remains unchanged and codes some ordinal, then the machine eventually writes some ordinal.

This way we can, using a pairing function, write down all codes for eventually writable ordinals together in a single real. Once we have done this, we can reduce this collection to a single ordinal by iteratively identifying the least elements from the combined set. The ordinal thus obtained is the supremum of all eventually writable ordinals,  $\zeta$ . Because this is writable by a space  $\Sigma+2$  machine, we obtain  $\zeta < \lambda_{\Sigma+2}$ .

We not only showed that  $\lambda_{\Sigma+2}$  is not equal to  $\lambda^h$ , but indeed that it is not equal to  $\lambda^{0^{\nabla(\alpha)}}$  for any eventually writable ordinal  $\alpha$ .

## References

- [HaLe] Joel David Hamkins and Andy Lewis: *infinite time Turing machines*, Journal Of Symbolic Logic 65 (2000), pp. 567-604
- [Ko] Peter Koepke, *Turing computations on ordinals*, Bulletin of Symbolic Logic 11 (2005), pp. 377-397
- [Lö] Benedikt Löwe: *Space bounds for infinitary computation*, in Arnold Beckmann, Ulrich Berger, Benedikt Löwe, John V. Tucker (eds.), *Logical Approaches to Computational Barriers, Second Conference on Computability in Europe, CiE 2006, Swansea, UK, July 2006, Proceedings*, Springer-Verlag, Berlin [Lecture Notes in Computer Science 3988] (2006), pp. 319-329
- [DeHaSc] Vinay Deolalikar, Joel David Hamkins, Ralf-Dieter Schindler,  $\mathbf{P} \neq \mathbf{NP} \cap \text{co-}\mathbf{NP}$  for infinite time Turing machines, Journal of Logic and Computation 15 (2005), pp. 577-592
- [Wi] Joost Winter: *Space Complexity in infinite time Turing Machines*, ILLC Scientific Publications, MoL-2007-14, 2007
- [Sc] Ralf Schindler,  $\mathbf{P} \neq \mathbf{NP}$  for infinite time Turing machines, Monatshefte der Mathematik 139 (2003), pp. 335-340
- [We] Philip D. Welch, *The Length of infinite time Turing machine Computations*, Bulletin of the London Mathematical Society 32 (2000), pp. 129-136

# Reverse Mathematics for Fourier Expansion

Keita Yokoyama

Department of Mathematics, Tokyo Institute of Technology

2-12-1 Oh-okayama, Meguro-ku, Tokyo 152-8551, JAPAN.

yokoyama@math.titech.ac.jp or k\_yoko\_tautology@infoseek.jp

**Abstract.** This research is motivated by the program of Reverse Mathematics. We investigate some theorems for the convergence of Fourier series within some weak subsystems of second order arithmetic, in order to determine which set existence axioms are needed to prove these theorems. We show that uniform convergence of Fourier series for  $C^1$ -functions and  $L^2$ -convergence of Fourier series for continuous functions are equivalent to  $\text{WKL}_0$  over  $\text{RCA}_0$ . We also show that  $L^2$ -convergence of Fourier series for bounded continuous functions is equivalent to  $\text{WWKL}_0$  over  $\text{RCA}_0$ .

**Key words:** Reverse Mathematics, second order arithmetic, Fourier expansion, weak König's lemma

## 1 Introduction

This paper is a contribution to the program of Reverse Mathematics, whose ultimate goal is to determine which set existence axioms are needed to prove theorems of ordinary mathematics. The Reverse Mathematics program was initiated by Friedman and carried forward most notably by Simpson. It is now known that many of theorems of analysis, algebra and other branches of mathematics are either proved in the system  $\text{RCA}_0$  or equivalent over  $\text{RCA}_0$  to particular set existence axioms.

In this paper, we deal with subsystems  $\text{RCA}_0$ ,  $\text{WKL}_0$  and  $\text{WWKL}_0$  of second order arithmetic.  $\text{RCA}_0$  is the system of recursive comprehension, which guarantees the existence of recursively definable sets, and  $\Sigma_1^0$  induction.  $\text{WKL}_0$  consists of  $\text{RCA}_0$  and a particular set existence axiom called weak König's lemma, which asserts that every infinite tree of sequences of 0's and 1's has an infinite path.  $\text{WWKL}_0$  consists of  $\text{RCA}_0$  and weak weak König's lemma, which asserts that if a tree  $T$  has no path, then

$$\lim_{n \rightarrow \infty} \frac{|\{\sigma \in T \mid \text{lh}(\sigma) = n\}|}{2^n} = 0.$$

$\text{WWKL}_0$  is weaker than  $\text{WKL}_0$ . For details of the definitions of these subsystems, see [1]. For  $\text{WWKL}_0$ , see also [4].

Reverse Mathematics for various parts of analysis were investigated by several people. We study Reverse Mathematics for Fourier expansions. In the next

section, we prepare basic parts of differential calculus for Fourier expansions. In section 3, we study some theorems for Fourier expansions within  $\text{RCA}_0$ ,  $\text{WKL}_0$  and  $\text{WWKL}_0$ . We show that uniform convergence of Fourier series for periodic  $C^1$ -functions and  $L^2$ -convergence of Fourier series for periodic continuous functions are equivalent to  $\text{WKL}_0$  over  $\text{RCA}_0$ . We also show that  $L^2$ -convergence of Fourier series for bounded continuous functions is equivalent to  $\text{WWKL}_0$  over  $\text{RCA}_0$ . Moreover, we give a local approximation for a continuous function within  $\text{WWKL}_0$ .

## 2 Derivative and integral

Within  $\text{RCA}_0$ , we can define the real number system  $\mathbb{R}$  by Cauchy sequences, and can also define continuous functions on  $\mathbb{R}$ , see [1]. We first define  $C^1$ -functions.

**Definition 1 ( $C^1$ -functions).** *The following definitions are made in  $\text{RCA}_0$ . Let  $U$  be an open subset of  $\mathbb{R}$ , and let  $f$  and  $f'$  be continuous functions from  $U$  to  $\mathbb{R}$ . Then a pair  $(f, f')$  is said to be  $C^1$  if and only if*

$$\forall a \in U \lim_{x \rightarrow a} \frac{f(x) - f(a)}{x - a} = f'(a).$$

A continuous function  $f$  is said to be  $C^1$  if we can find a (code for a) continuous function  $f'$  which is the derivative of  $f$ .

To prove basic properties of  $C^1$ -functions in  $\text{RCA}_0$ , we construct differentiable condition functions. A differentiable condition function for a  $C^1$ -function  $f$  expresses the condition of differentiability at each point of  $\text{dom}(f)$ . It also expresses the continuity of the derivative  $f'$ . Hence using a differentiable condition function, we can easily prove basic properties of  $C^1$ -functions in  $\text{RCA}_0$ .

**Theorem 1.** *The following is provable in  $\text{RCA}_0$ . Let  $U$  be an open subset of  $\mathbb{R}$ , and let  $f$  be a  $C^1$ -function from  $U$  to  $\mathbb{R}$ . Then, there exists a continuous function  $e_f$  from  $U \times U$  to  $\mathbb{R}$  such that*

$$\begin{aligned} \forall x \in U \quad e_f(x, x) &= 0; \\ \forall x, y \in U \quad f(y) - f(x) &= (y - x)(f'(x) + e_f(x, y)). \end{aligned}$$

We call this  $e_f$  a differentiable condition function for  $f$ .

*Proof.* See [3].

Note that we can effectively find a differentiable condition function for a  $C^1$ -function.

To integrate continuous functions effectively, we introduce a modulus of integrability. Let  $f$  be a continuous function and let  $\Delta$  be a partition of  $[a, b]$ , i.e.  $\Delta = \{a = x_0 \leq \xi_1 \leq x_1 \leq \dots \leq \xi_n \leq x_n = b\}$ . Then, we define  $S_{[a,b]}^\Delta(f)$  as

$$S_{[a,b]}^\Delta(f) = \sum_{k=1}^n f(\xi_k)(x_k - x_{k-1})$$

and define  $|\Delta|$  as  $|\Delta| = \max\{x_k - x_{k-1} \mid 1 \leq k \leq n\}$ .

**Definition 2 (modulus of integrability).** *The following definition is made in  $\text{RCA}_0$ . Let  $f$  be a continuous function from  $[a, b]$  to  $\mathbb{R}$ . A modulus of integrability on  $[a, b]$  for  $f$  is a function  $h$  from  $\mathbb{N}$  to  $\mathbb{N}$  such that for all  $n \in \mathbb{N}$  and for all partitions  $\Delta_1, \Delta_2$  of  $[a, b]$ ,*

$$|\Delta_1| < \frac{2^{-h(n)}}{b-a} \wedge |\Delta_2| < \frac{2^{-h(n)}}{b-a} \rightarrow |S_{[a,b]}^{\Delta_1}(f) - S_{[a,b]}^{\Delta_2}(f)| < 2^{-n+1}.$$

*f is said to be effectively integrable if f has a modulus of integrability.*

In  $\text{RCA}_0$ , there might exist an unbounded continuous function on a closed interval. Note that an unbounded continuous function on a closed interval is not Riemann integrable and does not have a modulus of integrability. The next theorem shows that to integrate continuous functions requires  $\text{WKL}_0$ .

**Theorem 2.** *The following assertions are pairwise equivalent over  $\text{RCA}_0$ .*

1.  $\text{WKL}_0$ .
2. *Every continuous function on  $[a, b]$  is Riemann integrable.*
3. *Every continuous function on  $[a, b]$  has a modulus of integrability.*

*Proof.* See [1, Theorem IV.2.7].

To integrate bounded functions, we only need  $\text{WWKL}_0$ .

**Theorem 3.** *The following assertions are pairwise equivalent over  $\text{RCA}_0$ .*

1.  $\text{WWKL}_0$ .
2. *Every bounded continuous function on  $[a, b]$  is Riemann integrable.*
3. *Every bounded continuous function on  $[a, b]$  has a modulus of integrability.*

*Proof.* Within  $\text{WWKL}_0$ , we can prove a weak version of Heine-Borel theorem appeared in [4]: if  $\{U_n\}_{n \in \mathbb{N}}$  is an open covering of  $[0, 1]$ , then there exists a sequence of finite sequences of intervals  $\langle \{[c_{ni}, d_{ni}]\}_{i < l_n} \mid n \in \mathbb{N} \rangle$  such that

$$\begin{aligned} \forall n \in \mathbb{N} \ [0, 1] &\subseteq \bigcup_{k < n} U_k \cup \bigcup_{i < l_n} [c_{ni}, d_{ni}]; \\ \lim_{n \rightarrow \infty} \sum_{i < l_n} (d_{ni}, c_{ni}) &= 0. \end{aligned}$$

By this theorem, we can prove  $1 \rightarrow 3$  easily. The implication  $3 \rightarrow 2$  is trivial.

To show  $2 \rightarrow 1$ , we define some notation. For a tree  $T \subseteq 2^{<\mathbb{N}}$ , define a set  $S_T \subseteq 2^{<\mathbb{N}}$  and  $\lambda_n^T \in \mathbb{N}$  as

$$\begin{aligned} S_T &:= \{\sigma \in 2^{<\mathbb{N}} \mid \sigma \notin T \wedge \forall \tau \subseteq \sigma (\tau \neq \sigma \rightarrow \tau \in T)\}; \\ \lambda_n^T &:= |\{\sigma \in T \mid \text{lh}(\sigma) = n\}|. \end{aligned}$$

For a finite sequence  $\sigma \in 2^{<\mathbb{N}}$ , define  $a_\sigma, b_\sigma \in \mathbb{Q}$  as

$$\begin{aligned} a_\sigma &:= \sum_{i < \text{lh}(\sigma)} \frac{\sigma(i)}{2^{i+1}}; \\ b_\sigma &:= a_\sigma + \frac{1}{2^{\text{lh}(\sigma)}}. \end{aligned}$$

Thus, if  $\sigma, \tau \in S_T$ , then,  $b_\tau \leq a_\sigma$  or  $b_\sigma \leq a_\tau$ . Note that a tree  $T$  has a path if and only if  $[0, 1] \not\subseteq \bigcup_{\sigma \in S_T} [a_\sigma, b_\sigma]$ .

Now, we show  $\neg 1 \rightarrow \neg 2$ . We reason within  $\text{RCA}_0$ . Assume  $\neg \text{WWKL}_0$ . Then, there exist  $q > 0$  and a tree  $T$  which has no path such that  $\lambda_n^T / 2^n > q$  for all  $n \in \mathbb{N}$ . Since  $[0, 1] \subseteq \bigcup_{\sigma \in S_T} [a_\sigma, b_\sigma]$ , we can define a continuous function  $f$  from  $[0, 1]$  to  $[0, 1]$  as

$$f(x) := \begin{cases} \frac{x-a_\sigma}{c_\sigma-a_\sigma} & x \in [a_\sigma, c_\sigma] \wedge \sigma \in S_T, \\ \frac{b_\sigma-x}{b_\sigma-c_\sigma} & x \in [c_\sigma, b_\sigma] \wedge \sigma \in S_T \end{cases}$$

where  $c_\sigma := (b_\sigma + a_\sigma)/2$ .

We show that this  $f$  is not Riemann integrable. Define partitions of  $[0, 1]$   $\Delta_k$  as

$$\Delta_k := \left\{ 0 \leq \frac{1}{2^k} \leq \frac{2}{2^k} \leq \cdots \leq \frac{2^k - 1}{2^k} \leq 1 \right\} = \{[a_\eta, b_\eta] \mid \eta \in 2^{<\mathbb{N}} \wedge \text{lh}(\eta) = k\}.$$

Note that we can easily take  $M_\sigma := \max\{f(x) \mid x \in [a_\sigma, b_\sigma]\}$  and  $m_\sigma := \min\{f(x) \mid x \in [a_\sigma, b_\sigma]\}$ . We show that for all  $k \in \mathbb{N}$ ,

$$\sum_{\eta \in 2^{<\mathbb{N}} \wedge \text{lh}(\eta)=k} (M_\eta - m_\eta) 2^{-k} > q.$$

If  $\eta \in T$ , then, there exists  $\sigma \in S_T$  such that  $\sigma \supseteq \eta$ , thus,  $[a_\eta, b_\eta] \supseteq [a_\sigma, b_\sigma]$ . Therefore,  $\eta \in T$  implies  $M_\eta - m_\eta = 1$ . Hence, for all  $k \in \mathbb{N}$ ,

$$\sum_{\eta \in 2^{<\mathbb{N}} \wedge \text{lh}(\eta)=k} (M_\eta - m_\eta) 2^{-k} \geq \sum_{\eta \in T \wedge \text{lh}(\eta)=k} 2^{-k} \geq \lambda_n^T 2^{-n} > q.$$

This completes the proof of  $2 \rightarrow 1$ .

Next, we construct series of continuous functions.

**Theorem 4.** *The following is provable in  $\text{RCA}_0$ . Let  $\{\alpha_n\}_{n \in \mathbb{N}}$  be a sequence of nonnegative real numbers whose series  $\sum_{n=0}^{\infty} \alpha_n$  is convergent. Let  $\{f_n\}_{n \in \mathbb{N}}$  be a sequence of continuous functions on  $[a, b]$  such that  $\forall x \in [a, b] |f_n(x)| \leq \alpha_n$  for all  $n \in \mathbb{N}$ . Then, there exists a continuous function  $f$  such that*

$$\forall x \in [a, b] f(x) = \sum_{n=0}^{\infty} f_n(x).$$

*Proof.* See [1, Lemma II.6.5].

We can show termwise differentiation and integration by using a differentiable condition function and a modulus of integrability. For the proof of the following theorems, see [3].

**Theorem 5 (termwise differentiation).** *The following is provable in  $\text{RCA}_0$ . Let  $U$  be an open interval of  $\mathbb{R}$ , and let  $\sum_{n=0}^{\infty} a_n$  and  $\sum_{n=0}^{\infty} b_n$  be nonnegative convergent series. Let  $\{(f_n, f'_n)\}_{n \in \mathbb{N}}$  be a sequence of  $C^1$ -functions from  $U$  to  $\mathbb{R}$  such that  $\forall x \in U |f_n(x)| \leq a_n$  and  $\forall x \in U |f'_n(x)| \leq b_n$  for all  $n \in \mathbb{N}$ . Then, there exists a  $C^1$ -function  $(f, f')$  from  $U$  to  $\mathbb{R}$  such that*

$$\forall x \in U f(x) = \sum_{n=0}^{\infty} f_n(x) \wedge f'(x) = \sum_{n=0}^{\infty} f'_n(x).$$

**Theorem 6 (termwise integration).** *The following is provable in  $\text{RCA}_0$ . Let  $\sum_{n=0}^{\infty} \alpha_n$  be nonnegative convergent series, and let  $\{f_n\}_{n \in \mathbb{N}}$  be a sequence of effectively integrable continuous functions from  $[a, b]$  to  $\mathbb{R}$  such that  $\forall x \in [a, b] |f_n(x)| \leq \alpha_n$  for all  $n \in \mathbb{N}$ . By Theorem 4, we define  $f = \sum_{n=0}^{\infty} f_n$ . Then,  $f$  is effectively integrable and*

$$\int_a^b f(x) dx = \sum_{n=0}^{\infty} \int_a^b f_n(x) dx.$$

*(We say that  $\{f_n\}_{n \in \mathbb{N}}$  is a sequence of effectively integrable continuous functions if there exists a sequence of functions  $\{h_n\}_{n \in \mathbb{N}}$  such that each  $h_n$  is a modulus of integrability for  $f_n$ .)*

By Theorems 4 and 5, we can construct trigonometric functions  $\sin x$  and  $\cos x$  as  $C^1$ -functions within  $\text{RCA}_0$ . We can also define  $\pi$  as a unique zero-point of  $\sin x$  on  $[2, 4]$ . Note that we can prove basic properties of  $\sin x$  and  $\cos x$  in  $\text{RCA}_0$ .

### 3 Fourier expansion

In this section, we show some results on Reverse Mathematics for some basic theories of Fourier expansions. See e.g. [2] for the usual theory of Fourier analysis. We write  $f \in \mathcal{P}^{2\pi}$  if  $f$  is a continuous periodic function with period  $2\pi$ . Let  $\{a_k\}_{k \in \mathbb{N}}, \{b_k\}_{k \in \mathbb{N}}$  be real sequences. Then, define  $S_n$  as

$$S_n[\{a_k\}\{b_k\}](x) = \frac{a_0}{2} + \sum_{k=1}^n a_k \cos kx + b_k \sin kx.$$

If  $f \in \mathcal{P}^{2\pi}$  is effectively integrable, then, define

$$S_n[f](x) = S_n[\{a_k\}\{b_k\}](x)$$

where  $a_k$  and  $b_k$  are Fourier coefficients, i.e.,

$$\begin{aligned} a_k &= \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos kx dx; \\ b_k &= \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin kx dx. \end{aligned}$$

We first prepare some lemmas.

**Lemma 1.** *The following assertions are pairwise equivalent over  $\text{RCA}_0$ .*

1.  $\text{WKL}_0$ .
2. *Every periodic  $C^1$ -function is bounded.*
3. *Every periodic  $C^1$ -function is uniformly continuous.*

*Proof.* Easy modification of the proof of [1, Theorem IV.2.3] (by means of piecewise parabolic functions).

**Lemma 2 (Bessel inequality).** *The following is provable in  $\text{RCA}_0$ . Let  $f \in \mathcal{P}^{2\pi}$  be effectively integrable and let  $a_k$  and  $b_k$  be Fourier coefficients of  $f$ . Then,*

$$2\pi \sum_{i=0}^n (|a_i|^2 + |b_i|^2) \leq \int_{-\pi}^{\pi} f(x)^2 dx$$

for all  $n \in \mathbb{N}$ .

*Proof.* Straightforward imitation of the usual proof.

**Lemma 3.** *The following is provable in  $\text{RCA}_0$ . Let  $f \in \mathcal{P}^{2\pi}$  be effectively integrable. If*

$$\int_{-\pi}^{\pi} f(x) \cos kx dx = 0 \wedge \int_{-\pi}^{\pi} f(x) \sin kx dx = 0$$

for all  $n \in \mathbb{N}$ , then  $f \equiv 0$ .

*Proof.* Straightforward imitation of the usual proof.

**Lemma 4.** *The following is provable in  $\text{RCA}_0$ . Let  $f \in \mathcal{P}^{2\pi}$  be a  $C^1$ -function and let  $f$  and  $f'$  be effectively integrable. Then the Fourier series  $S_n[f]$  uniformly converges to  $f$ .*

*Proof.* We reason within  $\text{RCA}_0$ . Let  $a_k$  and  $b_k$  be Fourier coefficients of  $f$ , let  $a'_k$  and  $b'_k$  be Fourier coefficients of  $f'$  and let  $K = \int_{-\pi}^{\pi} f'(x)^2 dx$ . Then,

$$\begin{aligned} \pi a_k &= \int_{-\pi}^{\pi} f(x) \cos kx dx = \frac{1}{k} \int_{-\pi}^{\pi} f'(x) \sin kx dx = \frac{\pi b'_k}{k}; \\ \pi b_k &= \frac{\pi a'_k}{k}. \end{aligned}$$

By Schwarz inequality and Lemma 2,

$$\sum_{k=n}^m |a_k| + |b_k| \leq \sqrt{2 \sum_{k=n}^m \frac{1}{k^2}} \sqrt{\sum_{k=n}^m |b'_k|^2 + |a'_k|^2} \leq \sqrt{\frac{K}{\pi} \sum_{k=n}^m \frac{1}{k^2}}.$$

Thus,  $\sum_{k=0}^{\infty} |a_k| + |b_k|$  converges. Then, by Theorem 4, there exists  $g \in \mathcal{P}^{2\pi}$  such that

$$g(x) = \lim_{n \rightarrow \infty} S_n[f](x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos kx + b_k \sin kx.$$

Let  $\bar{f} = g - f$ . Then, by Lemma 3,  $\bar{f} \equiv 0$ . This means  $S_n[f]$  uniformly converges to  $f$ . This completes the proof.

The first theorem is concerned with the uniform convergence of Fourier series.

**Theorem 7.** *The following assertions are equivalent over  $\text{RCA}_0$ .*

1.  $\text{WKL}_0$ .
2. *If  $f \in \mathcal{P}^{2\pi}$  is a  $C^1$ -function, then there exist real sequences  $\{a_k\}_{k \in \mathbb{N}}$  and  $\{b_k\}_{k \in \mathbb{N}}$  such that  $S_n[\{a_k\}\{b_k\}]$  uniformly converges to  $f$ .*

*Proof.* By Theorem 2 and Lemma 4, 1  $\rightarrow$  2 holds. For the converse, we assume 2. By Lemma 1, we only need to show that every periodic  $C^1$ -function (with period  $2\pi$ ) is uniformly continuous. Let  $f \in \mathcal{P}^{2\pi}$  be a  $C^1$ -function. Then, there exist  $\{a_k\}_{k \in \mathbb{N}}$  and  $\{b_k\}_{k \in \mathbb{N}}$  such that  $S_n[\{a_k\}\{b_k\}]$  uniformly converges to  $f$ . Since  $\sin x$  and  $\cos x$  are uniformly continuous, we can easily show that  $f$  is also uniformly continuous. This completes the proof.

Next, we argue about  $L^2$ -convergence of Fourier series.

**Definition 3 ( $L^2$ -convergence).** *The following definition is made in  $\text{RCA}_0$ . Let  $\{f_n\}_{n \in \mathbb{N}}$  be a sequence of functions in  $\mathcal{P}^{2\pi}$ . Then, we say that  $\{f_n\}_{n \in \mathbb{N}}$   $L^2$ -converges to  $f$  if for all  $i \in \mathbb{N}$  there exists  $k \in \mathbb{N}$  such that for all  $m \geq k$  there exists a continuous function  $g$  such that  $g^2$  is effectively integrable and*

$$\begin{aligned} |f_m(x) - f(x)| &\leq g(x), \\ \int_{-\pi}^{\pi} g(x)^2 dx &< 2^{-i}. \end{aligned}$$

**Lemma 5.** *The following is provable in  $\text{RCA}_0$ . Let  $f \in \mathcal{P}^{2\pi}$  be effectively integrable. Then, the Fourier series  $S_n[f]$   $L^2$ -converges to  $f$ .*

*Proof.* We reason within  $\text{RCA}_0$ . Let  $h : \mathbb{N} \rightarrow \mathbb{N}$  be a modulus of integrability for  $f$  on  $[-\pi, \pi]$ . We can construct a sequence of continuous functions on  $[-\pi, \pi]$   $\{\tilde{f}_i\}_{i \in \mathbb{N}}$  (by means of piecewise parabolic functions) which satisfies the following:

- $\tilde{f}_i$  is  $C^1$  and  $\tilde{f}_i$  and  $\tilde{f}_i'$  are effectively integrable;
- $\tilde{f}_i(t_{ij}) = f(t_{ij})$ ;
- $\tilde{f}_i$  is monotone on  $[t_j, t_{j+1}]$

where  $t_{ij} = -\pi + 2\pi j/2^{h(i)}$  ( $j \leq 2^{h(i)}$ ). Then,  $\{\tilde{f}_i\}_{i \in \mathbb{N}}$   $L^2$ -converges to  $f$ . By Lemma 2,

$$\begin{aligned} &\int_{-\pi}^{\pi} (f(x) - S_n[f](x))^2 dx \\ &\leq \int_{-\pi}^{\pi} \{(f(x) - \tilde{f}_i(x))^2 + (\tilde{f}_i(x) - S_n[\tilde{f}_i](x))^2 + (S_n[\tilde{f}_i](x) - S_n[f](x))^2\} dx \\ &\leq 2 \int_{-\pi}^{\pi} (f(x) - \tilde{f}_i(x))^2 dx + \int_{-\pi}^{\pi} (\tilde{f}_i(x) - S_n[\tilde{f}_i](x))^2 dx. \end{aligned}$$

Since  $\{\tilde{f}_i\}_{i \in \mathbb{N}}$   $L^2$ -converges to  $f$ ,

$$\lim_{i \rightarrow \infty} \int_{-\pi}^{\pi} (f(x) - \tilde{f}_i(x))^2 dx = 0.$$

By Lemma 4,  $\{S_n[\tilde{f}_i]\}_{n \in \mathbb{N}}$  uniformly converges to  $\tilde{f}_i$ . Thus,

$$\lim_{n \rightarrow \infty} \int_{-\pi}^{\pi} (\tilde{f}_i(x) - S_n[\tilde{f}_i](x))^2 dx = 0.$$

Hence,

$$\lim_{n \rightarrow \infty} \int_{-\pi}^{\pi} (f(x) - S_n[f](x))^2 dx = 0,$$

and this completes the proof.

**Theorem 8.** *The following assertions are equivalent over  $\text{RCA}_0$ .*

1.  $\text{WKL}_0$ .
2. *If  $f \in \mathcal{P}^{2\pi}$ , then there exist real sequences  $\{a_k\}_{k \in \mathbb{N}}$  and  $\{b_k\}_{k \in \mathbb{N}}$  such that  $S_n[\{a_k\}\{b_k\}]$   $L^2$ -converges to  $f$ .*

*Proof.* We reason within  $\text{RCA}_0$ . By Theorem 2 and Lemma 5,  $1 \rightarrow 2$  holds. For the converse, we show  $\neg 1 \rightarrow \neg 2$ . Let  $\neg \text{WKL}_0$ . Then, by Lemma 1, there exists an unbounded function  $f \in \mathcal{P}^{2\pi}$ . Thus, for any real sequences  $\{a_k\}_{k \in \mathbb{N}}$  and  $\{b_k\}_{k \in \mathbb{N}}$ ,  $|S_n[\{a_k\}\{b_k\}] - f|$  is unbounded. Therefore, if  $|S_n[\{a_k\}\{b_k\}] - f| \leq g$ , then  $g^2$  is not integrable. This means that  $\neg 2$  and this completes the proof of  $2 \rightarrow 1$ .

**Theorem 9.** *The following assertions are equivalent over  $\text{RCA}_0$ .*

1.  $\text{WWKL}_0$ .
2. *If  $f \in \mathcal{P}^{2\pi}$  and  $|f| \leq K$  for some  $K \in \mathbb{Q}$ , then there exist real sequences  $\{a_k\}_{k \in \mathbb{N}}$  and  $\{b_k\}_{k \in \mathbb{N}}$  such that  $S_n[\{a_k\}\{b_k\}]$   $L^2$ -converges to  $f$ .*

*Proof.* We reason within  $\text{RCA}_0$ . By Theorem 3 and Lemma 5,  $1 \rightarrow 2$  holds. For the converse, we show  $\neg 1 \rightarrow \neg 2$ . We use the notation  $S_T$  and  $\lambda_n^T$  defined in the proof of Theorem 3. Let  $\neg \text{WWKL}_0$ . Then, there exist  $q > 0$  and a tree  $T$  which has no path such that  $\lambda_n^T/2^n > q$  for all  $n \in \mathbb{N}$ . For a finite sequence  $\sigma \in 2^{<\mathbb{N}}$ , define  $a_\sigma, b_\sigma \in \mathbb{R}$  as

$$a_\sigma := -\pi + 2\pi \sum_{i < \text{lh}(\sigma)} \frac{\sigma(i)}{2^{i+1}};$$

$$b_\sigma := a_\sigma + \frac{2\pi}{2^{\text{lh}(\sigma)}}.$$

Since  $T$  has no path,  $[-\pi, \pi] = \bigcup_{\sigma \in S_T} [a_\sigma, b_\sigma]$ . Define a function  $f \in \mathcal{P}^{2\pi}$  as

$$f(x) := \begin{cases} \frac{8(x-a_\sigma)}{b_\sigma-a_\sigma} & x \in [a_\sigma, c_\sigma] \wedge \sigma \in S_T, \\ -\frac{8\{x-(a_\sigma+b_\sigma)/2\}}{b_\sigma-a_\sigma} & x \in [c_\sigma, d_\sigma] \wedge \sigma \in S_T, \\ \frac{8(x-b_\sigma)}{b_\sigma-a_\sigma} & x \in [d_\sigma, b_\sigma] \wedge \sigma \in S_T \end{cases}$$

where  $c_\sigma := (b_\sigma + 3a_\sigma)/4$  and  $d_\sigma := (3b_\sigma + a_\sigma)/4$ . Then,  $f(c_\sigma) = 2$ ,  $f(d_\sigma) = -2$  for any  $\sigma \in S_T$  and  $|f| \leq 2$ .

Now, we show that for any real sequences  $\{a_k\}_{k \in \mathbb{N}}$  and  $\{b_k\}_{k \in \mathbb{N}}$  and for any  $n \in \mathbb{N}$ , if  $|S_n[\{a_k\}\{b_k\}] - f| \leq g$ , then  $\int_{-\pi}^{\pi} g(x)^2 dx > \pi q$ . Let  $\{a_k\}_{k \in \mathbb{N}}$  and  $\{b_k\}_{k \in \mathbb{N}}$  be real sequences, let  $n \in \mathbb{N}$  and let  $g$  be a continuous function such that  $g^2$  is effectively integrable and  $|S_n[\{a_k\}\{b_k\}] - f| \leq g$ . Take  $M_0 \in \mathbb{Q}$  such that  $M_0 \geq \max\{|a_0|, \dots, |a_n|, |b_0|, \dots, |b_n|\}$  and define  $M := (n+1)^2 M_0$ . Then,  $|S_n[\{a_k\}\{b_k\}]'(x)| \leq M$  for any  $x \in [-\pi, \pi]$ . Thus, if  $\sigma \in S_T$  and  $2\pi/2^{\text{lh}(\sigma)} \leq 1/M$ , then,  $|S_n[\{a_k\}\{b_k\}](x)| < 1$  for all  $x \in [a_\sigma, b_\sigma]$  or  $|S_n[\{a_k\}\{b_k\}](x)| > -1$  for all  $x \in [a_\sigma, b_\sigma]$ . Therefore,  $g(c_\sigma) > 1$  or  $g(d_\sigma) > 1$  for any  $\sigma \in S_T$  such that  $2\pi/2^{\text{lh}(\sigma)} \leq 1/M$ . Let  $h$  be a modulus of integrability for  $g^2$  on  $[-\pi, \pi]$ . Take  $N \in \mathbb{N}$  such that  $2\pi/2^N \leq 1/M$  and  $2^{-N} < 2^{-h(i)}/2\pi$  where  $i = \min\{j \in \mathbb{N} \mid 2^{-j+2} < \pi q\}$ . As in the proof of Theorem 3, if  $\eta \in T$  and  $\text{lh}(\eta) = N$ , then there exists  $x \in [a_\eta, b_\eta]$  such that  $g(x)^2 > 1$ . Take  $\langle \alpha_\eta \in [a_\eta, b_\eta] \mid \eta \in 2^{<\mathbb{N}} \wedge \text{lh}(\eta) = N \rangle$  such that  $g(\alpha_\eta) > 1$  if  $\eta \in T$ . Then, as in the proof of Theorem 3,

$$\begin{aligned} \int_{-\pi}^{\pi} g(x)^2 dx &\geq \sum_{\eta \in 2^{<\mathbb{N}} \wedge \text{lh}(\eta) = N} g(\alpha_\eta)^2 (b_\eta - a_\eta) - 2^{-i+2} \\ &\geq \sum_{\eta \in T \wedge \text{lh}(\eta) = N} (b_\eta - a_\eta) - 2^{-i+2} \\ &\geq \frac{2\pi\lambda_N^T}{2^N} - \pi q \\ &> \pi q. \end{aligned}$$

This means that  $\neg 2$  and this completes the proof of  $2 \rightarrow 1$ .

Imitating the usual arguments for Fourier expansions in  $\text{RCA}_0$ , we can show the following theorems.

**Theorem 10 (Parseval equality).** *The following is provable in  $\text{RCA}_0$ . Let  $f \in \mathcal{P}^{2\pi}$  be effectively integrable, and let  $a_k$  and  $b_k$  be Fourier coefficients of  $f$ . Then,*

$$2\pi \sum_{i=0}^{\infty} (|a_i|^2 + |b_i|^2) = \int_{-\pi}^{\pi} f(x)^2 dx.$$

**Theorem 11 (Riemann-Lebesgue lemma).** *The following is provable in  $\text{RCA}_0$ . Let  $f$  be an effectively integrable continuous function on  $\mathbb{R}$ . Then, for all  $a, b \in \mathbb{R}$ ,*

$$\lim_{n \rightarrow \infty} \int_a^b f(x) \cos nx dx = 0.$$

**Theorem 12 (pointwise convergence).** *The following is provable in  $\text{RCA}_0$ . Let  $f \in \mathcal{P}^{2\pi}$  be of bounded variation on  $[-\pi, \pi]$ , i.e., there exist monotone increasing functions  $g_0, g_1$  such that  $f = g_0 - g_1$ . Then,  $f$  is effectively integrable and  $S_n[f]$  pointwise converges to  $f$ .*

**Theorem 13.** Let  $f_1, f_2 \in \mathcal{P}^{2\pi}$  be effectively integrable and let  $x_0 \in \mathbb{R}$ . Let  $f_1 \equiv f_2$  on some neighborhood of  $x_0$ . Then,  $S_n[f_1](x_0)$  converges if and only if  $S_n[f_2](x_0)$  converges. Moreover, if  $S_n[f_1](x_0)$  converges, then,

$$\lim_{n \rightarrow \infty} S_n[f_1](x_0) = \lim_{n \rightarrow \infty} S_n[f_2](x_0).$$

Finally, we argue about local approximation for continuous functions by trigonometric functions.

**Theorem 14.** The following assertions are equivalent over  $\text{RCA}_0$ .

1.  $\text{WWKL}_0$ .
2. If  $f$  is a continuous function on  $\mathbb{R}$  and  $x_0 \in \mathbb{R}$ , then there exist real sequences  $\{a_k\}_{k \in \mathbb{N}}$  and  $\{b_k\}_{k \in \mathbb{N}}$  such that  $S_n[\{a_k\}\{b_k\}]$   $L^2$ -converges to  $f$  on a neighborhood of  $x_0$ .

*Proof.* We reason within  $\text{RCA}_0$ .  $1 \rightarrow 2$  is a straightforward consequence of Theorem 9. For the converse, we show  $\neg 1 \rightarrow \neg 2$ . We reason within  $\text{RCA}_0$ . By  $\neg \text{WWKL}_0$ , define a continuous function  $f$  on  $[-\pi, \pi]$  as in the proof of  $2 \rightarrow 1$  of Theorem 9. Then, define continuous functions  $f_i$  on  $[0, \pi/2^i]$  as  $f_i(x) = 2^{-i}f(2^{i+1}x - \pi)$ . Note that  $|f_i| \leq 2^{-i+1}$ . Thus, we can define a continuous function  $\bar{f}$  on  $[-\pi, \pi]$  as

$$\bar{f}(x) := \begin{cases} f_{i+1}(x + \frac{\pi}{2^i}) & x \in [\frac{-\pi}{2^i}, \frac{-\pi}{2^{i+1}}], \\ f_{i+1}(x - \frac{\pi}{2^{i+1}}) & x \in [\frac{\pi}{2^{i+1}}, \frac{\pi}{2^i}], \\ 0 & x = 0. \end{cases}$$

Let  $U$  be a neighborhood of 0. Then, there exists  $i \in \mathbb{N}$  such that  $[\frac{\pi}{2^{i+1}}, \frac{\pi}{2^i}] \subseteq U$ . As in the proof of Theorem 9, there are no real sequences  $\{a_k\}_{k \in \mathbb{N}}$  and  $\{b_k\}_{k \in \mathbb{N}}$  such that  $S_n[\{a_k\}\{b_k\}]$   $L^2$ -converges to  $\bar{f}$  on  $[\frac{\pi}{2^{i+1}}, \frac{\pi}{2^i}]$ . This means that  $\neg 2$  and this completes the proof of  $2 \rightarrow 1$ .

## References

1. S. G. Simpson. *Subsystems of Second Order Arithmetic*. Springer-Verlag, 1999.
2. Elias M. Stein and Rami Shakarchi. *Fourier Analysis*. Princeton Lectures in Analysis. Princeton University Press, 2003.
3. Keita Yokoyama. *Standard and Non-standard Analysis in Second Order Arithmetic*. Doctoral thesis, Tohoku University, December 2007.
4. X. Yu and S. G. Simpson. Measure theory and weak König's lemma. *Archive for Mathematical Logic*, 30:171–180, 1990.

# Induced Matchings in Graphs of Maximum Degree Three

Grazyna Zwoźniak

Institute of Computer Science, Wrocław University, Poland  
[grazyna@ii.uni.wroc.pl](mailto:grazyna@ii.uni.wroc.pl)

**Abstract.** An induced matching is a matching  $M$  where each two edges  $e_1, e_2 \in M$  are at distance greater than one. In this paper we consider the problem of finding maximum induced matchings in graphs of maximum degree three. The problem is NP-hard. We present an algorithm which finds the maximum induced matching of size at least  $n/6 + O(1)$ , where  $n$  is the number of vertices in a graph. Using this bound we achieve an approximation ratio of 1.8 in cubic graphs.

**Key words:** induced matching, graphs, cubic, approximation algorithm

## 1 Introduction

Given a connected undirected graph  $G = (V, E)$ , the maximum induced matching problem is to find a maximum set of edges  $M$  which fulfils the following conditions: (a) no two edges of  $M$  share a common vertex (b) no two edges of  $M$  are joined by an edge of  $G$ . Induced matchings are also referred to as *strong matchings* [6, 7]. The problem has received considerable attention in the discrete mathematics community, since finding large induced matchings is a subtask of finding a strong edge-colouring in a graph [4, 5, 11, 14], a proper colouring of the edges, where no edge is incident to two edges of the same colour. There is also immediate connection between the size of an induced matching and the irredundancy number of a graph [7]. Stockmeyer and Vazirani [15] motivate the problem as the risk-free marriage problem, where each married person is compatible with no married person other than the one he (or she) is married to. On practical side, induced matchings have the applications for secure communication channels and network flow problems [8].

Cameron [2] showed that the maximum induced matching problem is NP-hard for bipartite graphs. Ko and Shepherd [12] proved the NP-hardness for cubic planar graphs. The problem was shown to be polynomial for chordal graphs and for interval graphs by Cameron [2]. Golumbic and Laskar [7] gave a polynomial time algorithm for circular arc graphs. Golumbic and Lewenstein [8] constructed polynomial time algorithms for trapezoid graphs, interval-dimension graphs and cocomparability graphs, and a linear time algorithm for the maximum induced matching problem in interval graphs. The linear time algorithms for trees were presented by Fricke and Laskar [6], Zito [16], Golumbic and Lewenstein [8].

There are several papers that focus on finding maximum induced matching in  $d$ -regular graphs. Zito [16] showed that for every  $k \geq 1$  there is a constant  $c > 1$  such that the problem of approximating maximum induced matching within a factor of  $c$  on  $4k$ -regular graphs is NP-hard. He also showed that the problem is approximable within  $d$ .

Duckworth, Manlove and Zito [3] proved that for any  $\epsilon > 0$  it is NP-hard to approximate maximum induced matching within a factor of  $\frac{1260}{1259} - \epsilon$  in graphs of maximum degree three and within a factor of  $\frac{7420}{7419} - \epsilon$  in cubic graphs. They also presented a greedy algorithm for approximating a maximum induced matching in  $d$ -regular graphs with a factor of  $d - 1$ . Gotthilf and Lewenstein [9] provided a simple greedy approach that yields an  $0.8d$  approximation factor.

In our paper we concentrate on graphs of maximum degree three. We present an algorithm which finds  $IM(G)$  – an induced matching of a graph  $G$  of size  $n/6 + O(1)$ . The approach to the problem is quite novel. Firstly a forest  $F$  with large number of leaves is constructed. Then the algorithm processes small subtrees of the trees  $T \in F$  and adds some edges to  $IM(G)$ . If after this step  $IM(G)$  is not maximal then some edges  $e \in E(G) \setminus E(F)$  are added to  $IM(G)$ . The properties of the forest  $F$  let us significantly reduce the number of possible cases which we have to consider looking at the subgraphs of  $G$ . It was shown [16] that the size of an optimum induced matching in a cubic graph is at most  $3n/10$ , so using our bound we achieve an approximation ratio of 1.8 for the maximum induced matching problem in cubic graphs.

The paper is organized as follows. In Section 2 we introduce some notation. In Section 3 we show how to construct the forest  $F$  and give some of its properties. In Section 4 we present the second part of the algorithm – adding edges to  $IM(G)$ . In Section 5 we give the main contribution of our paper. The proofs of the facts, lemmata and the theorem can be found in the full version of the paper.

## 2 Preliminaries

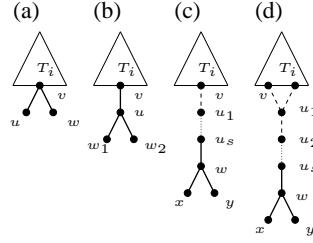
Let  $G$  be a connected undirected graph. We use  $V(G)$  to denote the set of vertices in  $G$  and  $E(G)$  to denote the set of edges in  $G$ . For a vertex  $v \in V(G)$  let  $\Gamma_G(v)$  denote the set of vertices  $\{w : (v, w) \in E(G)\}$ . The degree of  $v$  in  $G$ ,  $\deg_G(v)$ , is the number of edges incident to  $v$  in  $G$ . If  $T$  is a rooted tree, then we use  $LCA_T(u, w)$  to denote the lowest common ancestor of vertices  $u, w$  in  $T$ ,  $f(v)$  to denote the father of a vertex  $v \in V(T)$ , and  $h(T)$  to denote the height of the tree  $T$ . By  $L(T)$  we denote the set of leaves of  $T$ .

If  $u, v \in V(G)$  then  $dist(u, v)$  denotes the shortest distance between  $u, v$ . If  $e_1, e_2 \in E(G)$  then  $dist(e_1, e_2)$  denotes the shortest distance between  $e_1, e_2$ .

## 3 Construction of the forest $F$

In this section we consider a connected undirected graph  $G$  of maximum degree three, where at least one vertex  $v \in V(G)$  has degree three. We present the algorithm which constructs the forest  $F$  for  $G$ . This algorithm was also used in [13] and [17], were the forest  $F$  was the base structure for the presented algorithms.

In the first step our algorithm builds successive trees  $T_0, \dots, T_k$  of a forest  $F$ . Rule 1 puts to the tree  $T_i$  two vertices  $u, w \notin V(F)$  adjacent to a leaf  $v \in L(T_i)$ , see Fig. 1(a). Rule 2 puts to  $T_i$  a vertex  $u \notin V(F)$  adjacent to a leaf  $v \in L(T_i)$  together with both further neighbours  $w_1, w_2$  of  $u$ , where  $w_1, w_2 \notin V(F)$ , see Fig. 1(b). We name the two leaves added by Rules 1 and 2 the left and the right son of their father.



**Fig. 1.** (a) Rule 1. (b) Rule 2. (c), (d) Rule 3.

Rule 3 initiates a new tree  $T_j$ ,  $j > i$ , see Fig. 1(c)-(d). Let  $P$  be a path which starts at  $v \in L(T_i)$ , goes through vertices  $u_1, \dots, u_s \notin V(F)$ ,  $s \geq 1$  and ends at the first vertex  $w \notin V(F)$ , where  $\deg_G(w) = 3$ ,  $\Gamma_G(w) = \{u_s, x, y\}$  and  $x, y \notin V(F)$ . The vertex  $u_s$  precedes  $w$  on  $P$ . Rule 3 starts to build  $T_j$  rooted at  $u_s$  and adds  $u_s, w, x, y$  to  $V(T_j)$  and  $(u_s, w), (w, x), (w, y)$  to  $E(T_j)$ . We refer to  $T_i$  as the father of  $T_j$ . This relation determines a partial order in  $F$ , so we use some other related terms as e.g. ancestor of the tree.

Let  $F = \{T_0, \dots, T_k\}$ . In our algorithm and analysis we use the following notions:

**Definition 1.** A vertex  $v \in V(G)$  is an exterior vertex if  $v \notin V(F)$ . Let  $EX$  denote the set of all exterior vertices in  $G$ .

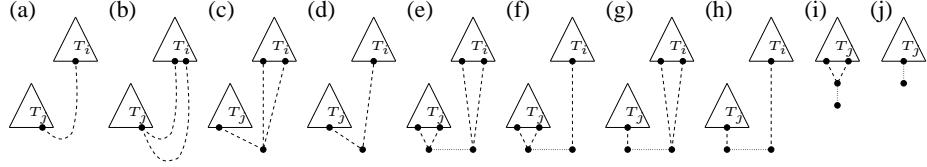
**Definition 2.** Let  $r_0$  be a root of  $T_0$ . Let  $R$  be the set of roots of the trees  $T_1, \dots, T_k$ .

The algorithm works as follows.

CONSTRUCT\_F( $G$ ).

1.  $F \leftarrow \emptyset$
2.  $V(T_0) \leftarrow \{r_0, v_1, v_2, v_3\}$ , where  $r_0$  is any degree three vertex of  $G$  and  $v_1, v_2, v_3 \in \Gamma_G(r_0)$ ;  $E(T_0) \leftarrow \{(r_0, v_1), (r_0, v_2), (r_0, v_3)\}$ ; let  $r_0$  be a root of  $T_0$ ;  $i \leftarrow 0$
3. if it is possible: find the leftmost leaf in  $T_i$  that can be expanded by the Rule 1 and expand it; go to step 3;  
else: go to step 4;
4. if it is possible: find the leftmost leaf in  $T_i$  that can be expanded by the Rule 2 and expand it; go to step 3  
else:  $F \leftarrow F \cup T_i$  and go to step 5
5. if it is possible: find the leftmost leaf  $v$  in  $T_i$  such that Rule 3 can be applied to  $v$  and apply this rule to  $v$ ;  $i \leftarrow i + 1$ ; let  $T_i$  be a new tree created in this step; go to step 3 with  $T_i$   
else: if  $T_i$  has a father: go to step 5 with the father of  $T_i$   
else: return  $F$ .

The properties of the forest  $F$  are described by Fact 1. The possibilities for edges  $e \in E(G) \setminus E(F)$  are presented in Fig. 1(c),(d) and 2.



**Fig. 2.** The edges  $(x, y) \in E(G) \setminus E(F)$ ,  $x, y \in EX \cup L(F)$  (dashed lines). Either  $T_i = T_j$  or  $T_i$  is an ancestor of  $T_j$ . Dotted lines denote the paths where all interior vertices have degree two in  $G$ .

**Fact 1** Let  $v \in V(T)$ ,  $T \in F$ .

1. If  $\deg_T(v) = 2$  and  $w \in \Gamma_T(v)$  then  $\deg_T(w) = 3$ .
2. If  $\deg_T(v) = 2$  and  $w \in \Gamma_G(v)$  then  $w \in V(T)$ .
3. If  $v$  is adjacent to the vertex  $w \in EX \cup R$ ,  $u \in \Gamma_G(v)$  and  $u \neq w$  then  $u \in V(T)$ .

The following facts let us consider the subtrees of the trees  $T \in F$  in some order described precisely in the next section.

**Fact 2** Let  $\deg_T(u) = 2$ ,  $\deg_T(w) = 2$ ,  $T \in F$ ,  $(u, w) \in E(G) \setminus E(F)$ ,  $a = LCA_T(u, w)$  and let  $u$  be on the left of  $w$  in  $T$ . Then  $w$  is the right son of  $a$ , and there are no vertices of degree 2 on the path from  $u$  to  $a$  in  $T$ .

**Fact 3** Let  $r$  be a root of a tree  $T \in F$ . Let  $u \in L(T)$  and let  $w$  be the first vertex of degree 2 on the path from  $u$  to  $r$  in  $T$ . If there is  $v \in V(T)$  such that  $\deg_F(v) = 2$  and  $(u, v) \in E(G) \setminus E(T)$  then  $w$  is an ancestor of  $v$  in  $T$ .

#### 4 Induced matching

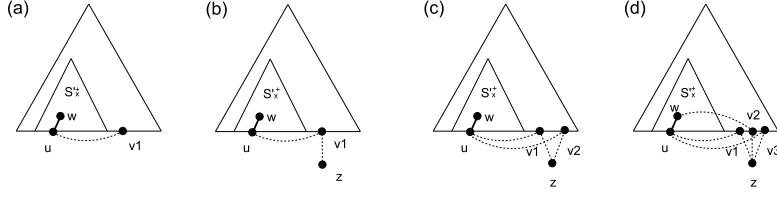
Let  $G'$  denote a current graph,  $F' = G' \cap F$ ,  $EX' = V(G') \setminus V(F')$ . At the beginning  $G' = G$ ,  $F' = F$  and  $EX' = EX$ . In successive steps of our algorithm we add some edges  $(u, v)$  to  $IM(G)$  and remove from  $E(G')$  these added edges with all edges  $(w, z)$  where  $(u, v), (w, z)$  are at distance at most one. A vertex  $u$  is removed from  $V(G')$  when the last edge incident to  $u$  is removed from  $E(G')$ .

Now we present the idea of our algorithm, precise description is given later.

There are two main phases of our algorithm. After the first one all edges  $E(F')$  and some edges  $E(G') \setminus E(F')$  are deleted from  $E(G')$ . Moreover, after this phase all vertices in  $V(G')$  have degree at most two. After the second phase the remaining edges  $E(G') \setminus E(F')$  are removed from  $E(G')$ .

Each phase consists of the steps. In a single step some edges  $\bar{E}$  are added to  $IM(G)$  and appropriate edges  $E_d$  and vertices  $V_d$  are deleted from  $G'$ . Our goal is to choose  $\bar{E}$  in such a way that  $|V_d| \leq 6|\bar{E}|$ .

Since after the first phase every connected component of  $G'$  is either a path or a simple cycle, the way of adding some of their edges to  $IM(G)$  is obvious. Now we will describe more precisely the first phase.



**Fig. 3.** Vertices  $v_1, v_2, v_3$  and  $z$  would be removed from  $V(G')$  if the edge  $(u, w)$  was added to  $IM(G)$ . In the cases (b)-(d) there exist edges  $(v_i, f(v_i)), (f(v_i), f(f(v_i))) \in E(F')$ .

Let  $T' = G' \cap T$ , where  $T \in F$ . Let  $S'_x$  denote the subtree of  $T'$  rooted at some vertex  $x \in V(T')$ . We define  $S'^+_x$  as a tree, where  $E(S'^+_x) = E(S'_x) \cup E^+$  and  $E^+ \subseteq E(G') \setminus E(F')$  denotes some edges incident to  $V(S'_x)$  (precise definition of  $S'^+_x$  is given later). We process bottom-up small subtrees  $S'_x$  of  $T'$  and we look for the edges  $\bar{E} = \{(u, w) : u, w \in V(S'^+_x), u, w \neq x\}$  such that if we added  $\bar{E}$  to  $IM(G)$  then

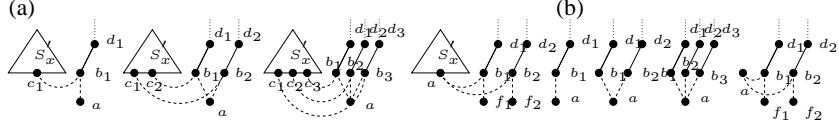
1. the set  $E(S'_x)$  would be empty;
2. the equality  $E(T') = E(S'_r)$  would hold, where  $r$  is the root of  $T$ ;
3. the vertices  $v \in EX'$  with  $\deg_{G'}(v) = 3$  and some neighbours in  $V(S'_x)$  would be removed or would change degree into 1 or 2.

In the special cases when  $x \in \{r_0\} \cup R$  and  $h(S'^+_x) = 1$  we add to  $IM(G)$  an edge incident to  $x$ .

The special care is needed when  $u \in V(S'_x)$ ,  $v \notin V(S'_x)$ ,  $\deg_{T'}(v) = 2$  and  $(u, v) \in E(G')$ , because if we added an edge incident to  $u$  to  $IM(G)$  then the equality  $E(T') = E(S'_r)$  might not hold (there could be two connected components:  $S'_r$  and the tree rooted at the son of  $v$ ). That is why in our algorithm the vertices  $y$ , where  $\deg_{T'}(y) = 2$  and  $\deg_{G'}(y) = 3$ , are treated as "milestones". More precisely, we consider in post-order subtrees  $S'_y$  of  $T'$ , where  $(\deg_{T'}(y) = 2 \text{ and } \deg_{G'}(y) = 3)$  or ( $y$  is the root of  $T$ ,  $T' = G' \cap T$ ), and process bottom-up subtrees  $S'_x$  of  $S'_y$ .

When we process  $S'^+_x$  some vertices  $v \notin V(S'^+_x)$  may be removed. To reduce the number of possible cases which we have to consider looking at these vertices we make some preprocessing and add to  $IM(G)$  some edges  $e \in E(G') \setminus E(F')$  which are close to  $S'_x$ . Later we will define precisely the sets  $C_1(S'_x)$  and  $C_2(S'_x)$  of such edges. After this operation we process  $S'^+_x$  and add some edges  $\bar{E}$  to  $IM(G)$ . Now a vertex  $z \notin V(S'^+_x)$  is removed if either  $z \in N$  or  $\Gamma_{G'}(z) \subseteq N$ , where  $N = \{v : (u, v) \in E(G'), u \in V(S'^+_x), v \notin V(S'^+_x) \text{ and } (u, w) \in \bar{E} \text{ for some } w \in V(S'^+_x)\}$ , see Fig. 3.

In our algorithm we use some kind of accounting analysis. We add some edges  $\bar{E}$  to  $IM(G)$  if they remove at most  $6|\bar{E}|$  vertices from  $V(G')$ . But in some cases we let the number of removed vertices be larger, let us say  $6|\bar{E}| + x$ . In these cases we choose  $x$  vertices which are still in  $V(G')$  and mark them. Later these marked vertices will be double counted in the process of adding edges to  $IM(G)$ . Every chosen vertex may be marked at most once. It is possible to mark a vertex  $a \in V(G')$  if



**Fig. 4.** A vertex  $a$  may be marked. (a) situations before adding  $\bar{E}$  to  $IM(G)$ , (b) situations after adding  $\bar{E}$  to  $IM(G)$ . It is possible that  $c_{i_1} = c_{i_2}$ ,  $i_1, i_2 \in \{1, 2, 3\}$ .

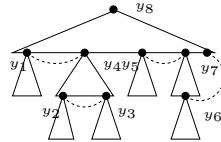
1. before adding edges  $\bar{E}$  to  $IM(G)$  there are edges  $(c_i, b_i), (b_i, a) \in E(G')$ ,  $(b_i, d_i) \in E(F')$ , such that  $\deg_{G'}(a) = j$ ,  $1 \leq j \leq 3$ ,  $1 \leq i \leq j$ , and after adding edges  $\bar{E}$  to  $IM(G)$  all edges  $(c_i, b_i)$  would be removed,  $a \in V(G')$ , at least one  $b_i \in V(G')$ , and if  $b_i \in V(G')$  then  $\deg_{G'}(b_i) = 2$ , or
2. before adding edges  $\bar{E}$  to  $IM(G)$  there are edges  $(a, b_i), (b_i, f_i) \in E(G')$ ,  $(b_i, d_i) \in E(F')$ ,  $\deg_{G'}(a) = 3$ ,  $\deg_{G'}(f_i) = 1$ , where  $i = 1, 2$  and adding edges  $\bar{E}$  to  $IM(G)$   $\deg_{G'}(a) = 2$  and  $(a, b_i), (b_i, f_i), (b_i, d_i) \in E(G')$ , see Fig. 4.

The choice of vertices which can be marked is intentional. In most of these cases we do not need to propagate the marking process, because we are sure that marked vertices are in the trees, where it is possible to add  $y$  edges to  $IM(G)$  and remove less than  $6y$  vertices from  $V(G')$ .

The algorithm starts with the procedure `CONSTRUCT_IM( $G'$ )`, which realizes two main phases of our algorithm: the first one which removes  $E(F')$  (step 1 of the procedure) and the second one which removes the remaining edges of  $G'$  (steps 2 and 3 of the procedure).

The procedure `MATCHING( $T'$ )` applies the procedure `MATCH()` to the subtrees of  $T'$ .

The procedure `MATCH( $S'_y$ )` starts with preprocessing steps. Then it tries to remove the edges of the trees  $S'_x$ , where  $x \in V(S'_y)$  and  $h(S'_x) = 2$ . If it is not possible, trees  $S'_x$  of height three are considered. If it is not possible, trees  $S'_x$  of height four are considered. If it is impossible to remove the edges of any tree of height two, three or four and there is a tree  $S'_x$  of height five then some edges are added to  $IM(G)$  and  $E(S'_x)$  are removed. In the special case when  $x = r_0$ ,  $\deg_{G'}(x) = 3$  and  $h(S'_x) > 1$  we split  $S'_x$  into two trees and successively process them.



**Fig. 5.** The tree  $T' \in F'$  rooted at  $y_8$ . The algorithm considers successively subtrees rooted at  $y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8$ .

**CONSTRUCT\_IM( $G'$ )**

1. apply the procedure MATCHING() in post-order to  $T'$ ;
2. while there is a path  $\{(v_1, v_2), \dots, (v_{k-1}, v_k)\}: \text{ADD}((v_{3i+1}, v_{3i+2}))$ ,  
 $i = 0, \dots, \frac{k-2}{3}$ ;
3. while there is a cycle  $\{(v_1, v_2), \dots, (v_k, v_1)\}: \text{ADD}((v_{3i+1}, v_{3i+2}))$ ,  
 $i = 0, \dots, \frac{k-3}{3}$ .

**MATCHING( $T'$ )**

1. apply the procedure MATCH() in post-order to the subtrees  $S'_y$  of  $T'$  such that  
 $(\deg_{T'}(y) = 2 \text{ and } \deg_{G'}(y) = 3) \text{ or } (y \text{ is the root of } T, T' = G' \cap T)$ .

**MATCH( $S'_y$ )**

1. if  $h(S'_y) > 0$  apply the procedure MATCH() to the trees rooted at the sons of  $y$ ;
2. while there is an edge  $(a, b) \in C_1(S'_y)$ : ADD( $(a, b)$ );
3. while there is an edge  $(a, b) \in C_2(S'_y)$ : ADD( $(a, b)$ );
4. for ( $i = 2; i \leq 5; i++$ ):
5. if there is a tree  $S_x'^+$ , where  $x \in V(S_x'^+)$ ,  $x \neq r_0$  or  $\deg_{F'}(x) \neq 3$ ,  $h(S_x'^+) = i$  and  $M(S_x'^+) \neq \emptyset$ : MAKE\_MATCH( $S_x'^+$ ) and goto 2;
6. if  $h(S_y'^+) = 1$  and  $y \in \{r_0\} \cup R$  or  $|E(G')| = 1$ : ADD( $(y, u)$ ) where  $u$  is a son of  $y$ ;
7. if  $y = r_0$  and  $\deg_{F'}(y) = 3$ :
  - let  $S_y'^*$  be a part of  $S_y'^+$  where  $E(S_y'^*) = E(S_{u_1}'^+) \cup E(S_{u_2}'^+) \cup (y, u_1) \cup (y, u_2)$  and  $h(S_{u_1}'^+) \geq h(S_{u_2}'^+) \geq h(S_{u_3}'^+)$ ;
  - (a) MAKE\_MATCH( $S_y'^*$ );
  - (b) if  $E(S_y'^+) \neq \emptyset$ : MAKE\_MATCH( $S_{u_3}'^+$ ).

**MAKE\_MATCH( $S_x'^+$ )**

1. ADD( $M(S_x'^+)$ );
2. while there is an edge  $e \in I(T')$ : ADD( $e$ ).

Now we will describe notations used in our procedures.

Let  $V_u(\bar{E})$  ( $V_m(\bar{E})$ ) denote the set of unmarked (marked) vertices which would be removed from  $V(G')$  if the edges  $\bar{E}$  were added to  $IM(G)$ . The process of adding edges to  $IM(G)$  is realized by the procedure ADD( $\bar{E}$ ) which adds the edges  $\bar{E}$  to  $IM(G)$  and removes appropriate vertices and edges from  $G'$ . Moreover, if  $|V_u(\bar{E})| + 2|V_m(\bar{E})| = 6|\bar{E}| + x$  and  $x > 0$  then the procedure ADD( $\bar{E}$ ) marks  $x$  vertices which are still in  $V(G')$ . The only exception is the situation when the last tree is processed. In

this case we have  $x = 0$ , but there are some cases when the number of removed vertices may be larger than  $6|\bar{E}|$ . That is why the size of  $IM(G)$  is  $n/6 + O(1)$ .

In the procedure  $\text{MATCH}(S'_y)$  subtrees  $S'_y$  of  $T' \in F'$  are considered. Let  $C_1(S'_y)$  and  $C_2(S'_y)$  be the sets of the edges which are removed in preprocessing phase (steps 2 and 3 of  $\text{MATCH}(S'_y)$ ).  $C_1(S'_y)$  contains the edges  $(v_1, v_2) \in E(G') \setminus E(F')$  such that  $v_1, v_2 \in EX'$ ,  $\deg_{G'}(v_1) = 3$  and  $v_1$  is at distance at most two from some vertex  $w \in V(S'_y)$ .

$C_2(S'_y)$  contains the edges  $e \in E(G') \setminus E(F')$  such that

1.  $e$  is at distance one from some leaf of  $S'_y$  or  $e$  is at distance two from some interior vertex of  $S'_y$ ;
2. for every  $S'_z \subseteq F'$ :  $e$  is at distance at least one from any leaf of  $S'_z$ ;
3. for every  $S'_z \subseteq F'$ :  $e$  is at distance at least two from any interior vertex of  $S'_z$ .

Let  $P'$  be the set of the edges which can be processed together with  $S'_y$ .  $P'$  contains the following paths:

1.  $P = \{(v_1, v_2)\}$ , where  $v_1 \in L(S'_y)$ , and  $v_2$  is not the endpoint of any edge  $e \in E(F')$ .
2.  $P \setminus (v_{k-1}, v_k)$ , where  $P = \{(v_1, v_2), \dots, (v_{k-1}, v_k)\}$ ,  $k \geq 3$ ,  $v_1, v_k \in V(S'_y)$ ,  $(v_{i-1}, v_i) \in E(G') \setminus E(F')$  where  $i = 2, \dots, k$ , and  $\deg_{G'}(v_j) = 2$  where  $j = 2, \dots, k-1$ .

We define a tree  $S'_y^+$  as  $S'_y \cup P''$ , where  $P'' \subseteq P'$  and if  $e \in P' \setminus P''$  then  $S'_y^+ \cup \{e\}$  contains a cycle.

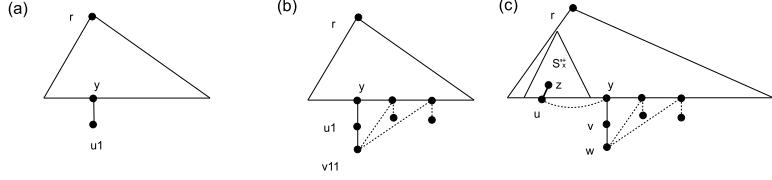
When we consider some tree  $S'_x^+$ , where  $x \in V(S'_y^+)$  then we select a set of the edges  $M(S'_x^+)$  which can be added to  $IM(G)$ . This set fulfills the following conditions:

1. if  $h(S'_x^+) \geq 2$  and  $(a, b) \in M(S'_x^+)$  then  $a, b \in V(S'_x^+) \setminus \{x\}$ ;
2. if  $h(S'_x^+) = 1$  and  $f(x)$  does not exist or is not in  $V(F')$  then  $a, b \in V(S'_x^+)$ ;
3. if  $e_1, e_2 \in M(S'_x^+)$  then  $\text{dist}(e_1, e_2) > 1$ ;
4. if  $|V_u(M(S'_x^+))| + 2|V_m(M(S'_x^+))| = 6|M(S'_x^+)| + z$  and  $z > 0$  then at least  $z$  vertices can be marked;
5. if we added  $M(S'_x^+)$  to  $IM(G)$  then the set  $E(S'_x^+)$  would be empty.

If it is possible to find such a set ( $M(S'_x^+) \neq \emptyset$ ) then we add  $M(S'_x^+)$  to  $IM(G)$ . The choice of  $M(S'_x^+)$  depends on the shape of  $S'_x^+$  and the edges incident to  $S'_x^+$ . When  $h(S'_x^+) = 2$  then in most of the cases it is possible to choose the edges presented in Fig. 7. Technical details are presented in the full version of the paper.

Let us introduce one more definition. We define  $I(T')$  as  $E(T') \setminus E(S_r)$ , where  $r$  is the root of  $T$  and  $T' = T \cap G'$ . Such edges may appear only if before removing  $E(S'_x^+)$  some vertex  $u$  of  $S'_x^+$  was connected by an edge with a vertex  $y$  of degree two in  $F'$ , and the tree rooted at  $y$  was not completely removed by  $\text{MATCH}(S'_y)$ , see Fig. 6.

In our procedures we use the following notions: vertices  $u_i$  are the sons of  $x$ , vertices  $v_{i,j}$  are the sons of  $u_i$ , vertices  $w_{i,j,k}$  are the sons of  $v_{i,j}$ ,  $i \in \{1, 2, 3\}$ ,  $j, k \in \{1, 2\}$ .



**Fig. 6.** (a) After  $\text{MATCH}(S'_y)$  one edge  $(y, u_1) \in E(S'_y)$  may be still in  $E(G')$ . In such a case  $\deg_{G'}(u_1) = 1$ . (b) After  $\text{MATCH}(S'_y)$  two edges  $(y, u_1), (u_1, v_{1,1}) \in E(S'_y)$  may be still in  $E(G')$ . In such a case  $\deg_{G'}(u_1) = 2$  and there are edges  $(v_{1,1}, b_i), (b_i, a_i), (b_i, c_i), \in E(G')$ ,  $i \in \{1, 2\}$  where  $\deg_G(a_i) = 1$  and  $(b_i, c_i), \in E(F')$ . (c) If we add  $(u, z)$  to  $IM(G)$  then  $(y, v)$  will be removed and  $(v, w)$  will be in  $E(T') \setminus E(S'_r)$

## 5 Main theorem

The main contribution of this paper is the following theorem.

**Theorem 1.** *Let  $G$  be a connected graph of maximum degree three where  $|V(G)| = n$ . Our algorithm constructs for  $G$  an induced matching  $IM(G)$  of size at least  $n/6 + O(1)$  in linear time.*

The following lemmata are used in the proof of Theorem 1.

**Lemma 1.** *Let us consider the procedure CONSTRUCT\\_IM( $G'$ ).*

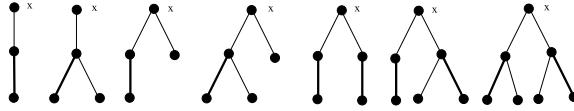
1. *After step 1  $E(F') = \emptyset$  and  $\forall_{v \in V(G')} \deg_{G'}(v) \leq 2$ .*
2. *After step 2  $\forall_{v \in V(G')} \deg_{G'}(v) = 2$ .*
3. *After step 3  $V(G') = \emptyset$ .*

**Lemma 2.** *After step 1 of the procedure MATCHING( $T'$ )  $E(T') = \emptyset$ .*

**Lemma 3.** *Let us consider the procedure MATCH( $S'_y$ ).*

1. *If  $y \in R \cup \{r_0\}$  then after  $\text{MATCH}(S'_y)$   $E(S'_y) = \emptyset$ .*
2. *If  $\deg_{T'}(y) = 2$  and  $\deg_{G'}(y) = 3$  then after  $\text{MATCH}(S'_y)$  at most two edges of  $S'_y$  may be still in  $E(G')$ , see Fig. 6(a)-(b).*
3. *If  $e \in C_1(S'_y)$  and the procedure ADD( $e$ ) is called in step 2 of the procedure  $\text{MATCH}(S'_y)$  then it removes at most 6 vertices from  $V(G')$ . All these vertices are unmarked.*
4. *If  $e \in C_2(S'_y)$  and the procedure ADD( $e$ ) is called in step 3 of the procedure  $\text{MATCH}(S'_y)$  then it removes at most 6 vertices from  $V(G')$ . All these vertices are unmarked.*

**Lemma 4.** *If  $h(S'_y^+) \geq 5$  then it is possible to find a tree  $S_x^{'+}$ , where  $x \in V(S_y^{'+})$  and  $M(S_x^{'+}) \neq \emptyset$ .*



**Fig. 7.** Basic shapes of  $S'_x^+$  where  $h(S'_x^+) = 2$ . The edges added to  $IM(G)$  are bold.

## References

1. P. Alimonti, V. Kann *Some APX-completeness results for cubic graphs*, Theoretical Computer Science, 237, pages 123–134, 2000.
2. K. Cameron *Induced matchings*, Discrete Applied Mathematics, 24, pages 97-102, 1989.
3. W. Duckworth, D. F. Manlove, M. Zito *On the approximability of the maximum induced matching problem*, The Journal of Discrete Algorithms, 3(1), pages 79-91, 2005.
4. P. Erdős *Problems and results in combinatorial analysis and graph theory*, Discrete Mathematics, 72, pages 81-92, 1988.
5. R. J. Faudree, A. Gyárfás, R. H. Schelp, Z. Tuza *Induced matchings in bipartite graphs*, Discrete Mathematics, 78, pages 83-87, 1989.
6. G. Fricke, R. Laskar *Strong matchings in trees*, Congressus Numerantium, 89, pages 239-244, 1992.
7. M. C. Golumbic, R. C. Laskar *Irredundancy in circular arc graphs*, Discrete Applied Mathematics, 44, pages 79-89, 1993.
8. M. C. Golumbic, M. Lewenstein *New results on induced matchings*, Discrete Applied Mathematics, 101, pages 157-165, 2000.
9. T. Gotthilf, M. Lewenstein *Tighter Approximations on Greedy for Maximum Induced Matchings in Regular Graphs*, Third Workshop on Approximation and Online Algorithms, LNCS 3879, pages 270-281, 2005.
10. R. Greenlaw, R. Petreschi *Cubic graphs*, ACM Computing Surveys, 27, pages 471-495, 1995.
11. P. Horák, H. Qing, W. T. Trotter *Induced matchings in cubic graphs*, Journal of Graph Theory, 17(2), pages 151-160, 1993.
12. C. W. Ko, F. B. Shepherd *Adding an identity to a totally unimodular matrix*, London School of Economics Operations Research Working Paper LSEOR.94.14, 1994.
13. K. Loryś, G. Zwoźniak *Approximation algorithm for the maximum leaf spanning tree problem for cubic graphs*, Proceedings of the 10th Annual European Symposium on Algorithms, LNCS 2461, pages 686–697, 2002.
14. A. Steger, M. Yu *On induced matchings*, Discrete Mathematics, 120, pages 291-295, 1993.
15. L. J. Stockmeyer, V. V. Vazirani *NP-completeness of some generalizations of the maximum matxhing problem*, Information Processing Letters, 15(1), pages 14-19, 1982.
16. M. Zito *Maximum induced matchings in regular graphs and trees*, The 25th International Workshop on Graph-Theoretic Concepts in Computer Science, LNCS 1665, pages 89-100, 1999.
17. G. Zwoźniak *Small independent edge dominating sets in graphs of maximum degree three*, Proceedings of the 32nd Conference on Current Trends in Theory and Practice of Computer Science, LNCS 3831, pages 556-564, 2006.

# Subsystems of Iterated Inductive Definitions

Bahareh Afshari and Michael Rathjen

University of Leeds, Leeds UK

In 1963, G. Kreisel [5] initiated the study of formal theories featuring inductive definitions. Subsystems of the theories of iterated inductive definitions ( $ID_n$ ) such as the fixed point theories  $\hat{ID}_n$  where investigated by Aczel and Feferman in connection with Hancock's conjecture about the strength of Martin-Löf type theories with universes. Another interesting type of theory lying between  $\hat{ID}_n$  and the usual  $ID_n$  is  $ID_n^*$ . To illustrate this in the case  $n = 1$ , in contrast to  $\hat{ID}_1$ ,  $ID_1^*$  has an induction principle for the fixed points but it is restricted to formulas in which other fixed points occur only positively. Results about the theories  $ID_n^*$  were obtained by Friedman, Feferman [4], and Cantini [2]. However, they did not settle the proof-theoretic strength of the theories  $ID_n^*$ . I would like to talk about our recent results revealing the strength of these theories.

## References

1. Wilfried Buchholz, Solomon Feferman, Wolfram Pohlers and Wilfried Sieg. *Iterated inductive definitions and subsystems of Analysis: Recent Proof-Theoretical Studies theories*, Springer-Verlag, Berlin, Heidelberg, 1981.
2. Andrea Cantini. A note on a predicatively reducible theory of elementary iterated induction, *Bollettino U.M.I.*, pp. 413-430, 1985.
3. Andrea Cantini. On the relation between choice and comprehension principles in second order arithmetic, *Journal of Symbolic Logic* **51**, pp. 360–373, 1986.
4. Solomon Feferman. Iterated inductive fixed-point theories: Application to Hancock's conjecture, *Patras Logic Symposium*, pp. 171–196, North-Holland, Amsterdam, 1982.
5. G. Kreisel. Generalized inductive definitions. Tech. rep., Stanford University, 1963.
6. Michael Rathjen. Auwahl und Komprehension in Teitsystemen der Analysis, M.Sc. thesis, University of Münster, Germany, 1985.
7. K. Schütte. *Proof Theory*, Springer-Verlag, Berlin, Heidelberg, 1977.
8. Helmut Schwichtenberg. Proof Theory: Some Applications of Cut-Elimination, *Handbook of Mathematical Logic*, pp. 868–895, North-Holland, 1977.
9. Stephen G. Simpson. *Subsystems of Second Order Arithmetic*, Springer-Verlag, Berlin, Heidelberg, 1999.

# Query Algorithms for Detecting Hamming and Reed-Solomon Codes \*

Rubens Agadžanjans

Institute of Mathematics and Computer Science  
University of Latvia, Raiņa bulv. 29, Rīga, LV-1459, Latvia.  
[ruben.agadzanyan@gmail.com](mailto:ruben.agadzanyan@gmail.com)

In this talk we will compare quantum and classical query complexity of some Boolean functions. This is the model where the Boolean function is known, but its arguments are unknown. So, to compute the function, the query algorithm asks for values of particular arguments. The complexity of the algorithm is the number of queries. Up to now there have been discovered some, though few, Boolean functions whose quantum query algorithm is better than classical. Here we will talk about another group of such functions, based on Hamming and Reed-Solomon error-correcting codes. There is a 25% improvement with the quantum algorithm for the Hamming codes and a 50% improvement for the Reed-Solomon codes (which repeats the previously best known achievement).

**Key words:** Boolean functions, polynomial degree, query algorithms, Hamming code, Reed-Solomon code.

## References

1. R. Agadzajans, J. Smotrovs. Efficient Quantum Query Algorithms Detecting Hamming and Reed-Solomon Codes. In *Proc. of the SOFSEM 06*, 2006
2. A. Ambainis. Polynomial degree vs. quantum query complexity. In *Proc. of the 44th IEEE FOCS*, 2003.
3. H. Buhrman and R. de Wolf. Complexity Measures and Decision Tree Complexity : A Survey. *Theoretical Computer Science*, v. 288(1): 21–43, 2002
4. R. Freivalds, M. Miyakawa, H. Tatsumi. An Exact Quantum Query Algorithm for a Specific Boolean Function. In *Proc. of the EQIS 04*, 2004.
5. J. Gruska. *Quantum Computing*. McGraw-Hill, 1999.
6. R. W. Hamming. Error detection and error correcting codes. *Bell System Technical Journal*, 26(2):147–160, 1950.
7. M. Nielsen, I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, New York, 700pp., 2000.
8. C. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, 500pp., 1994.
9. I. S. Reed, G. Solomon. Polynomial codes over certain finite fields. *J. Soc. Indust. Appl. Math.*, v.8, p.300–304, 1960.

---

\* Research supported by the European Social Fund.

# Expressive Power of Graph Logic

Timos Antonopoulos and Anuj Dawar

University of Cambridge

We present results on the expressive power of Graph Logic, a spatial logic for querying graphs introduced by Cardelli et al. [1] and studied further by Dawar et al. [2]. Graph Logic is an extension of First Order Logic with a second order quantifier over edges of restricted form, and a first order quantifier over labels of edges. In particular, if  $G$  is some graph, all one can do with the second order quantifier is express that there exists a set of edges  $X$  of the graph  $G$  such that some formula  $\phi$  is satisfied by the subgraph of  $G$  containing exactly the edges in  $X$ , and some formula  $\psi$  is satisfied by the subgraph of  $G$  with exactly the remaining edges not in  $X$ .

It has been observed that GL is a sublogic of Monadic Second Order Logic with quantification over edges and additional first order quantification over edge labels (MSO for short). Although it seems that GL is strictly less expressive than MSO, many interesting properties have been shown to be expressible in GL. Furthermore it was shown in [2] that GL is able to express complete problems on any level of the Polynomial Hierarchy and that GL and MSO are equi-expressive when restricted to words. Marcinkowski [3] showed that this richer form of MSO with quantification over edge labels, is more expressive than GL.

As this richer form of MSO is not the one we usually deal with, the case where we omit the first order quantification over edge labels from both logics is a more interesting one. We show that this restriction of GL is indeed strictly less expressive than the one of MSO. Moreover we show that this is the case even when restricted to the class of forests.

## References

1. L. Cardelli, P. Gardner and G. Ghelli, *A spatial logic for querying graphs*, ICALP 2002, Springer LNCS, 2380, 597–610.
2. A. Dawar, P. Gardner and G. Ghelli, *Expressiveness and Complexity of Graph Logic*, Information and Computation, **205** (2007) 263–310.
3. J. Marcinkowski, *On The Expressive Power of Graph Logic*, CSL 2006, Springer LNCS, 4207, 486–500.

# The Lost Melody Theorem: Infinite Time Register Machines

Merlin Carl and Peter Koepke

Mathematisches Institut, Universität Bonn, Germany

In ([1]) it is shown that, for ITTMs there are sets which are computable, but not writeable. We ask whether an analogous theorem holds for ITRMs as introduced in ([2]). We start with some definitions.

$P^x$  indicates that register machine program  $P$  is run with  $x$  as an oracle. A real  $r$  (a subset of  $\omega$ ) is **ITRM-decidable** (decidable) iff there is a program  $P$  such that  $P^x(0) = 1 \leftrightarrow x = r$  and the former is halts for every  $x$ . It is **ITRM-computable** (computable) iff there is  $P$  such that  $P^\emptyset(n) = 1 \leftrightarrow n \in r$ ,  $P(n)$  a total function. A countable  $\in$ -model  $M$  can be coded by a real by fixing a surjection  $s : \omega \rightarrow |M|$  and setting:  $r := \{i \in \omega : \exists m, n \in \omega [p(m, n) = i \wedge (b(m) \in b(n))^M]\}$ , where  $p$  is the Cantor pairing function.  $J_\alpha$  is the  $\alpha$ -th level of the Jensen hierarchy of constructible sets.

Now we can formulate our result:

Theorem: There is a real  $r$  which is decidable but not computable. In fact, we can take  $r$  to be the  $<_L$ -minimal code in the sense defined above for the  $\in$ -minimal model  $J_\alpha$  of  $ZF-$ .

Proof: Let  $r$  be as specified in the theorem statement. Then  $r$  is not computable, for if  $P$  computes  $r$ , the computation can be simulated in  $J_\alpha$  and since ITRM-computations are absolute between models  $ZF-$ , there is an  $\in$ -formula  $\phi$  such that  $\phi(n) \leftrightarrow n \in r$ , so  $r \in J_\alpha$ . But then  $r$  codes an element of  $J_\alpha$ , and  $\alpha$  is not minimal. But  $r$  is decidable: First, we check whether the  $\in$ -relation given by  $r$  is wellfounded, as described in [2]. From now on, we suppose that this is the case. The first surjection  $s : \omega^{>1} \rightarrow J_\alpha$  is an element of  $J_{\alpha+2}$ . So  $r \in J_{\alpha+2}$  and the question whether a real  $x$  satisfies the above conditions is equivalent to  $J_{\alpha+2} \models \phi(x)$  for some  $\phi$ . Now, elements of  $J_{\alpha+2}$  can be coded by terms of the form  $f(k_1, \dots, k_n)$ , where  $f$  is a composition of Goedel functions and  $k_i \in \omega$  for  $0 < i < n + 1$ , which again can be compressed into a single natural number by an appropriate use of  $p$ . Questions of this form are then solved recursively by unfolding the constructions given by the  $f$ 's, ending up with questions whose answers can be directly read from  $r$ .

## References

- [1] Hamkins,J.D. and Lewis,A.: Infinite Time Turing machines, J.of Symbolic Logic 65(2), (2000),567-604
- [2] Koepke,P. and Miller,R.: An Enhanced Theory of Infinite Time Register Machines. Logic and Theory of Algorithms, Fourth Conference on Computability in Europe, CiE 2008 (June 2008)

# Comparing Notions of Fractal Dimension\*

Chris J. Conidis

The University of Chicago, Chicago, IL USA

We construct a countable  $\Pi_1^0$ -class  $X \subset 2^\omega$  with effective packing dimension 1. This answers a question of Athreya, Hitchcock, Lutz, and Mayordomo [1] who asked if there is a correspondence principle for effective packing dimension, as in the case of effective Hausdorff dimension [4, 3].

## References

- [1] Athreya, K.B., Hitchcock, J.M., Lutz, J.H., Mayordomo, E.: Effective strong dimension in algorithmic information and computational complexity. *SIAM Journal on Computing* 37, 671–705 (2007)
- [2] Conidis, C.J.: Effective Packing Dimension of  $\Pi_1^0$ -classes. *Proceedings of the American Mathematical Society* (to appear).
- [3] Hitchcock, J.M.: Correspondence Principles for effective dimensions. *Theory of Computing Systems* 38, 559–571 (2005)
- [4] Lutz, J.H.: The Dimensions of Individual Strings and Sequences. *Information and Computation* 187, 49–79 (2003)

---

\* This work has been published in *The Proceedings of the American Mathematical Society* [2].

# Clockable Ordinals for Infinite Time Register Machines

Tim Fischbach, Peter Koepke, Miriam Nasfi, and Gregor Weckbecker

Mathematisches Institut, Universität Bonn, Germany

We do ordinal computability with the Infinite Time Register Machine (ITRM) model of Miller and Koepke ([1]). In analogy with the theory of Infinite Time Turing Machines (ITTM) ([2]) we introduce a notion of ITRM-clockable ordinals corresponding to the running time of a computation.

**Definition 1.** Let  $\alpha \in Ord$ .  $\alpha$  is ITRM-clockable iff there is an ITRM program  $P$  and a halting computation on input  $(0, \dots, 0)$  with zero oracle  $Z$

$$I : \alpha + 2 \rightarrow \omega, R : \alpha + 2 \rightarrow (\omega^\omega)$$

All natural numbers and all ordinals of the form  $\omega^{n-1} \cdot c_{n-1} + \dots + \omega \cdot c_1 + c_0$  for some  $n \in \omega$  and a sequence  $(c_i)_{i \in n}$  are clockable. Using finite vectors of bits we can also show that  $\omega^\omega$  is ITRM-clockable. In contrast to the ITTM situation there are *no gaps* in the ITRM-clockable ordinals:

**Theorem 1.**  $CLOCK := \{\alpha \mid \alpha \text{ ITRM-clockable}\}$  is a transitive initial segment of the ordinals and is properly contained in the ITTM-clockable ordinals.

The proof involves a halting criterion for ITRMs:

**Lemma 1.** Let

$$I : \theta \rightarrow \omega, R : \theta \rightarrow (\omega^\omega)$$

be a infinite time register computation by  $P$  with input  $(0, \dots, 0)$  and oracle  $Z$ . This computation does not stop iff there is some constellation  $(I', R')$  such that

$$\text{otp}(\{t < \theta \mid (I(t), R(t)) = (I', R')\}) \geq \omega^\omega$$

Furthermore, a finite speed-up lemma is used:

**Lemma 2 (Speed-up lemma).** Let  $\alpha + n$  be a clockable ordinal for some  $n \in \omega$ , then  $\alpha$  itself is clockable.

From theorem 1 we derive some closure properties of CLOCK and the following

**Theorem 2.** The class of ITRM-clockable ordinals coincides with the class of ITRM-computable ordinals.

## References

1. Koepke, P., Miller, R.: An enhanced theory of infinite time register machines. Logic and Theory of Algorithms, Fourth Conference on Computability in Europe, CiE 2008 (June 2008)
2. Hamkins, J.D., Lewis, A.: Infinite time Turing machines. J. Symbolic Logic **65**(2) (2000) 567–604

# Embedding the Enumeration Degrees in the $\omega$ -Enumeration Degrees

Hristo Ganchev

Sofia University, Faculty of Mathematics and Informatics  
5 James Bourchier blvd., 1165 Sofia, Bulgaria  
[ganchev@fmi.uni-sofia.bg](mailto:ganchev@fmi.uni-sofia.bg)

The structure of the  $\omega$ -enumeration degrees  $\mathcal{D}_\omega = (\mathbf{D}_\omega, \leq_\omega)$  is defined in [1]. It is shown, that  $\mathcal{D}_\omega$  is an upper semi-lattice and a jump operation is defined. Furthermore it is shown, that there is a natural embedding of the structure of the enumeration degrees  $\mathcal{D}_e = (\mathbf{D}_e, \leq)$  in  $\mathcal{D}_\omega$ , preserving *l.u.b* and jump operation. The image of the enumeration degrees under the natural embedding is denoted by  $\mathbf{D}_1$  and it is shown in [2] that  $\mathbf{D}_1$  is first order definable in  $(\mathcal{D}_\omega; \cup; ')$ .

We shall be concerned with embeddings of  $\mathcal{D}_e$  in  $\mathcal{D}_\omega$ , not realizable as the composition of an endomorphism of  $\mathcal{D}_e$  with the natural embedding. We prove that there are at least  $2^{\aleph_0}$  different embeddings, preserving the least upper bound operation, and at least  $\aleph_0$  — preserving the jump operation. We prove a necessary and sufficient condition for the existence of an embedding preserving both operations. From it, we conclude that the algebraic closure of  $(\mathcal{D}_e; \cup; ')$ , with respect to the least jump-invert operation (see [2]), is only embeddable in the algebraic closure  $\bigcup \mathcal{D}_n$  of  $(\mathbf{D}_1; \leq_\omega; \cup; ')$ . So,  $\bigcup \mathcal{D}_n$  may play an important role in the structural properties of  $\mathcal{D}_\omega$ . Finally, we show that  $\bigcup \mathcal{D}_n$  is first order definable substructure of  $(\mathcal{D}_\omega; \cup; ')$ .

## References

1. I. N. Soskov, *The  $\omega$ -enumeration degrees*, Journal of Logic and Computation **17** (2007), 1193–1214.
2. I. N. Soskov H. Ganchev, *The jump operator on the  $\omega$ -enumeration degrees*, to appear.

# Computable Models Spectras of Ehrenfeucht Theories\*

Alexander N. Gavryushkin

Novosibirsk State University, Russia

This paper is connected with next common problem. To characterize the set of models of an Ehrenfeucht theory (a complete theory with finite number of countable models up to isomorphism) having computable presentation. Types of isomorphism of a prime model and of a saturated model are well known in classical model theory. In [3] there are examples of Ehrenfeucht theories the only computably presentable model of which is the prime one. In [2] there is an example of Ehrenfeucht theory such that only saturated model of the theory has computable presentation. In [1] there is an example of Ehrenfeucht theory such that only one model of the theory has a computable presentation, and that model is neither prime nor saturated. Also there exist an Ehrenfeucht theory  $T$ , whose prime and saturated models have computable presentations, and a model of  $T$  which lacks in such. The example is in [1].

In [4] there is a syntactic characterization for the class of Ehrenfeucht theories. In [5] there are examples of Ehrenfeucht theories guaranteeing that all possible parameters given in the characterization theorem in [4] are realizable.

The result I want to present is formulated in terms of mentioned characterization and it couldn't be described using 1-page abstract. Then I have only to state that there are two groups of examples of Ehrenfeucht theories: 1) there are two models of a theory at the same level up to classification from [4] one of which has computable presentation and another is not computably presentable; 2) there are levels of the classification all models from which have computable presentations and another levels are without computably presentable models.

## References

1. Gavryushkin, A. N., Spectra of Computable Models for Ehrenfeucht Theories, *Algebra and Logic*, 46, No. 3, 149–157, 2007.
2. Khoussainov, B., Nies, A., Shore, R., Computable Models of Theories with Few Models, *Notre Dame Journal of Formal Logic*, 38, 165–178, 1997.
3. Peretyat'kin, M. G., On Complete Theories with Finite Number of Countable Models, *Algebra i Logika*, 12, No. 5, 550–576, 1973.
4. Sudoplatov, S. V., Complete Theories with Finitely Many Countable Models. I, *Algebra and Logic*, 43, No. 1, 62–69, 2004.
5. Sudoplatov, S. V., Complete Theories with Finitely Many Countable Models. II, *Algebra and Logic*, 45, No. 3, 180–200, 2006.

---

\* The work was partially supported by President Grant – 335.2008.1

# Deterministic Subsequential Transducers with Additional FIFO-memory

S. V. Gerdjikov

Department of Mathematical Logic and Applications  
Sofia University  
[st\\_gerdjikov@abv.bg](mailto:st_gerdjikov@abv.bg)

Regular rewriting rules play a significant role in text-processing techniques. It is well known that they can be expressed in terms of rational functions and consequently for each such rule one can efficiently construct a transducer or, equivalently a bimachine. Unfortunately the transducers, in general cannot be determinized and the bimachines do not allow to stream a text, and thus one cannot process a text on-line.

We propose a new formalism, which aims at preserving the determinism and still to be able to process the input sequentially, i.e. without disposing on the entire text in advance. To this end we incorporate an additional infinite memory, represented as FIFO. We model it in a way to guarantee the deterministic, linear traversal of an input text.

We call these machines *FIFO-transducers* and study their properties with respect to rational functions. Our main efforts concern the composition problem. We show that we can efficiently compose FIFO-transducers with subsequential transducers, but in general they are not closed under composition and they are unable to describe the class of all rational function. We also show a simple example of a function that is not rational but can be represented by such a machine.

Finally, we define a subclass of rational functions that can be recognized by FIFO-transducers but (in the general case) not by a subsequential transducer. We state sufficient conditions for a regular rule in order to be represented by a FIFO-transducer and show how to check these properties algorithmically.

# Proof Fragments and Cut-Elimination

Stefan Hetzl

Institute of Computer Languages  
Vienna University of Technology  
Favoritenstraße 9, 1040 Vienna, Austria  
[hetzl@logic.at](mailto:hetzl@logic.at)

Cut-elimination is a proof transformation of fundamental importance. Originally it was introduced by Gentzen together with the sequent calculus [2] and it builds the core of his consistency proof of Peano arithmetic [3]. It also plays an important role in the analysis of mathematical proofs and has deep connections to computation in functional programming languages.

Cut-elimination is usually presented as a set of local rewrite rules with some terminating strategy thus showing the existence of cut-free proofs for all provable sequents. The changes to the global structure of the proof that are caused by these local rewrite steps have traditionally been less investigated.

In this talk we consider proof skeletons (see e.g. [4]) which are abstract representations of the structure of proofs. We will describe the changes the skeleton of a proof undergoes during cut-elimination: Based on its skeleton, a proof with cuts can be split into several pieces (fragments) which are not broken up further by the local rewrite rules. The global effect of cut-elimination on the structure of a proof is therefore shown to be a re-composition of instances of these fragments. The proof is carried out by relying on methods based on cut-elimination by resolution [1].

This result allows to describe a certain kind of redundancy whose presence is a necessary condition for a cut-free proof to allow strong compression by introduction of cuts. From this characterization follows a lower bound on cut-introduction.

## References

1. Matthias Baaz and Alexander Leitsch. Cut-elimination and Redundancy-elimination by Resolution. *Journal of Symbolic Computation*, 29(2):149–176, 2000.
2. Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1934–1935.
3. Gerhard Gentzen. Die Widerspruchsfreiheit der reinen Zahlentheorie. *Mathematische Annalen*, 112:493–565, 1936.
4. Jan Krajíček and Pavel Pudlák. The Number of Proof Lines and the Size of Proofs in First Order Logic. *Archive for Mathematical Logic*, 27:69–84, 1988.

# $q$ -Overlaps in the Random Exact Cover Problem

Gabriel Istrate<sup>1\*</sup> and Romeo Negrea<sup>2</sup>

<sup>1</sup> e-Austria Institute, V.Pârvan 4, cam. 045B, Timișoara RO-300223, Romania  
[gabrielistrate@acm.org](mailto:gabrielistrate@acm.org).

<sup>2</sup> Department of Mathematics, Universitatea Politehnica din Timișoara  
Victoriei 2, 300006, Timișoara, Romania  
[negrea@math.uvt.ro](mailto:negrea@math.uvt.ro)

We prove lower and upper bounds for the threshold of the following problem: **given  $q \in (0, 1)$  and  $c > 0$  what is the probability that a random instance of the  $k$ -Exact Cover problem [KM05] has two solutions of overlap  $qn \pm o(n)$ ?** This problem is motivated by the study of *phase transitions in Combinatorial Optimization problems* has recently motivated (and brought to attention) the geometric structure of the solution space of a combinatorial problem. A remarkable recent advance in this area is due to Mézard et al. [MMZ05], [DMMZ08]. These papers have provided rigorous evidence that for the random  $k$ -satisfiability problem (with sufficiently large  $k$ ) the intuitions concerning the geometry of the solution space provided by the 1-RSB approach are correct. The approach in this paper is similar. We study the overlap distribution of the *random  $k$ -Exact Cover* problem. The phase transition in this problem has been studied in [KM05]. Zdeborová et al. [RSZ07],[MMR<sup>+</sup>07] have applied nonrigorous methods from Statistical Physics (the cavity approach) and have suggested that the *1-step Replica Symmetry Breaking* assumption is valid. This motivates us to study the problem  $q$ -overlap  $k$ -Exact Cover and prove lower and upper bounds on its satisfiability threshold.

## References

- [DMMZ08] H. Daudé, M. Mézard, T. Mora, and R. Zecchina. Pairs of SAT assignments and clustering in random boolean formulae. *Theoretical Computer Science*, 393(1-3):260–279, 2008.
- [KM05] Vamsi Kalapala and Cris Moore. The phase transition in exact cover. Technical Report cs/0508037, arXiv.org, 2005.
- [MMR<sup>+</sup>07] E. Maneva, T. Meltzer, J. Raymond, A. Sportiello, and L. Zdeborová. A hike in the phases of the 1-in-3 satisfiability problem. In J.P. Bouchaud, M. Mézard, and J. Dalibard, editors, *Lecture Notes of the Les Houches Summer School 2006*, pages 491–498. Elsevier, 2007.
- [MMZ05] M. Mézard, T. Mora, and R. Zecchina. Clustering of solutions in the random satisfiability problem. *Physical Review Letters*, 94(197205), 2005.
- [RSZ07] J. Raymond, A. Sportiello, and L. Zdeborová. The phase diagram of random 1-in-3 satisfiability. *Phys. Rev. E*, 76(011101), 2007.

---

\* Corresponding author. Supported by a Marie Curie International Reintegration Grant within the 6th European Community FP.

# Inductive Definitions over Domain Representable Spaces

Petter Kristian Køber

Department of Mathematics, University of Oslo, Norway  
[petterk@math.uio.no](mailto:petterk@math.uio.no)

One of the main constructions in domain theory is the solution of recursive domain equations. Of particular interest are the positive equations, which can be solved iteratively within set theory. The domains we consider are separable Scott domains.

The class of topological spaces with an admissible domain representation has been characterised ([1,2]) as the  $T_0$  quotients of countably based spaces ( $qcb_0$  spaces). In this talk, we look at how canonical fixed points can be constructed for strictly positive operators over  $qcb_0$  spaces, with the least solution of the equation

$$X = P \uplus [Q \Rightarrow X]$$

(where  $P$  and  $Q$  are fixed  $qcb_0$  spaces) as the most natural example.

The most useful approach to this problem is to consider partial equivalence relations on domains (domain-pers). We generalise the strictly positive operators as well as the inductive limit construction to domain-pers, and consider an important subclass of domain-pers which is closed under these operations. A strictly positive operator over  $qcb_0$  spaces can be represented by a strictly positive operator over domain-pers and the latter admits a least fixed point (w.r.t. certain well-behaved domain embeddings). Under certain natural initial conditions on the operator, this least fixed point gives rise to an admissible representation.

A different approach is to consider the operator over  $qcb_0$  spaces more generally as an operator over weak limit spaces. For the main example, this gives a least fixed point (w.r.t. continuous, injective maps) and the sequential topological space associated turns out to be homeomorphic to the one derived from the least fixed point of the operator over domain-pers. Combining these results, we obtain a least fixed point for the operator on  $qcb_0$  spaces.

The method used works only in the case of strictly positive inductive definitions, thus it remains open whether  $qcb_0$  spaces can be defined by positive induction in general. We also discuss whether it is possible to generalise the results by allowing the use of free algebra constructions in the defining equation.

## References

1. I.Battenfeld, M.Schröder, A.Simpson, A Convenient Category of Domains. Electronic Notes in Theoretical Computer Science **172**, 69-99, Elsevier 2007.
2. G.Hamrin, Admissible Domain Representations of Topological Spaces. U.U.D.M.Report 16, Uppsala University 2005.

# The Computable Dimension of Free Projective Planes

Nurlan Kogabaev

Sobolev Institute of Mathematics  
Koptyug Prospect 4, Novosibirsk 630090, Russia.  
[kogabaev@math.nsc.ru](mailto:kogabaev@math.nsc.ru)

Shirshov (cf. [1]) suggested to treat projective planes as partial algebraic systems. In the framework of this approach a *projective plane* is a structure  $\langle A, (A^0, {}^0A), \cdot \rangle$  with a disjunction of  $A$  into two subsets  $A^0 \cup {}^0A = A$ ,  $A^0 \cap {}^0A = \emptyset$  and commutative partial operation “ $\cdot$ ” which satisfy the following properties:

- (1)  $a \cdot b$  is defined iff  $a \neq b$  and  $a, b \in A^0$  (or  $a, b \in {}^0A$ ) with the product  $a \cdot b \in {}^0A$  ( $a \cdot b \in A^0$  respectively);
- (2) for all  $a, b, c \in A$  if  $a \cdot b, a \cdot c, (a \cdot b) \cdot (a \cdot c)$  are defined, then  $(a \cdot b) \cdot (a \cdot c) = a$ ;
- (3) there exist distinct  $a, b, c, d \in A$  such that products  $a \cdot b, b \cdot c, c \cdot d, d \cdot a$  are defined and pairwise distinct.

Any *free projective plane* is freely generated by the set of “points”  $\{b_1, b_2\} \cup \{a_i | i \in I\}$  and unique “line”  $c$  which is incident with  $a_i$  for each  $i$ . From the results of [1] it follows that any countable free projective plane has a computable presentation.

In the present paper we investigate the question of possible computable dimension of free projective planes and the existence problem of computable list for the class of all projective planes (up to computable isomorphism). Applying the Unbounded Models Theorem (cf. [2]) we obtained the following results:

**Theorem 1.** *Every countable free projective plane has computable dimension 1 or  $\omega$ . Furthermore, such a plane is computably categorical if and only if it has finite rank.*

**Theorem 2.** *The class of all projective planes is not computable (up to computable isomorphism).*

## References

1. Shirshov, A.I., Nikitin, A.A.: On the theory of projective planes. Algebra and Logic. 20, 330–356 (1981)
2. Goncharov, S.S.: Autostability of models and Abelian groups. Algebra and Logic. 19, 13–27 (1980)

# A Classification of Theories of Truth

Graham Leigh and Michael Rathjen

University of Leeds, Leeds UK

We augment the first-order language of Peano Arithmetic with a unary predicate  $T$  with  $T(x)$  intended to mean “ $x$  is the Gödel number of a true sentence of arithmetic”. In [2] Friedman and Sheard constructed a list of twelve axioms and rules of inference concerning the predicate  $T$ , each expressing some desirable property of truth, and classified all subsets of the list as either consistent or inconsistent. This gave rise to a collection of nine theories of truth, two of which have been treated in the literature (see [2], Halbach [3], Sheard [4] and Cantini [1] for more details). We uncover the proof-theoretic strength of the remaining seven and in the process construct a proof-theory of truth allowing the systems to be subject to an ordinal analysis.

## References

1. A. Cantini, A Theory of Formal Truth Arithmetically Equivalent to ID1, *Journal of Symbolic Logic*, 55, 1, 244–259, 1990.
2. H. Friedman and M. Sheard, An Axiomatic Approach to Self-referential Truth, *Annals of Pure and Applied Logic*, 33 1–21, 1987.
3. V. Halbach, A System of Complete and Consistent Truth, *Notre Dame Journal of Formal Logic*, vol. 35, 3, 311–327, 1994.
4. M. Sheard, Weak and Strong Theories of Truth, *Studia Logica*, 68, 89–101, 2001.

# Monotonicity Conditions over Characterisations of PSPACE

Bruno Loff<sup>1,3</sup> and Isabel Oitavem<sup>2,3</sup>

<sup>1</sup> Dept. of Mathematics, Instituto Superior Técnico, Technical Univ. of Lisbon

<sup>2</sup> Dept. of Mathematics, Faculdade de Ciências e Tecnologia, Univ. Nova de Lisboa

<sup>3</sup> Centro de Matemática e Aplicações Fundamentais (CMAF), Univ. of Lisbon

It is an open problem whether  $P = \text{PSPACE}$ . An algorithm working in polynomial space is allowed to erase previously used cells in order to reuse space, and if we observe the well-known algorithms for PSPACE-complete problems, they always seem to rely heavily on this possibility. Our intuition then indicates that this is the crucial difference between  $P$  and PSPACE. In this presentation, we begin by making this intuition rigorous, proving that the additional power of PSPACE arises exactly from the fact that we are allowed to reuse space. We consider a “write-only” Turing machine, given polynomial space, which is not allowed to erase (or re-write over) cells which have been previously written on; we then prove that under this restriction, polynomial-space-bounded Turing machines decide exactly  $P$ .

This will lead us to consider weaker forms of this “write-only” restriction. Consider the natural partial order over binary strings  $\preceq$ , where  $w \preceq v$  if  $|w| < |v|$  or if  $|w| = |v|$  and every symbol of  $w$  is less or equal to the corresponding symbol of  $v$  in the same position. Then by a detailed observation of the tape configurations of a write-only machine we find that consecutive tape configurations are monotonically increasing according to this order. It turns out that this monotonicity ensures that any set decided by a monotone polynomial-space bounded computation can also be decided in  $P$ . Then  $P$  vs. PSPACE is equivalent to asking whether a polynomial-space-bounded computation can be made monotone.

So in the second part of our presentation we apply these results to implicit characterisations of PSPACE. We introduce two multi-sorted function algebras  $\mathcal{A}$  and  $\mathcal{B}$ .  $\mathcal{A}$  is, essentially, the characterisation of PSPACE given by Isabel Oitavem in 97, and  $\mathcal{B}$  is  $\mathcal{A}$  with the input-sorted primitive recursion replaced by input-sorted primitive iteration. We show that, analogously to  $\mathcal{A}$ ,  $\mathcal{B}$  characterises PSPACE. When we restrict the recursion and iteration operators to functions obeying a monotonicity condition for  $\preceq$ , we obtain two algebras  $\hat{\mathcal{A}}$  and  $\hat{\mathcal{B}}$ .  $\hat{\mathcal{A}}$  is obtained by restricting input-sorted primitive recursion, and  $\hat{\mathcal{B}}$  by restricting input-sorted iteration. Our results are that  $\hat{\mathcal{B}}$  characterises  $P$ ,  $\hat{\mathcal{A}}$  characterises the polynomial hierarchy, and if we consider a hierarchy  $\hat{\mathcal{A}}_1, \dots, \hat{\mathcal{A}}_n, \dots$  by counting the rank of the operator of restricted input-sorted primitive recursion, then  $\hat{\mathcal{A}}_n$  corresponds to the functions computable with oracles in  $\Sigma_n \cap \Pi_n$ . The whole work suggests a new machine-based characterisation of PH and of each level  $\Sigma_n \cap \Pi_n$  by polynomial-space write-only deterministic machines coupled with clocks.

# Some Results on Local LR-degree Structures

Anthony Morphett\*

University of Leeds

A natural direction of study arising from recent work in algorithmic randomness is to investigate connections between the information content of a set (in the sense of its Turing degree) and the notion of relative randomness that is obtained by adding the set as an oracle. One approach to this is the LR(low for random)-reducibility: a set A is LR-reducible to B if the class of reals Martin-Löf random relative to oracle B is contained in the class of randoms relative to A. The associated degree structure is the LR-degrees. An LR-degree is c.e. ( $\Delta_2^0$  respectively) if it contains a c.e. ( $\Delta_2^0$ ) set.

Although many questions exist about the global and local LR-degree structures, some results have been obtained. For instance, Barmpalias, Lewis and Soskova [1] prove a splitting theorem for c.e. sets, and Barmpalias, Lewis and Stephan [2] prove a weak density theorem for the c.e. LR-degrees. It is not known if full density holds for the c.e. or  $\Delta_2^0$  LR-degrees. We describe some additional results about the c.e. and  $\Delta_2^0$  LR-degrees, including upward density results. We will also discuss some similarities and differences between these structures and the c.e. or  $\Delta_2^0$  Turing degrees.

## References

1. George Barmpalias, Andrew E. M. Lewis, Mariya Soskova, Randomness, Lowness and Degrees, *Journal of Symbolic Logic* vol.73, Issue 2, pp. 559-577 (2008)
2. George Barmpalias, Andrew E. M. Lewis, Frank Stephan,  $\Pi_1^0$  classes, LR degrees and Turing degrees, to appear in *Annals of Pure and Applied Logic*

---

\* Supported by MEST-CT-2004-504029 MATHLOGAPS Marie Curie Host Fellowship.

# The Axiomatic Derivation of Absolute Lower Bounds

Yiannis N. Moschovakis<sup>1,2</sup>

<sup>1</sup> Department of Mathematics, University of California in Los Angeles

<sup>2</sup> Graduate Program in Logic, Algorithms and Computation (MPLA)

Department of Mathematics, University of Athens

I will outline a method for deriving lower bounds for the complexity of problems (especially in arithmetic) which are absolute (universal), i.e., they apply to all algorithms; the key idea is to derive lower bounds from three axioms for algorithms, which are natural and easily shown to apply to all known models of computation.

# Exploring the Computational Contribution of a Non-constructive Combinatorial Principle

Diana Ratiu and Trifon Trifonov\*

Mathematics Institute, University of Munich, Germany  
`{ratiu,trifonov}@math.lmu.de`

We regard  $\Pi_2^0$ -formulas as specifications of the sort “Given an input  $x$  satisfying some property  $D$ , is there an algorithm producing an output  $y$  from  $x$  so that a given requirement  $G$  is met?” and look at proofs by contradiction of such statements. Here we will consider a particular combinatorial problem expressed as  $\forall x(D \rightarrow \forall y(G(x, y) \rightarrow \perp) \rightarrow \perp)$  and analyze the algorithms that we extract from its classical proof by two distinct methods — the refined  $A$ -Translation and Gödel’s functional (Dialectica) interpretation.

$A$ -Translation is a combination of Gödel-Gentzen double negation translation and Friedman’s trick [2], which substitutes  $\perp$ , viewed as a predicate variable, by  $\exists y G(x, y)$ . [1] proposed a refinement of the method, such that unnecessary double negations are avoided for certain classes of formulas  $D$  and  $G$ . By coupling this with Kreisel’s (modified) realisability, one can now associate with the  $A$ -translated proof an extracted term in a Curry-Howard fashion.

Gödel’s Dialectica interpretation [3] presents an alternative for collecting computational content from non-constructive proofs, namely by tracking counterexamples, i.e., terms instantiating universal assumptions. By employing simple finite types we can translate every formula  $A$  to a decidable relation  $A_D(z, y)$  between a solution  $z$  and a counterexample  $y$ . Furthermore, from a derivation of  $A$  we can extract a term  $t$ , which satisfies  $A_D(t, y)$  for every counterexample  $y$ .

We investigate a combinatorial result used to prove Ramsey’s foundational theorem, namely the Infinite Pigeon Hole (IPH) principle. IPH states that any finitely colored infinite sequence has an infinite monochromatic subsequence. In general, there is no computable functional producing such an infinite subsequence. However, IPH can be used to give a simple classical proof of the  $\Pi_2^0$ -statement that a finite monochromatic subsequence of any given length exists. We compare the programs extracted from this corollary using the aforementioned methods and discuss how they reflect the computational meaning of IPH.

## References

1. Berger, U. and Buchholz, W. and Schwichtenberg, H., Refined Program Extraction from Classical Proofs, Annals of Pure and Applied Logic, 114, pp 3–25, (2002)
2. Friedman, H., Classically and intuitionistically provably recursive functions, D.S. Scott and G.H. Müller, 669, Lecture Notes in Mathematics, pp 21–28, (1978)
3. Gödel, K, Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes, Dialectica, 12, pp 280–287, (1958)

---

\* The authors gratefully acknowledge financial support by MATHLOGAPS (MEST-CT-2004-504029), a Marie Curie Early Stage Training Site.

# Autostability of Automatic Linear Orders\*

Alexandra Revenko

Novosibirsk State University, Russia

The class of automatic structures is the easiest class of computable structures from an algorithmic point of view.

Our investigations concern the problem of existence a computable isomorphism between two automatic presentations of a structure. We say that a structure is autostable under automatic presentations if there is a computable isomorphism between any two automatic presentations of this structure [1]. Here we deal with linear orders.

Some description of automatic linear orders was found by B. Khoussainov, S. Rubin and others. Let  $\mathcal{L} = (L, \leq)$  be a countable linear order. We say that two elements  $x, y \in L$  are equivalent if there is a finite number of elements between  $x$  and  $y$ . The quotient structure with respect to this equivalence relation is a linear order  $\mathcal{L}_1$  (ordering is induced from the original structure). One can continue this procedure by transfinite induction. So the least ordinal  $\alpha$  for which  $\mathcal{L}_\alpha = \mathcal{L}_{\alpha+1}$  is a *FC*-rank of  $\mathcal{L}$ . Then the *FC*-rank of automatic linear order is finite [2].

It was earlier showed that all automatic ordinals and automatic scattered linear orders with FC-rank less than 3 are autostable under automatic presentations. We proved that moreover there exist an algorithm which, given two automatic presentations of a scattered linear order with FC-rank less than 3, constructs the computable isomorphism between them. Then every two automatic presentations of scattered linear order with *FC*-rank 3 are computable isomorphic.

## References

1. Ershov, Yu. L., Goncharov, S. S., *Constructive Models*, Siberian School of Algebra and Logic, 1999.
2. S. Rubin. *Automatic Structures*. A thesis submitted in partial fulfilment of the requirements for the Degree of Doctor of Philosophy. The University of Auckland, 2004.

---

\* The work was partially supported by President Grant – 335.2008.1

# Quantifiers on Automatic Structures

Sasha Rubin

Department of Computer Science, University of Auckland  
[rubin@cs.auckland.ac.nz](mailto:rubin@cs.auckland.ac.nz)

An *automatic structure* is one that has a presentation consisting of finite or infinite words or trees so that the coded domain and atomic operations are computable by finite automata operating synchronously on their inputs [KN95] [Blu99]. The fundamental fact about these structures is that they are effectively closed under first-order interpretations [BG00]. This result has been strengthened by extending first-order logic with certain generalised quantifiers (for instance, ‘there exist infinitely many’ and ‘there exist  $k$  modulo  $m$  many’) [BG00] [KRS04] [Col04] [KL05] [BKR07] [BKR08] [KL08].

Say that a generalised quantifier  $Q$  *preserves regularity* for a class of automatic structures  $\mathcal{C}$  if  $\mathcal{C}$  is closed under  $\text{FO} + Q$  interpretations. In this talk I will present some steps towards a fuller understanding of these quantifiers.

I will introduce a natural collection  $\mathfrak{Q}$  of quantifiers, each preserving regularity for the finite-word automatic structures  $\mathcal{C}_{\text{fw}}$ . The collection  $\mathfrak{Q}$  includes the known quantifiers that preserve regularity for  $\mathcal{C}_{\text{fw}}$ ; and every *unary* quantifier that preserves regularity for  $\mathcal{C}_{\text{fw}}$  is in  $\mathfrak{Q}$ .

This is part of ongoing work with Valentin Goranko and Moshe Vardi.

## References

- [BG00] A. Blumensath and E. Grädel. Automatic structures. In *15th Symposium on Logic in Computer Science (LICS)*, pages 51–62, 2000.
- [BKR07] V. Bárány, L. Kaiser, and A. Rabinovitch. Eliminating cardinality quantifiers from MLO, 2007. manuscript.
- [BKR08] V. Bárány, L. Kaiser, and S. Rubin. Cardinality and counting quantifiers on omega-automatic structures. In *STACS ’08: Proceedings of the 25th Annual Symposium on Theoretical Aspects of Computer Science*, 2008.
- [Blu99] A. Blumensath. *Automatic Structures*. Diploma thesis, RWTH Aachen, 1999.
- [Col04] T. Colcombet. *Properties and representation of infinite structures*. PhD thesis, University of Rennes I, 2004.
- [KL05] D. Kuske and M. Lohrey. First-order and counting theories of omega-automatic structures. Technical Report Fakultätsbericht Nr. 2005/07, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2005.
- [KL08] D. Kuske and M. Lohrey. Hamiltonicity of automatic graphs. *FIP TCS 2008*, 2008.
- [KN95] B. Khoussainov and A. Nerode. Automatic presentations of structures. *Lecture Notes in Computer Science*, 960:367–392, 1995.
- [KRS04] B. Khoussainov, S. Rubin, and F. Stephan. Definability and regularity in automatic structures. In *STACS 2004*, volume 2996 of *LNCS*, pages 440–451. Springer, Berlin, 2004.

# The Almost Zero $\omega$ -Enumeration Degrees

Ivan N. Soskov

Faculty of Mathematics and Computer Science, Sofia University  
5 James Bourchier Blvd., 1164 Sofia, Bulgaria,  
[soskov@fmi.uni-sofia.bg](mailto:soskov@fmi.uni-sofia.bg)

The jump operator “ $\prime$ ” on the  $\omega$ -enumeration degrees possesses a surprising inversion property that for every  $n$  and every degree  $\mathbf{a} \geq \mathbf{0}_\omega^{(n)}$  there exists a least degree  $I^n(\mathbf{a})$  among the degrees  $\mathbf{x}$  such that  $\mathbf{x}^{(n)} = \mathbf{a}$ , see [4].

A degree  $\mathbf{x}$  is *almost zero* (a.z.) if  $\mathbf{x} \leq \mathbf{0}'_\omega$  and for all  $n$ ,  $I^n(\mathbf{x}^{(n)}) = \mathbf{x}$ .

There exist non-zero a.z. degree. Moreover, there exist incomparable a.z. degrees. Another nice property of the a.z. degrees is that for every pair  $\mathbf{a}$  and  $\mathbf{b}$  of a.z. degrees we have  $(\forall n)((\mathbf{a} \oplus \mathbf{b})^{(n)} = \mathbf{a}^{(n)} \oplus \mathbf{b}^{(n)})$ . The a.z. degrees form an ideal called  $Az$  which has no minimal upper bound below  $\mathbf{0}'_\omega$ .

Suppose that  $C = (C, \leq)$  is a degree structure and let “ $\prime$ ” be a jump operator defined on the elements of  $C$ . Given elements  $\mathbf{a}$  and  $\mathbf{b}$  of  $C$ , let  $\mathbf{a} \preceq \mathbf{b}$  if for some  $n$ ,  $\mathbf{a}^{(n)} \leq \mathbf{b}^{(n)}$ . Let  $\mathbf{a} \sim \mathbf{b}$  if  $\mathbf{a} \preceq \mathbf{b}$  and  $\mathbf{b} \preceq \mathbf{a}$ . Finally for every  $\mathbf{a} \in C$ , set

$$\mathbf{a}^* = \{\mathbf{b} : \mathbf{a} \sim \mathbf{b}\}.$$

Let  $\mathbf{a}^* \preceq \mathbf{b}^*$  if  $\mathbf{a} \preceq \mathbf{b}$  and denote by  $\mathcal{J}_C$  the partial ordering  $(\{\mathbf{a}^* : \mathbf{a} \in C\}, \preceq)$ .

The ordering  $\mathcal{J}_R$  based on the r.e. Turing degrees is studied by LEMPP [2] who shows that it contains a minimal pair over  $\mathbf{0}^*$  and a splitting of  $(\mathbf{0}')^*$ . JOCKUSCH, LERMAN, SOARE AND SOLOVAY [1] show that this ordering is dense. The ordering  $\mathcal{J}_G$  based on the  $\Sigma_2^0$  enumeration degrees is of no particular interest since by a result of McEVoy [3] it is isomorphic to  $\mathcal{J}_R$ .

In the talk we shall present some results about the orderings  $\mathcal{J}_{Az}$  and  $\mathcal{J}_{G_\omega}$  based on the almost zero degrees and on the  $\Sigma_2^0$   $\omega$ -enumeration degrees respectively. We shall show that  $\mathcal{J}_{Az}$  is isomorphic to  $(Az, \leq_\omega)$  and that the orderings  $\mathcal{J}_{Az}$  and  $\mathcal{J}_{G_\omega}$  are dense.

## References

1. R. I. Soare R. M. Solovay C. G. Jockusch, M. Lerman, *Recurseively enumerable sets modulo iterated jumps and extensions of Arslanov's completeness criterion*, J. Symbolic Logic **54** (1989), 1288–1323.
2. S. Lempp, *Topics in recursively enumerable sets and degrees*, Ph.D. thesis, University of Chicago, 1986.
3. K. McEvoy, *Jumps of quasi-minimal enumeration degrees*, J. Symbolic Logic **50** (1985), 839–848.
4. I. N. Soskov and H. Ganchev, *The jump operator on the  $\omega$ -enumeration degrees*, Ann. Pure Appl. Logic, to appear.

# A Sequent Calculus for Intersection and Union Logic

Anastasia Veneti<sup>1</sup> and Yiorgos Stavrinos<sup>2</sup>

<sup>1</sup> Department of Computer Science, National Technical University of Athens  
GR-15773 Zografou, Greece, [tassiana98@gmail.com](mailto:tassiana98@gmail.com)

<sup>2</sup> MPLA, Department of Mathematics, University of Athens  
GR-15784 Zografou, Greece, [g.stavrinos@math.ntua.gr](mailto:g.stavrinos@math.ntua.gr)

We present a logical formalism for the *intersection and union type assignment system* IUT [1]. A first attempt to this end is the intersection and union logic IUL, a logic whose main structures are binary trees called *kits* [3, 4]. The rules of IUL are in natural deduction style and are categorized as *global* or *local* according to whether they affect all leaves of the kits involved or not. While rules for the intersection and union are meant to be local, the complex notation of kits conceals a certain kind of globality inherent in the union elimination rule. This becomes explicit if we abandon kits and resort to the linear structures employed in [2]. These structures are multisets of judgements (*atoms*) called *molecules*. A logic for IUT using molecules, but still in natural deduction style, incorporates a *union elimination rule*

$$\frac{[(\Gamma_i \vdash \varphi_i)]_{i < n} \bigcup [(\Gamma \vdash \sigma \cup \tau)] \quad [(\Gamma_i, \varphi_i \vdash \psi_i)]_{i < n} \bigcup [(\Gamma, \sigma \vdash \rho), (\Gamma, \tau \vdash \rho)]}{[(\Gamma_i \vdash \psi_i)]_{i < n} \bigcup [(\Gamma \vdash \rho)]} (\cup E)$$

which has both a global and a local behaviour. Here globality and locality refer to the number of atoms modified by the rule in the premise molecules.

In the present work we describe IUL with molecules in sequent calculus style. The main advantage of this formulation is that globality and locality of the union elimination rule are separated in the *cut rule* and *left union rule*, respectively.

$$\frac{[(\Gamma_i \vdash \varphi_i)]_{i < n} \quad [(\Gamma_i, \varphi_i \vdash \psi_i)]_{i < n}}{[(\Gamma_i \vdash \psi_i)]_{i < n}} (\text{cut}) \quad \frac{\mathcal{M} \bigcup [(\Gamma, \sigma \vdash \rho), (\Gamma, \tau \vdash \rho)]}{\mathcal{M} \bigcup [(\Gamma, \sigma \cup \tau \vdash \rho)]} (L \cup)$$

These improvements lead the way in yielding the relation of IUT to the logic IUL.

## References

1. Barbanera F., Dezani-Ciancaglini M., and de'Liguoro U., Intersection and Union Types: Syntax and Semantics, *Information and Computation* 119, 202–230 (1995)
2. Pimentel E., Ronchi Della Rocca S., and Roversi L., Intersection Types from a proof-theoretic perspective, *4th Workshop on Intersection Types and Related Systems*, Torino (2008)
3. Ronchi Della Rocca S. and Roversi L., Intersection Logic, *Proceedings of CSL'01*, LNCS 2142, 414–428 (2001)
4. Veneti A. and Stavrinos Y., Towards an intersection and union logic, *4th Workshop on Intersection Types and Related Systems*, Torino (2008)

# Anhomomorphic Logic: The Logic of Quantum Realism

Petros Wallden

Raman Research Institute, Sadashivanagar, Bangalore 560-080, India  
[petros.wallden@gmail.com](mailto:petros.wallden@gmail.com) or [petros@rrri.res.in](mailto:petros@rrri.res.in)

Anhomomorphic logic, is a novel interpretation of Quantum Theory (initiated by Sorkin) that comes as a development of the consistent histories approach and is an attempt to retain realism. When using logic to describe (classical) physics, we have a set of possible histories  $\Omega$ , a set of truth values ( e.g. {True, False}) and the possible maps ( $\phi_i$ ) that are homomorphisms between the Boolean algebra of subsets of  $\Omega$  and of the truth values. These maps give rise to different realizations (here is where the measure and thus the dynamics enter the picture). It is well known that the above picture cannot hold in quantum theory. One can either restrict the subsets of  $\Omega$  allowed (standard consistent histories) or change the set of truth values to some Heyting algebra (Isham) or finally, weaken the requirement that the map is homomorphism. This latter approach is taken in “Anhomomorphic Logic”. The weakening of the requirement to be a homomorphism is replaced by other conditions, that guarantee that most structure is preserved and the basic inference law (modus ponens) still applies. Thus we have a deductive logic (which is not the case in what is usually referred to as “quantum logic”). In this talk, we will first introduce anhomomorphic logic in some detail. Then we will deal with some recent developments on the emergence of classicality and the closely related issue of recovery of probabilistic predictions. The former, essentially means that in some scale of coarse graining, we know that classical (boolean) logic applies. Thus we have to show that the anhomomorphic logic, upon some coarse grainings, results to homomorphic logic (i.e. classical). Finally, the issues of how probabilities arise, is present in several attempts to retain realism in quantum theory (such as many worlds), and in this approach it is resolved with the use of the concept of “approximate preclusion” and by taking a frequentist’s view on probability rather than treating it as propensity.

## Author Index

Afshari, Bahareh .....	493	Hetzl, Stefan .....	502
Agadžanjan, Ruben .....	494	Horihata, Yoshihiro .....	157
Aihara, Kazuyuki .....	445	Hwang, Yoon-Hee .....	77, 165
Almeida, Marco .....	3	Irrgang, Bernhard .....	175
Aman, Bogdan .....	15	Istrate, Gabriel .....	503
Antonopoulos, Timos .....	495	Jain, Sanjay .....	185
Antunes, Luis .....	25	Jansen, Maurice .....	195
Ayala-Rincón, Mauricio .....	137	Jenhani, Ilyes .....	205
Azevedo, Tiago .....	35	Jervell, Herman Ruge .....	215
Beggs, Edwin .....	45	Kahle, Reinhard .....	224
Benevides, Mario .....	35	Karádais, Basil .....	234
Benferhat, Salem .....	205	Kim, Han-Doo .....	77, 165
Bonet, Isis .....	284	Kim, Jin-Gyoung .....	77
Caldwell, James .....	254	Kim, Seok-Tae .....	367
Calhoun, William .....	55	Koepke, Peter .....	496, 498
Carl, Merlin .....	496	Kogabaev, Nurlan .....	505
Chiarabini, Luca .....	64	Korovina, Margarita .....	246
Cho, Sung-Jin .....	77, 165, 367	Kothari, Sunil .....	254
Choi, Un-Sook .....	77, 165	Koutras, Costas .....	117
Ciobanu, Gabriel .....	15	Køber, Petter Kristian .....	504
Conidis, Chris .....	497	Lafitte, Grégory .....	264
Costa, Antônio Carlos .....	87	Le Roux, Stéphane .....	274
Dawar, Anuj .....	495	León, Maikel .....	284
de Miguel Casado, Gregorio .....	97	Leigh, Graham .....	506
Devlin, Keith .....	1	Li, Chung-Chih .....	294
Dimuro, Graçaliz .....	87	Loff, Bruno .....	507
Durand-Lose, Jérôme .....	107	Makowsky, Johann .....	304
Eleftheriou, Pantelis .....	117	Manousaridis, Angelos .....	324
Elouedi, Zied .....	205	Mora Mora, Higinio .....	97
Fischbach, Tim .....	498	Moreira, Nelma .....	3
Fokina, Ekaterina .....	127	Morphett, Anthony .....	508
Galdino, André Luiz .....	137	Moschovakis, Yiannis .....	509
Ganchev, Hristo .....	499	Mostowski, Marcin .....	332
García, Zenaida .....	284	Nagy, Benedek .....	435
García Chamizo, Juan Manuel ....	97	Nasfi, Miriam .....	498
Gavryushkin, Alexander .....	500	Negrea, Romeo .....	503
Gaßner, Christine .....	147	Nigam, Vivek .....	344
Gerber, Annelies .....	45	Nomikos, Christos .....	117
Gerdjikov, Stefan .....	501		

- Oitavem, Isabel ..... 507  
Papakyriakou, Michalis ..... 324  
Papaspyrou, Nikolaos ..... 324  
Pelupessy, Florian ..... 354  
Petrova, Katya ..... 361  
Piao, Yongri ..... 367  
Potgieter, Petrus ..... 377  
Protti, Fábio ..... 35  
Prunescu, Mihai ..... 387  
Qiu, Daowen ..... 397  
Rathjen, Michael ..... 493, 506  
Ratiu, Diana ..... 510  
Reis, Rogério ..... 3  
Revenko, Alexandra ..... 511  
Rubin, Sasha ..... 512  
Sadowski, Zenon ..... 407  
Seyfferth, Benjamin ..... 175  
Sihman, Marcelo ..... 35  
Solon, Boris ..... 361  
Soltys, Michael ..... 415  
Soskov, Ivan N. ..... 513  
Souto, André ..... 25  
Stavrinos, Yiorgos ..... 514  
Stephan, Frank ..... 185  
Tadaki, Kohtaro ..... 425  
Tajti, Ákos ..... 435  
Takahashi, Hayato ..... 445  
Trifonov, Trifon ..... 510  
Vardi, Moshe Y. ..... 2  
Vasilieva, Alina ..... 453  
Veneti, Anastasia ..... 514  
Vorobjov, Nicolai ..... 246  
Wallden, Petros ..... 515  
Weckbecker, Gregor ..... 498  
Weiermann, Andreas ..... 354  
Weiss, Michael ..... 264  
Wilson, Craig ..... 415  
Winter, Joost ..... 463  
Ye, Nan ..... 185  
Yokoyama, Keita ..... 157, 473  
Zwoźniak, Grażyna ..... 483