PRIFYSGOL CYMRU ABERTAWE
UNIVERSITY OF WALES SWANSEA

UNIVERSITY OF WALES SWANSEA

REPORT SERIES

Logical Approaches to Computational Barriers
Second Conference on Computability in Europe, CiE 2006
Swansea, UK, June/July 2006

edited by

Arnold Beckmann, Ulrich Berger, Benedikt Löwe and John V. Tucker

Report # CSR 7-2006

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, June 30-July 5, 2006
Proceedings

University of Wales Swansea
Report Series

Volume Editors

Arnold Beckmann
Ulrich Berger
John V. Tucker
University of Wales Swansea
Department of Computer Science
Singleton Park, Swansea SA2 8PP, UK
E-mail: {a.beckmann,u.berger,j.v.tucker}@swansea.ac.uk

Benedikt Löwe
Universiteit van Amsterdam
Institute for Logic, Language and Computation (ILLC)
Plantage Muidergracht 24, 1018 TV Amsterdam, The Netherlands
E-mail: bloewe@science.uva.nl

University of Wales Swansea
Report # CSR 7-2006

Preface

CiE 2006: Logical Approaches to Computational Barriers
Swansea, Wales, June 30 - July 5, 2006

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 upon 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, and computational learning. Applications are everywhere, especially, in algebra, analysis and geometry, or data types and programming.

This proceedings contains some of the talks given at the second CiE conference that was held at the Department of Computer Science, Swansea University, 30 June - 5 July, 2006. 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. This volume contains 30 of the contributed talks, all of which have been refereed, and 21 informal presentations. The other talks of the conference appeared in the Springer Lecture Notes in Computer Science series, Vol. 3988. There will be a number of post-proceedings publications, including special issues of *Theoretical Computer Science*, *Theory of Computing Systems*, and *Journal of Logic and Computation*.

The first meeting of CiE was at the University of Amsterdam, June 8 - 12, 2005. We are sure that all of the 200+ mathematicians and computer scientists attending that 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. We believe the same is certainly true of the Swansea meeting.

CiE 2005 and CiE 2006 are at the start of a new conference series *Computability in Europe*. The series is coordinated by the CiE Steering Committee:

S. Barry Cooper (Leeds)
Benedikt Löwe (Amsterdam, Chair)
Elvira Mayordomo (Zaragoza)
Dag Normann (Oslo)
Andrea Sorbi (Siena)
Peter van Emde Boas (Amsterdam)

We will reconvene 2007 in Siena, 2008 in Athens, 2009 in Heidelberg, and 2010 in Lisbon.

Organisation and Acknowledgements

CiE 2006 was organised by the logicians and theoretical computer scientists at Swansea: Arnold Beckmann, Ulrich Berger, Phil Grant, Oliver Kullmann, Faron Moller, Monika Seisenberger, Anton Setzer, John V. Tucker; and with the help of S. Barry Cooper (Leeds) and Benedikt Löwe (Amsterdam).

The Programme Committee was chaired by Arnold Beckmann and John V. Tucker and consisted of:

Samson Abramsky (Oxford)
Klaus Ambos-Spies (Heidelberg)
Arnold Beckmann (Swansea, Co-chair)
Ulrich Berger (Swansea)
Olivier Bournez (Nancy)
S. Barry Cooper (Leeds)
Laura Crosilla (Firenze)
Costas Dimitracopoulos (Athens)
Abbas Edalat (London)
Fernando Ferreira (Lisbon)
Ricard Gavaldà (Barcelona)
Giuseppe Longo (Paris)
Benedikt Löwe (Amsterdam)
Yuri Matiyasevich (St.Petersburg)
Dag Normann (Oslo)
Giovanni Sambin (Padova)
Uwe Schöning (Ulm)
Andrea Sorbi (Siena)
Ivan N. Soskov (Sofia)
Leen Torenvliet (Amsterdam)
John V. Tucker (Swansea, Co-chair)
Peter van Emde Boas (Amsterdam)
Klaus Weihrauch (Hagen)

We are delighted to acknowledge and thank the following for their essential financial support: the Department of Computer Science at Swansea, IT Wales, the Welsh Development Agency, the UK's Engineering and Physical Sciences Research Council, the British Logic Colloquium, the London Mathematical Society, and the Kurt Gödel Society of Vienna. Furthermore, we thank our sponsors the Association for Symbolic Logic, the European Association for Theoretical Computer Science, and the British Computer Society.

The high scientific quality of the conference was possible through the conscientious work of the Programme Committee, the special session organisers and the referees. We are grateful to all members of the Programme Committee for

their efficient evaluations and extensive debates, which established the final programme. We also thank the following referees:

A.Pastor	Bert Gerards	Milad Niqui
Klaus Aehlig	Sergey S. Goncharov	Dirk Pattinson
Klaus Ambos-Spies	Emmanuel Hainry	Hervé Perdry
Jeremy Avigad	Neal Harman	Bruno Poizat
George Barmpalias	Takis Hartonas	Chris Pollett
Andrej Bauer	Peter Hertling	Diane Proudfoot
N. K. Kosovsky	Pascal Hitzler	Michael Rathjen
A. P. Beltiukov	Jan Johannsen	Jan Reimann
Jens Blanck	Reinhard Kahle	Robert Rettinger
Stefan Bold	Ker-I Ko	Fred Richman
Ana Bove	Margarita Korovina	Piet Rodenburg
Vasco Brattka	Sven Kosub	Panos Rondogiannis
Robert Brijder	Oliver Kullmann	Anton Setzer
Andrei Bulatov	Simone Martini	Cosma Shalizi
Wesley Calvert	Ugo Dal Lago	Dieter Spreen
Véronique Cortier	Antoni Lozano	Umberto Straccia
K. Djemame	Maria Emilia Maietti	Bas Terwijn
L. Crosilla	Guillaume Malod	Neil Thapen
Victor Dalmau	Klaus Meer	John Tucker
Mário Jorge Edmundo	Wolfgang Merkle	Sergey Verlan
Ioannis Emiris	Christian Michaux	Jouko Väänänen
Juan L. Esteban	Joseph Miller	Philip Welch
Graham Farr	Faron Moller	Damien Woods
T. Flaminio	F. Montagna	Y. Yang
Hervé Fournier	Yiannis Moschovakis	Xizhong Zheng
Torkel Franzén †	Philippe Moser	Ning Zhong
Nicola Galesi	Sara Negri	Martin Ziegler
Nicola Gambino	Stela Nikolova	Jeff Zucker

Of course, the conference was primarily an event, and a rather complicated event at that. We are delighted to thank our colleagues on the Organising Committee for their many contributions and our research students for practical help at the conference. We owe a special thanks to Beti Williams, director of IT Wales, and her team for invaluable practical work. Finally, we thank Andrej Voronkov for his Easy Chair system which facilitated the work of the Programme Committee and the editors considerably.

Swansea and Amsterdam, April 2006

Arnold Beckmann
Ulrich Berger
Benedikt Löwe
John V Tucker

After completing this volume, we heard the sad news that our invited Special Session speaker, Torkel Franzén, died on April 19, 2006. Torkel Franzén's work on the philosophy of logic and mathematics had gained more and more international recognition in recent years. His death is a huge loss for the scientific community and he will be very much missed at CiE 2006. Torkel Franzén did send us an abstract of his planned contribution to this conference which is included in the other part of the proceedings, LNCS, Vol. 3988.

We are grateful to Andrew Hodges for giving a Special Session talk at Torkel Franzén's place. The abstract of Andrew Hodges' talk is included as the first item in this volume.

The Editors

Table of Contents

Gödel's Remarks on Turing's Work *(abstract of an invited special session talk)* 1
Andrew Hodges

Contributed Talks

Integrating Functional Programming Into C++: Implementation and Verification 2
Rose Hafsah Abdul Rauf

Relativity Theory for Logicians and New Computing Paradigms 12
Hajnal Andréka

Generalized Tabular Reducibilities in Infinite Levels of Ershov Hierarchy . 15
Marat Arslanov

Note on Reducibility Between Domain Representations 24
Jens Blanck

Gödel, Turing, the Undecidability Results and the Nature of Human Mind 37
Riccardo Bruni

The Conjecture $P \neq NP$ Presented by Means of Some Classes of Real Functions 47
Jose Felix Costa, Jerzy Mycka

Decidability of Arithmetic Through Hypercomputation: Logical Objections 58
Paolo Cotogno

On Generalising Predicate Abstraction 68
Birgit Elbl

Brownian Motion and Kolmogorov Complexity 78
Willem L. Fouché

A Structure with $P = NP$ 85
Christine Gaßner

Expansions of Structures with $P = NP$ 95
Christine Gaßner

On Complexity of Ehrenfeucht Theories with Computable Model. 105
Alexander Gavryushkin

Functional Interpretation and Modified Realizability Interpretation of the Double-Negation Shift 109
Philipp Gerhardy

Toward Combinatorial Proof of $P < NP$. Basic Approach 119
Lev Gordeev

Models of Timing Abstraction in Simultaneous Multithreaded and Multi-Core Processors 129
Neal A. Harman

Finite Prediction of Recursive Real-Valued Functions 140
Eiju Hirowatari, Kouichi Hirata, Tetsuhiro Miyahara

The Dyment Reducibility on the Algebraic Structures and on the Families of Subsets of ω 140
Iskander Kalimullin

Computing the Recursive Truth Predicate on Ordinal Register Machines . 160
Peter Koepke, Ryan Siders

Logic Programs with Uncertainty: Neural Computation and Automated Reasoning 170
Ekaterina Komendantskaya, Anthony Seda

Clocking Type-2 Computation in the Unit Cost Model 182
Chung-Chih Li

On the Calculating Power of Laplace's Demon 193
John Longley

Scaled Dimension of Individual Strings 206
Maria Lopez-Valdes

Solving a PSPACE-Complete Problem by a Linear Interval-Valued Computation 216
Benedek Nagy, Sándor Vályi

Hypercomputing the Mandelbrot Set? 226
Petrus Hendrik Potgieter

Definability of the Field of Reals in Admissible Sets 236
Vadim Puzarenko

The Algebra of Labeled Forests Modulo Homomorphic Equivalence 241
Victor Selivanov

Third-Order Computation and Bounded Arithmetic 251
Alan Skelley

On Inner Constructivizability of Admissible Sets 261
Alexey Stukachev

A Sharp Phase Transition Threshold for Elementary Descent Recursive Functions 268
Andreas Weiermann, Arnoud den Boer

Some Reflections on the Principle of Image Collection 282
Albert Ziegler

Abstracts of Informal Presentations

Information Content and Computability in the n-C.E. Hierarchy 289
Bahareh Afshari

Computing with Newtonian Machines 290
Edwin Beggs, John V. Tucker

Infinite Time Turing Computation 291
Barnaby Dawson

On a Problem of J. Paris 292
Costas Dimitracopoulos, Alla Sirokofskich

Risk Management in Grid Computing 293
Odej Kao, Karim Djemame

Some Mathematical Properties of Propositional Input Resolution Refutations with Non-Tautological Resolvents 294
Annelies Gerber

A Similarity Criterion for Proofs 295
Mircea-Dan Hernest

On Real Primitive Recursive Functions and Differential Algebraicity 296
Stefan Hetzl

NdE - Normalization during Extraction 297
Akitoshi Kawamura

Non-Unitary Quantum Walks: Exploring the Space Between Classical and Quantum Computing 298
Viv Kendon

Logical Characterization of the Counting Hierarchy 299
Juha Kontinen

Gödelian Foundations of Non-Computability and Heterogeneity In Economic Forecasting and Strategic Innovation 300
Sheri Markose

Sequent Calculi and the Identity of Proofs 301
Richard McKinley

The Key to the Universe, Part 2 302
Robert K. Meyer

Questions Concerning the Usefulness of Small Universal Systems 303
Liesbeth De Mol

Basic Model Theory for Bounded Theories 304
Morteza Moniri

The Method Of Approximation In Real Computation 305
Kerry Ojakian

Long Games with a Short Memory 306
Alexander Rabinovich, Amit Shomrat

Extensions of the Semi-lattice of the Enumeration Degrees 307
Ivan N. Soskov

Genericity and Nonbounding 309
Mariya Ivanova Soskova

Gödel's Remarks on Turing's Work

Andrew Hodges

Wadham College, Oxford, UK

Gödel originally endorsed Turing's 1936 definition of computability, but then, much later, questioned Turing's assumptions about finitely many states of mind. I will discuss this topic in the light of Turing's mechanistic theory of mind as it developed in the 1940s.

Integrating Functional Programming Into C++: Implementation and Verification

Rose Hafsah Abdul Rauf *

Department of Computer Science, University of Wales Swansea

Abstract. We describe a parser-translator program that translates typed λ-terms to C++ classes so as to integrate functional programming. We prove the correctness of the translation with respect to a denotational semantics using Kripke-style logical relations.

1 Introduction

C++ is a general purpose language that supports object oriented programming as well as procedural and generic programming, but unfortunately not functional programming. We have developed a parser-translator program that translates a simply typed λ-term to C++ statements so as to integrate functional progamming. This translated code uses the C++ object oriented concept of classes and inheritance in the definition of the λ-term. We build a mathematical model from the formal semantics of the translated code to prove its correctness. First, we give the denotational semantics of the typed λ-calculus. Then the correctness of the implementation of the typed λ-calculus by C++ classes is proved with respect to the denotational semantics. The correctness proof of the translated code is based on a Kripke-style of logical relation between the C++ class and the denotational model.

The parser-translator program that has been developed will parse a string represention of simply typed λ-term and translate it to a sequence of C++ statements. The translation of this λ-term will be discussed in the next section. How the translated code is executed will also be discussed along with the representation of the memory allocation. The mathematical model was based on the execution of the translated code. In building up this mathematical model, we will first give the denotational semantics of the typed λ-calculus. Then we will implement the C++ classes with the denotational semantics. These will be discussed in section 3. Some related and future work on integrating functional programing into C++ will be discussed at the end of this paper.

The approach of using denotational semantics and logical relation in proving the correctness of programs has been used before by researchers such as Plotkin[7], and many others. The method of logical relation can be traced back

* This paper is part of my PhD project and I would like to thank my supervisors Dr. Ulrich Berger and Dr. Anton Setzer for their knowledge and guidance making it possible for me to complete it.

at least to Tait[14] and has been used for a large variety of purposes (eg. Jung and Tiuryn[2], Statman[11] and Plotkin[6]). To our knowledge the verification of the implementation of λ-calculus in C++ using logical relation is new.

2 Translation

For the purpose of explaining how the λ-term is translated to its equivalent C++ statements and execution of the translated code, we will give an example of a λ-term input to the parser-translator program where the term must follow a syntax that has been determined. A λ-term $\lambda\, x^{\text{int}}\, .\, t$ where t is of the type `int` is written in our syntax as `\int x.int t` .
The statement shown below is the string that was entered to the parser-translator program:

```
int k =(\(int->int) f.\int x.int f^^(f^^x))^^(\int x. int x+2)^^3;
```

and it is equivalent to $k = (\lambda f^{(\text{int}\to\text{int})} \cdot \lambda x^{\text{int}} \cdot f(fx))(\lambda x^{\text{int}} \cdot x + 2)3$

First, the function type is defined as an abstract class with a virtual operator() method that will be overloaded in the definition of the λ-term and the type itself is the type pointers to an object of this abstract class. For the type class names we make use of letters `C` and `D` to represent open and close brackets respectively and an underscore for an arrow. For example, `Cint_intD` means (int→int). The concept of inheritance are involved in the definition of a λ-term where the function type abstract class will be the base class for the λ-term class.

The subterm `\int x.int f^^(f^^x)` in the statement above is translated as an instance of the class `Cint_intD_aux` ;

```
class lambda1 : public Cint_intD_aux{
 public :Cint_intD f;
 lambda1( Cint_intD f)  {   this-> f = f;};
 virtual int operator () (int x)
 { return (*(f))((*(f))(x)); };
};
```

The subterm `\(int->int) f.int x.int f^^(f^^x)` is translated as an instance of `CCint_intD_Cint_intDD_aux`:

```
class lambda0 : public CCint_intD_Cint_intDD_aux{
 public :
 lambda0( ) { };
 virtual Cint_intD operator () (Cint_intD f)
 { return new  lambda1( f); }
};
```

and the λ-term `\int x.int x+2` is translated as follows :

```
 class lambda2 : public Cint_intD_aux{
 public :
 lambda2( ) { };
 virtual int operator () (int x)
 { return x + 2; };
};
```

The term `k` will be finally translated as the expression :

```
int k = (*((*( new lambda0( )))( new lambda2( ))))(3);
```

The classes for the λ-terms are instantiated by statements new lambda0() and new lambda2() where pointers will be created that point to the addresses of the classes on the heap. The heap, which is also known as free store, is a dynamic store in the memory. Classes are created for each λ-term objects and each class has a pointer to its address on the heap. The evaluation for the expression above follows the call-by-value evaluation strategy (which will result in the value of 7). Note that the storage allocated for the instances of the classes are not deleted afterwards. The deletion depends on the garbage collection version of C++. One can also use smart pointers to enforce deletion.

3 Proof of correctness

Before we start building a mathematical model of the translated code, we list some of the mathematical preliminaries that will be frequently used in this section. The presentation of the proof follows the style of Winskel[15].

3.1 Mathematical preliminaries

Mappings

1. If X, Y are sets, then a list $m = (x_1 : y_1), \ldots, (x_n : y_n) \in \text{list}(\text{X} \times \text{Y})$ is considered as a finite map from X to Y which is defined as follows : If $x \in$ X, $y \in$ Y, then $m(x) := y$ where $x = x_i, y = y_i$ and $x \neq x_j$ for $j > i$.
2. We usually define dom(m) = the domain of $m = x_1, \ldots, x_n$.
   If $x \in$ X ,$y \in$ Y, then $m[x \mapsto y]$:= m, (x,y), the extension of the list m by (x,y). Note that dom($m[x \mapsto y]$) = dom(m) $\cup\{x\}$ and

$$m[x \mapsto y](x') = \begin{cases} y & \text{if } x' = x \\ m(x') & \text{if } x' \in \text{dom}(m) \backslash \{x\} \quad (x' \in \text{X}) \end{cases}$$

3.2 Implementation of the typed λ-calculus

a) **Types**
The set **Typ** of types is inductively given by :

i) Int $\in$ Typ

ii) if $A, B \in \text{Typ}$, then $A \to B \in \text{Typ}$

b)**Terms**
The **Terms** for the λ-calculus can be any of the following shown below.

i) $n \in \text{N}$ (any number)
ii) $x \in \text{Var}$ (where Var = String)
iii) $r\,s$ (term r is applied to term s)
iv) $\lambda x : A.r$ (term is an abstraction)
v) $f[r_1 \ldots r_n] = f[\boldsymbol{r}]$ ($f \in \mathcal{F}$ is a set of names for computable functions on N).The function denoted by f is written as $[\![f]\!]$

c) **Typing**
A **Context Γ** is a map from variables to types i.e. a list of variables and their type : **Context=list(Var× Typ)**
Context will be denoted as $\Gamma = x_1 : A_1, \ldots, x_n : A_n$
The **Typing** rules of the simply typed λ-calculus are :

i)
$$\frac{}{\Gamma, x : A \vdash x : A}$$

ii)
$$\frac{}{\Gamma \vdash n : \text{Int}}$$

iii)
$$\frac{\Gamma, x : A \vdash r : B}{\Gamma \vdash \lambda x r : A \to B}$$

iv)
$$\frac{\Gamma \vdash r : A \to B \qquad \Gamma \vdash s : A}{\Gamma \vdash rs : B}$$

v)
$$\frac{f : \text{Int} \times \ldots \times \text{Int} \to \text{Int} \qquad \Gamma \vdash r_1 : \text{Int} \ldots \Gamma \vdash r_n : \text{Int}}{\Gamma \vdash f[r_1, \ldots, r_n] : \text{Int}}$$

d) **Denotational semantics**
The sets of **functionals** of type A denoted as $\text{D}(A)$ are defined as follows :

i) $\text{D}(\text{Int}) = \text{N}$
ii) $\text{D}(A \to B) = \{f | f : \text{D}(A) \to \text{D}(B)\}$
iii) $\text{D} := \biguplus_{A \in \text{Typ}} \text{D}(A)$ where $\biguplus$ denotes disjoint union

A **Functional Environment** is a mapping of $\xi : \text{Var} \to \text{D}$. We let $\text{FEnv} := \text{Var} \to \text{D}$ be the set of all functional environments. If Γ is a context, then $\xi : \Gamma$ means $\forall x \in \text{dom}(\Gamma).\xi(x) \in \text{D}(\Gamma(x))$.

For every typed λ-term $\Gamma \vdash r : A$ and every functional environment $\xi : \Gamma$ the denotational value $[\![r]\!]\xi \in \text{D}(A)$ is defined as follows :

i) $[\![n]\!]\xi = n$
ii) $[\![x]\!]\xi = \xi(x)$

iii) $[\![r\ \ s]\!]\xi = [\![r]\!]\xi([\![s]\!]\xi)$
iv) $[\![\lambda x : A.r]\!]\xi(a) = [\![r]\!]\xi[x \mapsto a]$
v) $[\![f[\boldsymbol{r}]]\!] = [\![f]\!]([\![\boldsymbol{r}]\!]\xi)$

By an **implementation** of the typed λ-calculus we mean an (implementation of an) algorithm computing for every closed term r : Int the value $[\![r]\!] \in$ N.

3.3 Implementation by C++ classes

The classes that will be created depend on the λ-term that is being parsed, the more complex the term is the more level of classes will be created. When the class is instantiated, an address of the class will be stored on the heap, and further instantiation of other classes will create a stack of addresses on the heap with addresses of any variables which is bound to the classes (or λ-term).

Every class is instantiated by calling the constructor of the object i.e. the name of the class with or without any arguments.The body of a λ-abstraction is associated with an applicative term ($\in$App, below). The complete list of syntactic sets associated with the C++ classes is as follows:

- **Addr = Int**
  These are addresses of classes or variables on the heap.
- **Constr = String**
  Constructors are names of classes.
- **Val = Int + Addr**
  A value is either an integer or an address of a class or variables.
- **App = Int + Var +** $\mathcal{F}\times$ **list(App) + App × App + Constr × list(App)**
  Applicative terms representing bodies of λ-abstraction.
- **Abst = Var × Typ × Context × App**
  Abstractions consist of the variables and the type bound to the abstraction, and the context which is the list of variables and their types and the application.
- **Env = list(Var × Val)**
  An environment is the list of variables and their values.
- **Heap = list(Addr × Constr × list(Val))**
  The heap consists of list of addreses of constructors and their lists of values of the variables.
- **Class = list(Constr × Abst)**
  The class environment consists of list of constructors and their abstractions.

We assume that every $f \in \mathcal{F}$ is given by a side effect free C++ function

a) **The evaluation of λ-terms in C++**

When a λ-term is executed, a class address of the application of the λ-term is created on the heap and, with respect to the environment, a λ-term is evaluated to the value and an extended heap. This extended heap contains the address of

the value that has been evaluated for the λ-terms. Thus, the functionality of the evaluation function (**eval**) is :

$$\textbf{eval : Heap} \rightarrow \textbf{Env} \rightarrow \textbf{App} \rightarrow \textbf{Val} \times \textbf{Heap}$$

For a function application, where a λ-term is applied to another λ-term, the heap, which contains the class address of the two terms with the two values evaluated from the two terms, will evaluate to a value and an extended heap. Thus, the functionality of the application function (**apply**) is :

$$\textbf{apply : Heap} \rightarrow \textbf{Val} \rightarrow \textbf{Val} \rightarrow \textbf{Val} \times \textbf{Heap}$$

In the definition of the function **eval** and **apply** we fix some C:Class.
In presenting the evaluation rules we will follow the convention that :

- n ranges over numbers **N**
- x ranges over variables **Var**
- a, b range over application **App**
- v, w range over values **Val**
- k ranges over addresses **Addr**
- H ranges over **Heap**
- c ranges over constructors **Constr**
- C ranges over **Class**
- A, B range over **Typ**
- η ranges over **Env**

The metavariables we use to range over the syntactic categories can be primed or subscripted. For example, H, H', H'', H_k stand for heaps, C, C', C'' stand for classes and v_1, v' stand for values.

The rules for the evaluation of the λ-terms are as follows:

i) **Evaluation of a λ-term where application is a number:**

$$\text{eval}\, H\, \eta\, n = (n, H)$$

ii) **Evaluation of a λ-term where application is a variable:**

$$\text{eval}\, H\, \eta\, x = (\eta(x), H)$$

iii) **Evaluation of a λ-term where application is a function with a list of arguments:**

$$\text{eval}\, H\, \eta\, f[\boldsymbol{a}] = (\llbracket f \rrbracket(\boldsymbol{n}), H_k)$$

where $\boldsymbol{a} = a_1, \ldots, a_k, \boldsymbol{n} = n_1, \ldots, n_k$ and eval* $H\, \eta\, \boldsymbol{a} = (\boldsymbol{n}, H_k)$.
Here we define eval* $H\, \eta\, \boldsymbol{a} = (\boldsymbol{n}, H_k)$ if eval $H \eta\, a_1 = (n_1, H_1), \ldots,$ eval $H_{k-1}\, \eta\, a_k = (n_k, H_k)$. H_k is not changed by f because $f \in \mathcal{F}$ has no side effect.

iv) **Evaluation of a λ-term where the application is the application of one term to the other:**

$$\text{eval}\, H\, \eta\, (a\ \ b) = \text{apply}\, H''\, v\, w = (v', H''')$$

where $\text{eval}\, H\, \eta\, a = (v, H'),\ \text{eval}\, H'\, \eta\, b = (w, H'')$.
The definition of **apply** in detail is shown as follows :

$$\text{apply H}\, k\, v = \text{eval}\, H\, [x, \boldsymbol{y} \mapsto v, \boldsymbol{w}]a$$

where $H(k) = (c, \boldsymbol{w})$, $C(c) = (x : A; \boldsymbol{y} : \boldsymbol{B}; a)$ (assuming $c \in \text{dom}(C)$).

v) **Evaluation of a λ-term where the application is a constructor with a list of arguments:**

$$\text{eval}\, H\, \eta\, c[\boldsymbol{a}] = (k, H'[k \mapsto c[\boldsymbol{v}]]) \quad (k \in \textbf{Addr}, v \in \textbf{Val})$$

where eval* $H\, \eta\, \boldsymbol{a} = (\boldsymbol{v}, H')$ and $k =$ new H' (new H' is an address not in $\text{dom}(H')$).

In all other cases for the application, it is termed invalid and an error will be returned.

Lemma 1. *1. eval* $H\, \eta\, a = (v, H') \Longrightarrow H \subseteq H'$
2. apply $H\, v\, w = (v', H') \Longrightarrow H \subseteq H'$
*3. eval** $H\, \eta\, \boldsymbol{a} = (\boldsymbol{n}, H') \Longrightarrow H \subseteq H'$

The proof for Lemma 1 is by induction on the definition of **eval** and **apply**.

Note that, since **eval** and **apply** depend on C:Class, the true signatures of eval and apply are as follows :
eval : Class→Heap→Env→App→Val×Heap
apply : Class→Heap→Val→Val→Val×Heap
We write $\textbf{eval}_C\, H\, \eta\, a$ and $\textbf{apply}_C\, H\, v\, w$ if the argument C:Class is to be made explicit.

b) **The Parsing of a λ Term**

Traditionally, a λ-term that is input is parsed as a long string which will undergo several steps of parsing to get the translated code. The parsing will create classes for the λ-term where in the case of a complex λ-term it will create several levels of classes where the class of an upper level is an extension of the lower level class. In order to simplify things and to concentrate on the most important aspects of the problem we assume that the input is given as an abstract term rather than a string. The parsing from a string to a term is a traditional parsing problem which is of no interest here. What is interesting here is the process of creating a system of C++ classes that represents a λ-term.

In order to give a recursive description of this process, we must assume that the term in question is not the first term being parsed, but other terms (or subterms) have been parsed before having created a system of classes. Furthermore,

if the term has free variables, then the types of these variables must be fixed by an appropriate context. Therefore, the parser **P** has the following functionality :

$$\mathbf{P : Class{\to}Context{\to}Term \to App{\times}Class}$$

The rules for the parsing are as follows :

i) **Parsing when the term is a number:** $\mathrm{P}_C\Gamma n = (n, C)$
ii) **Parsing when the term is a variable:** $\mathrm{P}_C\Gamma x = (x, C)$
iii) **Parsing when the term is a function with a list of arguments:**

$$\mathrm{P}_C\Gamma f[\boldsymbol{r}] = (f[\boldsymbol{a}], C')$$

where $\mathrm{P}^*{}_C\Gamma\boldsymbol{r} = (\boldsymbol{a}, C')$ and P* is defined in a smilar way as eval*.
iv) **Parsing of an application:** $\mathrm{P}_C\Gamma(r\ \ s) = (a\ \ b, C'')$
where $\mathrm{P}_C\Gamma r = (a, C'),\ \mathrm{P}_{C'}\Gamma s = (b, C'')$
v) **Parsing of a λ abstraction :** $\mathrm{P}_C\Gamma(\lambda x : A.r) = (c[\boldsymbol{y}], C'[c \mapsto (x : A; \Gamma; a)])$
where $\boldsymbol{y} = \mathrm{dom}(\Gamma), \mathrm{P}_C\Gamma[x \mapsto A]r = (a, C'),$ and $c =$ new C'
meaning that c is a name of a class that is "new" i.e. has not been used before.

Remark: We only generate $c[\boldsymbol{x}] \in$ App with $\boldsymbol{x} \in$ list(Var) and not $c[\boldsymbol{a}]$ with arbitary $\boldsymbol{a} \in$ list(App)

Lemma 2. *i)* $P_C\Gamma r = (a, C') \Longrightarrow C \subseteq C'$
ii) $P^*{}_C\Gamma\boldsymbol{r} = (\boldsymbol{a}, C') \Longrightarrow C \subseteq C'$

The proof for Lemma 2 is by induction on r respectively $\boldsymbol{r}$.

3.4 The correctness of the translated code

The correctness proof of the translated code is based on a Kripke-style relation between the C++ representation of the term ($\in$ Val $\times$ Heap) and its denotational value ($\in \mathrm{D}(A)$). The relation is indexed by the class environment C and the type A of the term. Since in the case of an arrow type, $A \to B$, extensions of H and C have to be taken into account, this definition has some similarity with Kripke models. The relation

$$\sim^C_A \subseteq (\mathrm{Val} \times \mathrm{Heap}) \times \mathrm{D}(A) \text{ where } A \in \ \mathrm{Typ}\ , C \in \mathrm{Class}$$

is defined by recursion on A as follows:

$$\begin{aligned}(v, H) \sim^C_{\mathrm{Int}}\ n &:\Longleftrightarrow v = n\\ (v, H) \sim^C_{A\to B}\ f &:\Longleftrightarrow \forall C \subseteq C', \forall H \subseteq H', \forall (w, d) \in \mathrm{Val} \times \mathrm{D}(A) :\\ &\qquad (w, H') \sim^{C'}_A\ d \Longrightarrow \mathrm{apply}_{C'} H' v w \sim^{C'}_B\ f(d)\end{aligned}$$

We also set $(\eta, H) \sim^C_\Gamma \xi := \forall x \in\ \mathrm{dom}\ \Gamma(\eta(x), H) \sim^C_{\Gamma(x)} \xi(x) \in \mathrm{D}(\Gamma(x))$.

Lemma 3.

$$(v, H) \sim_A^C d, C \subseteq C', H \subseteq H' \Longrightarrow (v, H') \sim_A^{C'} d$$

The proof for Lemma 3 is by induction on A.

Our main theorem, which corresponds to the usual "Fundamental Lemma" or "Adequacy Theorem" for logical relations, reads as follows:

Theorem 1. *If* $\eta : Env, \xi : FEnv, \Gamma \vdash r : A, \xi : \Gamma, P_C \Gamma r = (a, C'), C' \subseteq C'', (\eta, H) \sim_\Gamma^{C''} \xi$, *and* $H \subseteq H'$, *then* $eval_{C''} H' \eta a \sim_A^{C''} [\![r]\!]\xi$.

The theorem can be proved by an induction on the typing judgement $\Gamma \vdash r : A$ using the Lemma 1-3 above. Due to limited space we omit details.

For a closed term r , we define $\mathrm{P}r = \mathrm{P}_\emptyset \emptyset r$.

Corollary 1 ((Correctness of the implementation)). *If* $\vdash r : Int, Pr = (a, C), C \subseteq C'$, *then for any heap* H, $eval_{C'}\, H\, \eta\, a = ([\![r]\!], H')$ *for some* $H' \supseteq H$.

4 Conclusion

The aim of this paper was to introduce a new approach of integrating functional programming into C++ and to show a method of proving the correctness of the translation code produced by denotational semantics and logical relation. In the past, several researches [3],[4] discovered that C++ can be used for functional programming by representing first class functions and higher order functions using classes, and by this technique we produced the translated code. There are other approaches that have made C++ a language that can be used for functional programming such as FC++ library [5] (a very elaborate approach), FACT! [13] (extensive use of templates and overloading) and [3] (creating macros that allow creation of single macro-closure in C++). The advantages of our solution are that it is very simple, it uses classes and inheritance in an essential way and, most importantly, we have a formal correctness proof.

In addition to the mathematical proof given in this paper, the correctness of the translated code produced by the parser-translator program has been verified by testing it with several types of λ-term from simple to complex ones.

Future work. This work can be extended by integrating lazy constructors (infinite structures) and lazy evaluation, having terms with side effect and integrating recursive higher order functions [1].

References

1. Abdul Rauf R.H., Berger U., Setzer A.: Functional Concepts in C++, To appear in Proceedings of TFP 2006, http://www.cs.nott.ac.uk/∼nhn/TFP2006.

2. Jung A., Tiuryn J.: A New Characterization of Lambda Definability. Typed Lambda Calculus and Applications, 1993.
3. Kiselyov O.: Functional Style in C++ : Closures, Late Binding, and Lambda Abstraction. Poster presentation, Int. Conf. on Functional Programming, 1998.
4. Läufer K.: A Framework For Higher Order functions in C++. Proc. Conf. Object Oriented Technologies(COOTS), Monterey, C.A., June 1995.
5. McNamara B., Smaragdakis Y.: Functional Programming in C++. ICFP '00, Montreal Canada,ACM Press, 2000.
6. Plotkin G. D.: Lambda Definability in the Full Type Hierarchy. To H.B. Curry; Essays on Combinatoric Logic, Lambda Calculus and Formalism., J.P.Seldin , J.R. Hindley, eds., 363-373, 1980.
7. Plotkin G. D.: LCF Considered As a Programming Language. Theoretical Computer Science, 5:223-255, 1977.
8. Polak W.: Program Verification Based On Denotational Semantics. Proceedings of the 8^{th} ACM, SIGPLAN-SIGACT Symposium on Principles Of Programing Analysis. ACM Press, Jan. 1981.
9. Scott, D., Strachey, C.: Mathematical Semantics For Computer Language. Tech. Monograph PRG-6, Programming Research Group, University Of Oxford, 1971.
10. Setzer A.: Java as a Functional Programming Language. Types for Proofs and Programs: International Workshop, Types 2002, Berg en Dal,April 24-28,2002. Selected Papers, Geuver H., Wiedijk F., eds, 279-298, LNCS 2646, 2003.
11. Statman R.: Logical Relation and the Typed λ Calculus. Information and Control, 65:85-97, 1985.
12. Stoy, J: Denotational Semantics - The Scot-Strachey Approach To Language Theory. MIT Press, Cambride, 1977.
13. Striegnitz J. : FACT!-The Functional Side of C++. http://www.fz-juelich.de/zam/FACT.
14. Tait W.: Intentional Intrepretation of Funtional of Finite Type I. Journal Of Symbolic Logic, 32(2):198-212, 1967.
15. Winskel, G. : The Formal Semantics Of Programming Languages : an Introduction. Massachusetts Institute Of Technology, 1993.

Relativity Theory for Logicians and New Computing Paradigms

Hajnal Andréka

Rényi Institute of Mathematics, Budapest P.O.Box 127, H-1364 Hungary,
andreka@renyi.hu,
WWW home page: http://www.renyi.hu/ andreka/

Abstract. Physical foundation for hypercomputing is provided by general relativity. In turn, here logical foundation for relativity is presented making relativistic hypercomputation self-contained for logicians and computer scientists. Further, new convergence phenomena between computing, AI, foundations of science, emergence, new cosmology, black hole physics are highlighted.

Because of hypercomputation and unconventional approaches to computation, new interconnections arose recently between computability theory, physics, and mathematical logic. This seems to be a part of an even broader new convergence phenomenon between branches of science. An example is the research area of relativistic computers [6], [9], [14].

In this perspective, there are two ways of looking at the present paper.

(i) We intend to provide the relativity theoretic background for the invited paper on relativistic approaches to hypercomputation by I. Németi in this volume [13]. No familiarity with physics will be presupposed, we will rely solely on mathematical logic. I.e. we present a logic based version of relativity theory designed for a logically trained audience and we will aim for explaining those theorems of relativity which serve as key ingredients in the design of relativistic hypercomputation.

(ii) Using the experience gained through the success-story of foundation of mathematics [7], we propose a logic-based foundation for the theory of spacetime and via this for theories of relativity. The foundation of mathematics is built up in mathematical logic, in particular, in first-order-logic (FOL), hence we build up spacetime theory (and relativity) also in FOL via a small number of clearly understandable and transparent axioms (formulated explicitly in FOL) [1], [11], [12]. Besides working on unambiguous foundation, we also aim for insights into the nature of relativity, and also for a conceptual analysis of the theory. In addition to the reasons coming from hypercomputation, motivation for building up a conceptual analysis for relativity comes from the historical fact that nowadays relativity, spacetime and cosmology are in the focus of attention and are going through revolutionary paradigm-shifts profoundly influencing our scientific world-view. Hence these fields both need and are worthy of a substantial help from the field of (mathematical) logic.

Besides the above, a further motivation for applying logic to relativity comes from a recent trend of convergence between ideas in relativity+cosmology (cf. e.g. Lee Smolin Life of Cosmos [15]), emergence, cybernetics, and life sciences, general system theory, evolving and self organizing systems, AI, e.g. Barry Cooper [4], Weidermann-Leeuwen [18]. Cf. also the homepage of Andrei Linde and [10].

References

1. Andréka, H., Madarász, J. X. and Németi, I., Logic of Spacetime. In: Logic of Space, eds: M. Aiello, J. van Benthem, and I. Hartman-Pratt, Kluwer.
2. Andréka, H., Madarász, J. X. and Németi, I., Logical axiomatizations of spacetime. In: Non-Euclidean Geometries, János Bolyai Memorial Volume. Ed. A. Prékopa and E. Molnár, Mathematics and its Applications Vol. 581, Springer 2006. pp.155-185.
3. Andréka, H., Madarász, J. X. and Németi, I. with contributions from Andai, A., Sain, I., Sági, G., Tőke, Cs. and Vályi, S., On the logical structure of relativity theories. Internet book, Budapest, 2000. http://www.math-inst.hu/pub/algebraic-logic/olsort.html
4. Cooper, B., Computability and Emergence. In: Mathematical Problems from Applied Logics. New Logics for the XXIst Century. Ed: Gabbay, D., International Mathematical Series Vol. 5, Springer, 2005. pp.194-233
5. Earman, J., Bangs, crunches, whimpers, and shrieks. Singularities and acausalities in relativistic spacetimes. Oxford university Press, Oxford, 1995.
6. Etesi, G. and Németi, I., Turing computability and Malament-Hogarth spacetimes. International Journal of Theoretical Physics 41,2 (2002), 342-370.
7. H. Friedman, On foundational thinking 1, Posting in FOM (Foundations of Mathematics) Archives www.cs.nyu.edu (January 20, 2004).
8. Gödel, K., Lecture on rotating universes. In: Kurt Gödel Collected Works, Vol. III. Eds.: Feferman, S., Dawson, J. S., Goldfarb, W., Parson, C. and Solovay, R. N., Oxford University Press, New York Oxford 1995. pp. 261-289.
9. Hogarth. M. L., Deciding arithmetic using SAD computers. Brit. J. Phil. Sci. 55 (2004), 681-691.
10. Linde, A., Inflation, quantum cosmology and the Anthropic Principle. In: "Science and Ultimate Reality: From Quantum to Cosmos", honoring John Wheeler's 90th birthday. J. D. Barrow, P.C.W. Davies and C. L. Harper eds. Cambridge University Press, 2003.
11. Madarász, J. X., Németi, I. and Székely, G., Twin Paradox and the logical foundation of space-time. Foundation of Physics, to appear. arXiv:gr-qc/0504118.
12. Madarász, J. X., Németi, I. and Székely, G., First-order logic foundation of relativity theories. In: New Logics for the XXIst Century II, Mathematical Problems from Applied Logics, International Mathematical Series Vol 5, Springer. To appear. arXiv:gr-qc/0604041.
13. Németi, I. and Andréka, H., Can general relativistic computers break the Turing barrier? In: CiE 2006, A. Beckmann et al. (eds.), LNCS 3988, Springer 2006. pp.398-412.
14. Németi, I. and Dávid, Gy., Relativistic computers and the Turing barrier. Journal of Applied Mathematics and Computation, to appear.
15. Smolin, L., The life of the Cosmos. Oxford University Press, Oxford, 1997.
16. Taylor, E. F. and Wheeler, J. A., Black Holes. Addison, Wesley, Longman, San Francisco, 2000.

17. Tegmark, M., Parallel Universes. Scientific American May 2003, pp.41-51.
18. Wiedermann, J. and van Leeuwen, J., Relativistic computers and non-uniform complexity theory. In: Calude et al (eds.) UMC 2002. Lecture Notes in Computer Science Vol. 2509, Springer-Verlag, Berlin, 2002. pp.287-299.

Generalized Tabular Reducibilities in Infinite Levels of the Ershov Difference Hierarchy

Marat M. Arslanov

Kazan State University, Kazan, Russia, *
Marat.Arslanov@ksu.ru

Abstract. In this paper we consider a collection of reducibilities which are intermediate between Turing and truth table reducibilities and study properties of these reducibilities relatively to infinite levels of the Ershov hierarchy which are defined by means of limit constructive ordinals.

1 Infinite Levels of the Difference Hierarchy

It is known that finite levels and the ω-level of the Ershov difference hierarchy are connected with bounded truth table and truth table reducibilities accordingly. In this paper I consider a collection of reducibilities which are intermediate between Turing and truth table reducibilities and have similar properties relatively to infinite levels of the Ershov hierarchy which are defined by means of limit constructive ordinals.

First we give a brief survey of principal definitions and results on the hierarchy. The following theorem was a sourse of all successive considerations.

Theorem 1. *(Shoenfield, Ershov) A set A is T-reducible to $\emptyset'$ if and only if there exists a uniformly computable sequence of c.e. sets $\{R_x\}_{x\in\omega}$ such that $R_0 \supseteq R_1 \supseteq \ldots$, $\bigcap_{x=0}^{\infty} R_x = \emptyset$, and*

$$A = \bigcup_{x=0}^{\infty} (R_{2x} - R_{2x+1}).$$

Definition 1. *A set A is n-computably enumerable (n-c.e. set), if either $n = 0$ and $A = \emptyset$, or $n > 0$ and there exist c.e. sets $R_0 \subseteq R_1 \subseteq R_2 \subseteq \ldots \subseteq R_{n-1}$ such that*

$$A = \bigcup_{i=0}^{\left[\frac{n-1}{2}\right]} \{(R_{2i+1} - R_{2i}) \cup (R_{2i} - R_{2i+1})\}.$$

(Here if n an odd number then $R_n = \emptyset$.)

Definition 2. *A set A belong to level Σ_n^{-1} of Ershov's hierarchy (A is Σ_n^{-1}-set), if it is n-c.e. set. A set A belong to level Π_n^{-1} of the hierarchy (A is Π_n^{-1}-set), if $\overline{A} \in \Sigma_n^{-1}$ and A is Δ_n^{-1}-set, if A and $\overline{A}$ both are Σ_n^{-1}-sets, i.e. $\Delta_n^{-1} = \Sigma_n^{-1} \cap \Pi_n^{-1}$.*

* The author is supported by RFBR Grant 05-01-00830

Theorem 2. *(Ershov; Epstein, Haas, Kramer) a) A set A is n-c.e. set for some $n \geq 0$ iff there is a computable function g of two variables s and x such that for all x $A(x) = lim_s g(s,x)$, $g(0,x) = 0$, and $|\{s|g(s+1,x) \neq g(s,x)\}| \leq n$.*

b) A set A is Δ^{-1}_{n+1}-set for some n, $1 \leq n < \omega$, iff there is a partial-computable function ψ such that for all x $A(x) = \psi(\mu t_{\leq n}(\psi(t,x)\downarrow), x)$.

Definition 3. *(Ershov; Epstein,Haas,Kramer) A set $A \subseteq \omega$ belong to level Σ^{-1}_{ω} of Ershov's hierarchy (A is Σ^{-1}_{ω}-set), if there exists a partial-computable function ψ such that for all x,*

$x \in A \to \exists s(\psi(s,x)\downarrow$ and $A(x) = \psi(\mu s(\psi(s,x)\downarrow), x)$;

$x \notin A \to$ either $\forall s(\psi(s,x))\uparrow$, or $\exists s(\psi(s,x)\downarrow)\&$
$A(x) = \psi(\mu s(\psi(s,x)\downarrow), x)$.

(In other words, $A \subseteq dom(\lambda x.\psi(\mu s(\psi(s,x)\downarrow), x)$, and for any $x \in dom(\lambda x.\psi(\mu s(\psi(s,x)\downarrow), x))$ we have $A(x) = \psi(\mu s(\psi(s,x)\downarrow), x))$.

A set A belong to level Π^{-1}_{ω} of the hierarchy (A is Π^{-1}_{ω}-set), if $\overline{A} \in \Sigma^{-1}_{\omega}$. At last, A is Δ^{-1}_{ω}-set, if A and $\overline{A}$ both are Σ^{-1}_{ω}-sets, i.e. $\Delta^{-1}_{\omega} = \Sigma^{-1}_{\omega} \cap \Pi^{-1}_{\omega}$.

Definition 4. *A set A is ω-c.e. set if and only if there are computable function g of two variables s and x and a computable function f such that for all x $A(x) = lim_s g(s,x)$, $g(0,x) = 0$, and*

$$|\{s|g(s+1,x) \neq g(s,x)\}| \leq f(x).$$

Theorem 3. *(Ershov;Epstein,Haas,Kramer) Let $A \subseteq \omega$. The following are equivalent:*

a) A is ω-c.e.;
b) A is a Δ^{-1}_{ω}-set;
c) there is a partial-computable function ψ such that for all x, $A(x) = \psi(\mu t(\psi(t,x)\downarrow), x)$;
d) there is a uniformly c.e. sequence of c.e. sets $\{R_x\}_{x\in\omega}$, such that $\bigcup_{x\in\omega} R_x = \omega$, $R_0 \subseteq R_1 \subseteq \ldots$, and $A = \bigcup_{n=0}^{\infty}(R_{2n+1} - R_{2n})$.

Theorem 4. *(Ershov) $A \in \Sigma^{-1}_{\omega}$ iff there is a uniformly computable sequence of c.e. sets $\{R_x\}_{x\in\omega}$ such that $R_0 \subseteq R_1 \subseteq ...$ (ω-sequence of c.e. sets), and $A = \bigcup_{x=0}^{\infty}(R_{2x+1} - R_{2x})$*

Theorem 5. *(Carstens) a) A set A is Δ^{-1}_{ω}-set if and only if it is tt-reducible to $\emptyset'$;*

b) For any $n \geq 1$ a set A is Δ^{-1}_{n+1}-set if and only if it is btt-reducible to $\emptyset'$ with norm n.

Let $P(x,y)$ be a computable predicate which on ω defines a partial ordering. (If $P(x,y)$ we write $x \leq_P y$.) A uniformly c.e. sequence $\{R_x\}$ of c.e. sets is $\leq_P$-sequence, if for all x, y, $x \leq_P y$ implies $R_x \subseteq R_y$.

Hereinafter we will use the Kleene system of notation $(\mathcal{O}, <_0)$. For $a \in \mathcal{O}$ we denote by $|a|_0$ ordinal α, which have $\mathcal{O}$-notation a. Therefore $|a|_0$ have the order

type $\langle\{x|x <_0 a\}, <_0\rangle$, and words "$a$-sequence of c.e. sets $\{R_x\}$" for $a \in \mathcal{O}$ have usual sense. If α a constructive ordinal and $a \in \mathcal{O}$ its notation, i.e. $|a|_0 = \alpha$, and $\lambda < \alpha$, then knowing a we can effectively find a notation b for λ, $|b|_0 = \lambda$. Sometimes in this situation we will also write $(\lambda)_0$, meaning under $(\lambda)_0$ this notation b for λ.

An ordinal is even, if it is either 0, or a limit ordinal, or a successor of an odd ordinal. Otherwise the ordinal is odd. Therefore, if α is even, then α' (successor of α) is odd and vise versa.

For system of notation $\mathcal{O}$ the parity function $e(x)$ is defined as follows: Let $n \in \mathcal{O}$. Then $e(n) = 1$, if ordinal $|n|_0$ is odd, and $e(n) = 0$, if $|n|_0$ is even.

The following definitions of infinite levels of the hierarchy and Theorems $7 - 11$ are from Ershov [1968,1970].

For any $a \in \mathcal{O}$ we first define operations S_a and P_a, which map a-sequences $\{R_x\}_{x<_0 a}$ to subsets of ω, as follows:

$S_a(R) = \{z|\exists x <_0 a(e(x) \neq e(a) \& z \in R_x \& \forall y <_0 x(z \notin R_y))\}$.

$P_a(R) = \{z|\exists x <_0 a(e(x) = e(a) \& z \in R_x \&$
$\& \forall y <_0 x(z \notin R_y))\} \cup \{\omega - \bigcup_{x<_0 a} R_x\}$.

It follows from these definitions that $P_a(R) = \overline{S_a(R)}$ for all $a \in \mathcal{O}$ and all a-sequences R.

Class Σ_a^{-1} (Π_a^{-1}) for $a \in \mathcal{O}$ is the class of all sets $S_a(R)$ (accordingly all sets $P_a(R)$), where $R = \{R_x\}_{x<_0 a}$ all a-sequences of c.e. sets, $a \in \mathcal{O}$. Define $\Delta_a^{-1} = \Sigma_a^{-1} \cap \Pi_a^{-1}$.

Theorem 6. *(Epstein, Haas, Kramer; Selivanov) Let $A \subseteq \omega$ and α be a limit ordinal which has a notation a in $\mathcal{O}$. Following three statements are equivalent:*

a) $A \in \Delta_a^{-1}$;

b) For some partial-computable function Ψ and any x,
$A(x) = \Psi((\mu\lambda < \alpha)_0(\Psi((\lambda)_0, x) \downarrow, x)$;

c) There is an a-sequence $\mathcal{R}$= $\{R_x\}_{x<_0 a}$ such that $A = S_a(\mathcal{R})$ and $\bigcup_{x<_0 a} R_x = \omega$.

Theorem 7. *Let $a, b \in \mathcal{O}$ and $a <_0 b$.*
Then $\Sigma_a^{-1} \cup \Pi_a^{-1} \subset \Sigma_b^{-1} \cap \Pi_b^{-1}$.

Corollary 1. *For any $a \in \mathcal{O}$, $\Sigma_a^{-1} \subset \Delta_2^0$.*

Theorem 8. $\bigcup_{a\in\mathcal{O}} \Sigma_a^{-1} = \bigcup_{a\in\mathcal{O}, |a|_{\mathcal{O}}=\omega^2} \Sigma_a^{-1} = \Delta_2^0$.

It follows from Theorem 9 that Theorem 8 cannot be strengthen:

Theorem 9. $\bigcup_{a\in\mathcal{O}, |a|_0<\omega^2} \Sigma_a^{-1} \neq \Delta_2^0$.

Theorem 10. *a) For any $a \in \mathcal{O}$ there is a path T_0 in $\mathcal{O}$ through a such that* $\bigcup_{b\in T_0} \Sigma_b^{-1} = \Delta_2^0$.

b) There is a path T in $\mathcal{O}$ with the length ω^3 and $\bigcup_{a\in T} \Sigma_a^{-1} = \Delta_2^0$.

c) If a path T in $\mathcal{O}$ have the length less than ω^3 then $\bigcup_{a\in T} \Sigma_a^{-1} \neq \Delta_2^0$.

2 Generalized Tabular Reducibilities

For convenience we will consider only constructive ordinals $\leq \omega^\omega$ but all our definitions by an obvious way can be generalized to all constructive ordinals so that Theorems 12 and 13 hold also in this general case.

It follows from the universality properties of $(\mathcal{O}, <_0)$, that for $\alpha < \omega^\omega$ for simplicity instead notations from $\mathcal{O}$ we may use ordinals meaning their representation in normal form

$$\alpha = \omega^m \cdot n_0 + ... + \omega \cdot n_{m-1} + n_m.$$

We first define the following classes of generalized truth-table conditions ($gtt(\alpha)$-conditions) $\mathcal{B}_\alpha$, $\alpha \leq \omega^\omega$.

$\alpha = n > 1$: $\mathcal{B}_\alpha$ consists from all tt -conditions with norm $< n$;
$\alpha = \omega$: $\mathcal{B}_\alpha$ consists from all tt-conditions;
$\alpha = \omega^m \cdot n + \beta$, $\beta < \omega^m$ ($n > 1$; if $n = 1$ then $\beta > 0$):
$\mathcal{B}_\alpha$ consists from all tt-conditions of the form

$$\sigma_1 \& \tau_1 \vee ... \vee \sigma_n \& \tau_n \vee \rho, \text{ or } \neg[\sigma_1 \& \tau_1 \vee ... \vee \sigma_n \& \tau_n \vee \rho],$$

where $\sigma_i \in \mathcal{B}_\omega$, $\tau_i \in B_{\omega^m}$, $\rho \in \mathcal{B}_\beta$;
$\alpha = \omega^{m+1}$: $\mathcal{B}_\alpha = \bigcup_n \mathcal{B}_{\omega^m \cdot n}$;
$\alpha = \omega^\omega$: $\mathcal{B}_\alpha = \bigcup_n \mathcal{B}_{\omega^n}$.

It follows from these definitions that for each $\alpha, \omega \leq \alpha \leq \omega^\omega$, gtt-conditions from $\mathcal{B}_\alpha$ are usual tt-conditions with a fixed inner structure of these conditions. Using this structure we now define by induction on α an enumeration $\{\sigma_n^\alpha\}_{n\in\omega}$ of *$gtt(\alpha)$-formulas* related to gtt-conditions from $\mathcal{B}_\alpha$.

We denote by σ_n^ω the n-th tt-condition (which is a formula of propositional logic constructed from atomic propositions $\langle k \in X\rangle$ for several $k \in \omega$, and the norm of the tt-condition is the number of its atomic propositions).

For $\alpha = \omega^m \cdot n + \beta$, $m \geq 1, n \geq 1, \beta < \omega^m$ ($n > 1$; if $n = 1$ then $\beta > 0$) the $gtt(\alpha)$-formula $\sigma^\alpha_{\langle l,p,q,r\rangle}$ with index $\langle l, p, q, r\rangle$ is the formula
$(\neg)[\sigma^\omega_{\Phi_p(0)} \& \sigma^\gamma_{\Phi_q(0)} \ \vee \ \ldots \ \vee \ \sigma^\omega_{\Phi_p(n-1)} \& \sigma^\gamma_{\Phi_q(n-1)} \vee \ \sigma^\beta_r]$,
where $l = 1$ ($l = 0$) means the presence (accordingly absence) of the negation in the beginning of the formula, $\gamma = \omega^m$, $\Phi_p(i)$ is the partial-computable function with index p, defined for all $i \leq n-1$, $\Phi_q(i)$ is the partial-computable function with index q.

Therefore, a $gtt(\alpha)$-formula σ_i^α with index $i = \langle l, p, q, r\rangle$ is a gtt-condition $\sigma \in \mathcal{B}_\alpha$, $\alpha = \omega^m \cdot n + \delta$, $m \geq 1, n \geq 1, \delta < \omega^m$, if and only if $l \leq 1$, $\Phi_p(x) \downarrow$ for all $x \leq n-1$ and r is an index for some gtt-condition from $\mathcal{B}_\delta$.

For $\alpha = \omega^{m+1}$ and $\alpha = \omega^\omega$ the enumeration of $gtt(\alpha)$-formulas $\{\sigma^\alpha\}$ is defined using a fixed effective enumeration of all gtt-formulas from $\bigcup_{n,i} \sigma_i^{\omega^m \cdot n}$ (accordingly from $\bigcup_{n,i} \sigma_i^{\omega^n}$).

For the convenience we add to integers two additional objects $true$ and $false$, for which σ^α_{true} is a tt-condition which is identically truth and σ^α_{false} is an inconsistent tt-condition.

From now on we identify the class $\mathcal{B}_\alpha$ with the class of all $gtt(\alpha)$-formulas.

Definition 5. *We say that a gtt-formula σ from $\mathcal{B}_\alpha$ converges on a set $A \subseteq \omega$, if*

1. *$\alpha \leq \omega$, i.e. any tt-condition from $\mathcal{B}_\alpha$, $\alpha \leq \omega$, converges on any set $A \subset \omega$, or*
2. *σ is equal to $(\neg)[(\bigvee\limits_{i \leqslant m} \sigma^\omega_{\Phi_p(i)} \& \sigma^\gamma_{\Phi_q(i)}) \vee \sigma^\beta_j]$, and for any $i \leq m$ if A satisfies $\sigma^\omega_{\Phi_p(i)}$ (see the definition below), then $\Phi_q(i)\downarrow$ and $\sigma^\gamma_{\Phi_q(i)}$ converges on A.*

Definition 6. *A gtt-formula σ from $\mathcal{B}_\alpha$ is satisfied by a set $A \subseteq \omega$ (written as $A \models \sigma$), if σ converges on A and*

- *If $\sigma \in \mathcal{B}_\omega$, then A satisfies to the tt-condition σ,*
- *If σ is equal to $(\bigvee\limits_{i \leqslant m} \sigma_i \& \tau_i) \vee \rho$, then $A \models \rho$ or there is an $i \leq m$ such that $A \models \sigma_i$ and $A \models \tau_i$,*
- *If σ is equal to $\neg[(\bigvee\limits_{i \leqslant m} \sigma_i \& \tau_i) \vee \rho]$, then $A \not\models \rho$ and for all $i \leq m$, if $A \models \sigma_i$ then $A \not\models \tau_i$.*

$A \not\models \sigma$ means $A \models \neg\sigma$.

Definition 7. *A set A is $gtt(\alpha)$-reducible to a set B (written as $A \leq_{gtt(\alpha)} B$), if there is a computable function f such that for any x*

(i) gtt-formula $\sigma^\alpha_{f(x)}$ converges on B, and
(ii) $x \in A \leftrightarrow B \models \sigma^\alpha_{f(x)}$.

Now it follows immediately from Theorem 5 the following

Corollary 2. *(i) For $\alpha < \omega$ $A \in \Delta^{-1}_{\alpha+1} \leftrightarrow A \leq_{gtt(\alpha)} K$.*
(ii) $A \in \Delta^{-1}_\omega \leftrightarrow A \leq_{gtt(\omega)} K$.

If the ordinal α is a successor then the reducibility $\leq_{gtt(\alpha)}$ is not transitive. By this reason the following theorem we formulate for limit ordinals only.

Theorem 11. *For $\alpha = \omega, \omega^2, ..., \omega^\omega$ reducibilities $\leq_{gtt(\alpha)}$ are reducibilities which are intermediate between tt- and T-reducibilities, and for different α all $\leq_{gtt(\alpha)}$ are different.*

Proof. By indexes i and j of σ_i^α, σ_j^α we can effectively compute an index k of the gtt-formula σ_k^α, which is obtained by substitution of σ_i^α into σ_j^α, which means that for $\alpha = \omega, \omega^2, ..., \omega^\omega$ the set $\mathcal{B}_\alpha$ effectively closed on substitutions, and the relation $\leqslant_{gtt(\alpha)}$ in this case transitive. It is easy to see that the relation $\leqslant_{gtt(\alpha)}$ is reflexive. To prove that $A \leqslant_{gtt(\alpha)} B \to A \leq_T B$ for all A, B, suppose that $x \in A$ if and only if $B \models \sigma^\alpha_{f(x)} = (\neg)[(\bigvee\limits_{i \leqslant m} \sigma^\omega_{\Phi_{g(x)}(i)} \& \sigma^\beta_{\Phi_{q(x)}(i)}) \vee \sigma^\gamma_{r(x)}]$ for some computable functions f, g, q and r. For each x using oracle of B we can list all $i \leq m$ such that $B \models \sigma^\omega_{\Phi_{g(x)}(i)}$. For each such i we have $\Phi_{q(x)}(i) \downarrow$, and we also check whether $B \models \sigma^\beta_{\Phi_{q(x)}(i)}$. Similarly for the $gtt(\gamma)$-formula $\sigma^\gamma_{r(x)}$. Now $x \in A$ if and only if for some $i \leq m$ we have $\models \sigma^\omega_{\Phi_{g(x)}(i)}$ and $B \models \sigma^\beta_{\Phi_{q(x)}(i)}$, or there is such i in $gtt(\gamma)$-formula $\sigma^\gamma_{r(x)}$.

Therefore, $gtt(\alpha)$-reducibilities are intermediate between Turing and truth-table reducibilities.

It is known that for all $a <_0 b$ the set of T-degrees of Δ_a^{-1}-sets is a proper subset of the set Δ_b^{-1}-sets and, therefore, it follows from Theorem 12 below that at least degrees of creative sets for these reducibilities are different.

Theorem 12. *Let* $a \in \mathcal{O},\ \omega \leq |a|_0 \leq \omega^\omega$. *Then*

$$A \in \Delta_a^{-1} \Leftrightarrow A \leq_{gtt(\alpha)} K.$$

Proof. We will consider the case $|a|_0 = \omega \cdot 2$. After that it will be clear how to prove the theorem for arbitrary $a \in \mathcal{O},\ \omega \leq |a|_0 \leq \omega^\omega$.

Suppose that $A \leq_{gtt(\omega \cdot 2)} K$, i.e. there is a computable function f such that $x \in A$ iff $K \models \sigma^{\omega \cdot 2}_{f(x)}$. It is easy to see that there are computable functions p and q such that for all x, $\sigma^{\omega \cdot 2}_{f(x)} = (\neg)[\sigma^\omega_{\Phi_{p(x)}(0)} \& \sigma^\omega_{\Phi_{q(x)}(0)} \vee \sigma^\omega_{\Phi_{p(x)}(1)} \& \sigma^\omega_{\Phi_{q(x)}(1)}]$, and for $i \leq 1$, if $K \models \sigma^\omega_{\Phi_{p(x)}(i)}$ then $\Phi_{q(x)}(i) \downarrow$.

Since $\sigma^\omega_{\Phi_{p(x)}(0)}$ is a usual truth-table condition, there are a Boolean function α_x and a finite set $F = \{u_1, u_2, \ldots, u_{n(x)}\}$ such that $\sigma^\omega_{\Phi_{p(x)}(0)} = (F, \alpha_x)$ (here $n(x)$ and α_x are computable functions on x). Similarly, if $\Phi_{q(x)}(0) \downarrow$, then let $\sigma^\omega_{\Phi_{q(x)}(0)} = (\{v_1, v_2, \ldots, v_{m(x)}\}, \beta_x)$ for some functions $m(x)$ and β_x. Let $g(s, x)$ be the following computable function:

$$g(s,x) = \begin{cases} 1, & \text{if } \alpha_x(K_s(u_1), K_s(u_2), \ldots, K_s(u_{n(x)})) = 1, \text{ and} \\ & \Phi_{q(x)}(0) \downarrow \& \beta_x(K_s(v_1), K_s(v_2), \ldots, K_s(v_{m(x)})) = 1, \\ 0, & \text{otherwise.} \end{cases}$$

Now define a partial-computable function ψ as follows: for all $x, \psi(n(x), x) = g(0, x)$. Now suppose that $\psi(n(x) - i, x)$ is defined as $g(s_i + 1, x)$ for some i, and there is a (least) $s_{i+1} > s_i$ such that $g(s_{i+1} + 1, x) \neq g(s_{i+1}, x)$. Define $\psi(n(x) - (i+1), x) = g(s_{i+1}, x)$.

And now we define a uniformly c.e. sequence of c.e. sets $R_i, i \geq 0$:

$R_0 = \{x|\psi(0,x)\downarrow = 0\}$,
$R_1 = R_0 \cup \{x|\psi(0,x)\downarrow = 1\}$,
. . . . . .
$R_{2m} = R_{2m-1} \cup \{x|\psi(m,x)\downarrow = 0\}$,
$R_{2m+1} = R_{2m} \cup \{x|\psi(m,x)\downarrow = 1\}$.

Exactly by the same way (using $\sigma^{\omega}_{\Phi_{p(x)}(1)}$ and $\sigma^{\omega}_{\Phi_{q(x)}(1)}$ instead $\sigma^{\omega}_{\Phi_{p(x)}(0)}$ and $\sigma^{\omega}_{\Phi_{q(x)}(0)}$) we define a uniformly c.e. sequence of c.e. sets $R_{\omega+i}, i \geq 0$, but now each $R_{\omega+i}$ contains also $\bigcup\{R_i : i < \omega\}$. We have an $\omega \cdot 2$-sequence $R_0 \subseteq R_1 \subseteq \ldots \subseteq R_\omega \subseteq R_{\omega+1} \subseteq \ldots$. It is easy to see that $\bigcup\{R_i : i < \omega \cdot 2\} = \omega$, and that $A = \{\bigcup_{i=0}^{\infty}(R_{2i+1} - R_{2i})\} \cup \{\bigcup_{i=0}^{\infty}(R_{\omega+2i+1} - R_{\omega+2i})\}$, which means (see Theorem 6) that $A \in \Delta^{-1}_{\omega\cdot 2}$.

Now suppose that $R_0 \subseteq R_1 \subseteq \ldots \subseteq R_\omega \subseteq R_{\omega+1} \subseteq \ldots$ be a $\omega \cdot 2$-sequence such that $\bigcup_{i<\omega\cdot 2} R_i = \omega$ and $A = \{\bigcup_{i=0}^{\infty}(R_{2i+1} - R_{2i})\} \cup \{\bigcup_{i=0}^{\infty}(R_{\omega+2i+1} - R_{\omega+2i})\}$.

The proof of $A \leq_{gtt(\omega\cdot 2)} K$ is similar to the proof of Theorem 5 (that any ω-c.e. set is tt-reducible to K).

Given x and using the condition $\bigcup_{i<\omega\cdot 2} R_i = \omega$, first find an $i < \omega$ such that either $x \in R_i$ or $x \in R_{\omega+i}$. In first case find (as in the proof of Theorem 5) a tt-condition $\sigma^{\omega}_{x_0}$ such that $x \in A$ if and only if $K \models \sigma^{\omega}_{x_0}$, similarly in the second case find a tt-condition $\sigma^{\omega}_{x_1}$ such that $x \in A$ if and only if $K \models \sigma^{\omega}_{x_1}$. Now $\sigma^{\omega}_{\Phi_{p(x)}(0)} \& \sigma^{\omega}_{\Phi_{q(x)}(0)} \vee \sigma^{\omega}_{\Phi_{p(x)}(1)} \& \sigma^{\omega}_{\Phi_{q(x)}(1)}$ is the required $gtt(\omega \cdot 2)$-formula which $gtt(\omega\cdot 2)$-reduces A to K. Here $\sigma^{\omega}_{\Phi_{p(x)}(0)} = x_0$, $\sigma^{\omega}_{\Phi_{p(x)}(1)} = x_1$, and $\Phi_{q(x)}(0)\downarrow$=true, if $\exists i < \omega(x \in R_i)$, $\Phi_{q(x)}(1)\downarrow$=true, if $\exists i < \omega(x \in R_{\omega+i})$.

Note that if A is a Δ^{-1}_a-set for some $a \in \mathcal{O}$ such that $|a|_0$ is not a limit ordinal, then we don't have the property $\bigcup_{i<_0\alpha} R_i = \omega$. In this case we proceed as follows: let $A = S_a(R_0)$ and $\omega - A = P_a(R_1)$ for some a-sequences R_0 and R_1. Given $x \in \omega$ we first enumerate $\{R_{0,i}\}_{i<_0 a}$ and $\{R_{1,i}\}_{i<_0 a}$ until either $x \in R_{0,i}$ or $x \in R_{1,i}$ for some $i <_0 a$. After that we proceed as above with the following difference: in case $x \in R_{1,i}$ the required $gtt(\alpha)$-formula contains the negation in the beginning of the formula.

The following theorem shows that the Turing reducibility is not exhausted by any collection of $gtt(\alpha)$-reducibilities.

Theorem 13. *There is a set $A \leq_T \emptyset''$ such that for all α $A \not\leq_{gtt(\alpha)} \emptyset''$.*

The proof of the theorem is based on the following

Lemma 1. *If $B \leqslant_{gtt(\alpha)} C$ for some α, then there exists a $\emptyset'$-computable function $\Phi_e^{\emptyset'}$ such that*

$$(\forall x)[\, x \in B \;\leftrightarrow\; C \models \sigma^{\omega}_{\Phi_e^{\emptyset'}(x)}].$$

Proof (of lemma). Let $B \leqslant_{gtt(\alpha)} C$ by a computable function f, i.e. for any x,

$$x \in B \leftrightarrow \ C \models \sigma^{\alpha}_{f(x)}.$$

By α and an index of f we can effectively find the following presentation

$$\sigma^{\alpha}_{f(x)} = (\neg)[(\bigvee_{i \leqslant m} \sigma^{\omega}_{g(i,x)} \& \sigma^{\beta}_{\psi(i,x)}) \vee \sigma^{\gamma}_{k(x)}]$$

If $\beta > \omega$ then we find such a presentation also for $\sigma^{\beta}_{h(i,x)}$ via new β' and γ' and similarly for $\sigma^{\gamma}_{k(x)}$ etc. Finally, we obtain an extended presentation

$$\sigma^{\alpha}_{f(x)} = (\neg)[\bigvee_{i \leqslant m} ((\bigwedge_{j_i \leqslant n_i} \sigma^{\omega}_{p_{i,j_i}(x)}) \& \sigma^{\omega}_{q_i(x)})],$$

where all $p_{i,j_i}(x), 0 \leq i \leq m, 0 \leq j_i \leq n_i$, are defined, but some $q_i(x), 0 \leq i \leq m$, for some i can be undefined. Now let for $0 \leq i \leq m$,

$$\tau_s(x) = \begin{cases} (\bigwedge_{j_i \leqslant n_i} \sigma^{\omega}_{p_{i,j_i}(x)}) \& \sigma^{\omega}_{q_i(x)}, & \text{if } q_i(x) \downarrow; \\ \sigma^{\omega}_{false}, & \text{if } q_i(x) \uparrow. \end{cases}$$

We have $\sigma^{\alpha}_{f(x)} = (\neg)(\tau_0(x) \vee \tau_1(x) \ldots \vee \tau_k(x))$. This is obviously a tt-condition which index in the enumeration of all tt-conditions can be computed using oracle of $\emptyset'$.

Proof (of theorem). Let $B = \{x | (\exists y)[\varphi^{\emptyset'}_x(x) = y \& \emptyset'' \models \sigma^{\omega}_y]\}$ and let $A = \omega - B$.

The reducibility $A \leq_T \emptyset''$ is obvious. If $A \leqslant_{gtt(\alpha)} \emptyset''$ for some α, then there exists $\Phi^{\emptyset'}_e(x)$ from the lemma. Then

$e \in A \ \leftrightarrow \ \emptyset'' \models \sigma^{\omega}_{\Phi^{\emptyset'}_e(x)} \ \leftrightarrow \ e \in \ B \leftrightarrow \ e \notin \ A$

From other side, the weak truth-table reducibility is a special case of the $gtt(\omega^2)$-reducibility.

Definition 8. *$A \leq_{wtt} B$, if $A = \Phi^B_e$ for some e and for all x $\varphi^B_e(x) \leq f(x)$ for some computable function f.*

Theorem 14. *If $A \leq_{wtt} B$, then $A \leq_{gtt(\omega^2)} B$.*

Proof. Let $A = \Phi^B$ and g a computable function such that $\varphi^B(x) < g(x)$ for all x. There are $2^{g(x)}$ subsets $X_i \subseteq \{0, 1, ..., g(x) - 1\}$. For each of them we compose a tt-formula $\sigma^{\omega}_{p(i)}$, $i \leq 2^{g(x)}$, as follows:

$X \models \sigma^{\omega}_{p(i)} \ \leftrightarrow \ X \lceil g(x) = \ X_i$.

Now consider the formula

$$\sigma^{\omega^2}_{h(x)} \leftrightharpoons \sigma^{\omega}_{p(1)} \& \sigma^{\omega}_{q(1)} \ \vee \ldots \ \vee \ \sigma^{\omega}_{p(2^{g(x)})} \& \sigma^{\omega}_{q(2^{g(x)})},$$

where $\sigma^{\omega}_{p(i)}$ from above and $q(x)$ is defined as follows:

$$q(x) = \begin{cases} true, & \text{if } \Phi^{X_i}(x) \downarrow = 1; \\ false, & \text{if } \Phi^{X_i}(x) \downarrow = 0; \\ \uparrow . & \text{if } \Phi^{X_i}(x) \uparrow \end{cases}$$

By a given x we can effectively compute an index $f(x)$ of the gtt-formula $\sigma^{\omega^2}_{h(x)}$. Now $\Phi^B(x) = 1 \leftrightarrow \ B \models \sigma^{\omega^2}_{f(x)}$, i.e. $A \leqslant_{gtt(\omega^2)} B$ by function $f(x)$.

Note that the converse of this theorem is not true [1]. Indeed, let $A \in \Delta^{-1}_{\omega^2} - \Delta^{-1}_{\omega}$. Then $A \leq_{gtt(\omega^2)} K$ but $A \not\leq_{wtt} K$, since $A \leq_{wtt} K$ if and only if $A \leq_{tt} K$ (see, for example, Rogers [1967, exercise 9.45, p.159]) if and only if $A \in \Delta^{-1}_{\omega}$.

References

1. Carstens, H.G.: Δ^0_2-mengen. Arch.Math. Log. Grundlag. **18** (1978) 55–65
2. Epstein, R. L., Haas, R., Kramer, R. L.: Hierarchies of sets and degrees below $\mathbf{0}'$. Lecture Notes in Math. **859** (1981) 32–48
3. Ershov, Y.L.: On a hierarchy of sets I. Algebra i Logika **7**, No.1 (1968) 47–73
4. Ershov, Y.L.: On a hierarchy of sets II. Algebra i Logika **7**, No.4 (1968) 15–47
5. Ershov, Y.L.: On a hierarchy of sets III. Algebra i Logika **9**, No.1 (1968) 34–51
6. Gold, E.M.: Limiting recursion. J. Symb. Logic **30** (1965) 28–48
7. Putnam, H.: Trial and error predicates and the solution to a problem of Mostowski. J. Symb. Logic **30** (1965) 49–57
8. Rogers H.,Jr.: Theory of Recursive Functions and Effective Computability, McGraw-Hill, New York, 1967
9. Selivanov, V.L.: On Ershov's hierarchy. Siberian Math. J. **26** (1985) 134-150
10. Soare, R. I.: Recursively Enumerable Sets and Degrees. Springer-Verlag, Berlin, 1987

[1] I thank an anonymous referee for simplifying this proof

Note on Reducibility Between Domain Representations

Jens Blanck

Swansea University, Singleton Park, Swansea, SA2 8PP, UK

Abstract. A notion of (continuous) reducibility of representations of topological spaces is introduced and basic properties of this notion are studied for domain representations.
A representation reduces to another if the representing map factors through it. Reductions form a pre-order on representations. A spectrum is a class of representations divided by the equivalence relation induced by reductions. The spectrum of dense domain representations has a top element and representations within this equivalence class are said to be admissible. The notion of admissibility generalises the notion of admissibility in Weihrauch's TTE, and is stronger than the notion of admissibility used by Hamrin. Admissible representations are of particular interest since any continuous operation on the represented space can be represented.
To illustrate the framework, some domain representations of real numbers are considered and it is shown that the usual interval domain representation, which is admissible, does not reduce to a binary expansion domain representation. However, a substructure of the interval domain more suitable for efficient computation of operations are on the other hand shown to be equivalent to the usual interval domain with respect to reducibility.

1 Introduction

A standard method of computing on a set X of data is to make a representation R of the data and to compute on R. Such methods have been called concrete computability theories [26, 27]. The question arises immediately; to what extent does computability on X depend on the *choice* of R? For any concrete computability there is the problem of clarifying the representations.

A general procedure to extend computability from the natural numbers $\mathbb{N}$ (or some other structure with a computability theory) to some countable structure X is to represent the objects of X via a map from $\mathbb{N}$ to X. This is known as a *numbering* of X.

The class of all possible numberings is huge and not all numberings of a particular space are useful for computations so some tools of classifying numberings is needed. Extensive studies of numberings and their properties have been carried out, see [18, 13–15, 23, 24].

Our aim is to study computability on uncountable structures (usually topological spaces). A simple numbering is not possible of an uncountable structure.

We therefor have to rely on computations on some numbered set of approximations. For example, real computations can be performed using the countable set of rational intervals as approximations. A general method of giving computability theory to a large class of topological spaces is to use *domain representations*.

Representations of topological spaces by domains or embeddings of topological spaces into domains have been studied by several people [31, 22, 23, 10, 9, 11, 8, 12, 20, 4, 2, 3, 19, 28]. Domain representations are also closely related to Type-2 Theory of Effectivity (TTE) introduced by Weihrauch [29, 25, 30].

Reductions between numberings (when a numbering factors through another) is one of the basic tools in studying numberings. We generalise reducibility to a very general class of representations of topological spaces and study basic properties of reducibility, in particular for domain representations.

Any T_0 space can be given domain representations. Some of these have nice properties such as density and an embedding property. These facilitate lifting of functions to the domain representations. Thereby opening up for a study of topological algebras.

The reducibility notion introduces a pre-order on domain representations and thereby an equivalence relation. A *spectrum* is a class of representations divided by the equivalence relation. We give examples showing that the structure of the spectrum of all representations is non-trivial.

The representations that have the embedding property is known as *retract* representations. Retract representations are invariant under our reducibility notion.

The importance of density has an information theoretic explanation in that non-dense representations contain non-consistent information or "garbage". When restricting our attention to dense representations, there is a top element in the spectrum, namely the equivalence class of dense retract representations. We call representations belonging to this equivalence class *admissible*. Our notion of admissibility is a stronger than the admissibility of domain representations considered by Hamrin [16]. Our notion also extends the notion of admissibility used in TTE.

Given an admissible representation of a set X, any continuous operation on X can be lifted to the representation.

To illustrate the framework, we conclude by studying some representations of real numbers. The usual interval domain representation of the reals is known to be admissible. We show that a particular substructure of the interval domain, where operations on exact reals can be more efficiently computed, also is admissible, although its domain properties are weaker (it is a bifinite domain, rather than a consistently complete domain). Finally, we show that a representation corresponding to binary expansions is not an admissible representation of the reals. This is highlighted by the example showing that addition is not a computable operation on the binary expansions of reals.

We thank John V. Tucker for many invaluable discussions on this paper.

2 Preliminaries

We give some background on representations of spaces. We give a more general setting than the domain representations considered in [6], but we still aim for representations using some type of domain. The terminology is adjusted to cope with a more general framework. For background on domains we refer to [1, 21].

Definition 1. *(i) A* weak representation *of a topological space X is a triple $(D, D^{\mathrm{R}}, \rho)$, where D is a topological space, $D^{\mathrm{R}} \subseteq D$ with the subspace topology, and $\rho : D^{\mathrm{R}} \to X$ is continuous and onto.*

(ii) A quotient representation *is a weak representation where ρ is a quotient map.*

The word representation will be used without qualification to mean a weak representation.

In [6] this notion of representation is studied where D is required to be a domain. We will always have domain representations in mind, but define the notion as general as possible.

When needed, we write *continuous cpo representation, domain representation*, etc., to specify the kind of space that D is. We will primarily focus on Scott–Eršov domains and algebraic cpos, since by Proposition 1 any continuous cpo representation can be used to construct an algebraic cpo representation without losing any property considered herein.

The introduced notion of representations above covers all *naming systems* used in TTE, i.e., both *notations* and *representations*. In fact, they are all domain representations since Cantor space together with finite sequences constitute an algebraic domain.

The set D^{R} above will be called the set of *representing elements*. For a domain-like structure D the set D^{R} is also known as a *totality* on D. If D is a domain then the ordering of the domain can be interpreted as an information ordering. With this interpretation the domain contains both proper approximations and total or complete representations of elements of X, the latter constituting the set D^{R}. Intuitively, D^{R} consists of those domain elements that contain sufficient information to completely determine an element in X via ρ.

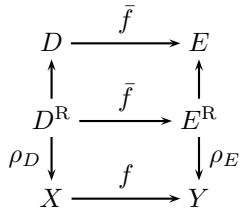
Beyond the type of domain D used in a representation, we make use of the following important characteristics of representations.

Definition 2. *(i) A representation $(D, D^{\mathrm{R}}, \rho)$ is* dense *if D^{R} is dense in D.*

(ii) A retract representation *of X is a quadruple $(D, D^{\mathrm{R}}, \rho, \eta)$ where $(D, D^{\mathrm{R}}, \rho)$ is a representation, and $\eta : X \to D^{\mathrm{R}}$ is a continuous function such that $\rho\eta = \mathrm{id}_X$.*

For a retract representation $(D, D^{\mathrm{R}}, \rho, \eta)$ we have that ρ is a quotient, and that $\eta\rho$ is a retraction on D^{R}. In fact, X will be homeomorphic to the retract of D^{R}. In a retract representation a canonical representative can be found continuously from any representation of an element of X.

Definition 3. *Let $(D, D^{\mathrm{R}}, \rho_D)$ and $(E, E^{\mathrm{R}}, \rho_E)$ be representations of X and Y respectively. A continuous function $f : X \to Y$ is* represented *by a continuous function $\bar{f} : D \to E$ if $\rho_E \bar{f}(x) = f\rho_D(x)$, for all $x \in D^{\mathrm{R}}$ (in particular, $\bar{f}[D^{\mathrm{R}}] \subseteq E^{\mathrm{R}}$).*



The functions between the subsets of representing elements are restrictions of functions. To avoid clumsy explicit restriction notation, as in $\bar{f}|_{D^{\mathrm{R}}} : D^{\mathrm{R}} \to E^{\mathrm{R}}$, we write $\bar{f} : D^{\mathrm{R}} \to E^{\mathrm{R}}$ and trust the reader to understand this as the restriction to the indicated domain of the function.

Let $(D, D^{\mathrm{R}}, \rho_D)$ and $(E, E^{\mathrm{R}}, \rho_E)$ be representations of X and Y respectively, and let $\bar{f} : D \to E$ be continuous such that $\bar{f}[D^{\mathrm{R}}] \subseteq E^{\mathrm{R}}$. If $\bar{f}$ respects the equivalence relations induced by ρ_D and ρ_E, then $\bar{f}$ represents a well-defined function $f : X \to Y$. Furthermore, if ρ_D is a quotient map, then f is continuous, since then the topology of X is fine enough.

We repeat some of the results from [6]. The following proposition sums up the results in Section 4 of [6] and shows why we may restrict our attention to representations from algebraic cpos.

Proposition 1. *Let $(D, D^{\mathrm{R}}, \rho_E)$ be a continuous cpo representation of X. Then there is a canonical algebraic cpo representation $(E, E^{\mathrm{R}}, \rho_E)$ retaining the properties of quotient, retract, and density if present in the original representation.*

For proofs of the following theorems, see Theorems 5.4, 5.6, and 9.3 of [6] respectively.

Theorem 1. *Any T_0 space has a dense retract domain representation.*

Theorem 2. *A space with a retract cpo representation is a T_0 space.*

Theorem 3. *Let $(D, D^{\mathrm{R}}, \rho_D)$ be a dense algebraic cpo representation of X, and let $(E, E^{\mathrm{R}}, \rho_E, \eta_E)$ be a retract domain representation of Y. Then every continuous function $f : X \to Y$ is represented by some continuous function $\bar{f} : D \to E$.*

3 Reducibility

In order to study representations and their applicability to various tasks we give here a notion of reduction between representations of a fixed space. We start with the topological version and postpone the effective version.

Definition 4. *Let $(D, D^{\mathrm{R}}, \rho_D)$ and $(E, E^{\mathrm{R}}, \rho_E)$ be representations of a topological space X.*

(i) Then D (continuously) reduces *to* E*, written* $D \leq E$*, if there exists a continuous function* $\phi : D \to E$ *such that* $\phi[D^{\mathrm{R}}] \subseteq E^{\mathrm{R}}$*, and* $\rho_D(d) = \rho_E\phi(d)$ *for any* $d \in D^{\mathrm{R}}$*.*

(ii) The representations are (continuously) equivalent*,* $D \equiv E$*, if* $D \leq E$ *and* $E \leq D$*.*

Rephrasing the definition we have that D reduces to E if ρ_D factors through ρ_E.

Lemma 1. *The relation* $\leq$ *is a pre-order, and* $\equiv$ *is an equivalence relation.*

Proof. The relation is reflexive since the identity is a continuous function reducing a representation to itself. Transitivity follows by composition.

That $\equiv$ is an equivalence relation follows from the definition by $\equiv$ in terms of the pre-order $\leq$. □

Lemma 2. *For representations* D *and* E *of* X *we have that* D *reduces to* E *if, and only if, the identity function on* X *is represented by some function from* D *to* E*.*

Proof. Any reduction function represents the identity function on X, and the identity is continuous on any topological space X. Any function representing the identity on X is a reduction function. □

Theorem 4. *Let* D *be a dense algebraic cpo representation of* X*, and* E *be a retract domain representation of* X*. Then* D *reduces to* E*.*

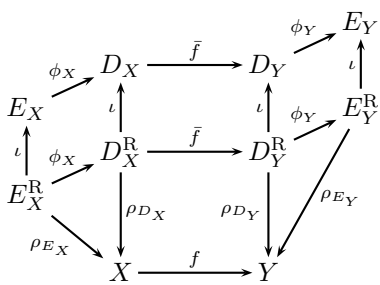
Proof. By Theorem 3 the identity function can be lifted to a continuous domain function. By Lemma 2 D reduces to E. □

Corollary 1. *Dense retract domain representations are unique up to* $\equiv$*.*

Proof. Immediate. □

Lemma 3. *Let* D_X*,* D_Y*,* E_X *and* E_Y *be representations of* X *and* Y *respectively. Assume that* E_X *reduces to* D_X *and that* D_Y *reduces to* E_Y*. If* $\bar{f} : D_X \to D_Y$ *represents* $f : X \to Y$ *then there exists* $\hat{f} : E_X \to E_Y$ *also representing* f*.*

Proof. Assume that ϕ_X reduces E_X to D_X and that ϕ_Y reduces D_Y to E_Y.



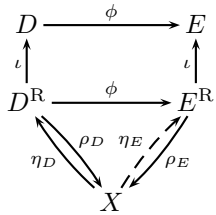
Let $\hat{f} = \phi_Y \bar{f} \phi_X$. □

The non-symmetric property of the above lemma makes it less valuable. As an example, consider a space X with representations D and E, where D reduces to E, and let f be an operation on X. Given that f is representable by an operation on D, is it also representable by some operation on E? Or vice versa? Unfortunately, the above lemma implies neither direction.

The following lemma shows that the property of retract is invariant under reductions.

Lemma 4. *Let $D \leq E$ be representations of X. If D is a retract representation, then so is E.*

Proof. Let $(D, D^{\mathrm{R}}, \rho_D, \eta_D)$, and $(E, E^{\mathrm{R}}, \rho_E)$ be the representations. By reducibility there exists a continuous $\phi : D \to E$ such that $\rho_D = \rho_E \, \phi$.



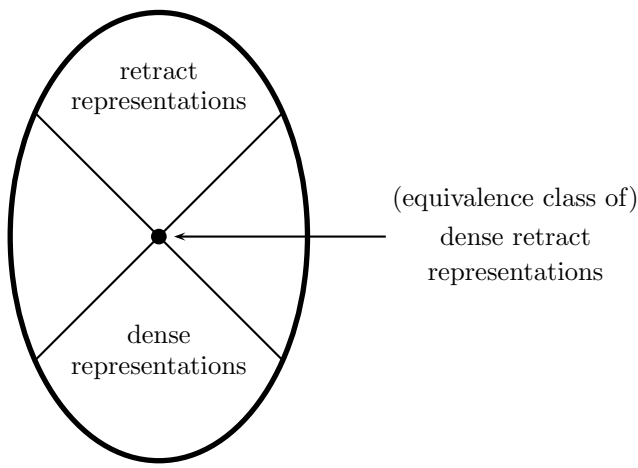
Let $\eta_E = \phi \, \eta_D$, then

$$\rho_E \, \eta_E = \rho_E \, \phi \, \eta_D = \rho_D \, \eta_D = \mathrm{id}_X$$

showing that $(E, E^{\mathrm{R}}, \rho_E, \eta_E)$ is a retract representation. □

Definition 5. *Let* **Rep** *be a class of representations of a space X. The* spectrum *of* **Rep** *is the class of all representations in* **Rep** *quoted by $\equiv$.*

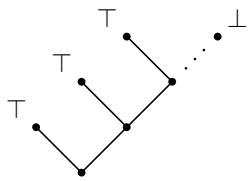
Let **Rep** be the class of all domain representations of a space X. Ordering the spectrum **Rep**/$\equiv$ with $\leq$ we have the following diagram.



We give examples showing that the diagram in general is non-trivial.

Example 1. Let Σ denote the Sierpinski space. That is, $\Sigma = \{\bot, \top\}$ and the topology is $\{\emptyset, \{\top\}, \Sigma\}$. Clearly, $(\Sigma, \Sigma, \mathrm{id})$ is a dense retract representation of Σ.

Create the domain D_Σ as:



where D_Σ^{R} consists of all maximal elements. The representation function $\rho : D_\Sigma \to \Sigma$ is defined as indicated in the figure. The representation $(D_\Sigma, D_\Sigma^{\mathrm{R}}, \rho)$ is dense.

By Theorem 4 D_Σ reduces to Σ. However, Σ does not reduce to D_Σ as any monotone map from $\Sigma = \Sigma^{\mathrm{R}}$ to D_Σ^{R} is constant.

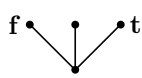
Example 2. Consider the following two retract representations of the booleans $\mathbb{B}$ where the representing elements are the ones indicated by **t** and **f**.

Note that $\mathbb{B}_\bot$ is a dense representation, but that $\mathbb{B}_{\bot,\top}$ is not. Clearly, $\mathbb{B}_\bot$ reduces to $\mathbb{B}_{\bot,\top}$ via the inclusion map.

However, $\mathbb{B}_{\bot,\top}$ does not reduce to $\mathbb{B}_\bot$ since there does not exist even a monotone function $\phi : \mathbb{B}_{\bot,\top} \to \mathbb{B}_\bot$ mapping the representing elements of $\mathbb{B}_{\bot,\top}$ to the corresponding representing elements of $\mathbb{B}_\bot$.

As a contrast to the previous example where a non-dense retract representation need not reduce to a retract representation we give the following example showing that the equivalence class of dense retract domain representations can contain non-dense representations.

Example 3. Let $\mathbf{3}_\bot$ be the domain



where $\mathbf{3}_\bot^{\mathrm{R}}$ are the labelled elements. This is a non-dense retract representation of $\mathbb{B}$. This representation reduces to $\mathbb{B}_\bot$ by mapping the compact witness of non-density to bottom in $\mathbb{B}_\bot$.

Recall that a common interpretation of the ordering relation of a domain representation is that it corresponds to information. High up in the domain means much information about an object of the space. So a non-dense representation can be viewed as a representation that contains non-consistent information or

"garbage". In practise it is sometimes desirable to cut away this non-consistent information (see, for example, Lemma 2.28 in [5]). This can be done in general for representations by taking the ideal completion over all approximations of representing elements, see [16]. Thus, we have a case to restrict ourselves to dense representations.

Theorem 5. *The spectrum of dense domain representations of a T_0 space has a top element, namely the equivalence class of dense retract domain representations.*

Proof. By Theorem 1 there exists a dense retract domain representation of the space and by Theorem 4 any dense representation can be reduced to it. □

4 Admissible representations

We will define a notion of admissible representations similar to the one used by Kreitz and Weihrauch [17].

Definition 6. *A representation is* admissible *if it is equivalent to a dense retract domain representation.*

By Example 3 an admissible representation need not be dense. However, any admissible representation is itself a retract representation.

Lemma 5. *Any admissible representation D is a retract representation.*

Proof. As the D is equivalent to a retract representation, say E, we have in particular that E reduces to D and by Lemma 4 D is itself a retract representation. □

The following shows that among dense representations admissibility is preserved by reductions.

Lemma 6. *If D is an admissible representation reducing to a dense representation E then E is admissible.*

Proof. Since E is a dense representation and since D is a retract representation the result follows by Theorem 4. □

The following lemma shows that admissible representations have a universality property.

Lemma 7. *Any dense representation of a space reduces to any admissible representation of the same space.*

Proof. By Lemma 5 and Theorem 4. □

The lemma also shows that our notion of admissibility implies the notion of admissibility introduced by Hamrin [16], where a representation is admissible exactly if the above universality property holds. Our notion is stronger in that representations such as $\mathbb{B}_{\perp,\top}$ in Example 2 are not admissible.

Theorem 6. *The class of spaces with admissible representations coincides with the class of T_0 spaces.*

Proof. By Theorem 1 there exist dense retract domain representations of all T_0 spaces, and these representations are by definition admissible. Since any admissible representation is a retract representation by Lemma 5 it follows by Theorem 2 that the represented space is T_0. □

The following immediate result shows why admissible representations are important.

Theorem 7. *Let $(D, D^{\mathrm{R}}, \rho_D, \eta_D)$ and $E, E^{\mathrm{R}}, \rho_E, \eta_E)$ be admissible representations of X and Y respectively. Then any continuous function $f : X \to Y$ has a representation $\bar{f} : D \to E$.*

Proof. By Theorem 3. □

5 Representations of the reals

Here we will look at three different representations of real numbers. The first is the customary interval domain, the second is a substructure of the interval domain that allow for more efficient computations, and the third corresponds to binary expansion of the reals.

Let $\mathcal{R}$ be the ideal completion of all closed rational intervals together with the real line ordered by reverse inclusion. The representing elements $\mathcal{R}^{\mathrm{R}}$ of this domain are all ideals that have singleton intersections; a representing ideal is mapped by $\rho_{\mathcal{R}}$ to the single element of its intersection. Define $\eta_{\mathcal{R}}$ by

$$\eta_{\mathcal{R}}(x) = \{[a, b] : a < x < b,\ a, b \in \mathbb{Q}\}\,.$$

Lemma 8. *$(\mathcal{R}, \mathcal{R}^{\mathrm{R}}, \rho_{\mathcal{R}}, \eta_{\mathcal{R}})$ is an admissible representation of the reals.*

Proof. A standard proof shows that the representation is a dense retract domain representation. □

In [7] centred dyadic approximations are considered for efficient implementations of exact real arithmetic. These form an interesting substructure of the interval domain.

Definition 7. *A* centred dyadic interval *is represented by a triple (m, e, s) of the form*

$$a = (m \pm e)2^{-s}\,,$$

where the mantissa *m and the* exponent *s are integers, and the* error term *e is a natural number. A real x is* approximated *by a if*

$$|x - m2^{-s}| \leq e2^{-s}\,,$$

or equivalently,

$$x \in [(m - e)2^{-s}, (m + e)2^{-s}]\,.$$

Fix $j > 0$. A centred dyadic j-approximation *is a centred dyadic interval where the error term is strictly bounded by 2^j.*

We will assume that j is fixed throughout and we will simply write *centred dyadic approximation*.

Let $\mathcal{R}_{\text{cda}}$ be the ideal completion of all centred dyadic approximations together with the real line ordered by reverse inclusion. Representing elements $\mathcal{R}^{\text{R}}_{\text{cda}}$ are again ideals with singleton intersection and the representing function $\rho_{\mathcal{R}_{\text{cda}}}$ is defined as before.

It is shown in [7, Lemma 3.7] that $\mathcal{R}_{\text{cda}}$ is not a domain, but that it is a bifinite domain (or SFP-domain). Nevertheless we will show that with respect to reducibility $\mathcal{R}_{\text{cda}}$ is equivalent to the interval domain. The following lemma shows that even though finite suprema does not exist in general in $\mathcal{R}_{\text{cda}}$ there is a sufficiently rich substructure of $\mathcal{R}_{\text{cda}}$ where finite suprema exist.

Lemma 9. *Within the substructure of all centred dyadic 1-approximations finite suprema exist.*

Proof. It is sufficient to show that the supremum of $a = (m \pm 1)2^{-s}$ and $b = (n \pm 1)2^{-t}$ is a centred dyadic 1-approximation. Assume without loss of generality that $t \geq s$.

If the distance between the centre points a and b, that is $m2^{-s}$ and $n2^{-t}$, is less than the radius of a, that is 2^{-s}, then the centre of b must be at least 2^{-t} away from boundary of a, meaning that b is contained in a, so b is the supremum.

The remaining case is that the centre of b is on the boundary of a. Assume that the centre of b is the upper end-point of a, i.e., $n2^{-t} = (m+1)2^{-s}$. The supremum of a and b is $((2n-1) \pm 1)2^{-t-1}$. □

Theorem 8. *The representations $\mathcal{R}$ and $\mathcal{R}_{\text{cda}}$ are equivalent.*

Proof. The inclusion map from $\mathcal{R}_{\text{cda}}$ to $\mathcal{R}$ represents the identity on the real line so $\mathcal{R}_{\text{cda}}$ reduces to $\mathcal{R}$.

For the other direction let ϕ be defined on compacts by

$$\phi([a,b]) = \bigsqcup_1 \{(m \pm 1)2^{-s} : [a,b] \subseteq (m \pm 1)2^{-s},\ 2^{-s} \leq |b-a|\}\,,$$

where $\bigsqcup_1$ gives finite suprema in the substructure of centred dyadic 1-approximations. Note that the supremum is taken over a finite set because there can only be finitely many 1-approximations containing the interval when the radius of the 1-approximations are bounded.

Extend ϕ to a continuous function. Then ϕ represents the identity on the real line. Thus, $\mathcal{R}$ reduces to $\mathcal{R}_{\text{cda}}$. □

Restricting the interval domain $\mathcal{R}$ to the unit interval gives an admissible representation of the unit interval which we denote by $\mathcal{R}_{[0,1]}$.

We will construct a representation corresponding to binary expansion. For simplicity we restrict ourselves to the unit interval. Let the compact elements be finite binary expansions, and let $D_{[0,1]}$ be the ideal completion over the prefix ordering of finite binary expansions. A maximal element of the domain corresponds to an infinite binary expansion. Let $\rho_D : D_{[0,1]} \to \mathbb{R}$ be the mapping of

an infinite binary expansion into the corresponding real number. Note that any interior dyadic point will have two representations in the domain, for example, $\frac{1}{2}$ is represented by both $.0111\ldots$ and $.1000\ldots$. The domain $D_{[0,1]}$ is a dense representation of the unit interval.

Construct a monotone map ϕ from compact elements of $D_{[0,1]}$ (that is, finite binary expansions) to $\mathcal{R}_{[0,1]}$ by mapping $.b_1 \ldots b_n$ to the interval

$$\left[\frac{a}{2^n}, \frac{a+1}{2^n}\right],$$

where a is the integer number $b_1 \ldots b_n$. Extend ϕ to a continuous function $\phi : D_{[0,1]} \to \mathcal{R}_{[0,1]}$. The function ϕ induces the identity on the unit interval so $D_{[0,1]}$ reduces to $\mathcal{R}_{[0,1]}$.

Lemma 10. *The representation $D_{[0,1]}$ is not admissible.*

Proof. Assume that $\mathcal{R}_{[0,1]}$ reduces to $D_{[0,1]}$. Then by Lemma 4 $D_{[0,1]}$ would be a retract representation. Consider where the embedding would send $\frac{1}{2}$. If the embedding of $\frac{1}{2}$ is to $.0111\ldots$ then the preimage of the basic open set $\uparrow.0$ is the non-open interval $[0, \frac{1}{2}]$. If $\frac{1}{2}$ instead is embedded into $.1000\ldots$ then the preimage of $\uparrow.1$ is the non-open interval $[\frac{1}{2}, 1]$. This contradicts the existence of a continuous embedding, and hence, that $D_{[0,1]}$ is a retract representation. Thus, $\mathcal{R}_{[0,1]}$ does not reduce to $D_{[0,1]}$. □

We know that representing the reals by binary expansions is not an appropriate choice when considering computability of operations on the domain. It is well-known that neither addition nor multiplication is computable on binary expansions of real numbers.

Example 4. Consider computing addition on infinite binary expansions. The sum of $\frac{1}{3} = 0.010101\ldots$ and $\frac{1}{6} = 0.0010101\ldots$ is $\frac{1}{2}$. However, for whatever finite amount of the inputs that is inspected even the first bit after the binary point of the output is not determined. Thus, we cannot effectively compute addition.

References

1. S. Abramsky and A. Jung. Domain theory. In S. Abramsky et al., editors, *Handbook of Logic in Computer Science*, volume III, pages 1–168. Oxford University Press, 1994.
2. U. Berger. Density theorems for the domains-with-totality semantics of dependent types. In J. Adamek, J. Koslowski, V. Pollara, and W. Struckmann, editors, *Proceedings of the Workshop Domains II*. Technische Universität Braunschweig, 1996.
3. U. Berger. Density theorems for the domains-with-totality semantics of dependent types. *Applied Categorical Structures*, 7:3–30, 1999.
4. L. Birkedal, A. Carboni, G. Rosolini, and D. S. Scott. Type theory via exact categories. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*, pages 188–198, Indianapolis, Indiana, 1998. IEEE Computer Society Press.

5. J. Blanck. Domain representability of metric spaces. *Annals of Pure and Applied Logic*, 83:225–247, 1997.
6. J. Blanck. Domain representations of topological spaces. *Theoretical Computer Science*, 247:229–255, 2000.
7. J. Blanck. Exact real arithmetic using centred intervals and bounded error terms. *Journal of Logic and Algebraic Programming*, 66:50–67, 2006.
8. P. di Gianantonio. Real number computability and domain theory. *Information and Computation*, 127:11–25, 1996.
9. A. Edalat. Domain theory and integration. *Theoretical Computer Science*, 151:163–193, 1995.
10. A. Edalat. Dynamical systems, measures, and fractals via domain theory. *Information and Computation*, 120:32–48, 1995.
11. A. Edalat and R. Heckmann. A computational model for metric spaces. *Theoretical Computer Science*, 193:53–73, 1998.
12. Y. L. Ershov. Model c of partial continuous functionals. In R. O. Gandy and J. M. E. Hyland, editors, *Logic Colloquium 76*, volume 87 of *Studies in Logic and Foundations in Mathematics*, pages 455–467. North-Holland, 1977.
13. Y. Eršov. Theorie der Numerierungen. *Zeitschrift für Math. Log.*, 19(4):289–388, 1973.
14. Y. Eršov. Theorie der Numerierungen II. *Zeitschrift für Math. Log.*, 21(6):473–584, 1975.
15. Y. Eršov. Theorie der Numerierungen III. *Zeitschrift für Math. Log.*, 23(4):289–371, 1977.
16. G. Hamrin. *Effective Domains and Admissible Domain Representations*. PhD thesis, Department of Mathematics, Uppsala University, 2005.
17. C. Kreitz and K. Weihrauch. Theory of representations. *Theoretical Computer Science*, 38:35–53, 1985.
18. A. I. Mal′cev. Constructive algebras. I. *Uspehi Mat. Nauk*, 16(3 (99)):3–60, 1961.
19. D. Normann. A hierarchy of domains with totality, but without density. In S. B. Cooper, T. A. Slaman, and S. S. Wainer, editors, *Computability, Enumerability, Unsolvability*, volume 224 of *London Mathematical Society Lecture Notes Series*, pages 233–257. Cambridge University Press, 1996.
20. D. S. Scott. A new category? Domains, spaces and equivalence relations. Manuscript, 1996.
21. V. Stoltenberg-Hansen, I. Lindström, and E. R. Griffor. *Mathematical Theory of Domains*. Cambridge University Press, 1994.
22. V. Stoltenberg-Hansen and J. V. Tucker. Complete local rings as domains. *Journal of Symbolic Logic*, 53:603–624, 1988.
23. V. Stoltenberg-Hansen and J. V. Tucker. Effective algebra. In S. Abramsky et al., editors, *Handbook of Logic in Computer Science*, volume IV, pages 357–526. Oxford University Press, 1995.
24. V. Stoltenberg-Hansen and J. V. Tucker. Computable rings and fields. In *Handbook of computability theory*, volume 140 of *Stud. Logic Found. Math.*, pages 363–447. North-Holland, Amsterdam, 1999.
25. V. Stoltenberg-Hansen and J. V. Tucker. Concrete models of computation for topological algebras. *Theoretical Computer Science*, 219:347–378, 1999.
26. J. V. Tucker and J. I. Zucker. Computable functions and semicomputable sets on many sorted algebras. In S. Abramsky et al., editors, *Handbook of Logic in Computer Science*, volume V, pages 317–523. Oxford University Press, 2000.
27. J. V. Tucker and J. I. Zucker. Abstract versus concrete computation on metric partial algebras. *ACM Transactions on Computational Logic*, 5(4):611–668, 2004.

28. G. A. Waagbø. *Domains-with-totality semantics for Intuitionistic Type Theory*. PhD thesis, University of Oslo, 1997.
29. K. Weihrauch. *Computability*. Number 9 in EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1987.
30. K. Weihrauch. *An Introduction to Computable Analysis*. Springer, 2000.
31. K. Weihrauch and U. Schreiber. Embedding metric spaces into cpo's. *Theoretical Computer Science*, 16:5–24, 1981.

Gödel, Turing, the Undecidability Results and the Nature of Human Mind

Riccardo Bruni

Università degli Studi di Firenze, Dipartimento di Filosofia
via Bolognese 52, Firenze 50139, Italy

Abstract. In this paper Turing and Gödel's standpoints toward the implications of the undecidability results are addressed. In the case of Turing, we show how his account on the issue was deeply connected to his project of actually building a computing machine showing an intelligent behaviour. Furthermore, we argue that his claim that the argument based on the halting problem offers no objection to that very project, is strenghtened by a general view on mathematical resoning and intelligence which has an anti-mechanistic flavour. As to Gödel's position, we reformulate, by enhancing its modal character, an argument that is contained in an unublished paper belonging to the early 1950's which ends in an open conclusion on minds and machines. We finally present Gödel's interpretation of the solution of the famous $P = NP$ problem contained in a letter to von Neumann as a further contribuition in this direction of working out the philosophical significance of certain mathematical achievements.

1 Introduction

Many years have passed since the first attempts to consider the very basic undecidability theorems in computability theory and metamathematics as directly estabilishing a bridge between mathematics and philosophy have appeared in literature. Roughly speaking, these results have been regarded as providing the means for a proof of the non-mechanical nature of human mind. The technical content of the argument and its relation to the mathematical theory are not very deep, and this makes even more striking that it may still be appealing despite all efforts that have been spent to show where the devised inferences break down.

The latest formulation of such an argument is due to Roger Penrose in his [12, 13]. As to this particular version, we have now come to what seems like a disproof of it, due to different although relatable analyses made by Pavel Pudlák [14], Stuart Shapiro [15] and Per Lindström [8, 9]. It turned out that Penrose's argument is either essentially relying on ambiguous notions (that is, the concept of 'unassailable mathematical truth' and that of 'a formal system which encapsulates all mathematical means of proofs accessible to human mind'), or it is wrong under a plausible definition of them. The failure of Penrose's gödelian argument seriously affects his project of a scientific investigation on

the true nature of human consciousness as based on mathematical, physical and philosophical considerations.

Since in our opinion these analyses have so far settled the matter, time seems right to us for a survey covering a more historical aspect of the issue which has somewhat been left unnoticed.

An inspection of the available sources reveals in fact that both Turing and Gödel had already considered similar implications of the undecidability results. The motivations that moved Turing and Gödel to direct their attention to them were different and in some sense opposed on to the other. Turing aimed at investigating intelligent behaviour as applied to the action of mechanical devices, while Gödel was in search for a rigorous argument that could confirm his beliefs concerning the status of mathematical concepts and the nature of mathematical reasoning. In both these cases, their analyses turned out to be surprisingly meaningful to Turing and Gödel's 'programs' in the fields of early Artificial Intelligence projects and in the philosophy of mathematics respectively.

Reconsidering the whole issue from their perspective may in turn help to do justice to it, and it may even give useful suggestions for some second thought concerning its implications.[1]

2 Turing and the "Mathematical Objection"

In a famous paper published on *Mind* in 1950, Turing discussed an argument based on the undecidability results as providing a "mathematical objection" to the idea of a machine that may successfully partecipate to what he called the Imitation Game, which is currently known as Turing's Test. It was not the first time that Turing was trying to tackle such an objection, but in this case he gave a particularly clear formulation of it.

The argument[2] applies to any (universal) machine $\mathsf{M}(x, y)$ which is supposed to answer questions concerning the behaviour of all (unary) machines (in a given enumeration) in such a way that $\mathsf{M}(m, n) = \texttt{YES}$ iff $\mathsf{M}_m(n) = \texttt{YES}$, and $\mathsf{M}(m, n) = \texttt{NO}$ otherwise. By bringing to contradiction the existence of such an M in the expected way,[3] one would be led to the conclusion that either such an M is unable to give always the correct answer to all questions, or that it may sometimes fail to give any answer. But this in turn would make it quite easy for an interrogator to distinguish between a machine and a human player, which would not be subject to the same constraints and could go freely in search for an appropriate method

[1] It seems unnecessary instead to spend some words about the up-to-date character of the problem in question. Despite all disproofs in fact, new attempts to restate the argument, even though with a critical attitude toward Penrose's, quickly arose (see [1]). Even in this case, the (brief) reconstruction of Gödel and Turing's views in particular seems questionable however, if not misleading.

[2] See [17, pp. 444–445].

[3] As usual, it would suffices to define a diagonal machine $\mathsf{D}_\mathsf{M}(x)$ such that $\mathsf{D}_\mathsf{M}(n) = \texttt{NO}$ in case $\mathsf{M}(n, n) = \texttt{YES}$ and $\mathsf{D}_\mathsf{M}(n) = \texttt{YES}$ otherwise, and then try to compute the values of $\mathsf{M}(d, d)$ for a description d of D_M.

and hopefully solve the problem in question. Thus, if Turing's Test is recognized as a suitable way to measure intelligence, machines cannot show an intelligent behaviour.

Turing's way out had been developed since 1947, when the british mathematician had directed his attention to this objection for the first time.[4] The first half of Turing's answer is not refined philosophically speaking, since it rests, as it may be expected, on the crucial requirement for the whole argument to apply, namely that the logical systems or the machine involved be consistent. The conclusion is then rejected by simply stressing that consistency is not an essential feature of intelligence.

As to its second half, Turing's analysis becomes more interesting. In fact, he points out that this kind of argument always applies to machines which have not been suitably trained, where, on the contrary, the abilities of a human mathematician can be viewed as the result of a continuous and stimulating interaction with other members of the community to which he belongs, including proper education.

So, while dropping the requirement of consistency may be sufficient by itself to escape the argument, Turing thought it necessary to do more in order to pursue the ambitious purpose of actually developing intelligent machineries, as he strenghtfully aimed at since the second half of the 1940's. The key idea was the possibility for a machine to act on its own istructions. This action had to be refined according to a specific training which needed to be directed from the outside on the basis of the application of suitably modified educational methods. As part of this project, Turing described, for example, an experiment in training an "unorganized" machine, that is a machinery built with no specific purpose, by means of external stimula of rewards and punishments.[5]

Since Turing never revealed a particular inclination to purely philosophical issues, it may come as a surprise that these pioneristic researches could be viewed as based on an analysis of mathematical reasoning and intelligence in general. It's even more surprising, since it contrasts with what is usually acknowledged to Turing,[6] that this analysis entails a non-mechanistic approach to human mind.

Turing's analysis of mathematical reasoning is contained in a section of his work on ordinal logics [16], and it is thought as providing a conceptual framework within which the results obtained by his logical investigation could be usefully discussed.

Mathematical reasoning is the result of two different faculties, namely *intuition* (which allows us to produce judgements on the truth/falsity of mathematical statements without any conscious train of thoughts), and *ingenuity* (which is responsible for the construction of proofs for intuitive judgements). With the

[4] See [18, pp. 87–88].

[5] See [19, pp. 122-125]. Even though he considered the result unsatisfactory since the whole process was not sufficiently close to the one which applies in the case of a child, Turing reported to have succeded in "organizing" such a device into a universal machine.

[6] See, e.g., [7, 1].

development of formal logic it became clear that no one of these two faculties could be dispensed with. In particular, it follows from Gödel's theorems that it is not possible to reduce intuition to ingenuity, that is to reduce mathematical activity to the choice within a given formal system of axioms of those steps which are the most efficient ones to build the proof of a given statement. On the contrary, it can be expected for a logical investigation only to give a more precise shape to the intuitive (that is, the non-mechanical) steps, without possibly eliminate them entirely.[7] It follows that mathematical knowledge is the result of a combination of mechanical and non-mechanical forms of reasoning.

In his later papers, where the case of intelligence in general is concerned, a similar scheme seems to apply.

In this case Turing speaks[8] of a need of both *discipline*, which is the ability of obeying orders and which is at best exemplified by the behaviour of an ordinary machine, and *initiative*. Even with respect to these two features, a purely mechanical approach would not allow an exaustive analysis. If the investigation is confined to the most general case, that is to those strategies which are applied to seek for a solution of existential problems,[9] it is true that it could be possible to account for a certain amount of cases by starting from a suitable logical system, and by relying our search on the results concerning the problem known as the extraction of programs from formal proofs.

In order to deal with *all* problems of this sort, however, it is necessary to think to more complicated processes, which are not trivially reducible to purely formal methods.[10] This makes it necessary a more radical turn: to change the kind of machines to be used for this task, and to build new ones on the basis of an appropriate analysis of those processes which are responsible for the turning of the child mind into the adult one.

3 Gödel's Modal Argument

While Turing addressed the "mathematical objection" as it actually has been used since then[11] (namely, as an argument showing that the human mind has

[7] In the case of Turing's investigation on ordinal logics for example, given a complete logic Λ (that is, in the terminology of [16], a function such that the collection $(\Lambda)o$, for any ordinal notation o, contains all true Π^0_2 statements), intuition is needed only to verify that a given expression is a notation for a constructive ordinal.

[8] See [19, pp. 125].

[9] Turing'opinion concerning statements of existential form as providing the most comprehensive collection of problems, was based (see [19, p. 126]) on a claim concerning the possibility of reducing all other forms of problem to this one via arithmetization.

[10] In particular, Turing indicated two additional searches for a solution, namely a *genetical search*, which consists in a process of genetical recombination so to obtain a new one which may result to be more suitable for the solution of the given problem, and an *intellectual search*, which is the one that results from the combined action of all the members of the community.

[11] In his retrospective article [11], the English philosopher J. R. Lucas, who is often wrongly credited as the first who provided the objection in question, stated that his

not a mechanical nature and that there can be no machine equivalent to it), Gödel arrived to it from a completely independent direction. In particular, he thought his version of the inference to help emphasizing the phenomenon of the "inexaustibility of mathematics", as he referred to it.

As he said in a lecture he delivered in 1951, the theorem on the undecidability of the sentence expressing consistency in formal mathematical systems "*makes it impossible that someone should set up a well-defined system of axioms and rules and consistently make the following assertion about it: All of these axioms and rules I perceive (with mathematical certitude) to be correct, and moreover I believe that they contain all of mathematics*".[12]

Recently, Bringsjord and Arkoudas in their [1] have presented a modal argument to show that computationalism (according to which, the human mind 'is' a Turing machine) is false. Further, even in the case of Penrose's latest formulation of the inference it is natural to give a modal reading of the informal notions used therein.[13] Then, in order to make a comparison more fruitful, it might be useful to present also Gödel's analysis of the basic inference by enhancing its modal flavour. Provided that, Gödel's argument goes as follows.

Assume $\mathsf{K}_M\alpha$ to mean that α is known with mathematical certitude. Then, it seems natural to assume that

$$\vdash \alpha \Rightarrow \mathsf{K}_M\alpha$$

holds (where the provability symbol should be understood in a broad and informal sense).

It follows that, for any given formal system $\mathcal{F}$, $\mathsf{K}_M(Corr_{\mathcal{F}} \rightarrow Con_{\mathcal{F}})$ where $Corr_{\mathcal{F}}$ and $Con_{\mathcal{F}}$ are the statements representing correctness and consistency of $\mathcal{F}$ respectively.[14] Assume K_M to be closed under *modus ponens*.[15] By the undecidability of $Con_{\mathcal{F}}$, we can conclude that if $\mathsf{K}_M Corr_{\mathcal{F}}$ then there exists a certain formula β such that $(\mathsf{K}_M\beta \wedge \neg\mathsf{Teor}_{\mathcal{F}}\beta)$ holds (where $\mathsf{Teor}_{\mathcal{F}}\alpha$ has an obvious meaning). This finally yields $\mathsf{K}_M\neg(\mathsf{K}_M\alpha \rightarrow \mathsf{Teor}_{\mathcal{F}}\alpha)$ for a generic α, from which it follows the desired conclusion $\neg\mathsf{K}_M(\mathsf{K}_M\alpha \rightarrow \mathsf{Teor}_{\mathcal{F}}\alpha)$ provided we have assumed $\neg(\mathsf{K}_M\alpha \wedge \mathsf{K}_M\neg\alpha)$.[16]

---

original writing [10] had been conceived as a reply to Turing's position on machine and intelligence as formulated in [17]. It seems thus natural to infer that Lucas didn't realize that the paper he argued against, already contained a formulation of the very same argument (even clearer than Lucas' own), and Turing's response to it.

[12] See [5, p. 309].

[13] See in particular Lindström [8, 9] and Shapiro [15] on this.

[14] As it is customary, $Corr_{\mathcal{F}}$ must be understood as some form or another of a reflection principle. Then, the implication of the system consistency would be provable (even in a formal framework) and then knowable according to the assumption above.

[15] In symbols, $\mathsf{K}_M(\alpha \rightarrow \beta) \rightarrow (\mathsf{K}_M\alpha \rightarrow \mathsf{K}_M\beta)$ holds.

[16] Our conclusion seems different than Gödel's since, under the assumption that we know with mathematical certainity that a system $\mathcal{F}$ is correct, it shows that we cannot know with the same certainity that this system contains all of mathematics. Gödel's conclusion instead dealt with two modalities, knowledge (with mathematical

Suppose now that for a certain formal system $\mathcal{F}^*$, we have

$$\mathsf{K}_M\alpha := \mathsf{Teor}_{\mathcal{F}^*}\alpha \quad (*)$$

which provides a definition for K_M as based on two assumptions, namely (i) that to know a mathematical proposition with certitude means to have a proof of it, and (ii) that there exists a certain formal system which comprises all the commonly acceptable means of proof that are accessible to human reasoning.

Then $\neg\mathsf{K}_M Corr_{\mathcal{F}^*}$ must hold. In Gödel's own words, it would follow that "the human mind (in the realm of pure mathematics) *is* equivalent to a finite machine that, however, is unable to understand completely its own functioning".[17] This conclusion has an immediate philosophical significance for the view entailed by (i) and (ii) above which would turn out to be, so to say, not self-contained since, under these assumptions, there would be no mathematically grounded justification for our belief in the validity of the commonly accepted inferences of deductive reasoning.

Furthermore, consider the collection $\mathbf{M}$ of all mathematical propositions which hold in an absolute sense. Let instead $\mathbf{K}$ be the class of all statements α such that $\mathsf{K}_M\alpha$ holds, and let us assume moreover $\mathbf{K} \subseteq \mathbf{M}$. Then, if $(*)$ holds for a given formal system, it follows that this inclusion is proper, namely there are mathematical truths which are 'inaccessible' to all the mathematical means of proof that the human mind can conceive. If on the contrary $(*)$ fails for all $\mathcal{F}$, it would follow for both $\mathbf{K}$ and $\mathbf{M}$ that these collections cannot be recursively enumerated where it remains possible for the above inclusion to be proper. This leads to Gödel's disjunctive conclusion[18] that either mathematics cannot be comprised in any finite rule, or there exist absolutely unsolvable mathematical problems, or both these alternatives are the case.[19]

Gödel's own solution of this disjunction is known. On the one hand, he thought that "Hilbert was right" in rejecting the existence of absolutely unsolvable mathematical problems among number-theoretic ones.[20] On the other,

---

certitude) and belief, the latter of which we've ommited for the sake of simplicity. We would have obtained a literal translation of Gödel's ending by introducing another modal operator B, and equivalently assuming that

$$\neg(\mathsf{K}_M\alpha \wedge \mathsf{B}\neg\alpha)$$

holds (which simply says that belief cannot contrast mathematical knowledge, since the latter has a stronger epistemic force).

[17] [5, pp. 309–310].

[18] See [5, p. 310].

[19] Due to an unpublished refinement of Gödel's theorem on arithmetical equivalents of the formally undecidable sentences (see [4]), the unsolvable problems would moreover have the form of diophantine statements of the type $\forall\vec{x}\exists\vec{y}P(\vec{x},\vec{y}) = 0$, where P is a polynomial with integer coefficients.

[20] See [20, p. 325]. Gödel's reference to Hilbert was motivated by the rationalistic attitude of the latter, as condensed in his famous slogan "In mathematics there's no *ignorabimus*".

Gödel was a supporter of an anti-mechanistic view of human reason. In a late and obscure remark,[21] he presented his position as based on the idea that (i) the human mind may be capable of thinking to infinitely many things and possibly even non-denumerably many, and that (ii) the way in which the abstract concepts enter human understanding seems of a *procedural* character (more and more abstract concept become accessible in the course of the mind development), but of a non-mechanical nature (a systematic method to actualize this development would result in a non-recursive arithmetic function).

Interestingly, in his Gibbs lecture Gödel argued that the conclusion of his argument could turn out to support some form or another of a Platonistic approach to mathematics: the existence of absolutely unsolvable problems to the human mind would go against the idea that mathematical concepts are our own construction since a creator knows all the properties of what he has created. Thus, it would be required to admit that these concepts, or at least some part of them, have an existence which is independent from our definitions and constructions.[22]

4 From Computability to Complexity

Obviously one may object that nothing like the disclosing of the true nature of human reasoning is *really* the purpose of the argument we've just surveyed. The crucial thesis one is dealing with is in fact comprised by statement $(*)$ above, which should be read as expressing the existence of a machine replicating the activity of human reason in the domain of mathematics when the latter is regarded extensionally. If compared to more recent attempts, the importance of Gödel's argument is primarily due to the refined analysis of the, so to say, linguistic side of the problem. Namely, it shows that there're reasonable interpretations for the concepts involved that lead to a meaningful and rigorously drawn conclusion which connects the mathematical investigation of the foundations to certain conceptual issues. But significantly, the conclusion provide no solution in the latter direction.

Indeed, the typical feature of this sort of arguments is their abstract character: they are concerned with an idealized human mathematician, whose actions are compared with a mathematical model of mechanical computation. This brings to an unsatisfactory ending which can be expressed philosophically by saying that the whole discussion is indifferent to how refined the assumed standpoint concerning the human mind might be. In this sense, the conclusion that can be drawn by Turing's analysis of the inference, which we would summarize by saying that the goals of a scientific research in the field of Artificial Intelligence turns out to be independent from the solution to the problem concerning the nature of human reasoning, is not surprising.

[21] See [3, p. 306] and [20, pp. 325–326].

[22] This argument is not that conclusive. One may argue against it, for example, by emphasizing that, due to Gödel's own theorems, the identification of mathematical concepts with their symbolic representation in a formal language does not lead to omniscience.

Then, one may try to refine the investingation by introducing some very basic restriction.

An attempt could be the one which starts from a view of human reasoning as the sum of actual processes which, like all kind of processes, are subject to certain practical constraints. One would consequently assume that only those processes which are feasible matter. Then, it is natural on the mathematical side to shift from computability to complexity issues.

Gödel is known to be among the firsts who adopted a related perspective. He did that in a letter to von Neumann dated 20 March 1956, which became known for containing one of the first formulation of the $P = NP$ problem.[23]

Indeed, Gödel questioned his correspondent about the following problem: to consider a Turing machine M that allows to decide, for every $n \in \mathbb{N}$ and every formula F of first order predicate logic, whether F has a proof of lenght n (n being the number of its symbols); then, if by $\psi(n, F)$ we indicate the number of steps required by the machine to reach a decision, Gödel's question was whether it is possible for

$$\varphi(n) := \max\{\psi(n, F) \mid F \text{ formula}\}$$

to grow linearly or quadraticly in the input n.

Today we know that it is possible to reduce the set of all satisfiable propositional formulas, which is NP-complete, to Gödel's problem.[24] This could make it surprising that Gödel was inclined to view such a solution as being "quite within the realm of possibilities" since (i) $\varphi(n) \geq k \cdot n$ is "the only estimate obtainable by generalizing the proof of the unsolvability of the Entscheidungsproblem", and (ii) $\varphi(n) \sim k \cdot n$ (or $\varphi(n) \sim k \cdot n^2$) "just means that the number of steps when compared to pure trial and error can be reduced from N to $\log N$ (or $\log N^2$)".

Furthermore, Gödel gave an interesting interpretation of the implications of this solution. If Gödel's question could be answered in the positive, then there would be a feasible algorithm yielding, among the formulas of first order logic, those which are provable in a reasonable sense of the word. Gödel thought this outcome to mean that the thinking of a human mathematician dealing with questions with a yes-or-no answer could be replaced, with just the exception of the formulation of the axioms for mathematics, by a machine. He thought this conclusion to refine what one can conclude under the unsolvability of the general decision problem, which would allow to consider any Turing machine as unfit to substitute the activity of a human mathematician. This conclusion would not in fact apply for a suitable choice of the input of the machine solving the problem above.

In fact, under the kind of pragmatic approach to human reasoning we've roughly described before, it is unreasonable to look for unbounded proofs. Then,

[23] See [6, pp. 373–376].

[24] Such a reduction, as perfomed by Stephen A. Cook (see [2, p. v]), goes as follows: given a propositional formula A with atoms $p_1, \ldots, p_n$, and given a new monadic predicate symbol Q and individual variables $x_1, \ldots, x_n$, let $A' \equiv A[p_i := Q(x_i)]$ $(1 \leq i \leq n)$ and $B \equiv \exists x \exists y(Q(x) \wedge \neg Q(y)) \rightarrow \exists x_1 \ldots \exists x_n A'$. Then $A \in SAT$ iff B is provable and it has a polinomially bounded proof.

if n is chosen sufficiently large, the activity of a human mathematician is in this sense reducible to that of a machine, provided Gödel's problem is solvable. In other words, although the human mind may be intensionally different from a Turing machine, it would be in fact equivalent to it for any practically feasible purpose.

5 Some General Comments

As a moral of our survey, one could naturally be brought to draw a balance of the interaction between a philosophically motivated problem and a mathematical treatement of it.

It could in fact be argued that the conclusion that follows from the complexity argument we've just accounted for, is basically the same of the one that was obtained by means of the modal argument. However, there is something in the approach to the problem of the previous section which is worth emphasizing as a distinguishing feature.

Gödel in fact thought the argument based on the unprovability of consistency and its disjunctive conclusion, to be the best possible result for a mathematical account of the situation concerning mind and machines. By that he plausibly meant that any further clarification, and even a solution of that disjunction should be sought by means of conceptual analysis. This would cause the whole problem to be subject to the unrigorous state of the philosophical investigation. The $P = NP$ argument may then represent, so to say, a mathematical way to such a solution which makes it unnecessary any further speculation of a philosophical nature.

This may have an unexpected consequence as to a comparison between Gödel and Turing's analysis. While in fact it seems natural to oppose the philosophical committment of the one to the neutrality of the other, Gödel's approach would end up with a conclusion which is even more radical than the one which can be attached to Turing's efforts. Indeed, one may find it justified to conclude that not only pure philosophy can be dispensed with in case of certain philosophically-committed scientific goals, but that it can be entirely substituted by making an appropriate use of certain results belonging to the most rigorous forms of a rational investigation. This claim would be further supported by the fact that the basic conceptual problem admits a straightforward and clear formulation (as attested by statement $(*)$ above), so that it can be subject to a mathematical treatement.[25] However, this general view is contrasted by the difficulty in reaching a definitive solution by treating the corresponding problem mathematically.

[25] Notice that this conclusion is only apparently opposed to the independent meaning of the philosophical investigation. Indeed, it could be suitably reformulated in such a way that it would recall some classical approaches to philosophy (the best and most obvious example of which would be Leibniz's view), which were based on the belief that its behaving according to the canons of deductive reasoning should be one of its required features.

In the case of Gödel's former argument, this difficulty is attested by the disjunctive conclusion that was obtained under a plausible definition of the concepts involved. As to Gödel's complexity argument instead, the weak aspect of it coincides with its depending on the solution of the $P = NP$ problem which, contrary to Gödel's expectations, would be nowadays recognized as unlikely.

References

1. Bringsjord, S., Arkoudas, K.: The modal argument for hypercomputing minds. Theoretical Computer Science **317** (2004) 167–190
2. Clote, P., Krajíček, J.: Arithmetic, Proof Theory, and Computational Complexity. Oxford University Press (1993)
3. Gödel, K.: Some Remarks on the Undecidability Results. *Collected Works vol. II: Publications 1938-1974.* Oxford University Press (1990) 305–306
4. Gödel, K.: Undecidable Diophantine Propositions. *Collected Works vol. III: Unpublished Essays and Lectures.* Oxford University Press (1995) 157–175
5. Gödel, K.: Some Basic Theorems on the Foundations of Mathematics and Their Implications. *Collected Works vol. III: Unpublished Essays and Lectures.* Oxford University Press (1995) 304–323
6. Gödel, K.: *Collected Works vol. V: Correspondence H-Z.* Oxford University Press (2003)
7. Hodges, A.: Alan Turing: The Enigma. Burnett Books, Simon and Schuster (1983)
8. Lindstöm, P.: Penrose's New Argument. Journal of Philosophical Logic **30** (2001) 241–250
9. Lindstöm, P.: Remarks on Penrose's "New Argument". To appear in the Journal of Philosophical Logic (online version already available at web page of this Journal)
10. Lucas, J. R.: Minds, Machines and Gödel. Philosophy **36** (1961) 112–127
11. Lucas, J. R.: Minds, Machines and Gödel: A Retrospect. P. Millican and A. Clark (eds) *Machines and Thought: The Legacy of Alan Turing.* Mind Association Occasional Series, Clarendon Press (1996) 103–124
12. Penrose, R.: Shadows of the Mind. Oxford University Press (1994)
13. Penrose, R.: Commentaries on 'Shadows of the Mind'. Psyche **2** (1995)
14. Pudlák, P.: A Note on the Applicability of the Incompleteness Theorems to Human Mind. Annals of Pure and Applied Logic **96** (1999) 335–342
15. Shapiro, S.: Mechanism, Truth, and Penrose's new argument. Journal of Philosophical Logic **32** (2003) 19–42
16. Turing, A. M.: Systems of Logic Based on Ordinals. Proc. of the London Math. Soc. **45** (1939) 161–228
17. Turing, A. M.: Computing Machinery and Intelligence. Mind **59** (1950) 433–460
18. Turing, A. M.: Lecture to the London Mathematical Society on 20 February 1947. D. C. Ince (ed.) *The Collected Works of A. M. Turing, vol. II.* North Holland (1992) 87–105
19. Turing, A. M.: Intelligent Machinery. D. C. Ince (ed.) *The Collected Works of A. M. Turing, vol. II.* North Holland (1992) 107–127
20. Wang, H.: *From Mathematics to Philosophy.* Routledge and Kegan Paul (1974)

The Conjecture $P \neq NP$ Presented by Means of Some Classes of Real Functions

José Félix Costa[1] and Jerzy Mycka[2⋆]

[1] Department of Mathematics, I.S.T.
Universidade Técnica de Lisboa
Lisboa, Portugal
fgc@math.ist.utl.pt
[2] Institute of Mathematics,
University of Maria Curie-Sklodowska
Lublin, Poland
Jerzy.Mycka@umcs.lublin.pl

Abstract. In this paper, we prove that there exists some condition, involving real functions, which implies $P \neq NP$.

Keywords: computation over the reals, computational complexity, open problems in computational complexity

1 Introduction and motivation

The theory of analog computation (see [1] for an introduction), where the internal states of a computer are continuous rather than discrete, has enjoyed a recent resurgence of interest. The first presentation of a theory, which transforms analog computation into the form similar to Kleene's classical theory of recursive functions over $\mathbb{N}$, was attempted by Cristopher Moore [5]. We present in this paper the concept of *(restricted) real recursive function* and the corresponding class $REC_R(\mathbb{R})$, which is based on the real recursive scalars $0, 1, -1$, and the real recursive projections with the following operators: composition and the solution of a Cauchy problem (or a initial value problem in mathematical analysis).

With these notions and the additional operator of bounded quantification (to define the concept of nondeterministic computation) we are able to introduce some important subclasses of real recursive functions. They are obtained by an imposition of restrictions on the growth of functions from $REC_R(\mathbb{R})$. Because the growth of values of functions is somehow restricted by time of computation such kind of restrictions is connected with complexity of real functions. But let us stress the fact that it is not our purpose to build such classes, which strictly inherit the properties of the classical complexity classes — rather, it is important for us to have classes with robust analytical definitions.

A real recursive function is said to be of exponential order if in any step of its construction, its components are bounded by the exponential function. By

⋆ Corresponding author

using of weak exponential restrictions ($e^{(logx)^k}$) we obtain the classes $DAnalog$ and $NAnalog$. We can also use a notion of admissible restrictions to transform real functions into natural ones not exceeding barriers of polynomial complexity.

Then we can prove that the condition $P = NP$ implies the identity of the above mentioned restrictions of $DAnalog$ and $NAnalog$. By contraposition we obtain an analytical test for the conjecture $P \neq NP$.

This paper is an attempt to use the above approach for the conjecture $P \neq NP$. With respect to [7], Section 2 introduces a more clear formulation of a (discrete) condition equivalent to $P = NP$, clarifying aspects that were obscure in [7], Section 3 is completely new and considers a generalized solution to the differential recursion scheme, Section 4 indeed intersects [7], but contains different definitions and modified concepts, capitalizing in some results from the precedent paper, and the new Section 5 motivates strongly our analog classes, relates them with linear systems, and with solutions of a feasible physics. In this Section, the conjecture $P \subset NP$ is lifted to the realm of Analysis.

With quantifier elimination (using techniques from the sixties, namely the concept of Richardson's map in [9]) in the non-deterministic class $NAnalog$, hopefully, we will succeed in the near future to present a quantifier free (full) analytic condition for the conjecture $P \subset NP$.

2 Polynomial time computable functions

The main model of computation, a Turing machine, has an obvious correspondence with the class of recursive functions.

A partial function f is said to be *computable* by a deterministic Turing machine M if (a) M accepts the domain of f and (b) if $\langle x_1, \ldots, x_n \rangle \in \mathrm{dom(f)}$, then the accepting computation writes in the output tape the value $f(x_1, \ldots, x_n)$. $P\mathcal{F}$ is the class of partial functions that can be computed in polynomial time by deterministic Turing machines, i.e., by deterministic Turing machines clocked with polynomials.

We adopted the definition of computable function given in [2]. Note that for functions in $P\mathcal{F}$ the halting problem is decidable. We have an inductive definition of the total functions in $P\mathcal{F}$, provided by Buss in 1986 (see, e.g., [8], Vol. II, p. 172):

Proposition 2.1. *The class of recursive functions computable in deterministic polynomial time is inductively defined from the basic functions* $Z = \lambda n.\ 0$ *and* $S = \lambda n.\ n + 1$, *the projections, basic functions* $\lambda n.\ 2n$, $\lambda n.\ 2n + 1$, $\lambda n.\ \lfloor \frac{n}{2} \rfloor$, *the characteristic function* δ *of "equality to* 0*", by the operations of composition, definition by cases, and polynomially bounded primitive recursion.* □

The intuition behind this characterization of the total functions in $P\mathcal{F}$ is the following: a Turing machine clocked in polynomial time p can write at most $p(|x|)$ bits in the output tape for the input x ($|x|$ is the length of x). This number is bounded by $2^{\lfloor \log(x+1) \rfloor^k}$, for some k.

We define now, for our purpose, a nondeterministic Turing machine in a way similar to which it is used in the probabilistic computational model (compare [2]). We impose the following conditions on the nondeterministic machines: (a) every step of a computation can be made in exactly two possible ways, which are considered different even if there is no difference in the corresponding actions (this distinction corresponds to two different bit guesses), (b) the machine is clocked by some time-constructible function and the number of steps in each computation is exactly the number of steps allowed by the clock; if a final state is reached before this number of steps, then the computation is continued, doing nothing up to this number of steps, (c) every computation ends in a final state, which can be either *accept* or *reject*, (d) if the machine computes a function, then all accepting computations write down to the output tape the value of the function (see [2], Chapter 2 for a comparison). It is irrelevant what the machine writes in the output tape if it reaches a rejecting state.

A partial function f is said to be *computable* by a nondeterministic Turing machine M if (a) M accepts the domain of f and (b) if $\langle x_1, \ldots, x_n\rangle \in \mathrm{dom(f)}$, then any accepting computation writes in the output tape the value $f(x_1, \ldots, x_n)$. $NP\mathcal{F}$ is the class of partial functions that can be computed in polynomial time by nondeterministic Turing machines, i.e., by nondeterministic Turing machines clocked with polynomials.

Now, we have the the following fact: $NP\mathcal{F}$ is the class of functions of the form $\lambda x_1 \ldots x_n.$ if $\langle x_1, \ldots, x_n\rangle \in A$ then $F(x_1 \ldots x_n)$, where $A \in NP$ and $F \in P\mathcal{F}$.

Let us use a different idea to think about nondeterminism.

Definition 2.1. *We define the class $\exists P\mathcal{F}$ as follows: a function $f : \mathbb{N}^n \to \mathbb{N}$ is in $\exists P\mathcal{F}$ if there exists a function $F : \mathbb{N}^{n+1} \to \mathbb{N}$ in $P\mathcal{F}$ and a polynomial $p : \mathbb{N}^n \to \mathbb{N}$ such that (a) $\langle x_1, \ldots, x_n\rangle \in \mathrm{dom(f)}$ if and only if there exists a number k such that $|k| \leq p(\sum_{i=1}^{n} |x_i|)$ and $\langle x_1, \ldots, x_n, k\rangle \in \mathrm{dom(F)}$ and (b) $f(x_1, \ldots, x_n)$ is defined and $f(x_1, \ldots, x_n) = y$ if and only if for the number k such that $|k| \leq p(\sum_{i=1}^{n} |x_i|)$ and $F(x_1, \ldots, x_n, k)$ is defined we have $F(x_1, \ldots, x_n, k) = y$, and, moreover for all such $|k| \leq p(\sum_{i=1}^{n} |x_i|)$ that $\langle x_1, \ldots, x_n, k\rangle \in \mathrm{dom(F)}$, we have $F(x_1, \ldots, x_n, k) = y$.*

With the last definition we can present the following two obvious statements: the class $NP\mathcal{F}$ coincides with $\exists P\mathcal{F}$; and $NP \subset P$ if and only if $\exists P\mathcal{F} \subset P\mathcal{F}$.

For the purpose of this paper we adopt the following convention: if f is undefined at x, then we take $f(x) = 0$ (adding 1 to a function allows coding for zeros without ambiguity). Then we can create the appropriate classes of total functions.

Definition 2.2. *Let $PF = \{\tilde{f} : f \in P\mathcal{F}\}$, where $\tilde{f}$, for a n-ary function f, is given by cases*

$$\tilde{f}(x_1, \ldots, x_n) = \begin{cases} f(x_1, \ldots, x_n) + 1 & \text{if } \langle x_1, \ldots, x_n\rangle \in dom(f) \\ 0 & \text{otherwise} \end{cases}$$

and let $NPF = \{\tilde{f} : f \in NP\mathcal{F}\}$*, where* $\tilde{f}$*, for a n-ary function* f*, is given by cases*

$$\tilde{f}(x_1, \ldots, x_n) = \begin{cases} f(x_1, \ldots, x_n) + 1 & \text{if } \langle x_1, \ldots, x_n \rangle \in dom(f) \\ 0 & \text{otherwise} \end{cases}$$

A function from NPF can be proved to be of the form: $\lambda\langle x_1, \ldots, x_n\rangle$. if $\langle x_1, \ldots, x_n\rangle \in A$ then $F(x_1, \ldots, x_n)$ else 0, for some set $A \in NP$ and some $F \in PF$. Then we have the desired result to work with functions rather than sets.

Proposition 2.2. *$NP \subset P$ if and only if $NPF \subset PF$.*

Proof. Assume that $NP \subset P$ and let $f \in NPF$. Then f is of the form $\lambda\langle x_1, \ldots, x_n\rangle$. if $\langle x_1, \ldots, x_n\rangle \in A$ then $F(x_1, \ldots, x_n)$ else 0, with $A \in NP$ and $F \in PF$. We conclude that $A \in P$ and, consequently, $f \in PF$. Conversely, assume that $NPF \subset PF$ and let $A \in NP$. Then the function f defined by $\lambda x.$ *if* $x \in A$ *then* 1 *else* 0 is in NPF. We conclude that $f \in PF$ and, consequently, $A \in P$. □

3 Recursive functions over $\mathbb{R}$

We will use the the concept of *a vector function* to denote a real function from $\mathbb{R}^k$ to $\mathbb{R}^n$, of *a scalar function* to denote a real function from $\mathbb{R}^k$ to $\mathbb{R}$.

We use in this paper the following idea of a solution for differential equation that differs from that one given, e.g., [4].[3] A solution to a system of equations (for $1 \leq i \leq n$) given by the following differential recursion

$$\partial_y h_i(x_1, \ldots, x_k, y) = g_i(x_1, \ldots, x_k, y, h_1(x_1, \ldots, x_k, y), \ldots, h_n(x_1, \ldots, x_k, y)),$$

satisfying the initial conditions $h_i(x_1, \ldots, x_k, 0) = f_i(x_1, \ldots, x_k)$ is a vector function $\hat{h}$ from $\mathbb{R}^{k+1}$ to $\mathbb{R}^n$ such that: a unique solution h to the system of differential equations exists in some open interval I containing 0; the vector function $\hat{h}$ satisfies the equations in a set $J \supseteq I$, such that J is an open interval up to a countable number of non-Zeno discontinuities,[4] in the sense that, for every $y \in J$, $\hat{h}_i(x_1, \ldots, x_k, y)$, $g_i(x_1, \ldots, x_k, y, \hat{h}_1(x_1, \ldots, x_k, y), \ldots, \hat{h}_n(x_1, \ldots, x_k, y))$, and $\partial_y \hat{h}_i(x_1, \ldots, x_k, y)$ are defined, for all $1 \leq i \leq n$, and it holds that

$$\partial_y \hat{h}_i(x_1, \ldots, x_k, y) = g_i(x_1, \ldots, x_k, y, \hat{h}_1(x_1, \ldots, x_k, y), \ldots, \hat{h}_n(x_1, \ldots, x_k, y));$$

the vector functions h and $\hat{h}$ coincide on I; the vector function $\hat{h}$ is the unique C^1 extension of h; and J is the largest such set.

[3] *A solution... is a function of the independent variable that, when substituted into the equation as the dependent variable, satisfies the equation for all values of the independent variable.* That is, a function $h(y)$ is a solution if it satisfies $\partial_y h(y) = g(y, h(y))$, for all y in $\mathbb{R}$.

[4] It means that for each finite open interval there exist only a finite number of discontinuities.

The vector function $\hat{h}$ is called *the generalized solution* of the system if it is the maximal solution according with the previous items. With this operator we can present the concept of *(restricted) real recursive function* and the corresponding class $REC_R(\mathbb{R})$ (based on the similar definition from [6]).

Definition 3.1. *The class $REC_R(\mathbb{R})$ of real recursive vector functions is generated from the real recursive scalars* 0, 1, -1, *the real recursive projections* $I^i_n(x_1, \ldots, x_n) = x_i$, $1 \leq i \leq n$, $n > 0$, *and the real recursive functions* $\theta_k(x) = x^k \Theta(x)$,[5] $k \geq 0$, *by the following operators:*

Composition: *if f is a real recursive vector function with n k-ary components and g is a real recursive vector function with k m-ary components, then the vector function with n m-ary components,* $1 \leq i \leq n$,

$$\lambda x_1 \ldots \lambda x_m.\ f_i(g_1(x_1, \ldots, x_m), \ldots, g_k(x_1, \ldots, x_m))$$

is real recursive.

Differential recursion: *if f is a real recursive vector function with n k-ary components and g is a real recursive vector function with n $(k + n + 1)$-ary components, then the vector function h of n $(k+1)$-ary components which is the solution of the Cauchy problem,* $1 \leq i \leq n$,

$$h_i(x_1, \ldots, x_k, 0) = f_i(x_1, \ldots, x_k),$$

$$\partial_y h_i(x_1, \ldots, x_k, y) = g_i(x_1, \ldots, x_k, y, h_1(x_1, \ldots, x_k, y), \ldots, h_n(x_1, \ldots, x_k, y))$$

is real recursive whenever h is a solution to the differential equation in the sense of the above explanation.

Assembling and designating components: *(a) arbitrary real recursive vector functions can be defined by assembling scalar real recursive function components into a vector function; (b) if f is a real recursive vector function, than each of its components is a real recursive scalar function.*

Let us give some examples of functions generated with the above definition. The scalar functions $+$, $\times$, $-$, exp, sin, cos, $\lambda x.\ \frac{1}{x}$, $/$, log, $\lambda xy.\ x^y$ are real recursive functions. See also [6], where many other examples and results can be found. To finish the current section let us observe that a simple modification of results from [7] gives us the inclusion of real extensions of classical computational classes like PF and NPF in the class of restricted recursive functions.

4 *DAnalog*, *NAnalog*, *fAP* and admissible restrictions

We know that, in computability theory, the growth of functions is an important factor of its complexity. We use this approach to define two subclasses of real recursive functions. A real recursive function is said to be of exponential order if in any step of its construction, its components are bounded by the exponential function.

[5] $\Theta(x) = 0$ for $x < 0$ and $\Theta(x) = 1$ for $x \geq 0$, it is the Heaviside function.

Concepts such as linear growth or sub-linear growth, exponential growth or subexponential growth can also be applied to some arbitrary function disregarding their component functions, i.e., their inductive construction. Thus, in what follows, we distinguish between *order* and *growth*.

The formulas

$$F(s) = \int_0^\infty f(\xi)\ e^{-s\xi}\ d\xi, \quad f(\xi) = \frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty} F(s)\ e^{s\xi}\ ds$$

present the Laplace transform $F = L[f]$ and the inverse Laplace transform $f = L^{-1}[F]$, respectively. The second integral is generally carried out by contour integration.

With the above notions, we have a precise boundary for the exponential order. *Exponentially bounded functions* are described by the following condition: for every total function f, $L[f](s)$ is defined for values of the complex variable s such that $\Re\, s > x_f$, where x_f depends on f. *Subexponentially bounded functions* can be introduced by the following condition: for every total function f, $L[f](s)$ is defined along the whole positive real axis $\Re\ s > 0$ We will also use *weak exponential bounds* on functions: in this case for given function f there exists some k such that $f(x) < e^{(log x)^k}$.

Consider a real recursive function f on the positive real axis such that: (i) f is continuous on $[0, \infty)$ except possibly for a finite number of jump discontinuities in every finite sub-interval; (ii) there is a positive number M such that $|f(\xi)| \leq Me^{k\xi}$, for all $\xi \geq 0$. Then we say that f belongs to the class L_k. Then, if $f \in L_k$ is a real recursive function, then the Laplace transform $L[f](x + iy)$ exists for $x > k$. Proof of the above statement can be found in [7]. If the Laplace transform of f exists, $L[f](s)$, then f is said to be of exponential order: it exists for $x = \Re\, s$ greater than some real number k.

We proposed in [7] the definition of the classes *DAnalog* and *NAnalog*, which can be interpreted as classes of real recursive functions computed with *weak exponential restrictions* on their values.

Definition 4.1. *The class DAnalog of real recursive vector functions is inductively defined as follows:*

Primitives: *Constants* 0, 1, *and* -1, *the projections* $I_n^i(x_1, \ldots, x_n) = x_i$, $1 \leq i \leq n$, *and the functions* $\theta_k(x) = x^k\Theta(x)$, $k \geq 0$, *are in DAnalog:*

Composition: *if* f *is a real recursive vector function with* n k*-ary components and* g *is a real recursive vector function with* k m*-ary components, all in DAnalog, then the vector function with* n m*-ary components,* $1 \leq i \leq n$,

$$\lambda x_1 \ldots \lambda x_m.\ f_i(g_1(x_1, \ldots, x_m), \ldots, g_k(x_1, \ldots, x_m))$$

is in DAnalog only if all its components grow less than a weak exponential.[6]

[6] This condition in this clause is not needed, since weak exponential bounded functions are closed for composition.

Differential recursion: *if f is a real recursive vector function with n k-ary components and g is a real recursive vector function with n $(k+n+1)$-ary components, both in DAnalog, then the vector function h of n $(k+1)$-ary components which is the solution of the Cauchy problem, $1 \leq i \leq n$,*

$$h_i(x_1, \ldots, x_k, 0) = f_i(x_1, \ldots, x_k),$$

$$\partial_y h_i(x_1, \ldots, x_k, y) = g_i(x_1, \ldots, x_k, y, h_1(x_1, \ldots, x_k, y), \ldots, h_n(x_1, \ldots, x_k, y))$$

is in DAnalog only if all its components grow less than a weak exponential.

Assembling and designating components: *(a) arbitrary real recursive vector functions in DAnalog can be defined by assembling scalar real recursive function components in DAnalog into a vector function; (b) if f is a real recursive vector function in DAnalog, than each of its components is a real recursive scalar function in DAnalog.*

Functions in DAnalog are said to be deterministic weak exponential.

Definition 4.2. *The class NAnalog of real recursive vector functions is obtained from real recursive vector functions in DAnalog:*

Admissible bounded quantification: *if $F : \mathbb{R}^{n+1} \to \mathbb{R}$ is a scalar function in DAnalog, $\phi : \mathbb{N}^n \to \mathbb{N}$ is a polynomial, and $f : \mathbb{R}^n \to \mathbb{R}$ is a function, such that (a) $f(x_1, \ldots, x_n) \neq 0$ if and only if there exists a positive integer k such that $|k| \leq \phi(||\lfloor x_1 \rfloor||, \ldots, ||\lfloor x_n \rfloor||)$ and $F(x_1, \ldots, x_n, k) \neq 0$ and (b) $f(x_1, \ldots, x_n) = y \neq 0$ if and only if there exists a positive integer k such that $|k| \leq \phi(||\lfloor x_1 \rfloor||, \ldots, ||\lfloor x_n \rfloor||)$, $F(x_1, \ldots, x_n, k) = y$, and, for all such positive integer $|k| \leq \phi(||\lfloor x_1 \rfloor||, \ldots, ||\lfloor x_1 \rfloor||)$ such that $F(x_1, \ldots, x_n, k) \neq 0$, we have $F(x_1, \ldots, x_n, k) = y$, then f is a scalar function in NAnalog.*

We write NAnalog = $\exists$DAnalog. Functions in NAnalog are said to be nondeterministic weak exponential.

We get the immediate result that $DAnalog \subset NAnalog$.

The functions $x + y$, xy, $x - y$, $\frac{1}{x+\epsilon}$, for all $\epsilon \in \mathbb{R}^+$, and $\frac{x}{y+\epsilon}$, for all $\epsilon \in \mathbb{R}$, belong to *DAnalog*. The reference functions $e^{\log(x+1)^k}$, for all $k \in \mathbb{N}$, is also in *DAnalog*. The function *exp* is not in the class *DAnalog*. Remember that functions in *DAnalog* are, strictly speaking, functions of the form $\Theta(x)f(\ldots, x, \ldots)$, according to Laplace transform conventions and notation, defined everywhere.

To consider some functions as subexponential, sometimes we have to make a small shift on the real variable to avoid a discontinuity at the origin. For example, the function $\lambda x.\ \frac{1}{x+\epsilon}$ is subexponential and its transform is $\lambda s.\ e^{\epsilon s} E_1(\epsilon s)$, where E_1 is the exponential integral of degree one, for positive ϵ as small as we want.

Notice that, if the function of expression $f(x, y)$ is obtained in *DAnalog*, then it can not grow faster than a weak exponential in one or in both variables. Now, if we take a univariable polynomial ψ and values of y such that $||\lfloor y \rfloor|| \leq \psi(||\lfloor x \rfloor||)$, then y grows less than a weak exponential of x and the combination of both growths, of f and y (now seen as function of x) can not grow faster than a weak exponential too. This simple explanation allows us to conclude that functions in *NAnalog* grow as fast as functions in *DAnalog*.

In [7] we proved that, for every function $f \in PF$, the Laplace transform $L[\hat{f}]$ of its canonical extension $\hat{f}$ is in *DAnalog*. Let us add that it can be proved that the classes *DAnalog* and *NAnalog* are closed for integration. Moreover, *DAnalog* is closed for differentiation (in the sense that if a function of one of these classes is differentiable then its derivative is also in the class).

Now we turn the direction of our consideration. We start from real functions and then we restrict them to the set of non negative integers.

Definition 4.3. *An indexed ordered set of real numbers $\phi = \{\phi_i\}_{i\in\mathbb{N}}$ is said to be* admissible *for a function $F : \mathbb{R} \to \mathbb{R}$ if ϕ is a real recursive function in DAnalog and $F(\phi_i) \in \mathbb{N}$, for all $i \in \mathbb{N}$. A function $f : \mathbb{N} \to \mathbb{N}$ is said to be an* admissible restriction *of F if there exists an admissible set $\{\phi_i\}_{i\in\mathbb{N}}$ such that $f(i) = F(\phi_i)$, for all $i \in \mathbb{N}$.*

We can consider functions with many variables.

Definition 4.4. *A tuple of indexed ordered sets of real numbers $\{\phi_i^1\}_{i\in\mathbb{N}}, \ldots, \{\phi_i^k\}_{i\in\mathbb{N}}$ is said to be* admissible *for a function $F : \mathbb{R}^k \to \mathbb{R}$, of arity k, if $F(\phi_{i_1}^1, \ldots, \phi_{i_k}^k) \in \mathbb{N}$, for all $i_1, \ldots, i_k \in \mathbb{N}$, and every sequence $\{\phi_j^i\}_{j\in\mathbb{N}}$, for $i = 1, \ldots, k$, is an admissible set.* Mutatis mutandis *we get the definition of an admissible restriction of F.*

Not all functions have admissible restrictions, like $\lambda x.\ e^{-x}$, or just like a constant $\frac{1}{2}$. Real recursive functions that extend functions over the integers have an infinite number of admissible restrictions.

Let us add the definition of the two following classes:

Definition 4.5. *Let $DAnalog^r$ and $NAnalog^r$ be the restrictions of the classes DAnalog and NAnalog to functions that in any step of their construction have admissible restrictions in PF and NPF, respectively.*

Now we propose the definition of the class fAP (the f stands for *feasible*), which can be interpreted a the class of real recursive functions computed with *quasi-polynomial restrictions* on their values and times of computation. Of course fAP — *analog quasi-polynomial time* in this context is just rhetoric, but as we will see these subclasses of real recursive functions arise in such natural way that they are not superseded by their classical counterparts.

Definition 4.6. *The class of real recursive functions designated by fAP is defined as DAnalog, but substituting the weak exponential order by subexponential order.*

Functions in fAP are said to be deterministic subexponential.

We see that our fAP indeed captures the meaning of Odifreddi words from Section VIII of [8] including all stepwise subexponential functions, providing a quite meaningful computational class. In classical terms, this class is not easy to capture or characterize, since there exists not a operational method to define it. We can only characterize the subexponential functions in the classical framework by means of quantifiers: for every total function f, f is subexponential if, for all $\epsilon > 0$, there exists a positive integer M, such that $f(x) < Me^{\epsilon x}$. Let us add that, of course, $DAnalog \subset fAP$.

5 Main results

We want to strongly motivate our concept of feasible functions, computable over the real numbers, i.e. the functions of fAP (and also of $DAnalog$). We define first a proper subclass $LIN(\mathbb{R})$ of $REC(\mathbb{R})$, by restricting differential recursion to linear differential recursion (see [3]).

Definition 5.1. *$LIN(\mathbb{R})$ The class $LIN(\mathbb{R})$ of real recursive vectors is generated from the real recursive scalars $0,\ 1,\ -1,\ \pi$, primitive functions $\theta_k(x) = x^k\Theta(x)$, for $k \geq 0$, and projections $I_n^i(x_1,\ldots,x_n) = x_i,\ 1 \leq i \leq n,\ n > 0$, by the following operators: composition, assembling and designating components and*

Linear differential recursion: *if f is a real recursive vector with n m-ary components and g is a real recursive matrix with $m \times m$ $(n+1)$-ary components, then the vector h of n m-ary components which is the solution of the Cauchy problem, $1 \leq i \leq n$, $h_i(x_1,\ldots,x_m,0) = f_i(x_1,\ldots,x_m)$, $\partial_y h_i(x_1,\ldots,x_m,y) = \sum_{j=1}^{m} g_{ij}(x_1,\ldots,x_m,y)\ h_j(x_1,\ldots,x_m,y)$ is real recursive.*

In [3] the following results can be found: $LIN(\mathbb{R})$ contains, e.g., sin, cos, $\lambda x.\ e^x$, the rational numbers, and real recursive extensions of successor, addition, and cut-off subtraction; moreover all functions in $LIN(\mathbb{R})$ are continuous and total. All functions in $LIN(\mathbb{R})$ can be approximated (in the sense of Grzegorczyk, e.g., like in [11]) by elementary functions, and all real recursive extensions of elementary functions are contained in $LIN(\mathbb{R})$. Proofs of these statements can be found in [3]. Then as a consequence we have the fact about $LIN(\mathbb{R})$ that it contains extensions of functions from PF and NPF.

Especially important for us is the connection between linear differential equations and the Laplace transform. Using in this context the Laplace transform, operations of differentiation and integration can be replaced with algebra. We consider in what follows the integral form of linear differential recursion: $h(x_1,\ldots,x_m,y) = h(x_1,\ldots,x_m,0) + \int_0^y g(x_1,\ldots,x_m,t)\ h(x_1,\ldots,x_m,t)\ dt + \int_0^y b(x_1,\ldots,x_m,t)\ dt$. Let γ be such that $\gamma(x_1,\ldots,x_m,t-y) = g(x_1,\ldots,x_m,y)$. Such function is in $LIN(\mathbb{R})$ because $\gamma(x_1,\ldots,x_m,y) = g(x_1,\ldots,x_m,t-y)$.

The integral form becomes now

$$h(x_1,\ldots,x_m,y) = h(x_1,\ldots,x_m,0) +$$

$$+ \int_0^y \gamma(x_1,\ldots,x_m,y-t)\ h(x_1,\ldots,x_m,t)\ dt + \int_0^y b(x_1,\ldots,x_m,t)\ dt.$$

We can prove the following theorem:

Proposition 5.1. *fAP $(\bigcap LIN(\mathbb{R}))$ consists of the functions which in any step of construction satisfy $L[\gamma] \leq 1$ in the whole positive real axis.*

Proof. Applying the Laplace transform to the integral form, we get

$$H(x_1,\ldots,x_m,s) =$$

$$= \frac{h(x_1,\ldots,x_m,0)}{s} + H(x_1,\ldots,x_m,s)\ L[\gamma(x_1,\ldots,x_m,y)](s) + \frac{B(x_1,\ldots,x_m,s)}{s},$$

from where it follows that $H(x_1,\ldots,x_m,s) = \frac{h(x_1,\ldots,x_m,0)+\frac{B(x_1,\ldots,x_m,s)}{s}}{s(1-L[\gamma(x_1,\ldots,x_m,y)](s))}$.

We conclude that the solution is in fAP if and only if $L[\gamma(x_1,\ldots,x_m,y)](s) \leq 1$, for all $\Re\ s > 0$, since from the inductive point a view, functions γ and b are already subexponential. □

The integral version of the linear differential recursion scheme is a generalized form of the *Volterra integral equation* that can be written in the standard form

$$y(t) = f(t) + \int_0^t y(\tau)\ K(t-\tau)\ d\tau,$$

for $t > 0$. The function K is called the *kernel* of the equation (cf. [4]). This result is quite interesting because an ordinary differential equation may be transformed into an integral equation. By *linear differential equation* (see, e.g., [12]) we mean a differential equation where the dependent variable appears only with exponent 0 or 1. All differential equations in the sense of this definition are in $LIN(\mathbb{R})$.Then we can state the following fact.

Proposition 5.2. *A linear differential equation has a solution in the class fAP ($\bigcap LIN(\mathbb{R})$) whenever ϕ is subexponential and the kernel k satisfy the Laplacian conditioning of Proposition 5.1.*

Proof. Proposition 5.1 offers all ingredients of the proof, applying the Laplace transform to the Volterra integral equation $u(x) = \phi(x) + \int_a^x\ k(x,t)\ u(t)\ dt$. Since $u(x) = y^{(n)}(x)$, to obtain the Laplace transform of y we have to solve the algebraic equation $U(s) = s^n\ Y(s) - s^{n-1}\ f(0) - s^{n-2}\ y'(0) - ... - y^{(n-1)}(0)$. □

One good idea seems to be defining the matrix of the linear differential recursion scheme in order to, with the help of the Laplace transform, characterize the subexponential world, and, within it, different classes, possibly including P, NP, and lower complexity classes. From the physical point a view, the last example is an example of a dissipative system, with a dumping term. A class of Hamiltonian systems and a class of dissipative systems is captured by our subexponential world containing extensions of $DAnalog$ and fAP.

With the above knowledge let us return to the main theme of the paper. We know from [7], that all functions in PF or in NPF have extensions in $DAnalog^r$ and $NAnalog^r$, respectively. In [7] we proved that (let us recall here that $NPF \subseteq PF$ iff $NP \subseteq P$):

Proposition 5.3. *If a function has an admissible restriction in any step of its construction, then it belongs to DAnalog.*

Proof. Suppose that a function f of arity one in these conditions is not in $DAnalog$. Then we can find, in some step of the construction a component g such that, for all $k \in \mathbb{N}$, $g(x) \geq 2^{\log(x)^k}$. The function g has an admissible restriction which is not quasi-polynomially bounded, contrarily to our hypothesis. Thus f is in $DAnalog$. □

The following statement although similar with final result in our previous paper [7], is more strongly based in a clear mathematical formulation (avoiding Proposition 31 of [7] that, it seems, it is not well-founded [7]).

Proposition 5.4. *If $NPF \subseteq PF$, then $NAnalog^r \subseteq DAnalog^r$.*

Proof. If f is a function in $NAnalog$, then all their admissible restrictions along its construction are nondeterministic quasi-polynomially bounded (i.e., they are in NPF), and then, by hypothesis, they are deterministic quasi-polynomially bounded (i.e., they are in PF). We end the proof applying Proposition 5.3. □

Since all functions in $DAnalog^r$ are built from components f satisfying (a) a differential equation that can be seen as linear and (b) a growing condition, then the differential equation itself can be solved by Laplace transforms. The above considerations justify the following interpretation of Proposition 5.4. **The problem whether P is a proper subclass of NP can be transformed into the problem of a proof $DAnalog^r \neq NAnalog^r$, which will be based on some sets of real functions with Laplace transforms**.

Acknowledgment.

We thank to Kerry Ojakian for point us a few potential problems in the formalization of [7]. These problems were avoided in the formulation of this paper.

References

1. J. R. Ashley. *Introduction to Analog Computation*, John Wiley and Sons, 1963.
2. J. L. Balcázar, J. Díaz, J. Gabarró. *Structural Complexity I*, Springer-Verlag, 1995.
3. M. L. Campagnolo, C. Moore, and J. F. Costa. An analog characterization of the Grzegorczyk hierarchy. *Journal of Complexity*, 18(4):977–1000, 2002.
4. A. C. King, J. Billingham, and S. R. Otto. *Differential Equations, Linear, Nonlinear, Ordinary, Partial*, Cambridge University Press, 2003.
5. C. Moore. Recursion theory on the reals and continuous-time computation. *Theoretical Computer Science,* 162: 23-44, 1996.
6. J. Mycka and J. F. Costa. Real recursive functions and their hierarchy. *Journal of Complexity*, 20(6): 835-857, 2004.
7. J. Mycka and J. F. Costa. The $P \neq NP$ conjecture in the context of real and complex analysis. *Journal of Complexity*, 22 (2): 287-303, 2006.
8. P. Odifreddi. *Classical Recursion Theory*, Elsevier, 1992.
9. D. Richardson. Some undecidable problems involving elementary functions of a real variable. *Journal of Symbolic Logic*, 33 (4):514-520, 1968.
10. C. Shannon. Mathematical theory of the differential analyzer. *J. Math. Phys. MIT*, 20:337-354, 1941.
11. K. Weihrauch. *Computable Analysis, An Introduction*, Springer-Verlag, 2000.
12. D. Zwillinger. *Handbook of Differential Equations*, Academic Press, Third Edition, 1989, 1998.

[7] Although it is not obvious, and more work has to be done with regard to cleaner and simpler formulation of such an analytic conditon — indeed, intended to shed new light on $P \subset NP$ conjecture.

Decidability of Arithmetic Through Hypercomputation: Logical Objections

Paolo Cotogno

paolo.cotogno@istruzione.it

Abstract. The theory of hypercomputation aims to extend effective computability beyond Turing reducibility, in particular by means of supertasks; physical models are considered in Newtonian, relativistic and quantum contexts. Apart from implementation differences, all approaches hold that supertask computations would decide arithmetic; in logical terms, this amounts to assume that the undecidability of Peano Arithmetic can be bypassed by the ω-rule. We argue that the assumption is not granted, as long as the system is consistent: this conclusion follows from alternative versions of Gödel's incompleteness, such as Rosser's and Yablo's, which are essentially indifferent to infinite deductions.

1 Introduction

In this note we deal with hypercomputation, the hypothetical extension of effective computability beyond Turing reducibility; as chief hypercomputing instrument we consider the supertask, an infinite succession of operations concluded in a finite span of time. We focus on the theoretical core of hypercomputation, rather than on the plausibility of physical models; in particular, we discuss the claim that supertask computations can decide arithmetic. Since this feat would apparently overcome classical undecidability results, we shall briefly rehearse the original arguments; we shall then interpret hypercomputation decidability in logical terms, as the claim that Peano Arithmetic becomes decidable through the effective ω-rule. Contrary to the assumption, this is not the case, since Rosser's sentence remains unsolvable even if the system allows infinite deductions, as long as it remains consistent. The same conclusion obtains also by proving undecidability through Yablo's paradox, an infinite form of the Liar.

2 Hypercomputation and physical supertasks

Effective computability concerns the computation processes that transform inputs into definite outputs, in principle. The theory of general recursion establishes the upper bound of effective computability as the degree **0'**, defined by Turing reducibility: apart from matters of efficiency and concrete implementation, all systems for expressing effectively computable functions – the universal programming systems – are reducible to the formalism of Turing Machines.

Above the Turing degree, computability could be just relative, depending on values produced by others sources.

The hypothesis of *hypercomputation*, or super-Turing computation, aims to extend the domain of effective computability beyond the 'barrier' of Turing reducibility. The basic idea consists in gaining more computing power by renouncing the finiteness of ordinary recursive algorithms: this should be achieved either by infinite extensions of discrete computation on integers, or by analogic computation, with real-valued operations as primitives. In Cotogno (2003) we point out some reasons for viewing the latter approach as essentially dependent on the former, so we do not consider analogic models here.

A rigorous treatment of infinite discrete computation is supplied by *infinite time Turing Machines* (Hamkins 2000, 2002). These are defined by finite transition tables, like ordinary Turing Machines, but may receive infinite strings as input, and their process may go on toward ω and beyond, along constructive ordinals. Configurations have a distinguished limit state, where cell tape values can be either passed unchanged from earlier computation steps, or set to the *lim sup* of the cell values before the limit.

Computations with infinitely many steps would be unable to produce outputs *per se*, but one can think of regaining effectivity by making the additional assumption that processing speed increases, so that total operation time converges to a finite value – what we may call a *supertask*. The notion, loosely related to Zeno's ancient paradoxes, is present in modern philosophical literature (e.g. Chihara 1965, Bringsjord and Zenzen 2002), and also in abstract computation theory (e.g., Boolos and Jeffrey 1989, Stewart 1991, Copeland 1998).
In recent years the topic took a new turn and received a remarkable boost, as some started investigating the possibility that supertasks are physically realizable, and not just theoretical fictions. For instance, Tipler (1994, Appendix G) speculates that non-collision singularities arising from Newton's n-body problem (Saari and Xia 1995) could be implemented by a ballistic computer, and realize effective infinite computations. In relativistic physics, one can conceive geometries where time contraction would allow an observer to access infinitely long geodesics in finite time (*Malament-Hogarth* spacetimes). Hogarth (1994), 2004) uses these spacetimes as basis for his theory of *SAD* computers (for *S*trings of *A*rithmetic *D*ecision; *ω-computers* in Wischik (1997), capable likewise of infinite processing sequences leading to finite results. Relativistic computation has been thoroughly criticized by Earman and Norton (1993), but Etesi and Németi (2002) revamp the topic, bringing it to higher levels of physical realism: they expect hypercomputational processes to occur as a by-product of spacetime dynamics within a Kerr black hole – say like the one that should be found in the centre of our galaxy.

If gravitational machines are to realize sequential supertasks, other models foresee the realization of parallel supertasks, through the infinite dimensionality present in the standard formalism of quantum mechanics. For instance, Kieu (2001, 2003) develops his approach on quantum field theory, and considers energy fluctuations of the void with creation-annihilation of particle systems

as hypercomputing operations. Sicard *et al.* (2003) consider hypercomputation through a different model, based on particles in an infinite square well. Calude and Pavlov (2002), Adamyan *et al.* (2003) propose instead a quantum scattering method for solving the infinite form of the *merchant problem* – a classical puzzle where n piles of m coins can contain false pieces of different weight, and only one weighting operation is allowed for checking coins, for $n, m \geq 0$.

3 Hypercomputation and decidability of arithmetic

The current discussion on hypercomputation focuses on the plausibility of the proposed physical models, and takes for granted the fundamental principle, superiority of supertasks over ordinary Turing-reducible algorithms; the typical claim is that hypercomputation would have the power of deciding formalized arithmetic – the first and foremost instance of universal programming system.
For instance: 'Arithmetic is decidable by the multi-string computer' (Hogarth 1994, p. 136); likewise, 'SAD_1 can decide 1-quantifier arithmetic' (Hogarth 2004, p. 685). The demonstration of these claims consists just in explaining the involved definitions: a gravitational computer SAD_1 should be able to check instances of any recursive predicate $R(x, z)$ for any chosen z and all $x \geq 0$, .
Another instance: 'Arithmetic truth is infinite time decidable' (Hamkins 2002, p. 9). Again, the proof is an immediate consequence of the definitions: given any Σ_1-sentence $\exists n \phi(n)$, a suitable infinite-time machine should 'simply try out all the possible values of n in turn', and test the truth of each $\phi(n)$.

The quantum approach of Kieu (2001, 2003) is instead aimed at deciding Hilbert's Tenth Problem, and promises to find solutions to any Diophantine equation – an 'equation with an arbitrary number of unknowns and with rational integer coefficients'. Since all recursively enumerable sets can be represented in Diophantine form, a general answer to the problem would go beyond Turing reducibility, and amount to decide arithmetic[1].

4 Peano Arithmetic and Gödel's Theorems

We shall now see in some more detail what such claims mean from the logical viewpoint. Let us refer to Peano Arithmetic (PA), the formal system based on classical first-order logic and Peano's non-logical axioms ax_{PA}, including the induction schema. An extensive treatment of the metamathematics of PA can be found in Kleene (1952), Háyek and Pudlák (1993).
Assume T is any consistent extension of PA and let x be the only free variable in the formula $\phi(x)$. T is Σ_1-*sound* when the denumerable succession of sentences

[1] As regards the approach of Calude and Pavlov (2002), we should observe that the 'infinite merchant' setting, unlike Diophantine equations, does not amount to a universal programming system; its decidability would not be sufficient to cover Peano Arithmetic.

$\neg\phi(0)$, $\neg\phi(1)$,... , $\neg\phi(n)$,... , for all $n \geq 0$, and the Σ_1 sentence $\exists x\phi(x)$ are not jointly provable in T. Σ_1-soundness evidently subsumes consistency; in a slightly more general form, it was named *ω-consistency* in Gödel (1931).
T is *complete* (in the sense of Post, or syntactic) when either $\vdash_T \neg\phi$ or $\vdash_T \phi$ hold in principle, for all T-sentences ϕ. T is *decidable* when there is an effective method for establishing syntactic completeness; any T-sentence ϕ is decidable in T when there is an effective method to decide whether $\vdash_T \neg\phi$ or $\vdash_T \phi$.

As regards the opposite notion of *undecidability*, we follow Goodstein (1971), and distinguish two different senses: (*i*) *practical*, when one just happens to be unable to assess provability of ϕ, by lack of resources or skill, or by some other unspecified reason; (*ii*) *logical*, when one does prove that ϕ cannot be decided, typically – but not exclusively – by applying the diagonal method and making a reduction to absurd.
Logical undecidability of arithmetic was first established by Gödel's Incompleteness Theorems. Gödel (1931), predating Turing's step-by-step analysis of effective computability, introduced the formalism of recursive functions, and used it to arithmetize the syntax of a 'system P', based on Russell-Whitehead's type theory; shortly after, he adapted the method to standard first-order systems like PA – see Davis (1982), Sieg (2006) for careful reconstructions of Gödel's stand from the computation-theoretic viewpoint.
The arithmetization process culminates in the proof predicate $\mathrm{Prov}_{PA}(x, y)$, where x is the Gödel number of any PA-formula ϕ, and y is the Gödel number of a formal proof ϕ_1 ... ϕ_m, with $\phi_m = \phi$. We shall also need the refutation predicate $\mathrm{Ref}_{PA}(x, y)$, where y is the Gödel number of a formal proof of the negation of the PA -formula of Gödel number x.
Following Kleene, let $A(x, y) = \mathrm{Prov}_{PA}(sub(x, \ulcorner x \urcorner, x), y)$, a self-referencing proof predicate, where *sub* arithmetizes the rule of free-variable substitution, and x ranges on Gödel numbers of open formulae with x as free variable; $B(x, y) = \mathrm{Ref}_{PA}(sub(x, \ulcorner x \urcorner, x), y)$ is the dual for refutation. Then, let $k = \ulcorner \forall y \neg A(x, y) \urcorner$; so, by substituting x with k, one obtains Gödel's sentence: $\mathrm{G} = \forall y \neg A(k, y)$. This Π_1 formula is a negative fixed point for the proof predicate; in a way, it formalizes the Liar paradox, as it asserts its own unprovability in PA via the numeral k.

Proposition 1. *(Gödel). If PA is Σ_1-sound then G is logically undecidable in PA.*

This is the syntactic content of Theorem VI in Gödel (1931), the famous *First Incompleteness Theorem*; the semantic aspects do not concern us here. An argument per absurdum shows $\nvdash_{PA}$ G, if PA is consistent. Hence, no integer n is the Gödel number of a proof of G, so one can enumerate all sentences
(*) $\vdash_{PA} \neg A(k, 0)$, $\vdash_{PA} \neg A(k, 1)$, ..., $\vdash_{PA} \neg A(k, n)$, ...
as theorems of PA, for each $n \geq 0$. Σ_1-soundness then yields $\nvdash_{PA} \exists y A(k, y)$, and this shows $\nvdash_{PA} \neg$G.

Proposition 2. *(Gödel). If PA is consistent then $\nvdash_{PA} Con_{PA}$.*

This is Theorem XI of Gödel (1931), or *Second Incompleteness Theorem*. As usual, Con_{PA} is any sentence formalizing the consistency of PA. The full proof requires a detailed definition of the provability predicate, and is concluded by a reduction to absurd, depending on the First Theorem.

5 Hypercomputation and the ω-rule

The arguments of decidability through supertasks are based, more or less explicitly, on the hypothesis that the undecidability of PA depends on the finite nature of recursive processes; infinitary deductions would be immune to such limitation, as full exhaustion would ensure syntactic completeness of PA. This is reasonable, if we think of any case of practical undecidability: e.g., Pitowsky (1990, p.83) speculates about discovering the truth about Fermat's Last Theorem, by having a relativistic hypercomputer test all quadruples of integers $x, y, z, n \geq 3$ for condition $\mathrm{x}^n + \mathrm{y}^n = \mathrm{z}^n$. We should observe, however, that nothing prevents other methods from delivering the same result: by definition, whatever decision strategy can be more efficient than the 'brute-force' search realized by infinitary processing[2]. The genuine breach in Turing's barrier, extending effective computation beyond **0'**, would open only if supertasks could do something absolutely impossible by recursive means – that is, solving logical undecidability as well as practical cases.

In terms of formal systems theory, we may reduce this perspective to the claim that arithmetic becomes decidable in PA_ω, the extension of PA where the induction schema is subsumed by the *ω-rule*:

$\phi(0)$, $\phi(1)$,... , $\phi(n)$,... $\vdash \forall x \phi(x)$

This schema describes a primitive operation, which draws a universal conclusion from denumerably many individual premisses, dispensing with the passage through the induction step[3]; it was introduced by Hilbert, and belongs to logical folklore also under other names – as infinite induction in Tarski (1933), as Carnap's rule in Rosser (1937). If each ϕ(n) is an effectively provable sentence, for any $n \geq 0$, the ω-rule is called effective: assuming it is applied only a finite number of times, its non-constructive content lies just in the infinity of premisses. Shoenfield (1959) showed that such restriction is immaterial for restoring completeness in PA, so we shall just mean ω-rule in effective sense.

PA_ω can generate its own Gödelian sentence G_ω, and therefore is still subject

[2] As history goes, Wiles succeeded in demonstrating Fermat's Theorem by means of the Taniyama-Shimura conjecture – a proof obtained through hard theoretical work, at the intersection of different areas of mathematics. Aczel (1996) has a lively account.

[3] See Lopez-Escobar (1977) for an introduction. The main reason for considering formal systems endowed with this rule was semantic: in Orey (1956), infinitely long deductions are presented as counterparts of the non-constructive elements inherent in algebraic completeness arguments.

to the argument of Proposition 1; however, since G_ω is based on a different proof-predicate, arithmetizing the ω-rule instead of induction, PA_ω seems to be a definitely more powerful system, capable of vanquishing the undecidability of PA. Rosser (1937, Theorem 3) took a straightforward stand, claiming that the ω-rule can prove Con_{PA}, notwithstanding Proposition 2, by deducing it from the infinitely many premisses $\neg Ref_{PA}(\ulcorner ax_{PA} \urcorner, n)$, for any $n \geq 0$. In a less explicit way, Gödel had already endorsed the same idea:

> (. . .) it can be shown that the undecidable propositions constructed here become decidable whenever appropriate higher types are added (for example, the type ω to the system P). (Gödel (1931), footnote 48)

Unlike contemporary hypercomputation theorists, he was not laying plans for infinitary computers; he was rather illustrating his notoriously anti-mechanistic philosophy of mind:

> Either mathematics is incompletable in this sense, that its evident axioms can never be comprised in a finite rule, that is to say, the human mind (even within the realm of pure mathematics) infinitely surpasses the powers of any finite machine, or else there exist absolutely unsolvable diophantine problems (Gödel (1951), p.310)

He observed that G, since it asserts its own unprovability and is indeed unprovable, should be a true sentence; as it seems, he could not help explaining this semantic insight as a proper inference from the denumerable premisses (*) to the universal conclusion $\forall$y $\neg$A(k, y)[4]. Tarski (1933) was somewhat more prudent; he admitted that G is true in the same structures as the theorems (*), yet is not a formal consequence thereof, and concluded that 'the formalized concept of consequence will, in extension, never coincide with the ordinary one'.

6 Undecidability through Rosser's sentence

Now, the problem with the 'infinite surpass' is that it depends essentially on the specific sentence used to establish Proposition 1: one is entitled hold that the ω-rule proves G just because, and as long as, infinitely many premisses are at hand. In Gödel's proof these are a by-product of Σ_1-soundness, but the assumption is, in fact, unnecessary: Rosser (1936) proved the logical undecidability of PA through a different sentence, requiring just consistency and the well-order of natural numbers. A brief reminder: one should start from the open formula $\forall y(A(x, y) \rightarrow \exists z(z \leq y \wedge B(x, z))$, of Gödel number h, where A and B are again Kleene's self-referential predicates. Then, replacing the free variable x with the numeral h produces Rosser's fixed point:

[4] For this interpretation see Bringsjord and Arkoudas (2004). The original context of Gödel's stand was chiefly set-theoretic: for a discussion, see Feferman (1999).

$\mathrm{R} = \forall y(A(h,y) \rightarrow \exists z(z \leq y \wedge B(h,z)))$.
This paradoxical sentence states that a proof of itself entails a shorter refutation; the role of Σ_1-soundness is played by the well-order of natural numbers, implicit in the clause $z \leq y$.

Proposition 3. *(Rosser). If PA is consistent then R is logically undecidable in PA.*

For the argument's sake, let us assume $\vdash_{PA}\mathrm{R}$; therefore, for some $v > 0$, $\vdash_{PA} A(h,v)$. Consistency then ensures provability of v non-refutation theorems:
(**) $\vdash_{PA} \neg B(h,0)$, ..., $\vdash_{PA} \neg B(h,v)$
These in turn yield $\vdash_{PA} \forall c(c \leq k \rightarrow \neg B(h,c))$; by the hypothesis, then, we have: $\vdash_{PA} \exists b(A(h,b) \wedge \neg\exists c(c \leq b \wedge B(h,c))$. Next, logical transformations yield $\vdash_{PA} \exists b\neg(\neg A(h,b) \vee \exists c(c \leq b \wedge B(h,c))$, and $\vdash_{PA} \neg\forall b(\neg A(h,b) \vee \exists c(c \leq b \wedge B(h,c))$. The last theorem actually amounts to $\nvdash_{PA} \neg\mathrm{R}$, against the hypothesis; hence, $\nvdash_{PA}\mathrm{R}$. An analogous argument shows $\nvdash_{PA} \neg\mathrm{R}$; for details see Kleene (1952, Theorem 29).

The interesting fact here is the difference between the successions (*) and (**): the ω-rule may deduce an otherwise unreachable conclusion from the former, but has no leverage on the latter, which contains but $v + 1$ sentences. If one tries to generalize on $A(h,y)$, the conclusion is vanified by the second clause $\exists z(z \leq y \wedge B(h,z))$, which is in any case bounded, and thus indifferent to the ω-rule; the consequence is that R remains undecidable in PA_ω.

7 Undecidability through Yablo's paradox

Another reason for being skeptical about the ω-rule as silver bullet for undecidability stems from the paradox introduced by Yablo (1985, p. 340); the argument consists of a denumerable succession of sentences, each asserting that its successors are all untrue, a sort of infinite-descent Liar. This was originally intended as a semantic device, but Priest (1997, footnote 4) remarks that it may be turned into a proof of the First Incompleteness Theorem; by the way, Gödel himself (1931, footnote 14) had already observed that his proof could be established through any other 'epistemological antinomy'.
Let us sketch the argument by means of the refutation predicate Ref_{PA}, instead of the 'untruth' predicate adopted in the original version:

Proposition 4. *If PA is Σ_1-sound then some denumerable succession of PA-sentences is logically undecidable.*

Let us stat from the infinite succession
(***) ϕ_0, ϕ_1, ..., ϕ_i,... ,
where each generic instance ϕ_i, for any $i \geq 0$, is a $\forall\exists$ sentence stating that all its successors are refutable:
$\phi_i = \forall j(j > i \rightarrow \exists z Ref_{PA}(\ulcorner\phi_j\urcorner, z)$

For the argument's sake, let us assume $\vdash_{PA} \phi_i$ for an arbitrary $i \geq 0$: by Σ_1-soundness, each existential clause $\exists z Ref_{PA}(\ulcorner \phi_j \urcorner, z)$ should be satisfied by some integer, so we may assert $\vdash_{PA} \neg\phi_j$ for all $j > i$. Then, let us consider the subsequent sentence ϕ_{i+1}: all its successors are refutable, therefore we should have $\vdash_{PA} \phi_{i+1}$, a contradiction. Vice versa, let us assume $\vdash_{PA} \neg\phi_i$: again, Σ_1-soundness yields that for some $z > i$ we may assert $\vdash_{PA} \phi_z$, so a contradiction obtains again for ϕ_{z+1}, by repeating the first part of the argument.

It is interesting to note that no sentence in (***) does ever refer to itself, unlike the fixed points G and R; as Hardy (1995) points out, Yablo's paradoxical engine is powered by the infinity inherent in Σ_1-soundness, rather than by self-reference.

Proposition 5. *PA_ω is consistent if and only if it is Σ_1-sound.*

Given a denumerable collection of premisses $\neg\phi(0)$, $\neg\phi(1)$,... , $\neg\phi$(n),..., for all $n \geq 0$, the ω-rule yields $\vdash_{PA\omega} \forall x \neg\phi(x)$. Then let us assume, for the argument's sake, that PA_ω is not Σ_1-sound: so, the same premisses would yield $\vdash_{PA\omega}$ $\exists$x $\phi(x)$, and this is prevented by the consistency of PA_ω. The other direction is immediate.

This yields that Proposition 4 extends to PA_ω, on the assumption of simple consistency: even if one could sweep the whole succession (***) in a single pass and reach a universal conclusion like $\forall\phi_i$, that would make no difference for deciding each ϕ_i. Unsolvability of Yablo-style sentences is a built-in feature of the whole infinite succession, and is not affected by the way one scans the elements, be it one at the time, or in batch.

8 Conclusion

The alternative arguments for Gödel's theorem just outlined are another way of observing that the logical undecidability of arithmetic is essentially different from the practical cases, and cannot be bypassed through the ω-rule. Therefore, even admitting that gravitational or quantum hypercomputers could somehow realize supertasks, one would still remain unable to reach a syntactic decision on arbitrary PA-formulae, as long as the system remains classical and consistent. Since all universal programming systems are Turing reducible to PA, the conclusion holds irrespective of any particular deduction or computation strategy. Common wisdom may view undecidability as a limitation, as if it was some fault of formal systems; that cliché is evidently derived from the early interpretation of Gödel's Theorems as negative, unpleasant results, as if they were but impediments to Hilbert's Programme. There is, however, a plain way (Cotogno 2006) of taking Gödel's proof from a more positive viewpoint: the First Incompleteness Theorem shows that the consistency of PA implies unprovability of some fixed-point sentence, so as unprovability of at least one formula implies consistency. In other words, logical undecidability *is* consistency, is just making sure that

formalized arithmetic is free of contradiction; hence, the bound established by Turing reducibility is no barrier to be overtaken, but just a side-effect of consistency.

References

Aczel, A.D.: *Fermat's Last Theorem*, Delta, New York, 1996.

Adamyan, V.A., Calude, C.S., Pavlov, B.S.: *Transcending the Limits of Turing Computability*, arXiv:quant-ph/0304128, 2003.

Bringsjord, S., Arkoudas, K.: 'The modal argument for hypercomputing minds', *Theoretical Computer Science* **317**, 2004, 167-190.

Bringsjord, S., Zenzen, M.: 'Toward a Formal Philosophy of Hypercomputation', *Mind and Machines* **2**, 2002, 241-258.

Calude, C.S., Pavlov, B.: 'Coins, Quantum Measurements, and Turing's Barrier', *Quantum Information Processing* **1**, 2002, 107-127.

Chihara, C.S.: 'On the Possibility of Completing an Infinite Process', *The Philosophical Review* **84**, 1965, 74-87.

Copeland, B.J.: 'Super Turing-Machines', *Complexity* **4**, 1998, 30-2.

Cotogno, P.: 'Hypercomputation and the Physical Church-Turing Thesis', *British Journal for the Philosophy of Science* **54**, 2003, 181-223.

Cotogno, P.: 'A Note on Gödel's Theorem and the Rejection of Hilbert's Programme', *Gödel Centenary Symposium*, Vienna 2006, forthcoming.

Davis, M.: 'Why Gödel did not have Church's Thesis', *Information and Control* **54**, 1982, 3-24.

Earman, J., Norton, J.D.: 'Forever is a Day: Supertasks in Pitowsky and Malament-Hogarth Spacetimes', *Philosophy of Science* **60**, 1993, 22-42.

Etesi, G., Németi, I.: 'Non-Turing computations via Malament-Hogarth space-times', *International Journal of Theoretical Physics* **41**, 2002, pp. 341-70.

Feferman, S.: 'Does mathematics needs new axioms?', *American Mathematical Monthly* **106**, 1999, 99-111.

Gödel, K. (1931): 'On formally undecidable propositions of Principia Mathematica and related systems I', in: S.G. Shanker [ed.], *Gödel's Theorem in Focus*, Routledge, London 1988, 17-47.

Gödel, K. (1951): 'Some Basic Theorems on the Foundations of Mathematics and their Implications', in S. Feferman *et al.* [eds.] *Collected Works III*, Oxford University Press, Oxford 1995, 304-323.

Goodstein, R.: *The Development of Mathematical Logic*, Logos Press, Glasgow 1971.

Hardy, J.: 'Is Yablo's paradox Liar-like?', *Analysis* **55**, 1995 197-198.

Háyek, P., Pudlák, P.: *Metamathematics of First-Order Arithmetic*, Springer Verlag, Berlin 1993.

Hogarth, M.: 'Non-Turing Computers and Non-Turing Computability', *PSA 1994*, Philosophy of Science Association, East Lansing, 1994, 126-138.

Hogarth, M.: 'Deciding Arithmetic Using SAD Computers', *British Journal for the Philosophy of Science* **55**, 2004, 681-691.

Kieu, T.D.: *Quantum Algorithm for the Hilbert's Tenth Problem*, arXiv:quant-ph/0110136, 2001.

Kieu, T.D.: 'Computing the Noncomputable', *Contemporary Physics* **44**, 2003, 51-71.

Kleene, S.C. (1952): *Introduction to Metamathematics*, North-Holland, Amsterdam 1974[7].

Lopez-Escobar, E.G.K.: 'Infinite rules in finite systems', in: A.I. Arruda, N.C.A. Da Costa, R. Chuaqui [eds.], *Non-classical Logics, Model Theory and Computability*, North-Holland, Amsterdam, 75-97.

Orey, S.: 'On ω-consistency and related properties', *Journal of Symbolic Logic* **21**, 1956, 246-252.

Pitowsky, I.: 'The Physical Church's Thesis and Physical Computational Complexity', *Iyyun* **39**, 1990, 81-99.

Priest, G.: 'Yablo's Paradox', *Analysis* **57**, 1997, 236-242.

Rosser, J.B.: 'Extensions of some theorems of Gödel and Church', *Journal of Symbolic Logic* **1**, 1936, 87-91.

Rosser, B.: 'Gödel Theorems for Non-constructive Logics', *Journal of Symbolic Logic* **2**, 1937, 129-137.

Saari, D.G., Xia, Z.J. [1995]: 'Off to Infinity in Finite Time', *Notices of the American Mathematical Society* **42**, 538-546.

Shoenfield, J.R.: 'On a Restricted ω-Rule', *Bulletin de L'Academie Polonaise de Sciences* **7**, 1959, 405-407.

Sicard, A., Vélez, M, Ospina, J.: *Computing a Turing-Incomputable Problem from Quantum Computing*, arXiv:quant-ph/0309198, 2003.

Sieg, W.: 'Gödel on computability', *Philosophia Mathematica*, 2006, forthcoming.

Stewart, I.: 'The Dynamics of Impossible Devices', *Nonlinear Science Today* **1**, 1991, pp. 8-9.

Tarski, A. (1933): 'Some Observations on the Concept of ω-Consistency and ω-Completeness', in: *Logics, Semantics, Metamathematics*, Clarendon Press, Oxford 1956, 279-295.

Tipler, F.: *The Physics of Immortality*, Doubleday, New York, 1994.

Wischik, L.: *A Formalization of Non-Finite Computation, dissertation*, University of Cambridge, 1997.

Yablo, S.: 'Truth and Reflection', *Journal of Philosophical Logic* **14**, 1985, 297-349.

On Generalising Predicate Abstraction

Birgit Elbl

Institut für Theoretische Informatik und Mathematik, Fakultät für Informatik, UniBw München, 85577 Neubiberg, Germany; Birgit.Elbl@unibw.de

Abstract. Starting from a simple term system, in which pure Prolog can be naturally embedded, the mechanism for building predicate terms is generalised. The resulting system has two forms of predicate abstraction which differ in semantics and are different from functional abstraction. It is more expressive than the original system, hence more expressive than pure Prolog. This is exemplified by defining some meta-logical predicates.

Keywords: predicate abstraction, term calculi, expressiveness, denotational semantics, logic programming

1 Introduction

Logic programs can be used to compute arithmetical functions. With respect to this task, even pure Prolog is complete: for every partial recursive function there is a pure Prolog program to compute it. Numbers, however, are not the standard data structure in logic programs. Predicates take *terms* as arguments and in case of a successful evaluation substitutions for their variables are returned. Terms, also terms with variables, are used to represent data. Typical programming techniques use open lists or open trees. To manipulate these terms, predicates which refer to the structure of terms are useful. One basic ingredient is, for example, the test whether a term is a variable using a predicate *var*. This predicate can not be explained by reference to the usual Herbrand models: *var*(t) fails for all ground terms but it can not be interpreted as the empty set, as *var*(x) succeeds. In a full Prolog system, a predicate for this test and similar operations are usually available as further built-ins. They can also be implemented using *cut*,*assert*,*retract*. Here we consider a completely different way to strengthen pure Prolog.

A pure logic program can be considered as a set of recursive predicate definitions, w.r.t. which a goal expression is evaluated. Similar to the theory of functional programs, we can use term systems as abstract modelling languages for them. The natural choice for a term system for pure Prolog would contain propositional constants, equality, connectives and quantifiers, recursion, and a form of predicate abstraction which corresponds to a definition $p(\bar{x}) \Leftrightarrow \mathit{Def}$ where *Def* is a formula with variables in $\bar{x}$. We consider a predicational language with a restricted abstraction like this and call it PL-(abs)-r. It is straightforward how to embed pure Prolog (i.e. without negation) in that system. As for the denotational semantics, there are several ways to define the meaning for predicate abstraction. We consider two variants. Although, for that small system, it

makes no difference which one we choose, we discuss both versions and introduce syntactic counterparts for them in a system PL-(2 abs)-r. It turns out that for one of them there is a natural way to generalise predicate formation to a version where some variables are not subject to the abstraction involved. The resulting system is called PL-(2 abs).

In the sequel we investigate the effect of varying the form of predicate abstraction and the system PL-(2 abs). First we show that in this system, the two versions of predicate abstractions are no longer equivalent. Furthermore, none of them is functional abstraction. In particular, β-reduction would not be a sound conversion for them. In the restricted systems the situation is different: both versions of abstraction are semantically equivalent. In PL-(2 abs), some meta-logical predicates can be defined, in contrast to the restricted systems. Hence the generalisation of predicate abstraction leads indeed to more expressiveness.

2 Syntax and Semantics of PL-(2 abs)

The syntactic components of PL-(2 abs) are constructor terms, formulas, and predicate terms of arity n, for which we use type ι, o and π_n respectively. We presuppose fixed sets LPV of (logic programming) variables, CONS_n of n-ary function or constructor symbols, and PRED_n of n-ary predicate symbols (or variables). The constructor terms are built from variables and constructor symbols as usual. We assume $|\text{CONS}_0| \geq 2$. Formulas are built from atomic formulas using propositional connectives and the quantifier E as usual. In the sequel we use the propositional constants $\mathbf{1}, \mathbf{F}, \mathbf{0}$ and the binary connectives $\otimes, *$. The goal $\mathbf{F}$ fails, evaluation of $\mathbf{1}$ is successful, yielding a single answer 'yes', and $\mathbf{0}$ does not terminate error-free. The symbols $\otimes, *, \mathsf{E}$ for conjunction, disjunction and quantification are chosen as in [4] where the logical background is discussed. The standard notation for classical logic is avoided, as the connectives differ significantly from the classical ones. In particular, the classical weakening and contraction rules are unsound for $\otimes$ and $*$, and these connectives are not commutative. Two different operations to build predicate terms explicitly are considered.

Definition 1. *The terms of the system PL-(2 abs) are defined inductively by:*

- $\mathbf{1}, \mathbf{F}, \mathbf{0}$ *and* $t = s$ *where* t, s *are constructor terms are formulas,*
- $p(\bar{t})$ *is a formula if* p *is an* n*-ary predicate term and* $\bar{t} = t_1, \ldots, t_n$ *are constructor terms,*
- $G_1 \otimes G_2$, $G_1 * G_2$, *and* $\mathsf{E}xG$, *where* x *is a variable, are formulas if* $G_1, G_2 \colon o$,
- *predicate variables are predicate terms,*
- $(\bar{x}.G) \colon \pi_n$ *if* $G \colon o$ *and* $\bar{x} = x_1, \ldots, x_n$ *are pairwise distinct* LP*-variables,*
- $(\bar{x} \mid G) \colon \pi_n$ *if* $G \colon o$ *and* $\bar{x} = x_1, \ldots, x_n$ *are pairwise distinct* LP*-variables so that* $\text{LPV}(G) \subseteq \{\bar{x}\}$,
- $(\mathtt{rec}\,\bar{p}.(\bar{P}))_i$ *is an* n_i*-ary predicate term if* $\bar{P} = P_1, \ldots, P_m$ *are predicate terms of arity* $\bar{n} = n_1, \ldots, n_m$ *respectively,* $\bar{p} = p_1, \ldots, p_n$ *are predicate variables of arity* $\bar{n} = n_1, \ldots, n_m$ *respectively, and* $i \in \{1, \ldots, m\}$

Here and in the sequel LPV$(\cdot)$ is used to refer to the set of free LP-variables. The quantification $\mathsf{E}x$, $(\bar{x}.\ )$, and $(\bar{x} \mid\ )$ bind the mentioned variable(s). Terms that are equal up to bound renaming are identified. The `rec`-construction corresponds to a combination of tupling, abstraction of predicate variables, fixed point operator, and projection. *Goals* are formulas that contain no free predicate names. Note that application of $(\bar{x}.\ )$ — in contrast to $(\bar{x} \mid\ )$ — is not subject to a condition on the occurrence of free LP-variables. In the restricted system PL-(2 abs)-r we change this. Furthermore we consider also a version with only one form of abstraction.

Definition 2. *The terms of the system PL-(2 abs)-r are those terms of PL-(2 abs) where* LPV$(G) \subseteq \{\bar{x}\}$ *holds for every subterm of the form* $(\bar{x}.G)$*. The terms of the system PL-(abs)-r are those terms of PL-(2 abs)-r which contain no subterm of the form* $(\bar{x} \mid G)$*. The systems PL-(2 abs)-(let), PL-(2 abs)-r-(let), and PL-(abs)-r-(let) are obtained from PL-(2 abs), PL-(2 abs)-r, and PL-(abs)-r respectively by the following additional rule:*

If G is a formula, $\bar{x} = x_1, \ldots, x_n$ are pairwise distinct LP*-variables and $\bar{t} = t_1, \ldots, t_n$ are constructor terms, then* `let` $\bar{x} = \bar{t}$ `in` G *is a formula.*

In `let` $\bar{x} = \bar{t}$ `in` G, the occurrences of $\bar{x}$ in G are bound but `let` has no effect on the variables in $\bar{t}$. In particular, we can apply bound renaming to ensure $\bar{x}$ not in $\bar{t}$. The term `let` $\bar{x} = \bar{t}$ `in` G is semantically equivalent to $G\{\bar{t}/\bar{x}\}$. As a consequence, every term is equivalent to a `let`-free term. It would be justified to write $(\lambda\bar{x}.G)(\bar{t})$ (or $(\lambda\bar{x}.G)\bar{t}$ if no x_i occurs in $\bar{t}$) instead of `let` $\bar{x} = \bar{t}$ `in` G. Hence the comparison of $(\bar{x}.G)(\bar{t})$, $(\bar{x} \mid G)(\bar{t})$ and `let` $\bar{x} = \bar{t}$ `in` G can serve to compare the predicate abstractions with functional abstraction. However, λ-abstraction is not introduced as a means to build predicate terms.

The language $\mathcal{L}_p$ in [5] is obtained from PL-(abs)-r by the restriction that $(\texttt{rec}\ )_i$ and $(\bar{x}.\ )$ are used in the combination $(\texttt{rec}\,\bar{p}.((\bar{x}^{(1)}.G_1), \ldots, (\bar{x}^{(m)}.G_m)))_i$ only. In $\mathcal{L}_p$ the expression $(\bar{x}.G)$ is strictly speaking no term but it could be introduced as a shorthand for a $(\texttt{rec}\ )$-expression which does not bind any predicate variable. The embedding of pure Prolog programs with goals in $\mathcal{L}_p$ now yields an embedding in PL-(abs)-r: assume that the program $\mathcal{P}$ has already been transformed according to Clark's completion [2] into a set of m predicate definitions $p_i(\bar{x}^{(i)}) \Leftarrow \mathit{Def}_i$ where we use $\otimes, *, \mathsf{E}$ as conjunction, disjunction, and existential quantification. Then Def_i contains only LP-variables in $\bar{x}^{(i)}$ free, and it is a disjunction of formulas of the form $\mathsf{E}\bar{u}(\bar{x}^{(i)} = \bar{t} \otimes A_1 \otimes \ldots \otimes A_k)$ where the A_ν are atoms with a predicate symbol in $\bar{p} := p_1, \ldots, p_m$ and all LP-variables in $\bar{t}$ or A_ν are bound by the quantifier. As a consequence, Def_i is a term in PL-(abs)-r satisfying LPV$(\mathit{Def}_i) \subseteq \{\bar{x}^{(i)}\}$. So $P_i :\equiv (\texttt{rec}\,\bar{p}.((\bar{x}^{(1)}.\mathit{Def}_1), \ldots, (\bar{x}^{(m)}.\mathit{Def}_m)))_i$, $i = 1, \ldots, m$, are predicate terms in PL-(abs)-r. The formula in PL-(abs)-r that corresponds to program $\mathcal{P}$ with goal $?- p_{i_1}(\bar{s}^{(1)}), \ldots, p_{i_k}(\bar{s}^{(k)})$ is $P_{i_1}(\bar{s}^{(1)}) \otimes \ldots \otimes P_{i_k}(\bar{s}^{(k)})$. This embedding could easily extended to include programs with negation, if a "not"-operator is added to PL-(abs)-r. As we consider pure Prolog, however, this "not"-operator would also correspond to Prolog's negation "fail-if", where non-ground goals are not delayed.

As for the semantics, we extend the denotational semantics based on stream functions which has been used for $\mathcal{L}_p$ to PL-(2 abs). For similar fixed point semantics for Prolog see [6, 3, 1]. The denotational semantics of $\mathcal{L}_p$ has been presented in [5] and builds on notions already introduced in [4]. There they serve to define a standard model which is used in the proof of the completeness of pure Prolog's evaluation w.r.t. a declarative semantics, namely biquantale semantics. So here we just recapitulate the main ingredients briefly, and refer for details and more discussion to [5, 4].

Finite substitutions and answers: A finite set $\{t_1/x_1, \ldots, t_n/x_n\}$ of bindings, where $x_1, \ldots, x_n$ are distinct variables and $t_1, \ldots, t_n$ are terms, is called *finite substitution*. Similar to [3], this notion of finite substitution takes into account the domain: substitutions may contain identical bindings x/x, and θ is usually different from $\theta \cup \{x/x\}$. The set of finite substitutions is denoted by FS. The domain of a finite substitution and the set of variables in its range are given by $\mathrm{DOM}(\{\bar{t}/\bar{x}\}) = \{\bar{x}\}$ and $\mathrm{RG}(\{\bar{t}/\bar{x}\}) = \mathrm{LPV}(t_1) \cup \ldots \cup \mathrm{LPV}(t_n)$, $\mathrm{FS}_d := \{\theta \in \mathrm{FS} \mid \mathrm{DOM}(\theta) = d\}$. A finite substitution $\delta = \{\bar{y}/\bar{x}\}$ where $\bar{y} = y_1, \ldots, y_n$ are pairwise distinct variables is called *finite renaming*. For finite renamings $\delta = \{\bar{y}/\bar{x}\}$ we let $\delta^{-1} := \{\bar{x}/\bar{y}\}$. Furthermore $\mathrm{id}_{\{\bar{x}\}}$ stands for $\{\bar{x}/\bar{x}\}$. If $\mathrm{DOM}(\theta) \cap \mathrm{DOM}(\sigma) = \emptyset$ and also $\mathrm{RG}(\theta) \cap \mathrm{RG}(\sigma) = \emptyset$, we call θ and σ *disjoint* and use the notation $\theta \uplus \sigma$ for $\theta \cup \sigma$. We use the following notations for operations on FS:

$\theta \downarrow d := \{t/x \mid (t/x) \in \theta \text{ and } x \in d\} \quad (d \subseteq \mathrm{LPV})$
$D(\theta) := \bigcap\{d \subseteq \mathrm{DOM}(\theta) \mid (\theta \downarrow d) \uplus \delta = \theta \text{ for some finite renaming } \delta\}$
$\theta - x := \theta \downarrow (\mathrm{DOM}(\theta) \setminus \{x\})$ for variables x
$\theta \cdot \sigma := \{t\sigma/x \mid (t/x) \in \theta\}$
$\theta \circ \sigma = (\theta \cdot \sigma) \cup (\sigma \downarrow (\mathrm{DOM}(\sigma) \setminus \mathrm{DOM}(\theta)))$

Finite substitutions that describe the "same" result when considered as output of the evaluation should be identified in the semantics. To this end, the relation $\sim$ is introduced. Consider for example a goal with variables x, y and the substitutions $\theta_1 = \{x/x, f(x)/y\}$, $\theta_2 = \{u/x, f(u)/y\}$. The auxiliary variables in the terms are different but θ_1, θ_2 describe the same result. In contrast to this, the substitution $\theta_3 = \{f(u)/y\}$ describes no restriction for x. Hence it is not equivalent to θ_1, θ_2, but it is equivalent to $\theta_4 = \{f(u)/y, v/x\}$ where there is also no restriction for the variable x, $x \in \mathrm{DOM}(\theta_4) \setminus D(\theta)$, which is just renamed. Finite substitutions θ, σ are *equivalent*, $\theta \sim \sigma$, if and only if there is a finite renaming δ with $\mathrm{DOM}(\delta) = \mathrm{RG}(\theta \downarrow D(\theta))$ so that $\sigma \downarrow D(\sigma) = (\theta \downarrow D(\theta)) \cdot \delta$. Equivalence classes $\theta^\sim$ w.r.t. $\sim$ are called *answers*, AS denotes the set of answers and $\mathrm{AS}_d := \{\theta^\sim \mid \theta \in \mathrm{FS} \text{ and } \mathrm{DOM}(\theta) = d\}$. For every $\theta \in \mathrm{FS}$, $\alpha \in \mathrm{AS}_{\mathrm{RG}(\theta)}$, $d \subseteq \mathrm{LPV}$, let $\theta \cdot \alpha := \theta \cdot \sigma^\sim$ for some $\sigma \in \alpha$ that satisfies $\mathrm{DOM}(\sigma) = \mathrm{RG}(\theta)$, and $\alpha \downarrow d := \sigma \downarrow d^\sim$ for some $\sigma \in \alpha$. (Note that $\theta \cdot \alpha$ and $\alpha \downarrow d$ are independent of the choice of σ.)

Streams: The set of *streams* over a set M consists of the infinite sequences $(m_i)_{i \in \omega}$, the finite total streams $[m_1, \ldots, m_n]$, and the finite partial streams $[m_1, \ldots, m_n, \bot]$ of elements $m_i \in M$. Stream concatenation is denoted by $*$. Operations on answers are extended to operations on streams of answers in the straightforward way. Let $F\colon M \longrightarrow \mathit{Stream}(M')$ for some sets M, M'. For $S \in \mathit{Stream}(M)$ we define $*_{m \in S} F(m)$ by

$$*_{m\in S}\, F(m) := \begin{cases} F(m_1) * \ldots * F(m_n) & \text{if } S = [m_1, \ldots, m_n] \\ F(m_1) * \ldots * F(m_n) * [\bot] & \text{if } S = [m_1, \ldots, m_n, \bot] \\ \sup_{n\in\omega}(F(m_1) * \ldots * F(m_n) * [\bot]) & \text{if } S = (m_i)_{i\in\omega} \end{cases}$$

Functions $\text{FS} \longrightarrow \mathit{Stream}(\text{AS})$: Evaluation of goals yields streams of answers. The denotational value of a (sub-)goal contains also information concerning the result of evaluating it when some bindings θ are already computed. So we arrive at considering functions $\text{FS} \longrightarrow \mathit{Stream}(\text{AS})$. For these functions we define $F \downarrow d$ by $(F \downarrow d)(\theta) := F(\theta \downarrow d)$ and $D(F) := \bigcap\{d \subseteq \text{LPV} \mid F = F \downarrow d\}$. Obviously, for every interpretation of a goal, there will be a finite set d, so that only bindings for elements of d are relevant, hence $D(F)$ will be finite.

Interpretation of types: The set of constructor terms is denoted by $\mathbb{T}$, and $\mathbb{P}_n$ is the set of functions $f : \mathbb{T}^n \longrightarrow \mathit{Stream}(\text{AS})$ satisfying:

1. $f(\bar{t}) \in \mathit{Stream}(\text{AS}_{\text{LPV}(\bar{t})})$ for every $(\bar{t}) \in \mathbb{T}^n$
2. $f(\bar{t}\delta) = \delta^{-1} \cdot f(\bar{t})$ for every finite renaming $\delta \in \text{FS}_{\text{LPV}(\bar{t})}$

Predicates of arity n are interpreted as elements of $\mathbb{P}_n$. The set $\mathbb{G}$ which is used as the interpretation of type o is the set of functions $F\colon \text{FS} \longrightarrow \text{Stream}(\text{AS})$ where $D(F)$ is finite and for all $\theta \in \text{FS}$:

1. $F(\theta) \in \mathit{Stream}(\text{AS}_{\text{RG}(\theta)})$
2. $F(\theta \cdot \delta) = \delta^{-1} \cdot F(\theta)$ for all renamings $\delta \in \text{FS}_{\text{Rg}(\theta)}$
3. $F(\theta \cup \delta) \downarrow \text{RG}(\theta) = F(\theta)$ for all renamings δ so that δ,θ disjoint.

Furthermore $\mathbb{G}(d) := \{F \in \mathbb{G} \mid D(F) \subseteq d\}$ for finite $d \subseteq \text{LPV}$. Every $\mathbb{G}(d)$ with the usual ordering is a cpo. Condition 1 above reflects the fact that we get restrictions only for variables in $\text{RG}(\theta)$. The functions F in $\mathbb{G}$ do not return the same result for equivalent substitutions: if the argument is $\{f(x)/y\}$, we get a binding for x, in case of success some $\{t/x\}^\sim$; if the argument is $\{f(u)/y\}$, we get a binding for u instead — but it has to be $\{t/u\}^\sim$. Condition 2 is a generalisation of this observation to arbitrary substitutions. Condition 3 and the fact that the full set FS is the domain of functions in $\mathbb{G}$ are helpful in defining operations on $\mathbb{G}$. For example, we obtain a simple interpretation of E.

If $A[\theta]$ is an expression which has a value $f(\theta)$ in $\text{Stream}(\text{AS}_{\text{RG}(\theta)})$ for all $\theta \in \text{FS}_d$ and $f(\theta \cdot \delta) = \delta^{-1} \cdot f(\theta)$ holds for all $\theta \in \text{FS}_d$ and finite renamings $\delta \in \text{FS}_{\text{RG}(\theta)}$, then there is a uniquely determined element F in $\mathbb{G}(d)$ coinciding with f on FS_d. In this situation we use $\lambda\theta : d.A[\theta]$ for this element of $\mathbb{G}$.

Unification: Let $\text{mgu}(\bar{t}; \bar{s})$ stand for the finite substitution $\{x\theta/x \mid x \in \text{LPV}(\bar{t}, \bar{s})\}$ where θ is the most general unifier of $\bar{t}$ and $\bar{s}$ as computed by Robinson's algorithm [7], and $\overline{\text{mgu}}(\bar{t}; \bar{s}) := [\theta^\sim]$ if $(\bar{t})$ and $(\bar{s})$ are unifiable with $\text{mgu}(\bar{t}; \bar{s}) = \theta$, $\overline{\text{mgu}}(\bar{t}; \bar{s}) = [\,]$ if $(\bar{t})$ and $(\bar{s})$ are not unifiable. The function $m_{\bar{t};\bar{s}}$ is defined by $m_{\bar{t};\bar{s}} := \lambda\theta : \text{LPV}(\bar{t}, \bar{s}).\overline{\text{mgu}}(\bar{t}\theta; \bar{s}\theta)$. It is used as the interpretation of $t = s$.

Operations on $\mathbb{G}$: Let $F, G \in \mathbb{G}$. The functions $F - x$ and $F * G$ are defined by $F - x := F \downarrow (D(F) \setminus \{x\})$ for all variables x and $(F * G)(\theta) = F(\theta) * G(\theta)$ for all $\theta \in \text{FS}$. These operations on $\mathbb{G}$ serve as interpretation of quantification E and disjunction $*$ respectively. The definition of $\otimes$ on $\mathbb{G}$ takes two steps: Given

$F, G \in \mathbb{G}$, we let $(F \times G)(\theta) := *_{\sigma \in S_\theta}(\sigma \cdot G(\theta \cdot \sigma))$ where S_θ is a stream in $\mathrm{Stream}(\mathrm{FS}_{\mathrm{RG}(\theta)})$ satisfying $S_\theta^\sim = F(\theta)$. For $F, G \in \mathbb{G}$, the function $F \times G$ is well-defined but not necessarily in $\mathbb{G}$. As interpretation of $\otimes$, we use $F \otimes G := \lambda\theta : D(F) \cup D(G).(F \times G)(\theta)$ which is in $\mathbb{G}$. These operations are continuous on every $\mathbb{G}(d)$ ($d \subseteq \mathrm{LPV}$ finite).

Having presented those parts of the semantics in [5, 4] that will be used in the sequel, we now turn to the semantics for PL-(2 abs)-(let). To this end, let ENV denote the set of functions $\varphi : \bigcup_{n\in\mathbb{N}} \mathrm{PRED}_n \longrightarrow \bigcup_{n\in\mathbb{N}} \mathbb{P}_n$ so that $\varphi(p) \in \mathbb{P}_n$ for $p \in \mathrm{PRED}_n$. As a notation for the environment obtained from φ by assigning the value $\bar{\wp}$ to $\bar{p}$ leaving the remaining assignments unchanged we use $\varphi[\bar{p} \leftarrow \bar{\wp}]$.

Definition 3. *The function $[\![\cdot]\!]$ which maps formulas to elements of* ENV $\times$ FS $\longrightarrow \mathbb{G}$ *and n-ary predicates to elements of* ENV $\times$ FS $\longrightarrow \mathbb{P}_n$ *is defined by:*

- $[\![\mathbf{F}]\!]_{\varphi,\sigma} := \lambda\theta\colon \emptyset.[]$ $\quad [\![\mathbf{1}]\!]_{\varphi,\sigma} := \lambda\theta\colon \emptyset.[\emptyset^\sim]$
  $[\![\mathbf{0}]\!]_{\varphi,\sigma} := \lambda\theta\colon \emptyset.[\bot]$ $\quad [\![t = s]\!]_{\varphi,\sigma} := m_{t\sigma;s\sigma}$
- $[\![G_1 \ op \ G_2]\!]_{\varphi,\sigma} := [\![G_1]\!]_{\varphi,\sigma} \ op \ [\![G_2]\!]_{\varphi,\sigma}$ *for* $op \in \{\otimes, *\}$
  $[\![\mathsf{E}xG]\!]_{\varphi,\sigma} := [\![G]\!]_{\varphi,\sigma} - x$ *where* $x \notin \mathrm{DOM}(\sigma) \cup \mathrm{RG}(\sigma)$
- $[\![P(\bar{t})]\!]_{\varphi,\sigma} = \lambda\theta\colon \mathrm{LPV}(\bar{t}\sigma).[\![P]\!]_{\varphi,\sigma}(\bar{t}\sigma\theta)$
- $[\![p]\!]_{\varphi,\sigma} = \varphi(p)$ *for* $p \in \mathrm{PRED}_n$
- $[\![(\mathtt{rec}\,\bar{p}.(P_1, \ldots, P_m))_i]\!]_{\varphi,\sigma} := (\mathtt{fix}(\lambda\bar{\wp}.([\![P_1]\!]_{\varphi[\bar{p}\leftarrow\bar{\wp}],\sigma}, \ldots [\![P_m]\!]_{\varphi[\bar{p}\leftarrow\bar{\wp}],\sigma})))_i$
  where $\mathtt{fix}$ *denotes the least fixed point and* $(\cdot)_i$ *stands for the projection to the ith component.*
- $[\![(\bar{x}.G)]\!]_{\varphi,\sigma} = \lambda\bar{t}.[\![G]\!]_{\varphi,\sigma}(\{\bar{t}/\bar{x}\})$ *where* $\bar{x} \notin \mathrm{DOM}(\sigma) \cup \mathrm{RG}(\sigma)$
  $[\![(\bar{x} \mid G)]\!]_{\varphi,\sigma} = \lambda\bar{t}.[\![G]\!]_{\varphi,\{\bar{t}/\bar{x}\}}(id_{\mathrm{LPV}(\bar{t})})$ *where* $\bar{x} \notin \mathrm{DOM}(\sigma) \cup \mathrm{RG}(\sigma)$
- $[\![\mathtt{let}\ \bar{x} = \bar{t}\ \mathtt{in}\ G]\!]_{\varphi,\sigma} = [\![G]\!]_{\varphi,\{\bar{t}/\bar{x}\}\circ\sigma}$ *where* $\bar{x} \notin \mathrm{DOM}(\sigma) \cup \mathrm{RG}(\sigma)$

For every term M we define $[\![M]\!]_\varphi := [\![M]\!]_{\varphi,\emptyset}$, and $[\![M]\!] := [\![M]\!]_\varphi$ if M contains no predicate name free. Furthermore, $[\![\![G]\!]\!]_\varphi := [\![G]\!]_\varphi(id_{\mathrm{LPV}(G)})$ and $[\![\![G]\!]\!] := [\![G]\!](id_{\mathrm{LPV}(G)})$ for goals G.

As the dependency on φ is continuous — remember that the operations on $\mathbb{G}$ used here are continuous on every $\mathbb{G}(d)$ where $d \subseteq \mathrm{LPV}$ is finite — the semantic function is well-defined. If G is a goal, then evaluation of G is the task of determining the stream $[\![\![G]\!]\!]$. Its elements are the answers for G.

Example 1. $G :\equiv (y.y = c)(x) * (y.y = d)(x)$ Then:
$$\begin{aligned}[\![G]\!](\{x/x\}) &= [\![(y.y = c)(x)]\!](\{x/x\}) * [\![(y.y = d)(x)]\!](\{x/x\}) \\ &= [\![(y.y = c)]\!](x) * [\![(y.y = d)]\!](x) \\ &= [\![y = c]\!](\{x/y\}) * [\![y = d]\!](\{x/y\}) = [\{c/x\}^\sim, \{d/x\}^\sim]\end{aligned}$$

Example 2. Let $G :\equiv (x \mid (y.y = x)(c))(u)$. Then:
$$\begin{aligned}[\![G]\!](\{d/u\}) &= [\![(y.y = x)(c)]\!]_{\varphi,\{d/x\}}(\emptyset) = [\![(y.y = x)]\!]_{\varphi,\{d/x\}}(c) \\ &= [\![y = x]\!]_{\varphi,\{d/x\}}(\{c/y\}) = m_{y;d}(\{c/y\}) = [\,]\end{aligned}$$

Lemma 1. *Let G be a formula in PL-(2 abs), $\theta \in$ FS, $\bar{x} = x_1, \ldots, x_n$ pairwise distinct variables, and $\bar{t} = t_1, \ldots, t_n$ constructor terms.*

1. $[\![G]\!]_{\varphi,\theta} = [\![G\theta]\!]_\varphi$
2. $[\![\,\texttt{let}\ \bar{x} = \bar{t}\ \texttt{in}\ G]\!]_\varphi = [\![G\{\bar{t}/\bar{x}\}]\!]_\varphi$
3. $[\![(\bar{x}.G)(\bar{t})]\!]_\varphi(\theta) = [\![G]\!]_\varphi(\{\bar{t}\theta/\bar{x}\})$ *if* $\text{DOM}(\theta) \supseteq \text{LPV}(\bar{t})$
4. $[\![(\bar{x} \mid G)(\bar{t})]\!]_\varphi(\theta) = [\![G\{\bar{t}\theta/\bar{x}\}]\!]_\varphi(id_{\text{LPV}(\bar{t}\theta)})$ *if* $\text{DOM}(\theta) \supseteq \text{LPV}(\bar{t})$

Proof. The first equation can be shown by a straightforward induction on G. The remaining equations are immediate from the definition and the first fact.

3 Predicate abstraction in PL-(2 abs)

First we show that the predicate abstractions are different from each other and different from functional abstraction.

Lemma 2. *There are constructor terms t, s and a goal G in PL-(2 abs) so that*

1. $[\![\texttt{let}\, x = t\, \texttt{in}\, G]\!] \neq [\![(x \mid G)(t)]\!] \neq [\![(x.G)(t)]\!]$
2. $[\![\texttt{let}\, x = s\, \texttt{in}\, G]\!] \neq [\![(x \mid G)(s)]\!] \neq [\![(x.G)(s)]\!]$

Proof. Let $G :\equiv (y.y = x)(c)$, $t :\equiv u$, $s :\equiv d$. Then, using Lemma 1, we obtain:

$[\![(x \mid G)(t)]\!](\{d/u\}) = [\,]$ (see Example 2)
$[\![\texttt{let}\, x = t\, \texttt{in}\, G]\!](\{d/u\}) = [\![(y.y = u)(c)]\!](\{d/u\}) = [\![y = u]\!](\{c/y\}) = [\emptyset^\sim]$
$[\![(x.G)(t)]\!](\{d/u\}) = [\![(y.y = x)(c)]\!](\{d/x\}) = [\![y = x]\!](\{c/y\}) = [\emptyset^\sim]$
$[\![(x.G)(s)]\!](\emptyset) = [\![(y.y = x)(c)]\!](\{d/x\}) = [\![y = x]\!](\{c/y\}) = [\emptyset^\sim]$
$[\![\texttt{let}\, x = s\, \texttt{in}\, G]\!](\emptyset) = [\![(y.y = d)(c)]\!](\emptyset) = [\![y = d]\!](\{c/y\}) = [\,]$
$[\![(x \mid G)(d)]\!](\emptyset) = [\![(y.y = d)(c)]\!](\emptyset) = [\![y = d]\!](\{c/y\}) = [\,]$

This implies that the equation $[\![(\bar{x} \mid G)(\bar{t})]\!]_\varphi \stackrel{?}{=} [\![G\{\bar{t}/\bar{x}\}]\!]_\varphi$ does not hold, but note that we have:

$$[\![(\bar{x} \mid G)(\bar{t})]\!]_\varphi = [\![G\{\bar{t}/\bar{x}\}]\!]_\varphi$$

Furthermore, Lemma 2 implies that $[\![G]\!]_\varphi(\{\bar{t}/\bar{x}\} \cdot \theta) \stackrel{?}{=} [\![G\{\bar{t}/\bar{x}\}]\!]_\varphi(\theta)$ does not hold. However, we can infer a similar property if we replace $\{\cdot/\cdot\}$ by a restricted substitution.

Definition 4. *Let G be a PL-(2 abs)-formula, $\bar{t}$ consist of n constructor terms and $\bar{x}$ of n pairwise distinct variables. Then $G[\bar{t}/\bar{x}]$ is the result of substituting simultaneously every free occurrence of x_i which is not in a predicate term by t_i.*

Lemma 3. *Let G be a PL-(2 abs)-formula, $\theta \in \text{FS}$, $\bar{t}$ constructor terms and $\bar{x}$ pairwise distinct variables so that* $\text{LPV}(G) \subseteq \{\bar{x}\}$ *and* $\text{LPV}(\bar{t}) \subseteq \text{DOM}(\theta)$. *Then:*

$$[\![G]\!]_\varphi(\{\bar{t}/\bar{x}\} \cdot \theta) = [\![G[\bar{t}/\bar{x}]]\!]_\varphi(\theta)$$

Proof. Observe first that $[\![P(\bar{s})]\!]_\varphi(\{\bar{t}/\bar{x}\} \cdot \theta) = [\![P]\!](\bar{s}\{\bar{t}/\bar{x}\}\theta) = [\![P(\bar{s}\{\bar{t}/\bar{x}\})]\!](\theta) = [\![(P(\bar{s}))[\bar{t}/\bar{x}]]\!](\theta)$ if $\text{LPV}(P(\bar{s})) \subseteq \{\bar{x}\}$. Now the claim can be proved by a straightforward induction on G.

Combining this with Lemma 1 we obtain:

Corollary 1. $[\![(\bar{x}.G)(\bar{t})]\!]_\varphi(\theta) = [\![G[\bar{t}/\bar{x}]]\!]_\varphi(\theta)$ *if* $\text{LPV}(G) \subseteq \{\bar{x}\}$, $\text{LPV}(\bar{t}) \subseteq \text{DOM}(\theta)$.

As a consequence we obtain $[\![(\bar{x}.G)(\bar{t})]\!] = [\![(\mathsf{E}\bar{y}\, G)[\bar{t}/\bar{x}]]\!]$ where $\bar{y}$ is a listing of $\text{LPV}(G) \setminus \{\bar{x}\}$. For every predicate abstraction, we have now shown a property which can be turned into a simplification rule as part of an operational semantics. Note, however, that the context in which these rules can be applied has to be restricted in a sound system.

Our next aim is to show that some meta-logical predicates can be defined in PL-(2 abs). To this end we consider the predicates $\text{var} \in \mathbb{P}_1$ and $\text{copy} \in \mathbb{P}_2$:

$$\text{var}(t) = \begin{cases} [\emptyset^\sim] & \text{if } t \text{ is a variable} \\ [\,] & \text{otherwise} \end{cases}$$

$$\text{copy}(t,s) = \overline{mgu}(s; t\delta) \downarrow \text{LPV}(s)$$
where δ is any finite renaming so that
$\text{DOM}(\delta) = \text{LPV}(t)$ and $\text{RG}(\delta) \cap \text{LPV}(s) = \emptyset$

The first predicate "var" is a test for the property "variable". Evaluating $\text{copy}(t,s)$ is achieved by unifying s with a fresh variant of t and returning the bindings for variables in s only. If s is a variable, a fresh copy of t is returned.

Lemma 4. *The predicates* var *and* copy *are definable in PL-(2 abs).*

Proof. Let $P :\equiv (x, y \mid (z.z = x)(y))$ and $Q :\equiv (x \mid (.x = c)(\,) \otimes (.x = d)(\,))$. For arbitrary constructor terms t, s we have:

$$\begin{aligned}
[\![P]\!](t,s) &= [\![(z'.z' = t)(s)]\!](\text{id}_{\text{LPV}(t,s)}) && \text{where } z' \text{ is fresh} \\
&= [\![z' = t]\!](\{s/z'\}) \\
&= (m_{z';t}(\{s/z'\} \cup \delta)) \downarrow \text{LPV}(s) && \text{where } \delta \text{ is a finite renaming so that} \\
& && \text{DOM}(\delta) = \text{LPV}(t), \text{RG}(\delta) \cap \text{LPV}(s) = \emptyset \\
&= \overline{mgu}(s; t\delta) \downarrow \text{LPV}(s) = \text{copy}(t,s) \\
[\![Q]\!](t) &= [\![(.t = c)(\,) \otimes (.t = d)(\,)]\!](\text{id}_{\text{LPV}(t)}) \\
&= \left\{ \begin{array}{ll} [\emptyset^\sim] & \text{if } t \text{ is unifiable with } c \text{ and } d \\ [\,] & \text{otherwise} \end{array} \right\} = \text{var}(t)
\end{aligned}$$

4 The restricted versions

Now we study the restricted systems PL-(2 abs)-r and PL-(abs)-r.

Lemma 5. *Let G be a formula in PL-(2 abs)-r with* $\text{LPV}(G) \subseteq \{\bar{x}\}$ *where $\bar{x}$ are n pairwise distinct variables, and let $\bar{t}$ be n constructor terms.*

1. *If θ is a finite substitution so that* $\text{LPV}(\bar{t}) \subseteq \text{DOM}(\theta)$ *then*
$$[\![G]\!]_\varphi(\{\bar{t}/\bar{x}\} \cdot \theta) = [\![G\{\bar{t}/\bar{x}\}]\!]_\varphi(\theta)$$
2. $[\![(\bar{x}.G)(\bar{t})]\!]_\varphi = [\![\mathit{let}\ \bar{x} = \bar{t}\ \mathit{in}\ G]\!]_\varphi = [\![(\bar{x} \mid G)(\bar{t})]\!]_\varphi$

Proof. The first fact is a consequence of Lemma 3, as predicate terms in PL-(2 abs)-r do not contain free LP-variables. Combining this with Lemma 1, we obtain the second part.

Theorem 1. *(Equivalence of PL-(2 abs)-r and PL-(abs)-r) For every formula or predicate term M in PL-(2 abs)-r there is a formula or predicate term M' in PL-(abs)-r respectively so that* $\textsc{lpv}(M') = \textsc{lpv}(M)$ *and* $[\![M]\!]_{\varphi,\theta} = [\![M']\!]_{\varphi,\theta}$ *for all environments φ and finite substitutions θ.*

Proof. Using Lemma 5, we can obtain the term M' by replacing every occurrence of $(\bar{x} \mid G)$ with the corresponding $(\bar{x}.G)$.

As the two versions of predicate abstraction are semantically equivalent in the restricted system, the system PL-(2 abs) can not be equivalent to PL-(2 abs)-r. Here we use the strict notion of equivalence as stated in the last theorem for the two restricted systems. This argument applies to every extension of PL which satisfies $[\![G]\!]_\varphi(\{\bar{t}/\bar{x}\} \cdot \theta) = [\![G[\bar{t}/\bar{x}]]\!]_\varphi(\theta)$ if $\textsc{lpv}(G) \subseteq \{\bar{x}\}$ and $\textsc{lpv}(\bar{t}) \subseteq \textsc{dom}(\theta)$, including an extension by a not-operator or by the var- and the copy-predicate described above: the corresponding system with unrestricted abstraction is not equivalent to restricted the version. However, this implies only that there is a non-ground term without a denotationally equivalent in the restricted system. In the last section we defined some meta-logical predicates by predicate terms *without* free variables. Given the fact that these predicates are not definable in pure Prolog, PL-(2 abs) is expected to be more expressive than PL-(2 abs)-r also w.r.t. definability by predicate terms without free variables. In the sequel we present a proof of this fact which makes no assumption concerning the relation to pure Prolog.

Theorem 2. *The system PL-(2 abs) is more expressive than PL-(2 abs)-r w.r.t. definability by predicate terms without free* LP*-variables.*

Proof. In the last section we saw that the predicates var and copy are definable in PL-(2 abs). It remains to show that this is not the case for PL-(abs). To this end we consider PL-(abs)-formulas of the form $\mathsf{E}\bar{u}^{(1)}(\bar{x}^{(1)} = \bar{t}^{(1)}) * \ldots * \mathsf{E}\bar{u}^{(n)}(\bar{x}^{(1)} = \bar{t}^{(n)})$ or $\mathsf{E}\bar{u}^{(1)}(\bar{x}^{(1)} = \bar{t}^{(1)}) * \ldots * \mathsf{E}\bar{u}^{(n)}(\bar{x}^{(1)} = \bar{t}^{(n)}) * \mathbf{0}$ where the lists $\bar{x}^{(i)}$ consist of pairwise distinct variables that do not occur in $\bar{u}^{(i)}$ and all variables in $\bar{t}^{(i)}$ are in $\bar{u}^{(i)}$. These formulas are called "solved", as the corresponding stream of answers $[\{\bar{t}^{(1)}/\bar{x}^{(1)}\}^\sim, \ldots, \{\bar{t}^{(n)}/\bar{x}^{(n)}\}^\sim]$ or $[\{\bar{t}^{(1)}/\bar{x}^{(1)}\}^\sim, \ldots, \{\bar{t}^{(n)}/\bar{x}^{(n)}\}^\sim, \bot]$ respectively is immediate. Now we observe:

> If G is a PL-(abs)-formula that contains no `rec` and no predicate variables, then there is a solved PL-(abs)-formula S so that $\textsc{lpv}(S) \subseteq \textsc{lpv}(G)$ and $[\![G]\!]_{\varphi,\theta} = [\![S]\!]_{\varphi,\theta}$ for all φ, θ

This can be proved by induction on G.

As a next step we consider, for every formula G, the approximating formulas, which are obtained by applying some unfolding steps to `rec`-expressions and replacing then every occurrence of the form $(\mathtt{rec}\,\bar{p}.(P_1, \ldots, P_n))_i(\bar{t})$ by $\mathbf{0}$. Using the fact that all operations that interpret constructions for building terms are monotone and continuous, we can show that the value $[\![G]\!]_\varphi$ is the least upper bound of the set of values $[\![G']\!]_\varphi$ of approximating formulas G'.

Now let G be a goal in PL-(abs) so that $[\emptyset^\sim] \sqsubseteq [\![G]\!]$. Then there is some approximating formula G' for G so that $[\emptyset^\sim] \sqsubseteq [\![G']\!]$. As G' contains no occurrence of `rec`, there is a solved goal S satisfying $[\![S]\!] = [\![G']\!]$, in particular $[\emptyset^\sim] \sqsubseteq [\![G']\!] = [\![S]\!]$. As S is solved, it must be semantically equivalent to $\mathbf{1}$, which in turn implies $[\![\mathbf{1}]\!] = [\![S]\!] = [\![G']\!] \sqsubseteq [\![G]\!]$. As a consequence, $[\emptyset^\sim] = [\![\mathbf{1}]\!](\theta) \sqsubseteq [\![G]\!](\theta)$ for every $\theta \in \text{FS}$.

Now we assume that there is a PL-(abs)-term P without free predicate variables so that $[\![P]\!] = \mathit{var}$. Then $[\emptyset^\sim] \sqsubseteq [\![P(x)]\!]$. As shown above, this implies $[\emptyset^\sim] \sqsubseteq [\![P(x)]\!](\{c/x\}) = [\![P]\!](c) = [\,]$, contradicting our assumption. A similar argument applies to the predicate copy.

5 Conclusion

Guided by semantical considerations, we introduced a predicational term calculus with two predicate abstractions, and presented a denotational semantics. The meaning of predicate abstractions is explained without refering to operational details.

Compared to a term calculus which corresponds to pure Prolog, the new system is more expressive. That strengthening is exemplified by defining some meta-logical predicates in the full system. However, the effect of generalising predicate abstraction is not equivalent to adding these predicates. Furthermore, generalised predicate abstraction differs from functional abstraction.

Remarkably, the new system is not obtained by adding new built-in-predicates or imperative elements but essentially by removing a restriction on variables in predicate formation.

References

1. M. Baudinet. Proving termination properties of Prolog programs: A semantic approach. *Journal of Logic Programming*, 14(1 and 2):1–29, October 1992.
2. K. L. Clark. Negation as failure. In J. M. H. Gallaire, editor, *Logic and Databases*, pages 293–322. Plenum Press, New York, 1978.
3. S. K. Debray and P. Mishra. Denotational and operational semantics for Prolog. *Journal of Logic Programming*, 5(1):61–91, March 1988.
4. B. Elbl. A declarative semantics for depth-first logic programs. *Journal of Logic Programming*, 41(1):27–66, 1999.
5. B. Elbl. A non-definability result for a predicational language with the usual control. *International Journal of Foundations of Computer Science*, 12(3):385–396, 2001.
6. N. D. Jones and A. Mycroft. Stepwise development of operational and denotational semantics for Prolog. In *Proceedings of the 1984 International Symposium on Logic Programming*, pages 281 – 288. IEEE, 1984.
7. J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

Brownian Motion and Kolmogorov Complexity

Willem L. Fouché

Department of Decision Sciences,
University of South Africa, PO Box 392, 0003 Pretoria, South Africa
fouchwl@unisa.ac.za

Abstract. We give a survey of the properties of Brownian motions which are represented by infinite binary strings which are random in the sense of Kolmogorov-Chaitin. We describe how the sensitive dynamics of such a Brownian motion depends on the recursive properties of the real where the dynamics is considered.
2000 Mathematics Subject Classification:68Q30 03D80, 60J65

1 Introduction

The results of this paper are based on [5, 6], where it was shown that each binary string α which is complex (random) in the sense of Kolmogorov-Chaitin (a KC-string) can be algorithmically transformed into a "generic" Brownian motion x_α. It is generic in the sense that every probabilistic event which holds almost surely with respect to the Wiener measure, is reflected in x_α, provided the probabilistic event has a suitably effective description. The class of generic Brownian motions coincides with a class $\mathcal{C}$ of functions which were introduced by Asarin and Prokovskiy in [1]. Each function in the latter class is a uniform limit of a sequence (x_n) of piecewise linear functions; moreover, every x_n can be encoded by a binary string s_n of length n such that, for some positive constant d, the Kolmogorov complexity of s_n is at least $n - d$, for all large values of n. For this reason we referred in [5, 6] to the elements of $\mathcal{C}$ as *complex oscillations*.

The complex oscillations have interesting recursion-theoretic properties. For example, it is shown in [5] that, if x is a complex oscillsation and r is a nonzero recursive real number in the unit interval, then $x(r)$ will not be a real number.

In [6] we showed that, for each $x \in \mathcal{C}$, one can compute from the values of x at the rational numbers a unique KC-string α such that $x = x_\alpha$. In this way one can identify interesting implicit structure in a generic Brownian motion. For example, the codes of many countable homogeneous relational structures can be computed from the values of a generic Brownian motion at the rationals in the unit interval. Recall that a relational structure X is homogeneous if any isomorphism $f : A \to B$ between finite substructures of X can be extended to an automorphism of X. The universal procedure which computes from the values of a complex oscillation x the KC-string α such that $x = x_\alpha$, also yields a code of a very interesting homogeneous structure, the so-called Rado graph [18]. Indeed, if α is a KC-string and $e_1, e_2, \ldots$ is a recursive enumeration, without repetition,

of the 2-element subsets of ω, let $R_\alpha = (\omega, E_\alpha)$ be the graph defined by:

$$e_i \in E_\alpha \leftrightarrow \alpha_i = 1.$$

Then the graph R_α is isomorphic to Rado's graph [4]. In this sense one could say that a Rado graph is "enfolded" in every complex oscillation.

In this paper, we take a closer look at the reverse process, namely the unfolding of KC-strings, not only to a generic Brownian motion as in [6], but also to the dynamical aspects of Brownian motion, as reflected in every complex oscillation. Our focus will be on the structure of the so-called rapid points of a complex oscillation. The fractal geometry of a complex oscillation will be studied in a sequel [7] to this paper.

Call a point $t \in (0,1)$ a *rapid point* of a continuous function X on the unit interval when

$$\overline{\lim_{h \to 0}} \frac{|X(t+h) - X(t)|}{\sqrt{|h| \log(1/|h|)}} > 0.$$

Denote the set of rapid points of X by $R(X)$. It was shown in [17] that Brownian motion has almost surely a set of rapid points of Hausdorff dimension 1. When X is one-dimensional Brownian motion, the set $R(X)$ has an extremely interesting structure. For example, Kaufmann [11] showed that, almost surely, $R(X)$ contains, for each $0 < \beta < 1$, a Salem set of Hausdorff dimension β. (Recall that a compact subset E of $\mathbf{R}^d$ of Hausdorff dimension $\beta > 0$ is said to be a Salem set, if β is the supremum of the reals $0 \leq \alpha < d$ for which there is some positive nonzero Radon measure μ with support contained in E, such that the Fourier transform $\hat{\mu}$ of μ satisfies $|\hat{\mu}(\xi)|^2 \ll |\xi|^{-\alpha}$, for all large values of $|\xi|$. In this case, E will generate $\mathbf{R}^d$ as an abelian group!)

The rapid points of a complex oscillation have a specific recursive structure. If x is a complex oscillation, then x has a dense set of rapid points. If x is the complex oscillation x_α associated with the KC-string α, a dense set of rapid points can be effectively retrieved from α. Indeed, there is a universal algorithmic procedure which, upon having access to an oracle for a KC-string α, will yield, for any closed dyadic interval I, a sequence (t_k) of rational numbers in I such that $|t_{k+1} - t_k| < 2^{-k}$ for all $k \geq 1$ and ,moreover, such that the limit t of the sequence (t_k) is a rapid point of the complex oscillation x_α associated with α. Furthermore, each rapid point of a complex oscillation is *not* a recursive real number. In fact, if $t \in (0,1)$ is a recursive real number, then t is an "ordinary" point of x. This means that Khintchine's law of the iterated logarithm [14] is reflected in x at every recursive t, i.e., if t is recursive, then

$$\overline{\lim_{h \to 0}} \frac{|x(t+h) - x(t)|}{\sqrt{2|h| \log\log(1/|h|)}} = 1.$$

In developing the analytical arguments which underly the proof of this result, the author benefited greatly from reading the papers [13, 12] of Kahane. The details of the proof will appear in [7].

2 Generic Brownian motions

The set of non-negative integers is denoted by ω and we write $\mathcal{N}$ for the product space $\{0,1\}^\omega$. The set of words over the alphabet $\{0,1\}$ is denoted by $\{0,1\}^*$. If $a \in \{0,1\}^*$, we write $|a|$ for the length of a. If $\alpha = \alpha_0\alpha_1\ldots$ is in $\mathcal{N}$, we write $\overline{\alpha}(n)$ for the word $\prod_{j<n}\alpha_j$. We use the usual recursion-theoretic terminology Σ^0_r and Π^0_r for the arithmetical subsets of $\omega^k \times \mathcal{N}^l,\ k,l \geq 0$. (See, for example [10] for the recursion-theoretic background.) We follow [10] by referring to a Σ^0_1 set of reals as a *semi-recursive* set instead of a recursively enumerable set. We write λ for the Lebesgue probability measure on $\mathcal{N}$. For a binary word s of length n, say, we write $[s]$ for the "interval" $\{\alpha \in \mathcal{N} : \overline{\alpha}(n) = s\}$. A sequence (a_n) of real numbers converges *effectively* to 0 as $n \to \infty$ if for some total recursive $f : \omega \to \omega$, it is the case that $|a_n| \leq (m+1)^{-1}$ when $n \geq f(m)$. A subset A of $\mathcal{N}$ is of *constructive* measure 0, if there is a total recursive $\phi : \omega^2 \to \{0,1\}^*$ such that $A \subset \cap_n \cup_m [\phi(n,m)]$, where $\lambda(\cup_m[\phi(n,m)])$ converges effectively to 0 as $n \to \infty$.

For any binary word a we denote its Kolmogorov complexity by $K(a)$. One can think of $K(a)$ as the shortest self-delimiting program for a universal Turing machine U which will output a from an empty input. We assume that U accepts self-delimiting programs only. It is well-known that if K_1, K_2 corresponds to universal Turing machines U_1, U_2, then $K_1(a) = K_2(a) + O(1)$ for all a. (See, for example, [21] for a discussion.) In the sequel, we shall regard our choice of K as fixed. An infinite binary string α is Kolmogorov-Chaitin complex if

$$\exists_d \forall_n\ K(\overline{\alpha}(n)) \geq n - d.$$

In the sequel, we shall denote this set by KC and refer to its elements as KC-strings.

A Brownian motion on the unit interval is a real-valued function $(\omega, t) \mapsto X_\omega(t)$ on $\Omega \times [0,1]$, where Ω is the underlying space of some probability space, such that $X_\omega(0) = 0$ a.s. and for $t_1 < \ldots < t_n$ in the unit interval, the random variables $X_\omega(t_1), X_\omega(t_2) - X_\omega(t_1), \cdots, X_\omega(t_n) - X_\omega(t_{n-1})$ are statistically independent and normally distributed with means all 0 and variances $t_1, t_2 - t_1, \cdots, t_n - t_{n-1}$, respectively. We say in this case that the Brownian motion is parametrised by Ω. Alternatively, the map X defines a Brownian motion iff for $t_1 < \ldots < t_n$ in the unit interval, the random vector $(X_\omega(t_1), \cdots, X_\omega(t_n))$ is Gaussian with correlation matrix $(\min(t_i, t_j) : 1 \leq i, j \leq n)$.

It is a fundamental fact that any Brownian motion has a "continuous version". This means the following: Write Σ for the σ-algebra of Borel sets of $C[0,1]$ where the latter is topologised by the uniform norm topology. There is a probability measure W on Σ such that for $0 \leq t_1 < \ldots < t_n \leq 1$ and for a Borel subset B of $\mathbf{R}^n$, we have

$$P(\{\omega \in \Omega : (X_\omega(t_1), \cdots, X_\omega(t_n)) \in B\}) = W(A),$$

where

$$A = \{x \in C[0,1] : (x(t_1), \cdots, x(t_n)) \in B\}).$$

(See, for example, pp46-50 of [9] or [8].) The measure W is known as the *Wiener measure*. We shall usually write $X(t)$ instead of $X_\omega(t)$.

We give a brief survey of the results from [1], [5] and [6]. The key idea in these papers is that of a so-called complex oscillation, which is a limit of a sequence of finitary random walks of growing Kolmogorov complexity, which is in a definite sense also a generic Brownian motion. We first introduce some notation. For $n \geq 1$, we write C_n for the class of continuous functions on the unit interval that vanish at 0 and are linear with slopes $\pm\sqrt{n}$ on the intervals $[(i-1)/n, i/n]$, $i = 1, \cdots, n$. With every $x \in C_n$, one can associate a binary string of length n by setting $a_i = 1$ or $a_i = 0$ according to whether x increases or decreases on the interval $[(i-1)/n, i/n]$. We call the sequence $a_1 \cdots a_n$ the code of x and denote it by $c(x)$. The following notion was introduced by Asarin and Prokovskiy in [1].

Definition 1. *A sequence (x_n) in $C[0,1]$ is complex if $x_n \in C_n$ for each n and there is a constant $d > 0$ such that $K(c(x_n)) \geq n - d$ for all n. A function $x \in C[0,1]$ is a complex oscillation if there is a complex sequence (x_n) such that $\|x - x_n\|$ converges effectively to 0 as $n \to \infty$.*

The class of complex oscillations is denoted by $\mathcal{C}$. It was shown by Asarin and Prokovskiy [1] that the class $\mathcal{C}$ has Wiener measure 1.

For the results in this paper, we shall require a recursive characterisation of the almost sure events, with respect to Wiener measure, which are reflected in each complex oscillation. In order to describe this characterisation, we use, as in [5], an analogue of a Π^0_2 subset of $C[0,1]$ which is of constructive measure 0. We introduce some notation. If F is a subset of $C[0,1]$, we denote by $\overline{F}$ the topological closure of F in $C[0,1]$. For $\epsilon > 0$, we let $O_\epsilon(F)$ be the set $\{f \in C[0,1] : \exists_{g \in F} \|f - g\| < \epsilon\}$. For convenience sake, we write F^0 for the complement of F and F^1 for F.

Definition 2. *A sequence $\mathcal{F}_0 = (F_i : i < \omega)$ in Σ is an effective generating sequence if*

1. *for $F \in \mathcal{F}_0$, for $\epsilon > 0$ and $\delta \in \{0,1\}$, we have, for $G = O_\epsilon(F^\delta)$ or for $G = F^\delta$, that $W(\overline{G}) = W(G)$,*
2. *there is an effective procedure that yields, for each sequence $0 \leq i_1 < \ldots < i_n < \omega$ and $k < \omega$ a binary rational number β_k such that*

$$|W(F_{i_1} \cap \ldots \cap F_{i_n}) - \beta_k| < 2^{-k},$$

3. *for $n, i < \omega$, a strictly positive rational number ϵ and for $x \in C_n$, both the relations $x \in O_\epsilon(F_i)$ and $x \in O_\epsilon(F_i^0)$ are recursive in x, ϵ, i and n.*

If $\mathcal{F}_0 = (F_i : i < \omega)$ is an effective generating sequence and $\mathcal{F}$ is the algebra generated by $\mathcal{F}_0$, then there is an enumeration $(T_i : i < \omega)$ of the elements of $\mathcal{F}$ (with possible repetition) in such a way, for a given i, one can effectively describe T_i as a finite union of sets of the form

$$F = F_{i_1}^{\delta_1} \cap \ldots \cap F_{i_n}^{\delta_n}$$

where $0 \leq i_1 < \ldots < i_n$ and $\delta_i \in \{0,1\}$ for each $i \leq n$. We call any such sequence $(T_i : i < \omega)$ a *recursive enumeration* of $\mathcal{F}$. We say in this case that $\mathcal{F}$ is *effectively generated* by $\mathcal{F}_0$ and refer to $\mathcal{F}$ as an *effectively generated algebra* of sets. A sequence (A_n) of sets in $\mathcal{F}$ is said to be $\mathcal{F}$*-semi-recursive* if it is of the form $(T_{\phi(n)})$ for some total recursive function $\phi : \omega \to \omega$ and some effective enumeration (T_i) of $\mathcal{F}$. (Note that the sequence (A_n^c), where A_n^c is the complement of A_n, is also an $\mathcal{F}$-semirecursive sequence.) In this case, we call the union $\cup_n A_n$ a $\Sigma_1^0(\mathcal{F})$ set. A set is a $\Pi_1^0(\mathcal{F})$ set if it is the complement of a $\Sigma_1^0(\mathcal{F})$ set. It is of the form $\cap_n A_n$ for some $\mathcal{F}$-semirecursive sequence (A_n). A sequence (B_n) in $\mathcal{F}$ is a *uniform* sequence of $\Sigma_1^0(\mathcal{F})$ sets if, for some total recursive function $\phi : \omega^2 \to \omega$ and some effective enumeration (T_i) of $\mathcal{F}$, each B_n is of the form

$$B_n = \bigcup_m T_{\phi(n,m)}.$$

In this case, we call the intersection $\cap_n B_n$ a $\Pi_2^0(\mathcal{F})$ set. If, moreover, the W-measure of B_n converges *effectively* to 0 as $n \to \infty$, we say that the set given by $\cap_n B_n$ is a $\Pi_2^0(\mathcal{F})$ set of constructive measure 0.

The following theorem explains in what sense a complex oscillation is a generic Brownian motion. The proof of this theorem appears in [5].

Theorem 1. *Let $\mathcal{F}$ be an effectively generated algebra of sets. If x is a complex oscillation, then x is in the complement of every $\Pi_2^0(\mathcal{F})$ set of constructive measure* 0.

The following theorem of [5] is a very useful practical tool for reflecting almost sure properties of Brownian motion in every complex oscillation. This result is repeatedly used in [6, 7].

Theorem 2. *If B is a $\Sigma_1^0(\mathcal{F})$ set and $W(B) = 1$, then $\mathcal{C}$, the set of complex oscillations, is contained in B.*

It follows that if a $\Pi_1^0(\mathcal{F})$ set A contains at least one complex oscillation, then $W(A) > 0$. For otherwise, the complement of A is a $\Sigma_1^0(\mathcal{F})$ set of measure 1 which fails to contain all the complex oscillations. An analogue for KC-strings of Theorem 2 appears in [4]. The following is an effective version of the Borel-Cantelli lemma restricted to Wiener processes.

Theorem 3. *If (A_k) is a uniform sequence of $\Sigma_1^0(\mathcal{F})$ sets with $\sum_k W(A_k) < \infty$, then, for each complex oscillation x, it is the case that $x \notin A_k$ for all large values of k.*

An analogue for KC-strings of this theorem appears in [19].

3 Rapid points of generic Brownian motion

The proofs of the theorems in this section will appear in a forthcoming paper [7]. Let x be a continuous function on the unit interval. We call $t \in (0,1)$ an

ordinary point of x when

$$\overline{\lim_{h\to 0}} \frac{|x(t+h)-x(t)|}{\sqrt{2|h|\log\log(1/|h|)}} = 1.$$

We call $t \in (0,1)$ a *rapid* point of x if

$$\overline{\lim_{h\to 0}} \frac{|x(t+h)-x(t)|}{\sqrt{|h|\log(1/|h|)}} > 0.$$

Theorem 4. *If x is a complex oscillation, then the set of rapid points of x is dense in the unit interval. Each recursive real in the unit interval is an ordinary point of x. Consequently, the rapid points of x are all non-recursive real numbers.*

We now discuss an effective version of Theorem 4. We first recall a few results from [6]. Let $g : (0,1) \to \mathbf{R}$ be the function defined by

$$\alpha = \int_{-\infty}^{g(\alpha)} \frac{e^{-t^2/2}}{\sqrt{2\pi}} dt, \quad \alpha \in (0,1).$$

Note that g is a recursive function, i.e., there is a uniform procedure that outputs $g(\alpha)$ up to arbitrary accuracy using only a finite number of bits of α. We fix a recursive bijection $<,>$ from ω^2 to ω. To any $\alpha \in \mathcal{N}$, we associate a sequence $B = (\beta_0, \beta_1, \beta_{jn} : j \geq 1, 0 \leq n < 2^j)$, where the sequence (β_{jn}) is lexicographically ordered with respect to the double indices jn, in such a way that the kth term of the sequence B is given by

$$\alpha_{k0}\alpha_{k1}\cdots.$$

Here, we have written kl instead of $< k, l >$. For $1 \leq j < \omega$, $0 \leq n < 2^j$, set $\xi_{jn} = g(\beta_{jn})$; in addition, set $\xi_k = g(\beta_k)$, for $k = 0, 1$. It follows that there is a uniform procedure that computes from $\alpha \in KC$, for each j, n, the number ξ_{jn} up to arbitrary accuracy. For $\alpha \in \mathcal{N}$ and $t \in [0,1]$ set

$$x_\alpha(t) = \xi_0 \Delta_0(t) + \xi_1 \Delta_1(t) + \sum_{j<\omega} \sum_{n<2^j} \xi_{jn} \Delta_{jn}(t).$$

It is shown in [6] that, if $\alpha \in KC$, then the series converges and that the function x_α is in fact a complex oscillation. Conversely, for every complex oscillation x, there is a unique KC-string α such that $x = x_\alpha$.

Theorem 5. *There is an oracle computation, that yields, upon having access to an oracle for a given $\alpha \in KC$, for every dyadic interval I contained in the unit interval, a sequence (s_k) of rationals in I which converges effectively to a rapid point of the complex oscillation x_α.*

References

1. Asarin, E.A. and Prokovskiy, A.V.: Primeenenie kolmogorovskoi slozhnosti k anlizu dinamiki upravlemykh sistem, *Automatika i Telemekhanika* **1** (1986), 25-33.
2. Chaitin, G.J.: On the length of programs for computing binary sequences, *J. Assoc. Comput. Mach.* **13** (1966), 547-569.
3. Chaitin, G.A.: *Algorithmic information theory*, Cambridge University Press, 1987.
4. Fouché, W.L.: Descriptive complexity and reflective properties of combinatorial configurations, *J. Lond. Math. Soc.* **54** (1996), 199-208.
5. Fouché, W.L.: Arithmetical representations of Brownian motion I, *J. Symb. Logic* **65** (2000), 421-442.
6. Fouché, W.L. : The descriptive complexity of Brownian motion, *Advances in Mathematics* **155** (2000), 3177-343.
7. Fouché, W.L. : Fractal geometry of Brownian motions of high Kolmogorov complexity, in preparation.
8. Freedman, D.: *Brownian motion and diffusion*, Holden-Day, 1971.
9. Hida, T.: *Brownian motion*, Springer-Verlag, New York, 1980.
10. Hinman, P.G.: *Recursion-theoretic hierarchies*, Springer-Verlag, New York,1978.
11. Kaufman, R.: Large increments of Brownian motion, *Nagoya Math. J.* **56** (1974), 139-145.
12. Kahane, J.-P.: Sur l'irrégularité locale du mouvement brownien, *C. r. hebd. séanc. Acad. Sci., Paris* **278** (1974), 331-333.
13. Kahane, J.-P.: *Some random series of functions* (second edition), Cambridge University Press, 1993.
14. Khinchine, A.Y.: *Asymptotische Gesetze der Wahrscheinlichkeitsrechnung*. Ergebn. Math. **2**, Berlin, 1933.
15. Kolmogorov, A.N.: Three approaches to the quantitative definition of randomness, *Probl. Inform. Transmission* **1** (1965), 1-7.
16. Martin-Löf, P.: The definition of random sequences, *Information and Control* **9** (1966), 602-619.
17. Orey, S. and Taylor, S.J.: How often on a Brownian path does the law of iterated logarithm fail?, *Proc. Lond. Math. Soc* **28** (1974), 174-192.
18. Rado, R.: Universal graphs and universal functions, *Acta Arith.* **9** (1964), 393-407.
19. Shen, A.Kh.: Connections between different algorithmic definitions of randomness, *Soviet Math. Dokl.* **38** (1989), 316-319.
20. van Lambalgen, M.: Von Mises' definition of random sequences reconsidered, *J. Symb. Logic* **52** (1987), 725-755.
21. Vitányi, P. and Li, M.: *An introduction to Kolmogorov complexity and its applications*, Springer-Verlag, New York, 1993.
22. Vov'k, V.G.: The law of the iterated logarithm for Kolmogorov random or chaotic sequences, *Theory Probal. Appl.* **32** (1987)), 413-425.

A Structure with P = NP

Christine Gaßner*

Institut für Mathematik und Informatik, Ernst-Moritz-Arndt-Universität,
F.-L.-Jahn-Straße 15 a, 17487 Greifswald, Germany
gassnerc@uni-greifswald.de

Abstract. Several NP-complete problems for the BSS model are known which correspond to classical NP-complete problems. By analogy with the BSS model one can define a Satisfiability Problem for each structure of finite signature. Here, we supplement a structure of strings by some new relation by means of which it is possible to decide a unary variant of the Satisfiability Problem with respect to the uniform model of computation in constant time. The corresponding Satisfiability Problem is decidable in polynomial time such that we obtain P = NP for the new structure. Thus, a solution of a problem posed by Bruno Poizat in his book "Les petits cailloux" is presented.

1 Introduction

The uniform model of computation over an arbitrary structure Σ is a generalization of the Blum-Shub-Smale (abbreviated BSS) model over the real numbers. The Σ-machines are the natural format for a structure Σ. We shall use the concepts explained in [8] in more details.

In this paper we construct some structure Σ_R over strings together with a relation R by means of which the corresponding Satisfiability Problem SAT_{Σ_R} is decidable with respect to the uniform model in polynomial time. The idea to construct a new structure Σ with $\mathrm{P}_\Sigma = \mathrm{NP}_\Sigma$ by introducing a new relation stems from Bruno Poizat. We define a unary variant of SAT_{Σ_R} and some R such that only the elements of this problem satisfy R. The operations of Σ_R allow us to encode the tuples by single elements and to pad the single elements, where padding means adding a fixed character many times. In this way we obtain some NP-complete satisfiability problem defined by formulae for which the quantifier domain is restricted to mainly small elements such that R can be defined essentially recursively on the padded elements. The recursive definition of R implies the decidability of SAT_{Σ_R} step by step. Moreover, the polynomial time reducibility of SAT_{Σ_R} to the unary variant of SAT_{Σ_R} follows from the definition of R since we can replace the parameters in the formulae by small elements

* I thank Gerald van den Boogaart, Petra Gummelt, Volkmar Liebscher, Mihai Prunescu, Rainer Schimming, and Michael Schürmann for discussions. Moreover, I also thank Günter Asser, Johann A. Makowsky, Kenneth Regan, and several referees for helpful hints, and I thank an anonymous referee for informing me about the forthcoming paper [15].

without changing the truth value of the formulae. The method to construct a relation on padded elements of an NP-complete problem was introduced in [5] in order to generate structures Σ with $\mathrm{P}_\Sigma = \mathrm{NP}_\Sigma$. In [5], [6], and [7] several structures Σ over trees or strings without identity relation and with identity relation were constructed by means of this method such that $\mathrm{P}_\Sigma = \mathrm{NP}_\Sigma$. The small elements are trees of small depth or short strings. For trees, padding means adding edges of one sort. Here, we present the proofs for the simplest structure of this kind.

The idea to define a relation on padded strings was also used in [10] and [15] in order to construct other structures Σ with $\mathrm{P}_\Sigma = \mathrm{NP}_\Sigma$. In [10] a new relation was formed from a known classical oracle $\mathcal{O}$ with $\mathrm{P}^{\mathcal{O}} = \mathrm{NP}^{\mathcal{O}}$ by encoding and padding the elements of $\mathcal{O}$. This construction works without knowing the elements of the oracle. The construction allows to shorten all inputs and guesses of the new machines working with the new relation in polynomial time such that these machines can be simulated by classical oracle machines in polynomial time. There, shortening also means replacing strings such that the equations and inequalities which determine the computation paths remain true. To this aim, for each element of the oracle, infinitely many padded versions satisfying R are given. In [15] a structure together with a relation was recursively defined by means of $\forall\exists$-formulae. The result is an infinite disjoint union of copies of a structure which is similar to the structure considered here. For the definition the existence of enough small strings is necessary since some parts of the $\forall\exists$-formulae contain a characterization which has to be also satisfied by small strings. The existence of enough small strings is guaranteed by the existence of an infinite number of copies of each string. The possible replacements are not given.

Our main goal is to give a construction and the proofs in detail for a general class of structures Σ with $\mathrm{P}_\Sigma = \mathrm{NP}_\Sigma$ over strings or trees. To this aim, we characterize all tuples of strings explicitly by logical terms with weighted variables, and we show the existence of enough small strings satisfying these characterizations by considering enough different small prefixes of strings. The structures are derived from a structure of trees which are used for the arrangement of data in computer science. The ideas of the method considered here go back to the investigations of computation paths and small guesses by Felipe Cucker, Pascal Koiran, M. Matamala, and Klaus Meer (see [3, 4, 11, 12]). The elements which satisfy the relation R correspond to the paths of some infinite tree which resembles the trees of the computation paths of machines which are traversed by the inputs step by step. For more details see [8].

2 The Structure Σ_R and Some Satisfiability Problems

We build Σ_R as follows.

Definition 1. *Let $\mathcal{A} = \{a, b\}^*$ be the set of strings over the alphabet $\{a, b\}$ where a and b are fixed symbols and let ε denote the empty string. The concatenation of two strings s and r is denoted by sr, and for any $r \in \mathcal{A}$ and any set $S \subseteq \mathcal{A}$, we write rS for $\{rs \mid s \in S\}$. For every $s \in \mathcal{A}$, let $|s|$ be the length of s. For*

$\mathcal{S} \subseteq \mathcal{A}$, let $\mathcal{S}^{(\leq k)} = \{r \in \mathcal{S} \mid |r| \leq k\}$, and the like. For every unary relation Rel, let Σ_{Rel} be the structure $(\mathcal{A}; \varepsilon; add_a, add_b, sub_a, sub_b; Rel, =)$ where the functions on strings $s \in \mathcal{A}$ are defined for the characters $x \in \{a, b\}$ by $add_x(s) = sx$, $sub_x(\varepsilon) = \varepsilon$, $sub_x(sx) = s$, $sub_a(sb) = sb$, and $sub_b(sa) = sa$.

We shall explicitly construct some relation R such that we obtain the following

Theorem 1. *There is some unary relation R such that $\mathrm{P}_{\Sigma_R} = \mathrm{NP}_{\Sigma_R}$ holds.*

To this aim let us define the Satisfiability Problem for formulae of first order logic with respect to Σ_{Rel}.

Definition 2. *Let $\mathcal{F}$ be the set of quantifier-free $(\neg, \wedge, \vee)$-formulae of first order logic corresponding to the structures Σ_{Rel} with literals of the form $Z = \varepsilon$, $Z_1 = Z_2$, $Z_1 \neq Z_2$, $Rel(Z)$, $\neg Rel(Z)$, $sub_x(Z_1) = Z_2$, and $add_x(Z_1) = Z_2$ where $x \in \{a, b\}$ and Z, Z_1, Z_2 stand for the variables $X_1, X_2, \ldots, Y_1, Y_2, \ldots$.*

Lemma 1. *There is an injective mapping of $\mathcal{F}$ into $\mathcal{A}^{(>0)}$ which can be computed step by step by translating the single characters $c \in \{\wedge, \vee, X, s, u, b, \ldots\}$ occurring in the formulae into strings $s_c \in \mathcal{A}^{(=n_0)}$, where n_0 is some positive integer, and by translating every index i into the string a^i.* □

Definition 3. *Let code be a mapping as in Lemma 1. For each formula ψ, let $code_{\mathrm{pad}}(\psi) = (code(\psi), \varepsilon, \ldots, \varepsilon) \in \mathcal{A}^{1+|code(\psi)|}$. For any positive integers l_0, l_1, let $\mathcal{F}_{l_0,l_1}$ be the set of all formulae $\psi \in \mathcal{F}$ for which $l_1 = |code(\psi)|$ holds and which only contain components of $\boldsymbol{X} = (X_1, \ldots, X_{l_0})$ and $\boldsymbol{Y} = (Y_1, \ldots, Y_{l_1})$ as variables. For every formula ψ containing only variables in $\{X_1, \ldots, X_{l_0}, Y_1, \ldots, Y_{l_1}\}$, we also write $\psi(\boldsymbol{X}, \boldsymbol{Y})$ instead of ψ.*

Note that we use the vector notation in a tuple $(\ldots, \boldsymbol{z}, \ldots)$ for a finite number of components $z_1, \ldots, z_l$ $(l > 0)$. We do not consider tuples of tuples of strings. Thus $(\boldsymbol{x}, \boldsymbol{y})$ stands for $(x_1, \ldots, x_{l_0}, y_1, \ldots, y_{l_1})$, and the like.

Definition 4. *Let the* Satisfiability Problem $\mathrm{SAT}_{\Sigma_{Rel}}$ *with respect to Σ_{Rel} be*

$$\mathrm{SAT}_{\Sigma_{Rel}} = \{(\boldsymbol{x}, code_{\mathrm{pad}}(\psi)) \mid (\exists l_0, l_1 \in \mathbb{N}^+)(\psi \in \mathcal{F}_{l_0,l_1} \;\&\; \boldsymbol{x} \in \mathcal{A}^{l_0} \;\&\; \Sigma_{Rel} \models \exists \boldsymbol{Y} \psi(\boldsymbol{x}, \boldsymbol{Y}))\} \ .$$

Clearly $\mathrm{SAT}_{\Sigma_{Rel}} \subseteq \mathcal{A}^\infty =_{\mathrm{def}} \bigcup_{n=1}^\infty \mathcal{A}^n$. In analogy with the classical Satifiability Problem we can show that $\mathrm{SAT}_{\Sigma_{Rel}}$ is NP-complete. We also define a unary variant of $\mathrm{SAT}_{\Sigma_{Rel}}$ by encoding tuples of strings in terms of $\langle \cdot, \ldots, \cdot \rangle$ and using a special form of padding defined by $(\cdot)_{dbl}$.

Definition 5. *For every string $s \in \mathcal{A}$, let the value $\langle s \rangle$ be recursively defined by $\langle \varepsilon \rangle = a$, $\langle ra \rangle = \langle r \rangle a^2$, and $\langle rb \rangle = \langle r \rangle ba$ for all $r \in \mathcal{A}$. For every integer $n > 1$ and every tuple $\boldsymbol{s} = (s_1, \ldots, s_n) \in \mathcal{A}^n$, let $\langle s_1, \ldots, s_n \rangle$ be the string $\langle s_1 \rangle b^2 \cdots \langle s_n \rangle b^2 b$. Moreover, we write $\langle \boldsymbol{s}, r \rangle$ for $\langle s_1, \ldots, s_n, r \rangle$. Let $(\cdot)_{dbl}$ be the function which doubles the lengths of the strings such that, for every string $s \in \mathcal{A}$, the value s_{dbl} is defined by $s_{dbl} = sa^{|s|}$.*

Now, we are able to partially fix the wished relations Rel. Later, we define R step by step on $\{\langle \boldsymbol{x}, code(\psi)\rangle_{dbl} \mid (\exists l_0, l_1 \in \mathbb{N}^+)(\boldsymbol{x} \in \mathcal{A}^{l_0} \,\&\, \psi \in \mathcal{F}_{l_0,l_1})\}$.

Definition 6. *Let* $c_1 = code(X_1 = X_1)$, $\mathcal{B} = \{\langle x, c_1\rangle r \mid x \in \mathcal{A} \,\&\, r \in \mathcal{A}\}$, $\mathcal{B}_1 = \{\langle x, c_1\rangle_{dbl} \mid x \in \mathcal{A}\}$, *and*

$$\begin{aligned}\mathrm{REL} = \{Rel : \mathcal{A} \mapsto \{true, false\} \mid\ & (\forall s \in \mathcal{B})(Rel(s) \leftrightarrow s \in \mathcal{B}_1)\\ & \&\ (\forall s \in \mathcal{A})(Rel(s) \rightarrow (\exists r \in \mathcal{A})(s = (rb)_{dbl}))\}\ .\end{aligned}$$

For every $Rel \in \mathrm{REL}$, we introduce some satisfiability problem $\text{RES-SAT}_{\Sigma_{Rel}}$ and the unary variant $\text{RES-SAT}^{(1)}_{\Sigma_{Rel}}$ where the quantifier domain is restricted for each formula. Then we define some $R \in \mathrm{REL}$ on the remaining domain $\mathcal{A} \setminus \mathcal{B}$ recursively such that $\text{RES-SAT}^{(1)}_{\Sigma_R}$ is decidable by means of R in constant time. The reduction of SAT_{Σ_R} to $\text{RES-SAT}^{(1)}_{\Sigma_R}$ is possible in polynomial time since we can restrict the quantifier domains as well as the domains of parameters to short strings without changing the truth values of formulae. The idea for the restrictions relies on the following lemma.

Lemma 2. *(1) Each* $(\psi \wedge sub_x(Z_1) = Z_2) \in \mathcal{F}$ *in which the variable* Z *does not occur is equivalent to* $(\psi \wedge add_x(Z_2) = Z_1) \vee \exists Z(\psi \wedge add_x(Z_2) = Z \wedge Z_1 \neq Z \wedge Z_1 = Z_2)$.

(2) For any $\psi \in \mathcal{F}$ *with* $|code(\psi)| = l$, *there are conjunctions* $\psi_1, \ldots, \psi_v \in \mathcal{F}$ *containing at most* l *variables and only literals in which* sub_a *and* sub_b *do not occur such that* ψ *is equivalent to* $\exists \boldsymbol{Z}\psi_1 \vee \cdots \vee \exists \boldsymbol{Z}\psi_v$.

(3) For each $Rel \in \mathrm{REL}$ *and any conjunction* $\psi \in \mathcal{F}$ *which contains at most* l *variables and in which* sub_a *and* sub_b *do not occur, there is some system of conditions whose solutions are exactly the strings which satisfy* ψ *over* Σ_{Rel}. *The conditions have the form* $Z_i = \varepsilon$, $Z_{i_1} \neq Z_{i_2}x_1 \cdots x_{v_1}$, $Z_j = Z_i x_1 \cdots x_{v_2}$, $[\neg]Rel(Z_i a^{v_1})$, *and* $[\neg]Rel(Z_i x_1 \cdots x_{w_1} b a^{w_2})$ *where* $j \in J$ *and* $i, i_1, i_2 \notin J$ *for some* $J = \{j_1, \ldots, j_k\}$, $v_1, v_2, w_1 + w_2 + 1 < l$, *and* $x_1, x_2, \ldots, \in \{a, b\}$. *If* ψ *is satisfiable over* Σ_{Rel}, *then* ψ *is also satisfied by strings in* $\mathcal{A}^{(<2l)} \cup \mathcal{B}$. □

Definition 7. *Let*

$$\begin{aligned}\text{SUB-SAT}_{\Sigma_{Rel}} = \{(\boldsymbol{x}, code_{\mathrm{pad}}(\psi)) \mid\ & (\exists l_0, l_1 \in \mathbb{N}^+)(\psi \in \mathcal{F}_{l_0,l_1}\\ & \&\ \boldsymbol{x} \in (\mathcal{A}^{(\leq m_0 + \bar{m}_0)} \cup \mathcal{B}^{(\leq max_0)})^{l_0}\ \&\ \Sigma_{Rel} \models \exists \boldsymbol{Y}\psi(\boldsymbol{x}, \boldsymbol{Y}))\}\ ,\end{aligned}$$

$$\begin{aligned}\text{RES-SAT}_{\Sigma_{Rel}} = \{(\boldsymbol{x}, code_{\mathrm{pad}}(\psi)) \mid\ & (\exists l_0, l_1 \in \mathbb{N}^+)(\psi \in \mathcal{F}_{l_0,l_1}\\ & \&\ \boldsymbol{x} \in \mathcal{A}^{l_0}\ \&\ \Sigma_{Rel} \models (\exists \boldsymbol{Y} \in (\mathcal{A}^{(\leq m_1 + \bar{m}_1)} \cup \mathcal{B})^{l_1})\psi(\boldsymbol{x}, \boldsymbol{Y}))\}\end{aligned}$$

where for each combination $l_0, l_1 > 0$ *and for each* $\boldsymbol{x} \in \mathcal{A}^{l_0}$,

$$\begin{aligned} m_0 = m_0^{(l_0,l_1)} &= (l_0 + 1)(l_1 + 1) - 2, & \bar{m}_0 = \bar{m}_0^{(l_0,l_1)} &= l_0(l_1 + 1) - 1,\\ m_1 = m_{1,\boldsymbol{x}}^{(l_1,0)} &= \max\{|x_1|, \ldots, |x_{l_0}|\} + l_1, & \bar{m}_1 = \bar{m}_1^{(l_1,0)} &= l_1 - 1\ ,\end{aligned}$$

and $max_0 = max_0^{(l_0,l_1)} = 4m_0 + 5\bar{m}_0 + 4|c_1| + 15$. *Since we use these notations in a fixed context, we shall omit the indices* (l_0, l_1), $(l_1, 0)$, *and* $\boldsymbol{x}$. *Moreover, let*

$$\text{RES-SAT}^{(1)}_{\Sigma_{Rel}} = \{\langle \boldsymbol{x}, c\rangle_{dbl} \mid c \in \mathcal{A}^{(>0)} \,\&\, \exists \boldsymbol{e}((\boldsymbol{x}, c, \boldsymbol{e}) \in \text{RES-SAT}_{\Sigma_{Rel}} \,\&\, \boldsymbol{e} \in \{\varepsilon\}^{|c|})\}.$$

Definition 8. *Let* $\hat{\mathcal{A}} = \{\langle \boldsymbol{x}, code(\psi)\rangle_{dbl} \mid (\exists l_0, l_1 \in \mathbb{N}^+)(\boldsymbol{x} \in \mathcal{A}^{l_0} \;\&\; \psi \in \mathcal{F}_{l_0,l_1})\}$ *and let* R *be the relation in* $\bigcap_{k\in\mathbb{N}} \mathrm{REL}_k$ *where*

$$\mathrm{REL}_0 = \{Rel \in \mathrm{REL} \mid (\forall s \in \mathcal{A} \setminus (\hat{\mathcal{A}} \cup \mathcal{B}))(Rel(s) = false)\} \ ,$$

$$\begin{aligned}\mathrm{REL}_k = \{Rel \in \mathrm{REL}_{k-1} \mid\ & (\forall l_0, l_1 \in \mathbb{N}^+)(\forall \boldsymbol{x} \in \mathcal{A}^{l_0})(\forall \psi \in \mathcal{F}_{l_0,l_1}) \\ & (|\langle \boldsymbol{x}, code(\psi)\rangle_{dbl}| = k \rightarrow (Rel(\langle \boldsymbol{x}, code(\psi)\rangle_{dbl}) \leftrightarrow \\ & (\exists \boldsymbol{y} \in (\mathcal{A}^{(\leq m_1+\bar{m}_1)} \cup \mathcal{B})^{l_1})(\forall Rel' \in \mathrm{REL}_{k-1})(\Sigma_{Rel'} \models \psi(\boldsymbol{x}, \boldsymbol{y}))))\} \ .\end{aligned}$$

Because of $\mathcal{B}_1 \subset \hat{\mathcal{A}}$ the unary relation R satisfies $(\forall s \in \mathcal{A})(R(s) \rightarrow s \in \hat{\mathcal{A}})$ and

Proposition 1. *(1) For any* $s \in \mathcal{A}$ *there is at most one* $i \in \mathbb{N}$ *such that* $R(sa^i)$ *holds.*

(2) For any $s \in \mathcal{A}$ *there is some* $r \in \mathcal{A}$ *such that* $R(s)$ *implies* $s = ra^{|r|}$. □

Proposition 2. *For all* $s \in \mathcal{A}$, $R(s)$ *is true iff* $s \in \text{RES-SAT}^{(1)}_{\Sigma_R}$ *holds.* □

3 Characterization and Replacement of the Tuples

For the reduction of SAT_{Σ_R} to $\text{RES-SAT}^{(1)}_{\Sigma_R}$ we need some propositions and corollaries. These result from the possibility to restrict the domain of parameters and the quantifier domain for the variables in each formula $\psi(\boldsymbol{X}, \boldsymbol{Y})$, to a domain of short strings without changing the truth value of this formula. Let us prepare some technical details. We want to consider some common prefixes of strings of low levels which are relevant for the satisfiability of the formulae in SAT_{Σ_R}. These prefixes correspond to the valuations of the variables Z_i, $i \notin J$ in (3) of Lemma 2 which are prefixes of levels k of variables $Z_j = Z_i x_1 \cdots x_k$ with $j \in J$, and so on. For this purpose, we first define the prefixes of several levels in Definition 9. The relevant levels depend on the number and the kind of the variables in such a formula. The relations $\succeq^{(M,n)}$ defined below will give even more information for the evaluation of formulae. They describe, for instance, connections between strings $z_j = z_i x_1 \cdots x_k$ and $z_{j'} = z_i x'_1 \cdots x'_{k'}$ $(x_v, x'_v \in \{a, b\})$ with a common prefix z_i of low levels $k, k' \leq n$, respectively. For each finite set $M_{\boldsymbol{z}} =_{\mathrm{def}} \{z_1, \ldots, z_l\}$ and all non-negative integers n and m, we define some decomposition

$$M_{\boldsymbol{z}} = S_0^{(\boldsymbol{z},n,m)} \cup p_1^{(\boldsymbol{z},n,m)} S_1^{(\boldsymbol{z},n,m)} \cup \cdots \cup p_{w(\boldsymbol{z},n,m)}^{(\boldsymbol{z},n,m)} S_{w(\boldsymbol{z},n,m)}^{(\boldsymbol{z},n,m)} \qquad (*)$$

by means of $\succeq^{(M_{\boldsymbol{z}},n)}$ where $p_1^{(\boldsymbol{z},n,m)}, p_2^{(\boldsymbol{z},n,m)}, \ldots$ are common prefixes of low levels of the elements of $M_{\boldsymbol{z}}$ such that $(*)$ can be used in order to describe the strings in $M_{\boldsymbol{z}}$ by logical terms containing variables for these prefixes and to evaluate equations and inequalities. We are also interested in the behaviour of these prefixes with respect to the relation R. Therefore, we introduce basic characters for these prefixes. For the complete characterization of the tuples $\boldsymbol{y}$ corresponding to the free variables $Y_1, \ldots, Y_{l_1}$ in a formula $\psi(\boldsymbol{X}, \boldsymbol{Y})$, we consider the integers $l = l_1$, $n = 0$, $m = m_1$, and $\bar{m} = \bar{m}_1$. We need $(l, n, m, \bar{m}) =$

$(l_0, l_1, m_0, \bar{m}_0)$ if we consider the parameters in $\boldsymbol{x}$ associated with the variables $X_1, \ldots, X_{l_0}$.

Definition 9. *For every $s \in \mathcal{A}$ and $k \in \mathbb{N}$, let the* prefix $s^{[k]}$ of the level k of s *be recursively defined by $r^{[0]} = r$, $\varepsilon^{[i+1]} = \varepsilon$, $(ra)^{[i+1]} = r^{[i]}$, and $(rb)^{[i+1]} = r^{[i]}$ for all $r \in \mathcal{A}$ and all $i \in \mathbb{N}$. Let $M \subset \mathcal{A}$ be finite and let n be a non-negative integer. For any $r, s \in M$, let $r \succeq^{(M,n)} s$ iff $r^{[n+1]} = s$ or $r^{[n]} \in \{s, s^{[1]}, \ldots, s^{[n]}\}$. For any $r_1, r_2, r_3 \in M$, let $r_3 \succeq^{(M,n)} r_2$ and $r_2 \succeq^{(M,n)} r_1$ imply $r_3 \succeq^{(M,n)} r_1$.*

Lemma 3. *For each finite $M \subset \mathcal{A}$ and $n \in \mathbb{N}$, $r_2 \succeq^{(M,n)} r_1$ implies $|r_2| \geq |r_1|$ and then there is a string $s \in \mathcal{A}^{(\leq |M|(n+1)-1)}$ such that $r_2 = r_1^{[n]} s$.* □

Definition 10. *For every set $M_{\boldsymbol{z}} = \{z_1, \ldots, z_l\} \subset \mathcal{A}$ and all $n, m \geq 0$, let the* decomposition $(*)$ *into the equivalence classes $M_v^{(\boldsymbol{z},n,m)} = p_v^{(\boldsymbol{z},n,m)} S_v^{(\boldsymbol{z},n,m)}$ be inductively defined. Let $p_0^{(\boldsymbol{z},n,m)} = \varepsilon$ and $S_0^{(\boldsymbol{z},n,m)} = \{z \mid \exists z'((z')^{[n]} \in \mathcal{A}^{(\leq m)} \,\&\, z \succeq^{(M_{\boldsymbol{z}},n)} z')\}$. If, for $v > 0$, $N_v =_{def} M_{\boldsymbol{z}} \setminus \bigcup_{i=0}^{v-1} M_i^{(\boldsymbol{z},n,m)} \neq \emptyset$, then we set $i_v = \min\{i \mid z_i \in N_v \,\&\, \neg(\exists z \in N_v)(z_i \succeq^{(M_{\boldsymbol{z}},n)} z \,\&\, z_i^{[n]} \neq z^{[n]})\}$, $p_v^{(\boldsymbol{z},n,m)} = z_{i_v}^{[n]}$, and $S_v^{(\boldsymbol{z},n,m)} = \{r \mid (\exists z \in N_v)(z \succeq^{(M_{\boldsymbol{z}},n)} z_{i_v} \,\&\, z = p_v r)\}$. In case of $N_v = \emptyset$, we set $w^{(\boldsymbol{z},n,m)} = v - 1$.*

Let us state properties of these equivalence classes. Note that (1) follows also from Lemma 3.

Proposition 3. *Let the set $M_{\boldsymbol{z}}$ be decomposed as $(*)$ according to Definition 10. Then, for all $1 \leq v \leq w^{(\boldsymbol{z},n,m)}$, $0 \leq v_1, v_2 \leq w^{(\boldsymbol{z},n,m)}$, $r_i \in S_{v_i}^{(\boldsymbol{z},n,m)}$, and $(k_1, k_2) \in \{0, \ldots, n\}^2 \cup \{(0, n+1)\}$ there hold*

(1) $p_v^{(\boldsymbol{z},n,m)} \in \mathcal{A}^{(>m)}$, $S_v^{(\boldsymbol{z},n,m)} \subseteq \mathcal{A}^{(\leq \bar{m})}$, and $S_0^{(\boldsymbol{z},n,m)} \subseteq \mathcal{A}^{(\leq m+\bar{m})}$,

(2) $(p_{v_1}^{(\boldsymbol{z},n,m)} r_1)^{[k_1]} = (p_{v_2}^{(\boldsymbol{z},n,m)} r_2)^{[k_2]}$ iff $(v_1 = v_2$ and $r_1^{[k_1]} = r_2^{[k_2]})$. □

For all n and m, we want to describe all elements of $M_{\boldsymbol{z}}$ with $(*)$ by logical terms $P_v r_i$ where $P_1, P_2, \ldots$ are the variables for the common prefixes $p_1^{(\boldsymbol{z},n,m)}, p_2^{(\boldsymbol{z},n,m)}, \ldots$. For the term $P_v r_i$ we select the index v and the string r_i. Moreover, each P_v will be weighted by a basic character $basic(v)$ such that $basic(v) \geq 0$ implies $R(p_v^{(\boldsymbol{z},n,m)} a^{basic(v)}) = true$.

Definition 11. *Set $m'' = (l+1)(n+1) - 2$. Let the set $M_{\boldsymbol{z}}$ be decomposed as $(*)$ according to Definition 10. Let then $char_{l,n}^m(\boldsymbol{z}) = (term, basic)$ where, for $i \in \{1, \ldots, l\}$ and $v \in \{0, \ldots, w^{(\boldsymbol{z},n,m)}\}$,*

$$term(i) = \begin{cases} (0, z_i) & \text{if } z_i \in S_0^{(\boldsymbol{z},n,m)} \\ (v, r) & \text{if } r \in S_v^{(\boldsymbol{z},n,m)},\ z_i = p_v^{(\boldsymbol{z},n,m)} r, \text{ and } v \neq 0 \end{cases},$$

$$basic(v) = \begin{cases} k & \text{if } R(p_v^{(\boldsymbol{z},n,m)} a^k) \text{ and } k \leq m'' \\ -1 & \text{if } \neg R(p_v^{(\boldsymbol{z},n,m)} a^k) \text{ for all } k \in \{0, \ldots, m''\} \end{cases}.$$

We also use the functions $term_1$ and $term_2$ with $term(i) = (term_1(i), term_2(i))$ for all $i \leq l$.

Now, we introduce functions which are used in order to reduce SAT_{Σ_R} to RES-$\mathrm{SAT}^{(1)}_{\Sigma_R}$.

Definition 12. *For each* $\boldsymbol{z} \in \mathcal{A}^l$, *let* $small^m_{l,n}(\boldsymbol{z}) = \boldsymbol{z}' \in (\mathcal{A}^{(\leq m+\bar{m})} \cup \mathcal{B}^{(\leq max)})^l$ *with* $max = 4m + 5\bar{m} + 4|c_1| + 15$ *and*

$$z'_i = \begin{cases} term_2(i) & \text{if } term_1(i) = 0 \\ \langle a^v b^{m+\bar{m}-v}, c_1\rangle a^{|\langle a^v b^{m+\bar{m}-v}, c_1\rangle| - basic(v)} term_2(i) & \text{if } term_1(i) = v \neq 0 \end{cases}$$

for all $i \in \{1, \ldots, l\}$ *where* $(term, basic) = char^m_{l,n}(\boldsymbol{z})$.

Proposition 4. *For every* $\boldsymbol{z} \in \mathcal{A}^l$, $char^m_{l,n}(small^m_{l,n}(\boldsymbol{z})) = char^m_{l,n}(\boldsymbol{z})$ *holds.* □

4 Preparation of the Reduction of SAT_{Σ_R} to RES-$\mathrm{SAT}^{(1)}_{\Sigma_R}$

Now we want to prove that in evaluating the formulae in $\mathcal{F}$ we need only short strings for the free variables. A consequence is the decidability of SAT_{Σ_R}. Because of (2) in Lemma 2 we have to consider only conjunctions in this proof.

Proposition 5. *Let* $l_0, l_1 \in \mathbb{N}^+$ *and* $\boldsymbol{x} \in \mathcal{A}^{l_0}$. *Let* $\psi(X_1, \ldots, X_{l_0}, Y_1, \ldots, Y_{l_1})$ *be a conjunction in* $\mathcal{F}$. *For all* l_1*-tuples* $\boldsymbol{y}^{(1)}$ *and* $\boldsymbol{y}^{(2)}$ *with* $char^{m_1}_{l_1,0}(\boldsymbol{y}^{(1)}) = char^{m_1}_{l_1,0}(\boldsymbol{y}^{(2)})$ *there holds*

$$\Sigma_R \models \psi(\boldsymbol{x}, \boldsymbol{y}^{(1)}) \leftrightarrow \psi(\boldsymbol{x}, \boldsymbol{y}^{(2)}) \ .$$

Proof. Let $\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)} \in \mathcal{A}^{l_1}$ with $char^{m_1}_{l_1,0}(\boldsymbol{y}^{(1)}) = char^{m_1}_{l_1,0}(\boldsymbol{y}^{(2)}) = (term, basic)$. Then, for some w, there are strings $p^{(1)}_0, \ldots, p^{(1)}_w$ and $p^{(2)}_0, \ldots, p^{(2)}_w$ such that we can describe the components of $\boldsymbol{y}^{(1)}$ and $\boldsymbol{y}^{(2)}$ by the terms in $M_{term} =_{\mathrm{def}} \{P_{term_1(1)} term_2(1), \ldots, P_{term_1(l_1)} term_2(l_1)\}$ where P_v stands for $p^{(1)}_v$ or $p^{(2)}_v$. Property (1) in Proposition 3 means that, for $v \neq 0$, $|P_v| > m_1$ and, consequently, the inequalities $x_j \neq P_v r$, $x_j^{[1]} \neq P_v r$, and $x_j \neq (P_v r)^{[1]}$ hold for all $r \in \mathcal{A}$ and for all $j \in \{1, \ldots, l_0\}$. Because of (2) in Proposition 3, for $k \in \{0, 1\}$ and for all $i, j \in \{1, \ldots, l_1\}$, the equation $P_{term_1(i)} term_2(i)^{[k]} = P_{term_1(j)} term_2(j)$ implies that $term_1(i) = term_1(j)$ and $term_2(i)^{[k]} = term_2(j)$. Therefore, in evaluating all equations and inequalities in $\psi(\boldsymbol{x}, \boldsymbol{Y})$ for $\boldsymbol{Y} \in \{\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}\}$, evaluation of the terms in M_{term} are sufficient.

Let us now assume that $R(p^{(1)}_{term_1(i)} term_2(i))$ and $\neg R(p^{(2)}_{term_1(i)} term_2(i))$ hold for some $i \in \{1, \ldots, l_1\}$ with $term_1(i) \neq 0$. In case of $basic(term_1(i)) \geq 0$, $R(s_1 a^{basic(term_1(i))})$ and $R(s_2 a^{basic(term_1(i))})$ are true for the strings $s_1 = p^{(1)}_{term_1(i)}$ and $s_2 = p^{(2)}_{term_1(i)}$. By (1) in Proposition 3 we have $|term_2(i)| \leq l_1 - 1$, $|s_1| > l_1$, and $|s_2| > l_1$. From (2) in Proposition 1 and the assumption of $R(s_1 term_2(i)) = true$ it follows that a string $r \in \mathcal{A}$ exists such that $s_1 term_2(i) = r a^{|r|}$ holds. If we compare the lengths of the strings, then we get $|s_1| > \frac{1}{2}|s_1 term_2(i)| = |r|$. Thus, the string r is a prefix of s_1 and consequently we have $term_2(i) = a^k$ for

some $k \leq l_1 - 1$ with $0 \leq k < |r|$. Therefore, because of $R(s_1 term_2(i)) = true$ we get $basic(term_1(i)) = k = |term_2(i)|$ and $term_2(i) = a^{basic(term_1(i))}$. But, in this case $R(s_2 term_2(i))$ is true. Hence we have $R(p^{(2)}_{term_1(i)} term_2(i))$. This contradicts our assumption.

Hence, the equivalence $\psi(\boldsymbol{x}, \boldsymbol{y}^{(1)}) \leftrightarrow \psi(\boldsymbol{x}, \boldsymbol{y}^{(2)})$ holds in Σ_R. □

Since each $small^{m_1}_{l_1,0}$ is a map into $(\mathcal{A}^{(\leq m_1 + \bar{m}_1)} \cup \mathcal{B})^{l_1}$, by the definition of RES-SAT_{Σ_R} we see that Lemma 1, Proposition 4, and Proposition 5 imply

Corollary 1. $\mathrm{SAT}_{\Sigma_R} = \text{RES-}\mathrm{SAT}_{\Sigma_R}$.

Therefore, the polynomial time reduction of SAT_{Σ_R} to SUB-SAT_{Σ_R} and of SUB-$\mathrm{SAT}_{\Sigma_R} \subseteq$ RES-SAT_{Σ_R} to RES-$\mathrm{SAT}^{(1)}_{\Sigma_R}$ is sufficient for $\mathrm{SAT}_{\Sigma_R} \in \mathrm{P}_{\Sigma_R}$. The first reduction follows from the next proposition in proof of which we also use

Definition 13. *For every strings $r, s \in \mathcal{A}$, let $r \subset_1 s$ be true iff $r = s^{[1]}$. For each tuple $\boldsymbol{h} = (h_1, \ldots, h_w) \in \mathcal{A}^w$, let $M_{\boldsymbol{h}}$ be the set $\{h_1, \ldots, h_w\}$. The set $M \subseteq M_{\boldsymbol{z}}$ is called a* maximal chain of $M_{\boldsymbol{z}}$ *if there is some $\boldsymbol{g} = (g_1, \ldots, g_v)$ such that $M = M_{\boldsymbol{g}}$, $g_{i-1} \subset_1 g_i$ for all $i \in \{2, \ldots, v\}$, and $\neg(\exists g \in M_{\boldsymbol{z}} \setminus \{\varepsilon\})(g \subset_1 g_1 \vee g_v \subset_1 g)$. For each $\boldsymbol{z} = (z_1, \ldots, z_l) \in \mathcal{A}^l$ and for any $z \in M_{\boldsymbol{z}}$, let $min_{\boldsymbol{z}}(z)$ be the smallest prefix r of z for which there is some maximal chain M of $M_{\boldsymbol{z}}$ with $r \in M$ and $z \in M$.*

Proposition 6. *Let $l_0, l_1 \in \mathbb{N}^+$ and $\psi(X_1, \ldots, X_{l_0}, Y_1, \ldots, Y_{l_1})$ be a conjunction in $\mathcal{F}$. For every $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ in $\mathcal{A}^{l_0}$ with $char^{m_0}_{l_0,l_1}(\boldsymbol{x}^{(1)}) = char^{m_0}_{l_0,l_1}(\boldsymbol{x}^{(2)})$, there holds*

$$\Sigma_R \models \exists \boldsymbol{Y} \psi(\boldsymbol{x}^{(1)}, \boldsymbol{Y}) \leftrightarrow \exists \boldsymbol{Y} \psi(\boldsymbol{x}^{(2)}, \boldsymbol{Y}) \ .$$

Proof. Let $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)} \in \mathcal{A}^{l_0}$ and $(term, basic) = char^{m_0}_{l_0,l_1}(\boldsymbol{x}^{(1)}) = char^{m_0}_{l_0,l_1}(\boldsymbol{x}^{(2)})$. Consider $p^{(1)}_0 = p^{(2)}_0 = \varepsilon$ and $(p^{(1)}_1, \ldots, p^{(1)}_w)$ and $(p^{(2)}_1, \ldots, p^{(2)}_w)$ in $(\mathcal{A}^{(>m_0)})^w$ such that $x^{(1)}_i = p^{(1)}_{term_1(i)} term_2(i)$ and $x^{(2)}_i = p^{(2)}_{term_1(i)} term_2(i)$ for all $i \in \{1, \ldots, l_0\}$. It is enough to consider the case $w > 0$. Moreover, we assume that there is a tuple $\boldsymbol{y}^{(1)} \in \mathcal{A}^{l_1}$ such that $\psi(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)})$ is true in Σ_R.

Let $\boldsymbol{z} = (z_1, \ldots, z_{l_0+l_1}) = (\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)})$. Let $\mathcal{V}$ be the set of all maximal chains of $M_{\boldsymbol{z}}$ and, for all $\boldsymbol{h}$ with $M_{\boldsymbol{h}} \subseteq M_{\boldsymbol{z}}$, let $Min_{\boldsymbol{h}}$ be the set $\{min_{\boldsymbol{z}}(z) \mid z \in M_{\boldsymbol{h}}\}$. Moreover, for all $g \in Min_{\boldsymbol{z}}$, let $\mathcal{V}_g$ be the set of all sets $M \in \mathcal{V}$ containing g.

Let us make the following replacements for all $g \in Min_{\boldsymbol{z}}$. If there is an $i \leq l_0$ with $x^{(1)}_i \in \bigcup_{M \in \mathcal{V}_g} M$, then we replace all remainders $p^{(1)}_{term_1(i)}$ in all elements of $\bigcup_{M \in \mathcal{V}_g} M$ by $p^{(2)}_{term_1(i)}$. Thus, we get a new set $M_{\boldsymbol{z}'}$ containing all $x^{(2)}_i$ and new strings $y^{(2)}_j$ in $\bigcup_{i=1}^{l_0} p^{(2)}_{term_1(i)} \mathcal{A}^{(\leq m_0)}$ derived from the elements $y^{(1)}_j \in M_{\boldsymbol{y}^{(1)}} \cap \bigcup_{g \in Min_{\boldsymbol{x}^{(1)}}} \bigcup_{M \in \mathcal{V}_g} M$. All equations and inequalities in ψ remain true for the new strings in $M_{\boldsymbol{z}'}$ because from $z_i = z_j$ there follows $min_{\boldsymbol{z}}(z_i) = min_{\boldsymbol{z}}(z_j)$ for all $z_i, z_j \in M_{\boldsymbol{z}}$. If we have made all these replacements and the elements of the chains in some $\mathcal{V}_{g'}$ were not replaced and one of these elements is equal to an element of $M_{\boldsymbol{z}'}$, then we replace the corresponding remainders $p^{(2)}_{v_0}$

in all elements of $\bigcup_{M\in\mathcal{V}_{g'}} M$ by $p_{v_0}^{(3)} = \langle a^{v_0}b^{max}, c_1\rangle a^{|\langle a^{v_0}b^{max},c_1\rangle|-basic(v_0)} \in \mathcal{B}$ where $max = \max\{|s| \mid s \in M_{\boldsymbol{z}'} \cup \{z \in M_{\boldsymbol{y}^{(1)}} \mid z \text{ was not replaced}\}\}$. By taking $y_i^{(2)} = y_i^{(1)}$ in the remaining cases we obtain the complete tuple $\boldsymbol{y}^{(2)}$ from $\boldsymbol{y}^{(1)}$.

If we assume that $R(x_i^{(1)})$ and $\neg R(x_i^{(2)})$ or $R(y_j^{(1)})$ and $\neg R(y_j^{(2)})$ hold for some i or j, then we get a contradiction analogous to the last proof.

Hence it follows that $\psi(\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)})$ is also true in Σ_R. □

Now, by Lemma 1, Proposition 4, and Proposition 6 there follows

Corollary 2. *Let $\psi(\boldsymbol{X}, \boldsymbol{Y})$ be a formula in $\mathcal{F}_{l_0,l_1}$. For any l_0-tuple $\boldsymbol{x}$ we have*

$$\Sigma_R \models \exists \boldsymbol{Y}\psi(\boldsymbol{x}, \boldsymbol{Y}) \leftrightarrow \exists \boldsymbol{Y}\psi(small_{l_0,l_1}^{m_0}(\boldsymbol{x}), \boldsymbol{Y}) \ .$$

5 P = NP for the Structure Σ_R

Proposition 7. *(1) The problem* $\text{RES-SAT}_{\Sigma_R}^{(1)}$ *is decidable by means of R in constant time.*

(2) Every problem in NP_{Σ_R} *can be reduced to* SAT_{Σ_R} *in polynomial time.*

Proof. (1) follows from Proposition 2. The proof of (2) can be done analogous to the proof of the NP-completeness for the classical problem SAT. □

Definition 14. *Let f_1 be the function of $\mathcal{A}^\infty$ into $\mathcal{A}^\infty$ defined by*

$$f_1(\boldsymbol{s}) = \begin{cases} (small_{l_0,l_1}^{m_0}(\boldsymbol{x}), c, \boldsymbol{e}) & \text{if } \boldsymbol{s} = (\boldsymbol{x}, c, \boldsymbol{e}), \boldsymbol{x} \in \mathcal{A}^{l_0}, c \in \mathcal{A}^{(>0)}, \text{ and } \boldsymbol{e} \in \{\varepsilon\}^{l_1} \\ \varepsilon & \text{otherwise .} \end{cases}$$

The first part of the next proposition follows from Corollary 2.

Proposition 8. *The function f_1 reduces* SAT_{Σ_R} *to* $\text{SUB-SAT}_{\Sigma_R}$*. The function f_1 can be computed by some Σ_R-machine in polynomial time.* □

Definition 15. *Let the function f_2 of $\mathcal{A}^\infty$ into $\mathcal{A}$ be defined by*

$$f_2(\boldsymbol{s}) = \begin{cases} \langle \boldsymbol{x}, c\rangle_{dbl} & \text{if } \boldsymbol{s} = (\boldsymbol{x}, c, \boldsymbol{e}),\ c \in \mathcal{A}^{(>0)}, \text{ and } \boldsymbol{e} \in \{\varepsilon\}^{|c|} \\ \varepsilon & \text{otherwise .} \end{cases}$$

By the definition of $\text{RES-SAT}_{\Sigma_R}^{(1)}$ and by Corollary 1 we obtain

Proposition 9. *The function f_2 reduces* SAT_{Σ_R} *to* $\text{RES-SAT}_{\Sigma_R}^{(1)}$. □

In order to show that a further polynomial time reduction of $\text{SUB-SAT}_{\Sigma_R}$ to $\text{RES-SAT}_{\Sigma_R}^{(1)}$ by means of f_2 is possible, we use the following proposition.

Proposition 10. *There is some deterministic Σ_R-machine which computes the strings $\langle s_1, \ldots, s_n\rangle$ from $(s_1, \ldots, s_n) \in \mathcal{A}^\infty$ in a time which depends on the sum of the lengths of the strings $s_1, \ldots, s_n$ linearly.* □

This allows us to show

Proposition 11. *The function f_2 can be computed on the domain*

$$\{(\boldsymbol{x}, c, \boldsymbol{e}) \mid (\exists l_0, l_1 \in \mathbb{N}^+)(\boldsymbol{x} \in (\mathcal{A}^{(\leq m_0 + \bar{m}_0)} \cup \mathcal{B}^{(\leq max_0)})^{l_0} \,\&\, c \in \mathcal{A}^{(>0)} \,\&\, \boldsymbol{e} \in \{\varepsilon\}^{l_1})\}$$

by some Σ_R-machine in polynomial time. □

We arrive at the main result which mainly follows from the proposition 5 and 6.

Theorem 2. $\mathrm{P}_{\Sigma_R} = \mathrm{NP}_{\Sigma_R}$ *holds.*

Proof. By the propositions 8, 9, and 11, $f_2 \circ f_1$ reduces the problem SAT_{Σ_R} to $\mathrm{RES\text{-}SAT}^{(1)}_{\Sigma_R}$ in polynomial time. Therefore, by (1) in Proposition 7 the problem SAT_{Σ_R} is decidable in polynomial time. Consequently, by (2) in Proposition 7 each problem in NP_{Σ_R} is decidable in polynomial time. □

References

1. Blum, L., F. Cucker, M. Shub, and S. Smale, *Complexity and Real Computation.* Springer-Verlag (1998).
2. Blum, L., M. Shub and S. Smale, 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 (1989), 1–46.
3. Cucker, F. and M. Matamala, On digital nondeterminism. *Math. Systems Theory* 29 (1996), 635–647.
4. Cucker, F., M. Shub, and S. Smale, Separation of complexity classes in Koiran's weak model. *Theoretical Computer Science* 133 (1994), 3–14.
5. Gassner, C., Über die Konstruktion von Strukturen endlicher Signatur mit P = NP. *Preprint* 1 (2004).[1]
6. Gassner, C., NP ⊂ DEC und P = NP für Expansionen von Erweiterungen von Strukturen endlicher Signatur mit Identitätsrelation. *Preprint* 13 (2004).[1]
7. Gassner, C., Eine Struktur endlicher Signatur mit Identitätsrelation und mit P = NP. *Preprint* 14 (2004).[1]
8. Gassner, C., Expansions of structures with P = NP (to appear in *Computer Science Report Series of the University of Wales Swansea*).
9. Hemmerling, A., Computability and complexity over structures of finite type. *Preprint* 2 (1995).[1]
10. Hemmerling, A., P = NP for some structures over the binary words. *Journal of Complexity* 21 (2005), 557-578.
11. Koiran, P., Computing over the reals with addition and order. *Theoretical Computer Science* 133 (1994), 35–47.
12. Meer, K., A note on a P ≠ NP result for a restricted class of real machines. *Journal of Complexity* 8 (1992), 451–453.
13. Meer, K., Real number models under various sets of operations. *Journal of Complexity* 9 (1993), 366–372.
14. Poizat, B., *Les Petits Cailloux.* Aléa (1995).
15. Prunescu, M., Structure with fast elimination of quantifiers. *Journal of Symbolic Logic* 71 (2006), 321–328.

[1] *Preprint-Reihe Mathematik, E.-M.-Arndt-Universität Greifswald*

Expansions of Structures with P = NP

Christine Gaßner*

Institut für Mathematik und Informatik, Ernst-Moritz-Arndt-Universität,
F.-L.-Jahn-Straße 15 a, 17487 Greifswald, Germany
gassnerc@uni-greifswald.de

Abstract. We consider an arbitrary structure of finite signature. For an extension of this structure, an additional relation is constructed such that a unary variant of the corresponding Satisfiablility Problem is decidable by means of this relation with respect to the uniform model of computation in constant time. This implies P = NP for the new structure.

1 Introduction

In 1989, Lenore Blum, Michael Shub, and Steve Smale (BSS) developed a uniform model of computation in order to instigate the complexity of the computation over structures containing the real or other numbers. By analogy with this uniform model and similar to the model of computation over groups given in [5] we introduce register machines over any structures of finite signature which have the form

$$\Sigma = (A; d_1, \ldots, d_t; f_1, \ldots, f_u; r_1, \ldots, r_v, =), \quad t \geq 2$$

where $d_1, \ldots, d_t \in A$ are the constants of this structure, $f_1, \ldots, f_u$ are operations of the arities $n_{f_1}, \ldots, n_{f_u}$, and $r_1, \ldots, r_v$ are relations of the arities $n_{r_1}, \ldots, n_{r_v}$.

Each *Σ-machine* $\mathcal{M}$ is provided with registers $Z_1, Z_2, \ldots$ for elements of A and with a fixed number of registers $I_1, I_2, \ldots, I_{k_\mathcal{M}}$ for indices in $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$. Every input from the input space $\mathcal{I} = \bigcup_{n=1}^{\infty} A^n$ is processed by the machine by means of $L_\mathcal{M}$ instructions labelled by $1, \ldots, L_\mathcal{M}$. The instructions are executed according to the order of their labels. There are branch and non-branch instructions. The successor of a non-branch instruction labelled by $l \in \{1, \ldots, L_\mathcal{M} - 1\}$ is the instruction labelled by $l + 1$. The two successors of a branch instruction are fixed by the instruction itself. The output can be a tuple. Each instruction must be of one of the following types.

The *input instruction* is labelled by 1. Every input $(x_1, \ldots, x_n) \in \mathcal{I}$ is assigned to the registers $Z_1, \ldots, Z_n$. To simplify matters, the registers $Z_{n+1}, Z_{n+2}, \ldots$ obtain the constant d_1 as value. The index register I_1 gets the content n and the other index registers get the content 1. Moreover, in the nondeterministic

* I thank Volkmar Liebscher, Rainer Schimming, and several referees for helpful comments to [7] and to this paper. Moreover, I thank Gerald van den Boogaart and Michael Schürmann for discussions.

case it is possible that the input instruction includes a *guess instruction*. By means of this instruction a *nondeterministic* machine can redefine the content of a finite number of registers $Z_{n+1}, \ldots, Z_{n+m}$ by guessing the number m and some finite sequence $(y_1, \ldots, y_m) \in A^m$ and putting it into $Z_{n+1}, \ldots, Z_{n+m}$ before the second instruction is executed. The *computation*, *branch*, and *copy instructions* have the form

l: $Z_j := f_k(Z_{j_1}, \ldots, Z_{j_{n_{f_k}}})$; $\qquad$ l: $Z_j := d_k$;
l: $I_j := I_j + 1$; $\qquad$ l: $I_j := 1$;
l: IF *cond* THEN GOTO l_1 ELSE GOTO l_2; $\qquad$ l: $Z_{I_j} := Z_{I_k}$;

where $l \in \{2, \ldots, L_{\mathcal{M}} - 1\}$ and *cond* can have the form $Z_j = Z_k$, $I_j = I_k$, and $r_k(Z_{j_1}, \ldots, Z_{j_{n_{r_k}}})$, respectively. Indirect addresses are permitted only in the copy instructions. The *output instruction* is labelled by $L_{\mathcal{M}}$. If this instruction is reached, then $(Z_1, \ldots, Z_{I_1})$ is the output and the machine halts.

Whereas the nondeterministic machines can guess, the *deterministic* machines work without guessing. Note that by definition the class of nondeterministic machines contains the deterministic machines.

A nondeterministic Σ-machine $\mathcal{M}$ *accepts* an input $(x_1, \ldots, x_n) \in \mathcal{I}$ *nondeterministically* if there is some finite sequence $(y_1, \ldots, y_m)$ such that $\mathcal{M}$ guesses exactly the elements $y_1, \ldots, y_m$ in one step and if $\mathcal{M}$ outputs d_1 for the input $(x_1, \ldots, x_n)$ and these guesses $(y_1, \ldots, y_m)$. A deterministic Σ-machine *accepts* or *rejects*, respectively, an input $(x_1, \ldots, x_n) \in \mathcal{I}$ if the machine outputs d_1 or d_2, respectively, without guessing.

The execution of one instruction is one step of the computation. Each step can be executed in one fixed time unit. A Σ-machine works *in polynomially bounded time* if there is a polynomial p such that the machine produces an output for every input $(x_1, \ldots, x_n) \in \mathcal{I}$ within at most $p(n)$ time units. A problem $\mathcal{P} \subseteq \bigcup_{n=1}^{\infty} A^n$ is *nondeterministically recognizable* in polynomial time if there is some nondeterministic Σ-machine which works in polynomially bounded time and accepts an input $(x_1, \ldots, x_n) \in \mathcal{I}$ iff it belongs to $\mathcal{P}$. A problem $\mathcal{P} \subseteq \bigcup_{n=1}^{\infty} A^n$ is *decidable* [*in polynomial time*] if there is a deterministic Σ-machine which accepts all inputs in $\mathcal{I} \cap \mathcal{P}$ and which rejects all inputs in $\mathcal{I} \setminus \mathcal{P}$ [in polynomially bounded time].

P_Σ and NP_Σ are the usual complexity classes of all problems which are decidable and nondeterministically recognizable, respectively, in polynomial time. DEC_Σ is the class of all decidable problems. There holds $\mathrm{P}_\Sigma \subseteq \mathrm{DEC}_\Sigma$ and $\mathrm{P}_\Sigma \subseteq \mathrm{NP}_\Sigma$ whereas $\mathrm{NP}_\Sigma \not\subseteq \mathrm{DEC}_\Sigma$ is possible. Analogous to the classical model and to the BSS model, for every structure of finite signature with two constants we can define NP-complete satisfiability problems, several kinds of universal machines, the halting problems, and so on. For each Σ-machine $\mathcal{M}$, a universal Σ-machine can simulate one step of $\mathcal{M}$ for an input of $\mathcal{M}$ in polynomial time if the universal machine gets a suitable code of this machine as an additional part of its input. Sometimes the number of steps to be simulated is also given as a unary code. Then, the corresponding universal nondeterministic machine recognizes an NP-complete problem. As in the classical case, the halting problems are not decidable over the considered structures.

A *computation path* is a sequence of configurations where the *kth configuration* describes the state of a machine after k steps. The *state of a machine* $\mathcal{M}$ is given by the label of the instruction executed last, and the content of the registers $I_1, I_2, \ldots, I_{k_\mathcal{M}}, Z_1, Z_2, \ldots$. To simplify matters, we sometimes identify the computation paths with the corresponding sequences of the covered labels, the latter are also called *nodes* of the path. All possible paths can be composed to a rooted tree where the root corresponds to the input and the nodes corresponding to the branching instructions have two successors whereas the other nodes have at most one successor. The leaves stand for the outputs.

In [7], a method for the construction of structures of finite signature with P = NP is presented for a simple case. This construction was found while we searched for structures which contain the real numbers and for which P = NP is valid with respect to the uniform model of computation (cf. also [6]). The idea to expand a structure of real numbers (to extent the signature of this structure) in order to obtain results for the problem "P versus NP" goes back to a paper of Klaus Meer [11]. He shows how the ring of real numbers can be expanded such that P $\neq$ NP holds for the new structures. Of course, the alternative question how one obtains P = NP by expanding the real numbers is interesting, too. One idea is to choose an NP-complete problem UNI_R recognized non-deterministically by a universal machine and to introduce an additional relation R by means of which this problem can be decided deterministically in polynomial time. Another possibility is to use the corresponding Satisfiablility Problem defined by first order formulae and parameters as NP-complete problem (see [7]). The values of the reduction functions which reduce arbitrary problems in NP to some usual NP-complete problem $\mathcal{P}$ are tuples of any arity. They play an important role in such a construction. If all tuples can be encoded by tuples of a fixed arity or by single elements of the universe, then, maybe, it is possible to decide which values of a reduction function belong to $\mathcal{P}$ by means of a suitable relation R satisfied by the codes of the tuples in $\mathcal{P}$. For the tuples of real numbers, possibilities of encoding in polynomial time are given in extensions like structures of trees whose leaves are labelled by real numbers, structures of paths whose nodes are labelled by real numbers or structures of strings over the real numbers, and so on.

In defining R, we have to be careful as the following example shows. Let $\mathrm{UNI}_R^{(3)}$ be the ternary problem which contains exactly the triples consisting of the code of an input $(x_1, \ldots, x_n)$, the code of a nondeterministic machine $\mathcal{M}$, and the code of a number n if and only if $\mathcal{M}$ accepts $(x_1, \ldots, x_n)$ within n steps. If we could reduce UNI_R to $\mathrm{UNI}_R^{(3)}$ in polynomial time and if we could decide $\mathrm{UNI}_R^{(3)}$ by means of the relation R in constant time, then we would have P = NP for the corresponding classes P and NP. But, let us assume that $\mathrm{UNI}_R^{(3)}$ is decidable by a machine $\mathcal{M}_0$ in polynomial time. Moreover, let us consider the machine which accepts the code of any input $(x_1, \ldots, x_n)$ and the code of any machine $\mathcal{M}$ by guessing the code of some number of steps n and by checking whether $\mathcal{M}$ accepts $(x_1, \ldots, x_n)$ within n steps by analogy with $\mathcal{M}_0$. This machine recognizes exactly the set of the pairs of the first two components of

all tuples in $\mathrm{UNI}_R^{(3)}$ non-deterministically. However, this set includes the known halting problem with respect to the considered structure. Therefore, the halting problem would belong to NP. Thus, $\mathrm{UNI}_R^{(3)} \in \mathrm{P}$ implies that NP contains undecidable problems and consequently $\mathrm{P} \neq \mathrm{NP}$ for the considered structure. Hence, the decision of $\mathrm{UNI}_R^{(3)}$ by means of the relation R is not possible in constant time. The main reason could be that the guesses of non-deterministic machines can not be replaced by small guesses. On the other side we know results as in [9] which imply the decidability of the problems in NP for a structure over the real numbers. There, the decidability follows from the possibility to replace the large real numbers which could be the guesses by "small" numerators and denominators of coefficients of linear combinations of the inputs. Therefore, we want to define a structure which allows to replace arbitrary guesses by small guesses in each case.

A great problem is also that the codes of machines and the formulae which are relevant for the definition of the values of $R(x)$ for the codes x can contain the code of R or the symbol R. Therefore, we shall define R inductively, or essentially inductively, for instance, on the length of strings. It is possible to solve this problem without great difficulties if we give a relation R which can be described by an infinite tree T (for details see [7] and Fig. 1). The paths of T remind of the computation paths of machines which are traversed by the inputs step by step (see [1, 3, 4, 9, 10]).

Here, the elements of the new structure are strings. The single characters of these strings, for instance the real numbers, correspond to the labels of the nodes of paths, and the strings which have to satisfy R correspond to the paths of T starting with the root and ending with leaves, in reverse order. We consider only three string operations. We permit to add one character to a string, to read the last character of a string, and to delete the last character. Thus, we can modify the strings only slowly, step by step by changing the last character. Moreover, we extend the operations and the relations of the basic structure in a trivial way such that the new structure reminds of a structure of two sorts of elements. The strings with a length greater than 1 do not satisfy the extended relations. Moreover, the extended functions assign the empty string to the tuples whose components are not from the basic structure. These definitions make it possible that there is an analogy between the large strings satisfying R and large computation paths. We shall see that most prefixes of the large strings are not important for the evaluation of the free variables in the formulae. It is possible to consider small prefixes instead of the large prefixes without changing the truth value of a considered formula. Thus, we can restrict the domain of the free variables to a domain of mainly small strings. The necessary length of the small strings is only dependent on the length of the considered formula and on the length of the considered parameters. Similar restrictions of the domains of the parameters in the tuples of the Satisfiability Problem enable the decision of the Satisfiability Problem by means of R in polynomial time.

Because this kind of construction is possible with any structure of finite signature as starting point, we shall give expansions of extensions for arbitrary

structures of finite signatures. The new structures contain strings whose characters are the elements of the basic structures. Moreover, to simplify matters, we shall also use two additional constants a and b for the encoding of the formulae such that the characters of the considered strings can also be the new constants. Then, the structure over $\{a, b\}^*$ considered in [7] is the result of expanding of an extension of the empty structure. Strictly speaking, the additional constants are not necessary if the basic structure has two constants. In this case we could construct the new structure without new characters. The structure considered in [7] would be the result of expanding of an extension of the structure $\Sigma_0 = (\{a, b\}; a, b; ; =)$ where the computation over Σ_0 corresponds to the classical computation. However, in this case the two constants a and b have a special purpose, too.

2 The Structure Σ_R and Four Satisfiability Problems

For an arbitrary structure Σ, let the elements of Σ and two new characters be the symbols of an alphabet. We remark that this alphabet may be infinite. We extend Σ to a structure over the strings over this alphabet.

Definition 1. *Let Σ be an arbitrary structure of finite signature of the form $(A; d_1, \ldots, d_t; f_1, \ldots, f_u; r_1, \ldots, r_v, =)$ where each f_i is an n_{f_i}-ary function, r_i is an n_{r_i}-ary relation, and $A = \emptyset$, $t = 0$, $u = 0$, and/or $v = 0$ are possible. Let a and b be two new constants with $a, b \notin A$, let $\mathcal{A} = (A \cup \{a, b\})^*$ be the set of the strings over the alphabet $A \cup \{a, b\}$, and let ε be the symbol for the empty string. The concatenation of two strings s_1 and s_2 is denoted by $s_1 s_2$ as usual. For every $s \in \mathcal{A}$, let $|s|$ be the length (the number of the characters) of s. For any non-negative integer k and $\mathcal{S} \subseteq \mathcal{A}$, let $\mathcal{S}^{(\leq k)} = \{r \in \mathcal{S} \mid |r| \leq k\}$, $\mathcal{S}^{(=k)} = \{r \in \mathcal{S} \mid |r| = k\}$, and so on. For any unary relation Rel, let $\Sigma_{Rel} = Exp(\Sigma, Rel)$ be the structure*

$$(\mathcal{A}; d_1, \ldots, d_t, a, b, \varepsilon; f'_1, \ldots, f'_u, add, sub_{\mathrm{r}}, sub_{\mathrm{l}}; r'_1, \ldots, r'_v, Rel, =) \ .$$

$d_1, \ldots, d_t, a, b$, and ε are the only constants. add is a binary function for adding a character to a string. sub_{r} and sub_{l} are unary functions for computing the last character and the remainder of a string, respectively. That means that these functions are defined for the strings $s \in \mathcal{A}$, $r \in \mathcal{A} \backslash (A \cup \{a, b\})$, and $x \in A \cup \{a, b\}$ by $add(s, x) = sx$, $sub_{\mathrm{r}}(sx) = s$, $sub_{\mathrm{l}}(sx) = x$, $add(s, r) = \varepsilon$, $sub_{\mathrm{r}}(\varepsilon) = \varepsilon$, and $sub_{\mathrm{l}}(\varepsilon) = \varepsilon$.

For each $i \leq u$, let f'_i be an n_{f_i}-ary function of $\mathcal{A}^{n_{f_i}}$ into $A \cup \{\varepsilon\} \subset \mathcal{A}$. For all $i \leq u$ and for all $(s_1, \ldots, s_{n_{f_i}}) \in \mathcal{A}^{n_{f_i}}$, let

$$f'_i(s_1, \ldots, s_{n_{f_i}}) = \begin{cases} f_i(s_1, \ldots, s_{n_{f_i}}) & \text{if } (s_1, \ldots, s_{n_{f_i}}) \in A^{n_{f_i}} \\ \varepsilon & \text{otherwise} \end{cases}$$

be given. For all $i \leq v$, let r'_i be an n_{r_i}-ary relation on $\mathcal{A}$. Let $r'_i(s_1, \ldots, s_{n_{r_i}})$ be true in Σ_{Rel} if and only if $(s_1, \ldots, s_{n_{r_i}}) \in A^{n_{r_i}}$ and $r_i(s_1, \ldots, s_{n_{r_i}})$ is true in Σ.

Note that, if one of the relations of the structure Σ, for instance, r_1 is a unary relation satisfied by all elements of the universe A, then any formula of first order logic with respect to Σ which has the form $\exists Z(r_1(Z) \wedge \ldots)$ or $\forall Z(r_1(Z) \rightarrow \ldots)$ where all quantifiers are bounded by r_1 is valid in the constructed expansion if and only if it is valid in Σ.[1]

The satifiability problems can be defined by means of the following formulae analogous to [7].

Definition 2. *Let $\mathcal{F}$ be the set of quantifier-free $(\neg, \wedge, \vee)$-formulae of first order logic corresponding to the structures Σ_{Rel} such that $d_1, \ldots, d_t, a, b, \varepsilon$, $f'_1, \ldots, f'_u$, $add, sub_{\mathrm{r}}, sub_{\mathrm{l}}$, $r'_1, \ldots, r'_v$, and Rel are the symbols for the corresponding constants, functions, and relations and the formulae only contain literals of the form $add(Z_1, Z_2) = Z_3$, $sub_{\mathrm{r}}(Z_1) = Z_2$, $sub_{\mathrm{l}}(Z_1) = Z_2$, $f'_i(Z_1, \ldots, Z_{n_{f_i}}) = Z$, $Z = C$, $Z_1 = Z_2$, $Z_1 \neq Z_2$, $r'_i(Z_1, \ldots, Z_{n_{r_i}})$, $\neg r'_i(Z_1, \ldots, Z_{n_{r_i}})$, $Rel(Z)$, and $\neg Rel(Z)$ where $Z, Z_1, Z_2, \ldots$ stand for $X_1, X_2, \ldots, Y_1, Y_2, \ldots$ and C stands for $d_1, \ldots, d_t, a, b$ or ε.*

Definition 3. *Let code be an injective mapping of $\mathcal{F}$ into $\{a, b\}^+$ where every index i is represented by the string a^i and every other character of the strings $\psi \in \mathcal{F}$ is represented by other suitable strings in $\{a, b\}^{(=n_0)}$ for one fixed integer n_0. Moreover, for each formula ψ, let $code_{\mathrm{pad}}(\psi) = (code(\psi), \varepsilon, \ldots, \varepsilon) \in \mathcal{A}^{1+|code(\psi)|}$. For any positive integers l_0, l_1, let $\mathcal{F}_{l_0,l_1}$ be the set of all formulae $\psi \in \mathcal{F}$ which only contain components of $\boldsymbol{X} = (X_1, \ldots, X_{l_0})$ and $\boldsymbol{Y} = (Y_1, \ldots, Y_{l_1})$ as variables and for which $l_1 = |code(\psi)|$ holds. For every formula ψ containing only variables in $\{X_1, \ldots, X_{l_0}, Y_1, \ldots, Y_{l_1}\}$, we also write $\psi(\boldsymbol{X}, \boldsymbol{Y})$ instead of ψ.*

Let us sketch the definitions using the notations of [7].

Definition 4. *For every string $s \in \mathcal{A}$, let the value $\langle s \rangle$ be recursively defined by $\langle \varepsilon \rangle = a$, $\langle ra \rangle = \langle r \rangle a^2$, and $\langle rb \rangle = \langle r \rangle ba$ for all $r \in \mathcal{A}$. For each integer $n > 1$ and each tuple $\boldsymbol{s} = (s_1, \ldots, s_n) \in \mathcal{A}^n$, let $\langle s_1, \ldots, s_n \rangle$ be the string $\langle s_1 \rangle b^2 \cdots \langle s_n \rangle b^2 b$. Moreover, we write $\langle \boldsymbol{s}, r \rangle$ for $\langle s_1, \ldots, s_n, r \rangle$. Let $(\cdot)_{dbl}$ be the function which doubles the lengths of the strings such that, for every string $s \in \mathcal{A}$, the value s_{dbl} is defined by $s_{dbl} = sa^{|s|}$. Let us define $c_1 = code(X_1 = X_1)$ and $\mathcal{B} = \{\langle x, c_1 \rangle r \mid x \in \mathcal{A} \;\&\; r \in \mathcal{A}\}$.*

SAT_{Σ_R} will be defined by formulae of the form $\psi(\boldsymbol{X}, \boldsymbol{Y})$ and parameters in $\boldsymbol{x}$. Moreover, let us present the problems $\text{SUB-SAT}_{\Sigma_R} \subseteq \mathrm{SAT}_{\Sigma_R}$, $\text{RES-SAT}_{\Sigma_R} \subseteq \mathrm{SAT}_{\Sigma_R}$, and a unary variant $\text{RES-SAT}^{(1)}_{\Sigma_R}$ where the domain of parameters or the quantifier domain is restricted for each formula.

Definition 5. *For every unary relation Rel, let the* problems $\mathrm{SAT}_{\Sigma_{Rel}}$, $\text{SUB-SAT}_{\Sigma_{Rel}}$, $\text{RES-SAT}_{\Sigma_{Rel}} \subseteq \mathcal{A}^\infty$, *and* $\text{RES-SAT}^{(1)}_{\Sigma_{Rel}} \subseteq \mathcal{A}$ *be defined by*

[1] The remark with respect to the conservation of theories is a first answer to a question posed by Dimitri Grigoriev (EPIT 2005).

$$\begin{aligned}\mathrm{SAT}_{\Sigma_{Rel}} &= \{(\boldsymbol{x}, code_{\mathrm{pad}}(\psi)) \mid (\exists l_0, l_1 \in \mathbb{N}^+)(\exists \psi \in \mathcal{F}_{l_0,l_1} \ \&\ \boldsymbol{x} \in \mathcal{A}^{l_0} \\ &\quad \&\ \ \Sigma_{Rel} \models \exists \boldsymbol{Y} \psi(\boldsymbol{x}, \boldsymbol{Y}))\} \ ,\end{aligned}$$

$$\begin{aligned}\text{SUB-SAT}_{\Sigma_{Rel}} &= \{(\boldsymbol{x}, code_{\mathrm{pad}}(\psi)) \mid (\exists l_0, l_1 \in \mathbb{N}^+)(\exists \psi \in \mathcal{F}_{l_0,l_1} \\ &\quad \&\ \ \boldsymbol{x} \in (\mathcal{A}^{(\leq m_0 + \bar{m}_0)} \cup \mathcal{B}^{(\leq max_0)})^{l_0} \ \ \&\ \ \Sigma_{Rel} \models \exists \boldsymbol{Y} \psi(\boldsymbol{x}, \boldsymbol{Y}))\} \ ,\end{aligned}$$

$$\begin{aligned}\text{RES-SAT}_{\Sigma_{Rel}} &= \{(\boldsymbol{x}, code_{\mathrm{pad}}(\psi)) \mid (\exists l_0, l_1 \in \mathbb{N}^+)(\exists \psi \in \mathcal{F}_{l_0,l_1} \ \&\ \boldsymbol{x} \in \mathcal{A}^{l_0} \\ &\quad \&\ \ \Sigma_{Rel} \models (\exists \boldsymbol{Y} \in (\mathcal{A}^{(\leq m_1 + \bar{m}_1)} \cup \mathcal{B})^{l_1}) \psi(\boldsymbol{x}, \boldsymbol{Y}))\} \ ,\end{aligned}$$

$$\text{RES-SAT}^{(1)}_{\Sigma_{Rel}} = \{\langle \boldsymbol{x}, c \rangle_{dbl} \mid c \in \mathcal{A}^{(>0)} \,\&\, \exists \boldsymbol{e}((\boldsymbol{x}, c, \boldsymbol{e}) \in \text{RES-SAT}_{\Sigma_{Rel}} \,\&\, \boldsymbol{e} \in \{\varepsilon\}^{|c|})\}$$

where for each combination $l_0, l_1 > 0$ and for each $\boldsymbol{x} \in \mathcal{A}^{l_0}$,

$$\begin{aligned} m_0 &= m_0^{(l_0,l_1)} = (l_0+1)(l_1+1) - 2, & \bar{m}_0 &= \bar{m}_0^{(l_0,l_1)} = l_0(l_1+1) - 1, \\ m_1 &= m_{1,\boldsymbol{x}}^{(l_1,0)} = \max\{|x_1|, \ldots, |x_{l_0}|\} + l_1, & \bar{m}_1 &= \bar{m}_1^{(l_1,0)} = l_1 - 1 \ , \end{aligned}$$

and $max_0 = max_0^{(l_0,l_1)} = 4m_0 + 5\bar{m}_0 + 4|c_1| + 15$. Since we use these notations only in this context, we shall omit the indices (l_0, l_1),$(l_1, 0)$, and $\boldsymbol{x}$.

Definition 6. *Let $\hat{\mathcal{A}} = \{\langle \boldsymbol{x}, code(\psi) \rangle_{dbl} \mid (\exists l_0, l_1 \in \mathbb{N}^+)(\psi \in \mathcal{F}_{l_0,l_1} \ \&\ \boldsymbol{x} \in \mathcal{A}^{l_0})\}$ and $\mathcal{B}_1 = \{\langle x, c_1 \rangle_{dbl} \mid x \in \mathcal{A}\}$. Let R be the relation in $\bigcap_{k \in \mathbb{N}} \mathrm{REL}_k$ where*

$$\mathrm{REL} \ = \{Rel : \mathcal{A} \mapsto \{true, false\} \mid (\forall s \in \mathcal{B})(Rel(s) \leftrightarrow s \in \mathcal{B}_1)\} \ ,$$

$$\mathrm{REL}_0 = \{Rel \in \mathrm{REL} \mid (\forall s \in \mathcal{A} \setminus (\hat{\mathcal{A}} \cup \mathcal{B}))(Rel(s) = false)\} \ ,$$

$$\begin{aligned}\mathrm{REL}_k = \{&Rel \in \mathrm{REL}_{k-1} \mid (\forall l_0, l_1 \in \mathbb{N}^+)(\forall \boldsymbol{x} \in \mathcal{A}^{l_0})(\forall \psi \in \mathcal{F}_{l_0,l_1}) \\ &(|\langle \boldsymbol{x}, code(\psi) \rangle_{dbl}| = k \rightarrow (Rel(\langle \boldsymbol{x}, code(\psi) \rangle_{dbl}) \leftrightarrow \\ &(\exists \boldsymbol{y} \in (\mathcal{A}^{(\leq m_1 + \bar{m}_1)} \cup \mathcal{B})^{l_1})(\forall Rel' \in \mathrm{REL}_{k-1})(\Sigma_{Rel'} \models \psi(\boldsymbol{x}, \boldsymbol{y}))))\}\end{aligned}$$

for all $k > 0$.

3 A Tree Describing the Problem RES-SAT$^{(1)}_{\Sigma_R}$

The ideas for the definition of the relation R go back to the consideration of some tree T (see Fig. 1) which resembles a tree of computation paths. The tree $T = T_R$ describes the problem RES-SAT$^{(1)}_{\Sigma_R}$ containing the strings which satisfy R. The strings correspond to paths from the root of T to the leaves of T in reverse order. The nodes are labelled by the elements of the basic structure Σ or by a or b. The edges are labelled by $\mathcal{L}$ or $\mathcal{R}$ or they are not labelled. The corresponding paths are defined by $path_1$ and $path_2$ given below.

Let $path_1(\varepsilon)$ be the empty graph. For any $w \in \mathcal{A}$ and $x \in A \cup \{a, b\}$, let $path_1(wx)$ be the path with the root denoted by x and with the following properties. The root has an outgoing edge labelled by $\mathcal{R}$ iff $path_1(w)$ is not the empty graph. In this case this edge connects the root x to the root of $path_1(w)$.

For any integer $k > 0$ and any $w \in \mathcal{A}^{(=k-1)}$, let $path_2(wba^k)$ be the path consisting of the path $path_1(a^k)$, one additional edge labelled by $\mathcal{L}$, one additional node $node_k$ denoted by b, one additional non-labelled edge, and the path $path_1(w)$, in the given order.

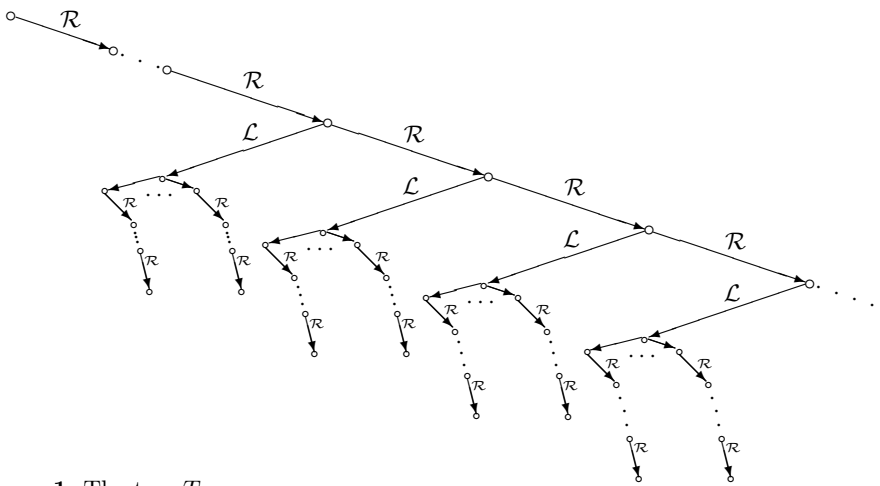


Figure 1. The tree T

Let *tree* T be an infinite connected directed circuit-free graph with the following properties.

1. There is exactly one infinite path P. All edges of this path are labelled by $\mathcal{R}$. For each integer $k > 0$, the kth node of P can have a second outgoing edge labelled by $\mathcal{L}$. This second edge incomes in a further node $node_k$ which is connected, for some strings $w \in \mathcal{A}^{(=k-1)}$, to the roots of the paths $path_1(w)$ by non-labelled edges.
2. For every string $v \in \mathcal{A}$, $v \in \text{RES-SAT}^{(1)}_{\Sigma_R}$ holds if and only if there are some positive integer k and some $w \in \mathcal{A}^{(=k-1)}$, such that $v = wba^k$ is valid and $path_2(v)$ is a subgraph of T containing the root of T and some leaf of T.

Note that the complete subtree with $node_k$ as root represents the strings of the length k which are codes of the tuples in $\text{RES-SAT}_{\Sigma_R}$.

In evaluating R we have a situation resembling the evaluation of a tree of computation paths of a BSS-machine or of a Σ_R-machine. If we are interested in the l first steps of such a machine, then we can make a cut after l nodes in each of the computation paths. For a special case we show that it is similar for our strings satifying R. We consider a simple combination of l strings $z_1, \ldots, z_l$ with $sub_1(z_i) = z_{i-1}$ for all $i \in \{2, \ldots, l\}$ and a conjunction $\psi(z_1, \ldots, z_l)$ with $\psi \in \mathcal{F}$. For a large string z_1 with $|z_1| \geq l - 1$, we only have to know whether $path_2(z_1)$ is a suitable subgraph of the following kind. If $path_2(z_1 a^{l'})$ is a subgraph of T containing the root of T and a leaf of T for some l', then $R(z_1 a^{l'})$ is true

and $R(z_1 a^{l''})$ is false for $l'' \neq l'$. Apart from the existence of such an l' with $l' < l$, the form of z_1 is not important for the truth value of the conjunction $\psi(z_1, \ldots, z_l)$. We could replace the string z_1 with $R(z_1 a^{l'}) = \mathit{true}$ by a small string p with $R(pa^{l'}) = \mathit{true}$. In each other case, instead of the large string z_1 we could also take a corresponding small string and we could replace the prefix z_1 of $z_2, \ldots, z_l$ in a suitable way described in [7] such that the truth value for the considered conjunction remains the same one in Σ_R. That means that in evaluating a formula with parameters belonging to the problem SAT_{Σ_R} we can also use this fact and reduce SAT_{Σ_R} to the problem $\mathrm{SUB\text{-}SAT}_{\Sigma_R}$, further to $\mathrm{RES\text{-}SAT}_{\Sigma_R}$, and to $\mathrm{RES\text{-}SAT}^{(1)}_{\Sigma_R}$.

4 P = NP for Σ_R

Formulae with parameters in any $\boldsymbol{x}$ remain true if we replace any large strings by small strings in $\mathcal{A}^{(\leq m_1 + \bar{m}_1)} \cup \mathcal{B}$ and $\mathcal{A}^{(\leq m_0 + \bar{m}_0)} \cup \mathcal{B}$ as in [7] such that the existence of same prefixes and the other basic properties of these prefixes which are relevant for satisfying formulae stay the common features. The relevant properties can be described by functions $\mathit{char}^{m_1}_{l_1,0}$ and $\mathit{char}^{m_0}_{l_0,l_1}$ as in [7]. The replacements are also possible by functions $\mathit{small}^{m_1}_{l_1,0}$ with the range $\mathcal{A}^{(\leq m_1 + \bar{m}_1)} \cup \mathcal{B}^{(\leq max_1)}$ and $\mathit{small}^{m_0}_{l_0,l_1}$ with the range $\mathcal{A}^{(\leq m_0 + \bar{m}_0)} \cup \mathcal{B}^{(\leq max_0)}$ as in [7] where $max_i = 4m_i + 5\bar{m}_i + 4|c_1| + 15$. The reason is that we get the following propositions and our main result analogous to [7].

Proposition 1. *For every $\boldsymbol{z} \in \mathcal{A}^l$, $\mathit{char}^m_{l,n}(\mathit{small}^m_{l,n}(\boldsymbol{z})) = \mathit{char}^m_{l,n}(\boldsymbol{z})$ holds.* □

Proposition 2. *For $l_0, l_1 \in \mathbb{N}^+$, let $\psi(X_1, \ldots, X_{l_0}, Y_1, \ldots, Y_{l_1})$ be a conjunction in $\mathcal{F}$.*

For all $\boldsymbol{x} \in \mathcal{A}^{l_0}$ and any $\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)} \in \mathcal{A}^{l_1}$ with $\mathit{char}^{m_1}_{l_1,0}(\boldsymbol{y}^{(1)}) = \mathit{char}^{m_1}_{l_1,0}(\boldsymbol{y}^{(2)})$, there holds

$$\Sigma_R \models \psi(\boldsymbol{x}, \boldsymbol{y}^{(1)}) \leftrightarrow \psi(\boldsymbol{x}, \boldsymbol{y}^{(2)}) \ .$$

For any tuples $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ in $\mathcal{A}^{l_0}$ with $\mathit{char}^{m_0}_{l_0,l_1}(\boldsymbol{x}^{(1)}) = \mathit{char}^{m_0}_{l_0,l_1}(\boldsymbol{x}^{(2)})$, there holds

$$\Sigma_R \models \exists \boldsymbol{Y} \psi(\boldsymbol{x}^{(1)}, \boldsymbol{Y}) \leftrightarrow \exists \boldsymbol{Y} \psi(\boldsymbol{x}^{(2)}, \boldsymbol{Y}) \ .$$

Proof. The proofs are as in [7]. However, the literals which contain the functions and the relations resulting from the basic structure have also to be considered. But, since we do not replace the elements of the basic structure, we can transfer the proof without difficulties. □

Theorem 1. *There holds $\mathrm{P}_{\Sigma_R} = \mathrm{NP}_{\Sigma_R}$.*

Proof. By Proposition 1 and Proposition 2 there holds $\mathrm{SAT}_{\Sigma_R} = \mathrm{RES\text{-}SAT}_{\Sigma_R}$ and the problem SAT_{Σ_R} can be reduced to $\mathrm{SUB\text{-}SAT}_{\Sigma_R}$. This implies $\mathrm{SUB\text{-}SAT}_{\Sigma_R} \subset \mathrm{RES\text{-}SAT}_{\Sigma_R}$ and, consequently, that $\mathrm{SUB\text{-}SAT}_{\Sigma_R}$ can be reduced to a subset of $\mathrm{RES\text{-}SAT}^{(1)}_{\Sigma_R}$. The reductions are even possible in polynomial time.

Since $(\forall s \in \mathcal{A})(R(s) \leftrightarrow s \in \text{ RES-SAT}^{(1)}_{\Sigma_R})$ follows from the definition of R, RES-$\text{SAT}^{(1)}_{\Sigma_R}$ is decidable by means of R in constant time. Thus SAT_{Σ_R} is decidable in polynomial time. The fact that every problem in NP_{Σ_R} can be reduced to SAT_{Σ_R} in polynomial time implies the assumption. □

References

1. Blum, L., F. Cucker, M. Shub, and S. Smale, *Complexity and Real Computation.* Springer-Verlag (1998).
2. Blum, L., M. Shub and S. Smale, 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 (1989), 1–46.
3. Cucker, F. and M. Matamala, On digital nondeterminism. *Math. Systems Theory* 29 (1996), 635–647.
4. Cucker, F., M. Shub, and S. Smale, Separation of complexity classes in Koiran's weak model. *Theoretical Computer Science* 133 (1994), 3–14.
5. Gassner, C., The P-DNP problem for infinite abelian groups. *Journal of Complexity* 17 (2001), 574–583.
6. Gassner, C., NP ⊂ DEC und P = NP für Expansionen von Erweiterungen von Strukturen endlicher Signatur mit Identitätsrelation. *Preprint* 13 (2004).[2]
7. Gassner, C., A structure of finite signature with identity relation and with P = NP - A formal proof with more details. *Preprint* 9 (2005).[2] (The abridged version appears in *Computer Science Report Series of the University of Wales Swansea.*)
8. Hemmerling, A., Computability and complexity over structures of finite type. *Preprint* 2 (1995).[2]
9. Koiran, P., Computing over the reals with addition and order. *Theoretical Computer Science* 133 (1994), 35–47.
10. Meer, K., A note on a P ≠ NP result for a restricted class of real machines. *Journal of Complexity* 8 (1992), 451–453.
11. Meer, K., Real number models under various sets of operations. *Journal of Complexity* 9 (1993), 366–372.
12. Poizat, B., *Les Petits Cailloux.* Aléa (1995).

[2] *Preprint-Reihe Mathematik, E.-M.-Arndt-Universität Greifswald*

On Complexity of Ehrenfeucht Theories with Computable Model*

Alexander N. Gavryushkin

Novosibirsk State University, Russia

Abstract. This work is devoted to investigation of complexity of theories with computable model and finite number of countable models up to isomorphism and also possibilities of location of computable model in the spectra of Ehrenfeucht theory.
In the work for all $m \in \omega$ there are examples of Ehrenfeucht theories of complexity $\mathbf{0}^{(m)}$ with computable model. Also there is example of theory with finite number of countable models which all models are computable but not decidable. And example of Ehrenfeucht theory which has uncomputable prime and saturated models but one model of the theory has computable presentation.

1 Introduction

One of the interesting problems in modern constructive model theory is problem of existence of computable model of elementary theory. Let a theory T has computable model. Then what is its algorithmic complexity? If a theory T has computable model then what about model theoretic properties of such model?

Let now give some definitions and results which we need in this work. Basic definitions and denotions are from [1], [7], [8].

In this paper we'll identify sentences with their Gödel numbers i. e. a phrase like "complexity of the theory T" means "complexity of set of Gödel numbers of all sentences of the theory T".

Let L be computable language.

Definition 1. *Algebraic system $\mathfrak{M}$ of L is* **computable** *if its universum, basic functions and predicates are uniformly computable.*

Definition 2. *Algebraic system $\mathfrak{M}$ of L* **has computable presentation** *if it is isomorphic to any computable model.*

Definition 3. *Let $\omega(T)$ be a number of countable models of the theory T up to isomorphism. Theory T is* **Ehrenfeucht** *theory if $3 \leq \omega(T) < \omega$.*

Theorem 1. (Peretyat'kin[6]). *For all $n \in \omega$, $n \geq 3$, there exists complete decidable theory T_n of finite signature which has n countable models up to isomorphism. Only the prime model of T_n is computable.*

* The work was partially supported by President Grant – 4413.2006.1

Theorem 2. (Khoussainov, Nies, Shore[4]). *There exists a theory with three countable models up to isomorphism such that only saturated model of the theory has computable presentation.*

Theorem 3. (Goncharov, Khoussainov[3]). *For any natural number $n \geq 1$ there exists ω-categorical theory T of finite signature such that T is equivalent to $\mathbf{0}^{(n)}$.*

Let L be finite language without functional symbols and let $\mathfrak{A} =< A; P_0^{n_0}, \ldots, P_m^{n_m} >$ is a structure of L. For all predicates P of the structure sets $A^k \setminus P$ and P are infinite where k is arity of P.

Let X be infinite set such that $A \cap X = \varnothing$.

Definition 4. *Predicate $P_\exists$ is* **Marker $\exists$-expansion** *of predicate P if next properties hold:*

1. *If $P_\exists(a_1, \ldots, a_{k+1})$ then $P(a_1, \ldots, a_k)$ and $a_{k+1} \in X$.*
2. *For all $a \in X$ there exists the unique $(a_1, \ldots, a_k) \in A^k$ such that $P_\exists(a_1, \ldots, a_k, a)$.*
3. *If $P(a_1, \ldots, a_k)$ then there exists the unique a such that $P_\exists(a_1, \ldots, a_k, a)$.*

Definition 5. *Predicate $P_\forall$ is* **Marker $\forall$-expansion** *of predicate P, if next properties hold:*

1. *If $P_\forall(a_1, \ldots, a_{k+1})$, then $a_1, \ldots, a_k \in A$ and $a_{k+1} \in X$.*
2. *For all $(a_1, \ldots, a_k) \in A^k$ there exist not more then one element $a_{k+1} \in X$ such that $\neg P_\forall(a_1, \ldots, a_{k+1})$.*
3. *If $P_\forall(a_1, \ldots, a_{k+1})$ for all $a_{k+1} \in X$, then $P(a_1, \ldots, a_k)$.*
4. *For all $a \in X$ there exists $(a_1, \ldots, a_k) \in A^k$ such that $\neg P_\forall(a_1, \ldots, a_k, a)$.*

Definition 6. *The set X in any $\exists$- or $\forall$-expansion is* **companion** *of P.*

Definition 7. *Let $\mathfrak{A} =< A; P_0^{n_0}, \ldots, P_m^{n_m} >$ is model. Let*

$$\mathfrak{A}_\exists \leftrightharpoons < A \cup \bigcup_{i=0}^{m} X_i, P_0^{n_0+1}, \ldots, P_m^{n_m+1}, X_0, \ldots, X_m >,$$

$$\mathfrak{A}_\forall \leftrightharpoons < A \cup \bigcup_{i=0}^{m} Y_i, Q_0^{n_0+1}, \ldots, Q_m^{n_m+1}, Y_0, \ldots, Y_m >,$$

where every predicate $P_i^{n_i+1}$ is Marker $\exists$-expansion of $P_i^{n_i}$ with companion X_i, $i \in (m+1)$, every $Q_i^{n_i+1}$ is Marker $\forall$-expansion of $P_i^{n_i}$ with companion Y_i, $i \in (m+1)$. $X_i \cap X_j = \varnothing$ and $Y_i \cap Y_j = \varnothing$ for $i \neq j$. Then $\mathfrak{A}_\exists$ and $\mathfrak{A}_\forall$ are **Marker $\exists$- and $\forall$-expansions** *of model $\mathfrak{A}$ respectively.*

Marker expansions let us to expand the basic model inductive next way. Let $\mathfrak{A}$ be a model and w be a word in the alphabet $\{\exists, \forall\}$. Define $\mathfrak{A}_w$ by induction. If w is empty then let $\mathfrak{A}_w = \mathfrak{A}$. If $w = w'\exists$ or $w = w'\forall$ and $\mathfrak{B} = \mathfrak{A}_{w'}$ then $\mathfrak{A}_{w'\exists} = \mathfrak{B}_\exists$ and $\mathfrak{A}_{w'\forall} = \mathfrak{B}_\forall$.

Theorem 4. (Goncharov, Khoussainov[3]). *If a model* $\mathfrak{A}$ *is* X'*-computable then* $\mathfrak{A}_{\exists\forall}$ *is* X*-computable.*

In [2] there is a proof of correctness Marker expansions application in Ehrenfeucht theories case i. e.

$$\mathfrak{N}_\exists \equiv \mathfrak{M}_\exists \Leftrightarrow \mathfrak{N} \equiv \mathfrak{M} \Leftrightarrow \mathfrak{N}_\forall \equiv \mathfrak{M}_\forall,$$

moreover T, $T_\exists$ and $T_\forall$ has the same number of countable models up to isomorphism.

2 The Result

Let $\mathfrak{M}_1 =< M_1; \Sigma_1 >$ and $\mathfrak{M}_2 =< M_2; \Sigma_2 >$ are countable models of signatures Σ_1 and Σ_2 respectively such that $\Sigma_1 \cap \Sigma_2 = \varnothing$, there are no functional symbols in Σ_1 and Σ_2, $M_1 \cap M_2 = \varnothing$.

Definition 8. *Let* $\mathfrak{M} =< M; \Sigma_1, \Sigma_2, P_1^1, P_2^1 >$ *where* $M = M_1 \cup M_2$, $P_i(x) \leftrightarrow x \in M_i$, $i = 1, 2$, *values of constants from* Σ_1 *and* Σ_2 *are the same as it was in* $\mathfrak{M}_1$ *and* $\mathfrak{M}_2$, *predicates from* Σ_1 (Σ_2) *let be false on any tuples not from* M_1 (M_2). *Model* $\mathfrak{M}$ *is* **disjunctive amount** *of models* $\mathfrak{M}_1$ *and* $\mathfrak{M}_2$. *Denote it* $\mathfrak{M} = \mathfrak{M}_1 \oplus \mathfrak{M}_2$.

Let T_1 and T_2 be theories of signatures Σ_1 and Σ_2 respectively. $\mathfrak{N}_i \models T_i$, $i = 1, 2$.

Definition 9. *Elementary theory of model* $\mathfrak{N}_1 \oplus \mathfrak{N}_2$ $Th(\mathfrak{N}_1 \oplus \mathfrak{N}_2)$ *is* **disjunctive amount** *of* T_1 *and* T_2. *Denote it* $T_1 \oplus T_2$.

Definition 9 is correct i. e. it isn't depend on selection of models. The proof of this fact there is in [2]. Also in [2] there is next clear fact: $T_1 \oplus T_2$ is $deg(T_1) \oplus deg(T_2)$-computable.

Proposition 1. *For all* $m \in \omega$ *there exists Ehrenfeucht theory* $T_1^{(m)}$ *with computable model such that* $T_1^{(m)} \equiv_T \mathbf{0}^{(m)}$.

$\lhd$ Len m be a natural number, T_e be Ehrenfeucht theory which only prime model has computable presentation (see 1). And T_ω be ω-categorical theory such that $T_\omega \equiv \mathbf{0}^{(m)}$ (see 3). In this case $T' = T_e \oplus T_\omega$ will be $\mathbf{0}^{(m)}$-computable Ehrenfeucht theory. And let $T_1^{(m)} = (T')_w$ where $w = \exists\forall \ldots \exists\forall$, $|w| = 2m$. There exists computable model of $T_1^{(m)}$ because of 4. Inasmuch as T' is definable in $T_1^{(m)}$ and T' is $deg(T_e) \oplus deg(T_\omega) = \mathbf{0}^{(m)}$-computable as $T_1^{(m)} \equiv_T \mathbf{0}^{(m)}$. $T_1^{(m)}$ is Ehrenfeucht theory because of the note in the end of previous section. $\rhd$

Proposition 2. *There exists Ehrenfeucht theory* T_2 *which all models are computable but not decidable.*

$\lhd$ Let $T_1 = T_1^{(m)}$ for any $m \geq 1$. Note that there exists $k \in \omega$ such that for all $\mathfrak{M} \models T_1$ $\mathfrak{M} \leq_T \mathbf{0}^{(k)}$. Considere $v = \exists\forall \ldots \exists\forall$, $|v| = 2k$, and $T_2 = (T_1)_v$. If $\mathfrak{N} \models T_2$ then there exists $\mathfrak{A} \models T_1$ such that $\mathfrak{N} \cong \mathfrak{A}_v$. Inasmuch as $\mathfrak{A}_v$ is computable by 4 as $\mathfrak{A}_v$ is computable presentation of $\mathfrak{N}$.

Note that for all $m \in \omega$ one can choose T_2 to be $\mathbf{0}^{(m)}$-computable but not $\mathbf{0}^{(m-1)}$-computable. And if $\mathfrak{N} \models T_2$ is decidable then T_2 should be decidable too but it isn't so. $\rhd$

Proposition 3. *There exists Ehrenfeucht theory T_3 with only one computable model such that neither prime nor saturated model of T_3 has computable presentation.*

$\lhd$ Let T_e be Ehrenfeucht theory which only computable model is prime (see 1) and T_s be a theory with three countable models which only computable model is saturated (see 2). $T_3 = T_e \oplus T_s$.

Let $\mathfrak{A}_0 \models T_e$ is prime and $\mathfrak{B}_2 \models T_s$ is saturated model. Obviously $\mathfrak{A}_0 \oplus \mathfrak{B}_2$ is computable model of T_3. If all countable models of T_e are $Spec(T_e) = \{\mathfrak{A}_0, \ldots, \mathfrak{A}_{k-1}\}$, all countable models of T_s are $Spec(T_s) = \{\mathfrak{B}_0, \mathfrak{B}_1, \mathfrak{B}_2\}$ then it isn't hard to show that $Spec(T_3) = \{\mathfrak{A}_i \oplus \mathfrak{B}_j | i \in k, j \in 3\}$. Note that $\mathfrak{C} \models T_3$ is computable if and only if $\mathfrak{C} \cong \mathfrak{A}_0 \oplus \mathfrak{B}_2$. If $\mathfrak{A}_0 \oplus \mathfrak{B}_2$ is prime (saturated) then both $\mathfrak{A}_0$ and $\mathfrak{B}_2$ should be prime (saturated). Thus neither prime nor saturated model of T_3 has computable presentation. $\rhd$

References

1. Chang, C. C., Keisler, H. J., *Model Theory*, North-Holland, Amsterdam, 1990.
2. Gavryushkin, A. N., Complexity of Ehrenfeucht Models, the article is in *Algebra i Logika* journal editorial.
3. Goncharov, S., Khoussainov, B., Complexity of Categorical Theories with Computable Models, *Algebra and Logic*, 43, No. 6, 365–373, 2004.
4. Khoussainov, B., Nies, A., Shore, R., Computable Models of Theories with Few Models, *Notre Dame Journal of Formal Logic*, 38, 165–178, 1997.
5. Marker, D., Non-Σ_n-axiomatizable Almost Strongly Minimal Theories, *The Journal of Symbolic Logic*, 54, No. 3, 921-927, 1989.
6. Peretyat'kin, M. G., On Complete Theories with Finite Number of Countable Models, *Algebra i Logika*, 12, No. 5, 550-576, 1973.
7. Rogers, H. J., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, New York-Toronto-London, 1967.
8. Soare, R. I., *Recursively Enumerable Sets and Degrees. A Study of Computable Functions and Computably Generated Sets*, Soringer Verlag, Berlin-New York, 1987.

Functional Interpretation and Modified Realizability Interpretation of the Double-negation Shift

Philipp Gerhardy

BRICS[⋆], Department of Computer Science, University of Aarhus, Aabogade 34, DK-8200 Aarhus N, Denmark,peegee@daimi.au.dk

Abstract. In this paper, we investigate the relationship between the functional, resp. the modified realizability, interpretation of finite and infinite variants of the double-negation shift. The aim of these investigations is to compare and gain a better understanding of the existing (bar-recursive) interpretations of the full, infinite double-negation shift. In particular, we obtain a new modified realizability interpretation of the double-negation shift that is derived by extending a corresponding interpretation (derived from a simple proof of) the intuitionistic principle $\neg\neg P_0 \wedge \neg\neg P_1 \to \neg\neg(P_0 \wedge P_1)$ to the infinite case.

Key words: Proof theory, functional interpretation, modified realizability.

1 Introduction

In [1], Spector gives his famous consistency proof for classical analysis, using bar-recursion to give a computational interpretation (in the sense of Gödel's functional interpretation) of the double-negation shift

$$\textbf{DNS:}\ \ \forall x^0 \neg\neg P(x) \to \neg\neg \forall x^0 P(x),$$

for arbitrary formulas P. The negative translation of the axiom of countable choice is derivable in $\mathbf{WE\text{-}HA}^\omega$ + **CC** + **DNS**, weakly extensional Heyting arithmetic in all finite types + the axiom of choice + the double-negation shift. This allows Spector to prove that classical analysis $\mathcal{A}^\omega :\equiv \mathbf{WE\text{-}PA}^\omega$ + **CC**, weakly extensional Peano arithmetic in all finite types + the axiom of countable choice, has a functional interpretation in the (Spector) bar-recursive functionals, **T** + SBR, where **T** are the Gödel primitive recursive functionals and SBR is Spector's scheme of bar-recursion.

Bezem, Berardi and Coquand ([2]) and, inspired by this, Berger and Oliva ([3]), developed a modified realizability interpretation of the double-negation shift. The interpretation uses a variant of bar-recursion first defined in [2] and

⋆ **BRICS**, Basic Research in Computer Science, is funded by the Danish National Research Foundation.

coined modified bar-recursion (in short: MBR) by Berger and Oliva. However, the mr-interpretation of **DNS** using MBR is subject to important limitations, which we will discuss in greater detail below.

Such computational interpretations of classical analysis are of special interest as they allow one to extract effective realizers or bounds from ineffective proofs of $\forall\exists A_{qf}$ statements, where A_{qf} is quantifier-free. In recent years, the extraction of bounds from ineffective proofs in analysis using monotone variants of functional interpretation has shown to be particularly fruitful (see e.g. [4–6]). Both the functional interpretation and the mr-interpretation of arithmetic can be carried out by terms in Gödel's **T**. The interpretation of **DNS** requires some form of bar-recursion. For the extraction of bounds using monotone variants of such interpretations, it is crucial that the chosen form of bar-recursion and the interpretation of **DNS** is valid in Bezem's type structure $\mathcal{M}^\omega$ of the majorizable functionals ([7]). While Spector's interpretation of **DNS** is valid in $\mathcal{M}^\omega$ and SBR is effectively majorizable, the mr-interpretation of **DNS** using MBR uses a continuity principle that is not valid in $\mathcal{M}^\omega$. Moreover, even though MBR exists in $\mathcal{M}^\omega$ it is only majorizable ineffectively.

In this paper, we investigate a finite version of the double-negation shift:

$$\textbf{fDNS:}\ \ \forall k\big(\forall x \le k \neg\neg P(x) \to \neg\neg \forall x \le k P(x)\big).$$

Since this finite version of the double-negation shift follows by induction on k from the intuitionistic principle

$$\neg\neg P_0 \wedge \neg\neg P_1 \to \neg\neg(P_0 \wedge P_1),$$

it has both a functional interpretation and a modified realizability interpretation by *primitive* recursive (in the sense of Gödel) functionals.

These investigations are motivated by two main points:

1. Spector's solution of the functional interpretation of the full **DNS** is easy to verify, but the intuition behind the solution is non-trivial. By investigating the interpretation (both in the sense of functional interpretation and the mr-interpretation) of the finite version of **DNS**, we may get a better understanding of the interpretation of the full **DNS**.
2. As mentioned above, the mr-interpretation of **DNS** is subject to certain restrictions relative to Spector's solution. By comparing the different interpretations of the finite version of **DNS**, we hope to gain some insight into the differences between the two interpretations of **DNS** and maybe even lift some of the restrictions on the mr-interpretation of the full **DNS**.

The investigations carried out in this paper indicate that every computational interpretation of some proof of (the intuitionistic principle) **fDNS**, essentially $\neg\neg P_0 \wedge \neg\neg P_1 \to \neg\neg(P_0 \wedge P_1)$, contains the germ of a corresponding computational interpretation of full **DNS**. Spector's solution can be shown to be an extension to the infinite case of the functional interpretation of a simple intuitionistic proof of **fDNS**. From an mr-interpretation of that same proof we will obtain a

new mr-interpretation of the full **DNS**. However, this interpretation requires a very strong form of bar-recursion and is not feasible for e.g. program extraction. The mr-interpretation of **DNS** using MBR may be understood as an extension of the mr-interpretation of a different, more complicated proof of **fDNS**. The question is then, whether there is a proof of **fDNS** from which an idea for an mr-interpretation using only Spector's bar-recursion, which is weaker than all the other forms of bar-recursion considered, can be obtained.

Throughout the paper we will assume basic familiarity with Peano arithmetic in all finite types, as well as Gödel's functional interpretation[8], Kreisel's modified realizability interpretation[9] and their use in interpreting classical arithmetic and analysis. Given a functional C of type $0 \to \rho$, i.e. a sequence of objects of type ρ, we write $\langle C0, \dots, C(n-1)\rangle$ for the initial segment of length n of that sequence continued with the constant 0-element of type ρ. We may sometimes also write $\langle C0, C1, \dots\rangle$ for the infinite sequence C.

2 Spector's consistency proof

In this section, we briefly recall Spector's ingenious solution for the functional interpretation of the double-negation shift. First, let the Gödel $()^D$-transform of the formula $P(x)$ be $\exists a \forall b P(x,a,b)$, then we need to solve the functional interpretation of

$$\forall x \neg\neg \exists a \forall b P(x,a,b) \to \neg\neg \forall y \exists c \forall d P(y,c,d), \tag{1}$$

(recall that the types of x, y are 0 (i.e. natural numbers)) of which the $()^D$-transform is

$$\forall A, Y, D \exists x, B, C \big(P(x, A(x,B), B(A(x,B))) \to P(Y(C), C(Y(C)), D(C))\big), \tag{2}$$

which means we wish to obtain x, B, C as functionals in A, Y, D solving the system of equations

$$x = Y(C), \qquad A(x,B) = C(Y(C)), \qquad B(A(x,B)) = D(C). \tag{3}$$

Here, x is of type 0, C of type $0 \to \rho$ for some type ρ and B is of type $\rho \to \tau$ corresponding to the type $(0 \to \rho) \to \tau$ of D for some type τ; the types of Y and A are $(0 \to \rho) \to 0$ and $0 \to ((\rho \to \tau) \to \rho)$ respectively.

Spector solves these equations with the following special form of bar-recursion (which is primitive recursively definable in SBR):

$$\varphi(x, C, n) = \begin{cases} Cn & \text{if } n < x, \\ \mathbf{0} & \text{if } n > x \wedge Y(\langle C0, \dots C(x-1)\rangle) < x, \\ \varphi(x+1, \langle C0, \dots C(x-1), a_0\rangle) & \text{otherwise .} \end{cases}$$

where $a_0 = G_0(x, \lambda a.\varphi(x+1, \langle C0, \dots C(x-1), a\rangle))$, Y, G_0 are further parameters for φ, $\varphi(x, C) = \lambda n.\varphi(x, C, n)$ and $\mathbf{0}$ is the constant 0-functional of type ρ.

Defining:

- $G_0 = \lambda m, E.A(m, \lambda a.D(E(a))$,
- $C = \varphi(0, \langle\rangle)$, where $\langle\rangle$ denotes the empty sequence,
- $x = Y(C)$,
- $E_m = \lambda a.\varphi(m+1, \langle C(0), \dots C(x-1), a\rangle)$,
- $B_m = \lambda a.D(E_m(a))$,

one easily checks that x, B_x and C as defined above solve the equations (3). Likewise, one easily checks that there is no circularity in the definitions, as each functional only depends on the parameters A, Y, D, the bar-recursor φ or functionals already defined above. Finally, assuming that all functionals Y of type $(0 \to \rho) \to 0$ satisfy the following well-foundedness condition

$$\text{(WF):}\ \ \forall C^{0\to\rho}\exists n^0(Y(\langle C0, \dots C(n-1)\rangle) < n),$$

the solution is well-defined, as eventually the stopping condition $Y(\langle C0, \dots C(x-1)\rangle) < x$ will be true. The above condition (WF) holds both for the continuous functionals (see e.g. [10, 11]) and the majorizable functionals (see [7]), which both are models of the bar-recursive functionals $\mathbf{T}$ + SBR.

3 Functional interpretation of the finite double-negation shift

In section 10 of [1], Spector suggests that there is a much simpler interpretation of the following finite version of the double-negation shift:

$$\forall k\big(\forall x \le k \neg\neg\exists a\forall b P(x,a,b) \to \neg\neg\forall y \le k\exists c\forall d P(y,c,d)\big). \tag{4}$$

which corresponds to having an a-priori bound k on $Y(C)$ for all sequences C.

Finite **DNS** is provable in Heyting arithmetic by induction on k and hence has a functional interpretation by primitive recursive functionals. The argument is the following: For $k = 0$ the proof is trivial. For the induction step $k \Rightarrow k+1$ we heavily use that

$$(\neg\neg\exists a_0\forall b_0 P_0 \wedge \neg\neg\exists a_1\forall b_1 P_1) \to \neg\neg(\exists c_0\forall d_0 P_0 \wedge \exists c_1\forall d_1 P_1) \tag{5}$$

is intuitionistically provable and hence has a functional interpretation by primitive recursive functionals.

However, one may also obtain an interpretation of **fDNS** using the fact that the (Spector) bar-recursive definition can be completely evaluated (i.e. yielding a primitive recursive functional) if we have an a-priori bound k on $Y(C)$ for all C. We compare these two approaches by extracting the functional realizers for the case $k = 2$, i.e. compare the two-element sequence C obtained by bar-recursion with the realizing terms for c_0 and c_1 when interpreting an intuitionistic proof of (5).

The full $()^D$-translation of the equation (5) yields the following functional equations:

$$A_0(t_{B_0}) = t_{C_0}, \qquad t_{B_0}(A_0(t_{B_0})) = D_0(t_{C_0}, t_{C_1}),$$
$$A_1(t_{B_1}) = t_{C_1}, \qquad t_{B_1}(A_1(t_{B_1})) = D_1(t_{C_0}, t_{C_1}).$$

The intuition behind a solution for these equations is as follows: Assume we have already produced a solution for t_{C_0}. Then t_{B_1} has to be a function that given $t_{C_1} = A_1(t_{B_1})$ has the same value as $D_1(t_{C_0}, t_{C_1})$, so the obvious solution is $t_{B_1} = \lambda c.D_1(t_{C_0}, c)$. If we then set $t_{C_1} = A_1(t_{B_0})$ the lower two equations are satisfied. The solutions for t_{B_0}, t_{C_0} can be obtained in a similar way: For t_{B_0} we construct a function that passes any solution for t_{C_0} to D_0, i.e. $\lambda a.D_0(a, [])$, but now we need to explicitly construct a potential solution for t_{C_1} to plug into [] as well. However, within the scope of λa we have a potential realizer a for t_{C_0} available, so we just repeat the above construction for t_{C_1}, only this time with a for t_{C_0}.

Based on e.g. the intuitionistic proof of $\neg\neg P_0 \wedge \neg\neg P_1 \to \neg\neg(P_0 \wedge P_1)$ in [12], one may – employing Gödel's functional interpretation – obtain the following terms for t_{C_0} and t_{C_1}:

$$\begin{aligned} t_{C_0} &:= A_0(\lambda a.D_0(a, A_1(\lambda b.D_1(a, b)))), \\ t_{C_1} &:= A_1(\lambda c.D_1(t_{C_0}, c)), \end{aligned}$$

which exactly implement the intuitive solution sketched above.

Alternatively, evaluating the bar-recursive definition C for $Y(C) < 2$, one obtains the following sequence C:

$$\langle A(0, \lambda a.D(\langle a, A(1, \lambda b.D(\langle a, b\rangle))\rangle)),\ A(1, \lambda c.D(\langle [C_0(0)], c\rangle))\rangle$$

where again for simplicity we write $[C_0(0)]$ in the definition of the second element instead of repeating the first element of C_0 in detail.

Note, that the approach via interpreting $\neg\neg P_0 \wedge \neg\neg P_1 \to \neg\neg(P_0 \wedge P_1)$ and the approach via evaluating the bar-recursive solution for $Y(C) < 2$ – modulo some renaming – result in exactly the same realizing terms. This shows that Spector's solution for the functional interpretation of **DNS** is an infinite, bar-recursive version of a specific primitive recursive interpretation of finite **DNS**. For the finite case we merely solve a finite number of equations. In the full case we have a sequence of equations, enumerated by possible values $0, 1, 2, \ldots$ of $x = Y(C)$, and hence we construct a corresponding sequence C. Since it is required that the functional Y satisfies the condition (WF), the bar-recursive construction of the sequence C eventually terminates, and the solution for the "initial segment" of the sequence of equations is then a solution for the functional interpretation of full **DNS**.

4 Modified realizability interpretation of the finite and infinite double-negation shift

Modified realizability interpretation differs conceptually from functional interpretation. Whereas functional interpretation reduces the task of witnessing a formula to solving a corresponding set of essentially logic-free functional equations, modified realizability asks to construct a functional realizer for the formula,

leaving the formula unchanged. Furthermore, the interpretation of *classical* systems requires the use of A-translation ([13, 14]) on top of negative translation, before mr-interpretation can be applied. For modified realizability a negative formula $\neg B := B \to \bot$ is realized by the empty realizer, as there is no realizer for $\bot$(false), and thus the modified realizability interpretation of the negative translation of a given formula B would not contain any useful information.

Thus interpreting the (A-translated) double-negation shift

$$\forall x^0((P^A(x) \to A) \to A) \to (\forall x^0 P^A(x) \to A) \to A$$

boils down to constructing a realizer for A from realizers:

$$\begin{array}{ll} G^{0\to((\rho\to *)\to *)} & \text{mr } \forall x^0((P^A(x) \to A) \to A) \\ Y^{(0\to\rho)\to *} & \text{mr } \forall x^0 P^A(x) \to A \end{array}$$

where ρ is the type of a realizer for the A-translation $P^A(x)$ of $P(x)$ and $*$ the type of a realizer of the formula A in the A-translation.

In [3], Berger and Oliva solve the modified realizability interpretation[1] of **DNS** under one significant restriction (relative to the solution for functional interpretation): The functional Y is assumed to be continuous, whereas for functional interpretation and Spector's solution it suffices that Y satisfies the condition (WF). Due to this restriction, the mr-interpretation of **DNS** is only valid for the continuous functionals, although the modified bar-recursor itself exists both in the continuous and the majorizable functionals. As mentioned earlier, the validity in the majorizable functionals is important for the extraction of effective bounds from ineffective proofs in classical analysis.

Moreover, the interpretation crucially depends on the fact that the formula $P^A(x)$ in the double-negation shift is the A-translation of some formula $P(x)$[2]. From this, one may derive the existence of a closed term $H^{*\to\rho}$ that satisfies $\forall n H$ mr $(A \to P^A(n))$, where H is independent of n. This is essentially the mr-interpretation of the A-translation of "ex falso quodlibet": $\bot \to P$ for arbitrary P. It should be clear that such an H cannot exist for general formulas $P(x)$.

The interpretation of the double-negation shift is then given using the following weak variant of modified bar-recursion([3]):

$$\Phi(\langle C0, \dots C(n-1)\rangle) =_0 Y(\langle C0, \dots C(n-1), a, a, \dots\rangle),$$

where $a = H(\langle C0, \dots C(n-1)\rangle, \lambda b.\Phi(\langle C0, \dots C(n-1), b\rangle))$. The functional Y is of type $(0 \to \rho) \to 0$ and H takes a finite sequence and a functional $\rho \to 0$ and

[1] In [3], Berger and Oliva employ a variant of the the mr-interpretation in which a term t realizes $\bot$ if it realizes a yet to be determined formula $P_\bot$. This can be viewed as directly incorporating the A-translation into the mr-interpretation. To avoid confusing the two variants we will base the discussion below on explicitly employing the A-translation.

[2] In the variant of the mr-interpretation employed by Berger and Oliva, the requirement is that all atoms in the formula P(x) in the double-negation shift occur negatively. This requirement is fulfilled when interpreting the negative translation of countable, resp. dependent choice.

produces an element of type ρ. Finally, the weak modified bar-recursor Φ takes a finite sequence of elements of type ρ and the parameters Y and H as its input and outputs a natural number.

Given the realizers Y and G from the mr-interpretation of **DNS** and the functional H satisfying $\forall n H$ mr $(A \to P^A(n))$, Berger and Oliva the present the following realizer for A:

$$\Phi(\langle C0, \ldots C(n-1)\rangle) = Y(\langle C0, \ldots C(n-1), a, a, \ldots\rangle),$$

where $a = H(G(n, \lambda b.\Phi(\langle C0, \ldots C(n-1), b\rangle)))$.

The realizer $\Phi(\langle\rangle)$ is verified using quantifier-free pointwise bar-induction relativized to the set $\mathcal{S}$ of sequences realizing $\forall x P(x)$ (for the precise definition of this scheme, see [3]). As mentioned above the proof depends on the continuity of Y and – to secure the existence of the functional H – the fact that $P^A(x)$ is the A-translation of some formula $P(x)$.

In fact, one may show that the realizing strategy for **DNS** using MBR basically corresponds to the following, more complicated proof of **fDNS**, i.e. $\neg\neg P_0 \wedge \neg\neg P_1 \to \neg\neg(P_0 \wedge P_1)$: We start with the assumptions (1) P_0, (2) P_1, (3) $\neg(P_0 \wedge P_1)$ and (4) $\neg\neg P_0 \wedge \neg\neg P_1$. From (1) and (2) we obtain $P_0 \wedge P_1$ and together with (3) we obtain $\bot$. Discharging the assumption (1) we obtain $\neg P_0$ and together with (4) we again get $\bot$. Now, one uses "ex falso quodlibet" to obtain P_0, and it is this which after A-translation leads to the application of the realizer H mentioned above. Symmetrically, we may dispose of the assumption (2), again obtaining a proof of $\bot$. Another application of "ex falso quodlibet" yields $P_0 \wedge P_1$ and together with (3) a proof of $\bot$. Two final implication introductions with (3) and (4) yield the proof of $\neg\neg P_0 \wedge \neg\neg P_1 \to \neg\neg(P_0 \wedge P_1)$ as desired. The mr-interpretation of **DNS** using MBR can thus be understood as an infinitary version of the mr-interpretation of (the A-translation of) this proof.

We now turn to an alternative, primitive recursive interpretation **fDNS**, obtained from the intuitionistic proof of (the A-translation) of $\neg\neg P_0 \wedge \neg\neg P_1 \to \neg\neg(P_0 \wedge P_1)$ in [12], from which Spector's solution can be obtained. More precisely, we here must construct a realizer for A from realizers:

$$\begin{array}{ll} G_0^{(\rho\to *)\to *} & \text{mr } (P_0 \to A) \to A \\ G_1^{(\rho\to *)\to *} & \text{mr } (P_1 \to A) \to A \\ Y^{(\rho\times\rho)\to *} & \text{mr } (P_0 \wedge P_1) \to A \end{array}$$

where $*$ is the type of a realizer for A. Again, strictly speaking P_0 and P_1 would here also be the A-translations of some formulas, but for the mr-interpretation to be given next, obtained from the proof in [12], this is not necessary:

Given x_0, x_1 realizing (arbitrary formulas) P_0 and P_1 we may obtain a realizer $Y(x_0, x_1)$ for A. Then $\lambda x_1.Y(x_0, x_1)$ realizes $P_1 \to A$ and $G_1(\lambda x_1.Y(x_0, x_1))$ realizes A. Doing the same for x_0 we obtain the realizer $G_0(\lambda x_0.G_1(\lambda x_1.Y(x_0, x_1)))$ for A. Finally, λ-abstracting G_0, G_1 and Y, we obtain the desired realizer for $\neg\neg P_0 \wedge \neg\neg P_1 \to \neg\neg(P_0 \wedge P_1)$. This strategy easily generalizes to the general finite cases of **DNS**.

Note, that while in the solution for full **DNS** by Berger and Oliva the overall strategy is to produce a sequence realizing $\forall x P(x)$ and then apply Y to this sequence, here the focal point of the realizing strategy is a nested application of the G_i's to one Y. Also note, that the finite solution does not depend on the P_i being the A-translation of some formula, but works for arbitrary $P(x)$.

Inspired by the above primitive recursive mr-interpretation for **fDNS**, we suggest an alternative solution for the interpretation of (the A-translation of) full **DNS**. The idea is to apply Y to a sequence of realizers for $P(x)$, and if Y is continuous this will only depend on a finite initial segment of such a sequence. Then with appropriate nested λ-abstractions and applications of $G_i = G(i, \cdot)$ we obtain the desired realizer.

We first define the following variant of bar-recursion, which we call *continuous* bar-recursion (in short: CBR):

$$\psi(n, C) = \begin{cases} Y(\langle C0, \dots, C(n-1)\rangle) & \text{if } n \text{ is a point of continuity for } Y \text{ on } C, \\ G(n, C, \lambda a.\psi(n+1, \langle C0, \dots C(n-1), a\rangle) & \text{otherwise .} \end{cases}$$

where Y, G are as above for the mr-interpretation of **DNS** and n is a point of continuity for Y on C, if for all sequences C' agreeing on the first n elements $Y(C) = Y(C')$. This is essentially Spector's bar-recursion, but with a stopping condition based on the continuity of Y, rather than Y satisfying (WF). This form of bar-recursion has previously been discussed in Kreisel's review([15]) of [16]. The stopping condition here is slightly ineffective. To be able to effectively determine the point of continuity of Y, we would have to explicitly ask for a continuous[3] modulus of continuity $\omega : (0 \to \rho) \to 0$ for Y, as a "modulus of continuity"-functional does not exist in all models of the continuous functionals (see [18]).

By defining an appropriate G' in terms of the pararameters Y, G, H for MBR, one easily sees that MBR is primitive recursively definable in CBR. Hence, as MBR is not S1-S9 computable in the continuous functionals (see [3]), neither is CBR. As in Kohlenbach's PhD-thesis [19], one may furthermore show that, similar to the variant of bar-recursion defined there (Def. 3.74, also discussed as Kohlenbach bar-recursion in [3]) the functional

$$\mu Y C := \min n[Y(\langle C0, \dots C(n-1)\rangle) = Y(C)],$$

is definable in CBR. Hence, CBR is not majorizable and thus strictly stronger than MBR, which is majorizable, although only ineffectively.

The proof that the mr-interpretation of **DNS** can be carried out using this variant of bar-recursion is very similar to the proof in [3]: Let $\mathcal{S}$ be the set of sequences realizing $\forall x P(x)$[4] and define

$$\begin{aligned} S(x, n) &:\equiv x \text{ mr } P(n), \\ R(n, C) &:\equiv \psi(n, \langle C0, \dots, C(n-1)\rangle) \text{ mr } A, \end{aligned}$$

[3] In [17], Kohlenbach shows that unless the modulus of continuity itself is continuous the point of continuity of a continuous function cannot be determined effectively.

[4] For a finite sequence, $\langle C0, \dots, C(n-1)\rangle \in \mathcal{S}$ means that the first n elements are realizers, i.e. $\forall i < n(C(i) \text{ mr } P(i))$.

then we prove that $R(0, \langle\rangle)$ by quantifier-free pointwise bar-induction.

(i) $\forall C \in \mathcal{S} \exists n R(n, C)$: Let $C \in \mathcal{S}$ be fixed and let n be the point of continuity for Y on C, then $\psi(n, C) = Y(\langle C0, \ldots, C(n-1)\rangle) = Y(C)$ and as $Y(C)$ mr A also $\psi(n, \langle C0, \ldots, C(n-1)\rangle)$ mr A and so $R(n, C)$.

(ii) The induction step consists of proving $\forall \langle C0, \ldots, C(n-1)\rangle \in \mathcal{S}(\forall a[S(a, n) \to R(n+1, \langle C0, \ldots C(n-1), a\rangle)] \to R(n, \langle C0, \ldots C(n-1)\rangle))$: Again, let the sequence $\langle C0, \ldots, C(n-1)\rangle \in \mathcal{S}$ be fixed. If n is already a point of continuity for Y on C, then trivially $R(n, \langle C0, \ldots C(n-1)\rangle)$ and we are done. Otherwise, suppose

$$\forall a[S(a, n) \to R(n+1, \langle C0, \ldots C(n-1), a\rangle)],$$

i.e. $\forall a[a \text{ mr } P(n) \to \psi(n+1, \langle C0, \ldots C(n-1), a\rangle) \text{ mr } A$. Then clearly

$$\lambda a.\psi(n+1, \langle C0, \ldots C(n-1), a\rangle) \text{ mr } P(n) \to A$$

and so

$$G(n, \lambda a.\psi(n+1, \langle C0, \ldots C(n-1), a\rangle) \text{ mr } A.$$

But $G(n, \lambda a.\psi(n+1, \langle C0, \ldots C(n-1), a\rangle) = \psi(n, \langle C0, \ldots C(n-1)\rangle)$ and so again $R(n, \langle C0, \ldots C(n-1)\rangle)$. From this quantifier-free bar-induction relativized to $\mathcal{S}$ yields $R(0, \langle\rangle)$ as desired.

In conclusion, one can ask why there is a direct correspondence between the functional interpretation of **fDNS** and full **DNS**, whereas for mr-interpretation, the extension of a quite simple interpretation of an intuitionistic proof of **fDNS** to the infinite case requires the (relatively) strongest and most impractical form of bar-recursion (compared to modified bar-recursion and Spector bar-recursion). At the same time, a better mr-interpretation of the full **DNS**, using MBR, is related to a more complicated strategy for interpreting **fDNS**, obtained from a more complicated proof of **fDNS**. Thus, one can speculate whether there is any proof of **fDNS** which would yield an mr-interpretation of **DNS** that only requires Spector's bar-recursion, or, more pessimistically, whether such an mr-interpretation of **DNS** is possible at all. An answer to that question would certainly be of interest, both for the extraction of programs from proofs as well as for the general comparison of the two existing main computational interpretations of full classical analysis.

5 Acknowledgements

The author would like to thank U. Kohlenbach for several discussions on the subject of this paper and for proof-reading an earlier draft version of this paper. Likewise, the author would like to thank P. Oliva for useful comments on a draft version of this paper. Finally, the author would like to thank the reviewers who made many useful suggestions for improving the presentation of this paper.

References

1. Spector, C.: Provably recursive functionals of analysis : a consistency proof of analysis by an extension of principles formulated in current intuitionistic mathematics. In Dekker, J., ed.: Proceedings of Symposia in Pure Mathematics. Volume 5., AMS, Providence, R.I. (1962) 1–27
2. Berardi, S., Bezem, M., Coquand, T.: On the computational content of the axiom of choice. J. of Symbolic Logic **63** (1998) 600–622
3. Berger, U., Oliva, P.: Modified bar recursion and classical dependent choice. In Baaz, M., Friedman, S.D., Krajcek, J., eds.: Proceedings of the Annual European Summer Meeting of the Association for Symbolic Logic, Vienna, Austria. Volume 20 of Lecture Notes in Logic., ASL and A K Peters (2001) 89–107
4. Kohlenbach, U., Oliva, P.: Proof mining: a systematic way of analyzing proofs in mathematics. Proc. Steklov Inst. Math **242** (2003) 136–164
5. Kohlenbach, U.: Some logical metatheorems with applications in functional analysis. Trans. Amer. Math. Soc. **357** (2005) 89–128
6. Gerhardy, P., Kohlenbach, U.: General logical metatheorems for functional analysis (2005) Submitted, 42pp.
7. Bezem, M.: Strongly majorizable functionals of finite type: a model of bar recursion containing discontinous functionals. J. of Symbolic Logic **50** (1985) 652–660
8. Gödel, K.: Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. Dialectica **12** (1958) 280–287
9. Kreisel, G.: Interpretation of analysis by means of constructive functionals of finite types. In Heyting, A., ed.: Constructivity in Mathematics. North-Holland Publishing Company, Amsterdam (1959) 101–128
10. Scarpellini, B.: A model for bar-recursion of higher types. Compositio Mathematica **23** (1971) 123–153
11. Troelstra, A.S., ed.: Metamathematical investigation of intuitionistic arithmetic and analysis. Volume 344 of Springer LNM. Springer-Verlag, Berlin (1973)
12. van Dalen, D.: Logic and Structure. Springer-Verlag, Berlin (1997)
13. Friedman, H.: Classical and intuitionistically provably recursive functions. In Müller, G., Scott, D., eds.: Higher Set Theory. Volume 669 of Springer LNM. Springer Verlag, Berlin (1978) 21–27
14. Dragalin, A.G.: New kinds of realizability and the Markov rule. Dokl. Akad. Nauk. SSSR **251** (1980) 534–537 English translation in: *Soviet Math. Dokl.*, 21: 461-464, 1980.
15. Kreisel, G.: Review of [16]. in Zentralblatt MATH (1976) ZBL 0312.02034.
16. Ershov, Y.L.: The model G of BR. Soviet Math., Doklady **15** (1975) 1158–1161 Translation of Doklady Akad. Nauk SSSR 217: 1004–1006 (Russian), 1974.
17. Kohlenbach, U.: Foundational and mathematical uses of higher types. In: Reflections on the Foundations of Mathematics: Essays in Honor of Solomon Feferman. Volume 15 of Lecture Notes in Logic. ASL and A K Peters (2002) 92–116
18. Kreisel, G.: On weak completeness of intuitionistic predicate logic. J. of Symbolic Logic **27** (1962) 139–158
19. Kohlenbach, U.: Theorie der stetigen und majorisierbaren funktionale und ihre anwendung bei der extraktion von schranken aus inkonstruktiven beweisen: Effektive eindeutigkeitsmodule bei besten approximationen aus ineffektiven eindeutigkeitsbeweisen. PhD thesis, Frankfurt, pp. xxii+278 (1990)

Toward Combinatorial Proof of P<NP

Basic Approach

L. Gordeev, FTICA

Logik u. Sprachtheorie, WSI f. Informatik, Tübingen University
gordeew@informatik.uni-tuebingen.de

1 Introduction

1.1 Summary

We present a plausibe "school-algebraic" condition C_0 that infers, in Peano Arithmetic, the negative solution (abbr.: $\mathcal{P} < \mathcal{NP}$) to the familiar open problem $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ (cf. e.g. [C], [F], [P], [T]). C_0 expresses that a slight modification of the ordinary DNF conversion algorithm needs exponential size inputs in order to produce a certain big and complex output. This output is explicitly defined and its structure can be analyzed by standard methods of asymptotic combinatorics in order to achieve a desired proof of C_0. C_0 also admits purely combinatorial tree-presentation. We believe that our approach might accelerate fulfillment of Harvey Friedman's prophecy: *"2050, $\mathcal{P} \neq \mathcal{NP}$. Detailed combinatorial work on easier problems, leading up to the full result"* (see [F]).

1.2 Background

Boolean space Let $\mathbb{B}^n$ be the space of $n-$dim boolean polynomials. Polynomials (or circuits) containing only positive literals are called *positive*. Polynomials having the same boolean values as positive polynomials are called *semi-positive*. We recall a familiar "school-algebra" algorithm $f \mapsto \mathtt{BASE}(f)$ (see Chapter 2.2 below) such that for every semi-positive input f, $\mathtt{BASE}(f)$ is the uniquely determined minimal positve DNF having the same values as f. Any such $\mathtt{BASE}(f)$ is also referred to as a *basic polynomial*. For any basic polynomial g, denote by $\mathtt{BASE}^{-1}(g)$ the set of semi-positive polynomials f such that $\mathtt{BASE}(f) = g$ (mod boolean commutative and associative laws). We pose a following question:

Q : Given a big and complex basic polynomial g, how big must be every $f \in \mathtt{BASE}^{-1}(g)$?

Since nontrivial valid DNF are not semi-positive, $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ is not related to Q. However, an appropriate analytic upgrade provides us with the desired link.

Borel space Keeping this in mind we consider a suitable real space of n^2-dim boolean-valued Borel polynomials, $\mathbb{B}_0^{n^2}$, whose variables are ranging over the non-zero part of the real continuum, $\mathbb{R}_{\neq 0}$. We elaborate basic algebraic theory along the lines of $\mathbb{B}^n$ (as above), except that in the Borel case, our proof tools are geometrical by nature (cf. Chapter 3.2 below). In particular, we generalize boolean "school-algebra" algorithm BASE and establish its basic properties (see Chapter 3.4 below); let $\mathtt{BASE}_0$ denote the corresponding Borel generalization. Our Borel modification of Q reads as follows, in $\mathbb{B}_0^{n^2}$:

Q_0: Given a big and complex basic Borel polynomial g, how big must be every $f \in \mathtt{BASE}_0^{-1}(g)$?

We observe that the whole DNF family of $\mathbb{B}^n$ (call it $\mathbb{D}_n$) is representable in $\mathbb{B}_0^{n^2}$ by one "universal" positive CNF Φ_n; such Φ_n is explicitly defined (see Definition 3 below). Furthermore, we observe that $\mathtt{BASE}_0(\Phi_n)$ is a DNF whose clauses are characteristic to minimal valid elements of $\mathbb{D}_n$ (see Chapter 5 below); it is readily seen that $\mathtt{BASE}_0(\Phi_n)$ is big and complex, relative to n. We conjecture that the following C_0 answers the corresponding specification of Q_0.

Definition 1. *Denote by C_0 the condition "for every $c \in \mathbb{N}$ there is a $n \in \mathbb{N}$ so large that the size of any basic polynomial f, in $\mathbb{B}_0^{n^2}$, satisfying $\mathtt{BASE}_0(f) = \mathtt{BASE}_0(\Phi_n)$ (mod boolean commutative and associative laws), is bigger than n^c ".*

Next we observe that the hypothesis $\mathcal{P} = \mathcal{NP}$ is actually stronger than its "naive" discrete translation. Loosely speaking, $\mathcal{P} = \mathcal{NP}$ infers that Φ_n can be characterized by equivalent polynomial-size algebraic polynomials f. The equivalence $\Phi_n \sim f$ in question usually refers to extensional equality $\Phi_n \sim_D f :\Leftrightarrow (\forall x \in D)(\Phi_n(x) = f(x))$ in a given discrete domain $D \subset \mathbb{N}$, that by standard encoding can be replaced by $\mathbb{N}$ (for brevity we identify $\subset$ with $\subseteq$). However, a closer look at the computing nature of $\mathcal{P} = \mathcal{NP}$ enables us to regard D as an arbitrary model of an appropriate simple algebra; in particular, we can set $D := \mathbb{R}_{\neq 0}$ (see [GK] for more exhaustive algebraic elaboration). Now clearly the condition $\Phi_n \sim_{\mathbb{R}_{\neq 0}} f$ is more restrictive than $\Phi_n \sim_{\mathbb{N}} f$, and hence passing from $\mathcal{P} = \mathcal{NP}$ to $\Phi_n \sim_{\mathbb{R}_{\neq 0}} f$, instead of $\Phi_n \sim_{\mathbb{N}} f$, can provide us with more insight into the structure of any hypothetical f in question. Actually, from $\Phi_n \sim_{\mathbb{R}_{\neq 0}} f$ (that we in the sequel denote by $\Phi_n \sim_0 f$) we infer $\mathtt{BASE}_0(f) = \mathtt{BASE}_0(\Phi_n)$ (mod boolean commutative and associative laws) and thereby arrive (see Theorem 3 below) at

Conclusion 1 *C_0 infers $\mathcal{P} < \mathcal{NP}$.*

Remark 1. C_0 is not a lower bound problem - rather, it says that the set of valid $n-$dim boolean DNF is not representable by n^2-dim Borel polynomials whose size is polynomial in n. Moreover, C_0 admits purely combinatorial interpretation CS (see Appendix below). The restriction of CS to positive inputs f seems provable by purely combinatorial methods. This partial result reminds of the known separation $\mathcal{P} \neq \mathcal{NP}$ in the model of monotone circuits, but we believe

that our proof techniques (to be presented elsewhere) can be strengthened to the required combinatorial proof of C_0. That despite its transparency the underlying "school-algebraic" question Q_0 was not elaborated yet enables us to regard C_0 as a desired missing link between $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ and basic "working mathematics".

2 Boolean space $\mathbb{B}^n$

2.1 Preliminaries

In the sequel without loss of generality we assume that in boolean polynomials the negation $^-$ can be applied only to variables from the list $\mathfrak{v}_1, \cdots, \mathfrak{v}_n$. f and g are called *equivalent* (abbr.: $f \sim g$) iff they have the same boolean values $Val\,(f) = Val\,(g)$, for all boolean evaluations of the variables. Positive DNF (abbr.: PDNF) $f = \bigvee_{i=1}^{k} \bigwedge_{j=1}^{s_i} x_{i,j}$ and $g = \bigvee_{i=1}^{l} \bigwedge_{j=1}^{t_i} y_{i,j}$ are called *isomorphic* (abbr.: $f \cong g$), iff $k = l$ and $\{\{x_{i,j} \mid j = 1, \cdots, s_i\} \mid i = 1, \cdots, k\}$ and $\{\{y_{i,j} \mid j = 1, \cdots, t_i\} \mid i = 1, \cdots, l\}$ are equal sets. Moreover, a PDNF $f = \bigvee_{i=1}^{k} \bigwedge_{j=1}^{s_i} x_{i,j}$ is called *minimal* iff for $1 \leq q < r \leq k$, $X_q := [x_{q,j} \mid j = 1, \cdots, s_q]$, $X_r := [x_{r,j} \mid j = 1, \cdots, s_r]$ (the lists) are sets which are mutually incomparable with respect to $\subset$, i.e. $X_q \not\subseteq X_r \not\subseteq X_q$. Minimal PDNF are also called *basic polynomials* (abbr.: BP). Polynomials equivalent to positive (basic) polynomials are called *semi-positive* (*semi-basic*).

Lemma 1. *For any BP f and g, $f \sim g$ iff $f \cong g$.*

Proof. Suppose $f = \bigvee_{i=1}^{k} \bigwedge_{j=1}^{s_i} x_{i,j}$ and $g = \bigvee_{i=1}^{l} \bigwedge_{j=1}^{t_i} y_{i,j}$. Let
$Var_i\,(f) := \{x_{i,j} \mid j = 1, \cdots, s_i\}$ and $Var_i\,(g) := \{y_{i,j} \mid j = 1, \cdots, t_i\}$
and note that:
$Val\,(f) = 1$ iff $(\exists i = 1, \cdots, k)\,(\forall x \in Var_i\,(f))\,(Val\,(x) = 1)$,
$Val\,(g) = 1$ iff $(\exists i = 1, \cdots, l)\,(\forall y \in Var_i\,(g))\,(Val\,(y) = 1)$.
Hence

$$f \sim g \leftrightarrow \begin{pmatrix} (\forall i = 1, \cdots, k)\,(\exists \imath = 1, \cdots, l)\,(Var_i\,(f) \subset Var_\imath\,(g)) \\ \wedge\,(\forall \imath = 1, \cdots, l)\,(\exists i = 1, \cdots, k)\,(Var_i\,(g) \subset Var_i\,(f)) \end{pmatrix}$$

Now by the minimality of BP, we can replace above every $\subset$ by $=$ and arrive at $f \sim g \leftrightarrow f \cong g$, as required.

2.2 "School-algebra" conversions

Algorithm 1 : BASE. *For any boolean polynomial f we construct a BP* $\mathtt{BASE}\,(f)$ *in four subsequent steps, as follows (for brevity we rename $\vee$ and $\wedge$ by $+$ and $\cdot$, respectively, and also omit $\cdot$ as usual in school algebra).*

1. *Apply rewriting rules* $(t+s)\cdot r \hookrightarrow t\cdot r + s\cdot r$ *and* $r\cdot(t+s) \hookrightarrow r\cdot t + r\cdot s$ *to arbitrary subterms* r, s, t *so long as possible.*
2. *Delete all inconsistent clauses, i.e. maximal conjunctions containing both* x *and* $\overline{x}$, *for any variable* x.
3. *Delete all negative literals.*
4. *Delete all but minimal clauses, i.e. those whose sets of variables are proper extensions of other conjunctions' sets of variables.*
   Denote by $\mathtt{BASE}(f)$ *the output.*

Example 1. Let $f = ((x+y)\cdot(z+x)+(u+x)\cdot(x+\overline{x}))\cdot(y+\overline{y})$ and $g = (x+y)\cdot(z+x)+u+x$. Obviously g is positive and equivalent to f. Hence f is semi-positive.

1. Step 1 yields plain DNF conversion that is given by the familiar school-algebra conversion EXPAND:

$$\begin{aligned}\mathtt{EXPAND}(f) = {} & xzy + xz\overline{y} + xxy + xx\overline{y} + yzy + yz\overline{y} + xyy + xy\overline{y} \\ & + uxy + ux\overline{y} + u\overline{x}y + u\overline{x}\,\overline{y} + xxy + xx\overline{y} + x\overline{x}y + x\overline{x}\,\overline{y}\end{aligned}$$

2. Steps 1-2 upgrade EXPAND to EXPANDC by deleting all inconsistent clauses:

$$\begin{aligned}\mathtt{EXPANDC}(f) = {} & xzy + xz\overline{y} + xxy + xx\overline{y} + yzy + xyy + uxy + ux\overline{y} + \\ & u\overline{x}y + u\overline{x}\,\overline{y} + xxy + xx\overline{y}\end{aligned}$$

3. Steps 1-3 upgrade EXPANDC to EXPANDCP by further deleting all negative literals:

$$\begin{aligned}\mathtt{EXPANDCP}(f) = {} & xzy + xz + xxy + xx + yzy + xyy + uxy + ux + \\ & uy + u + xxy + xx\end{aligned}$$

4. Finally, we remove all repetitions and all but minimal products and arrive at the desired BP of f:

$$\mathtt{BASE}(f) = x + yz + u$$

Theorem 1. *The following holds for any semi-positive polynomials* f *and* g.

1. $\mathtt{BASE}(f)$ *is a BP*
2. $f \sim \mathtt{EXPAND}(f) \sim \mathtt{EXPANDC}(f) \sim \mathtt{EXPANDCP}(f) \sim \mathtt{BASE}(f)$
3. f *is semi-basic*
4. $f \sim g$ *infers* $\mathtt{BASE}(f) \cong \mathtt{BASE}(g)$, *and hence* $\mathtt{BASE}(f) = \mathtt{BASE}(g)$ *(mod boolean commutative and associative laws)*

Proof. 1. Straightforward by the definition (see the example).

2. It will suffice to verify that the passage from EXPANDC to EXPANDCP (see the example) preserves $\sim$. This can be shown by induction on the number of negative literals occurring in $\mathtt{EXPANDC}(f)$, as follows. Let $g = (\overline{x}\wedge y_1\wedge\cdots\wedge y_l)\vee$

$\bigvee_{i=1}^{k}\bigwedge_{j=1}^{s_i}\ell_{i,j}$, $g' := (y_1 \wedge \cdots \wedge y_l) \vee \bigvee_{i=1}^{k}\bigwedge_{j=1}^{s_i}\ell_{i,j}$, $h = \bigvee_{i=1}^{r}\bigwedge_{j=1}^{t_i}x_{i,j}$ such that $g \sim h$, where $x \notin \{y_1, \cdots, y_l\}$, since g is consistent. Clearly $Val(x) := 0$, $Val(y_1) = \cdots = Val(y_l) := 1$ and $Val(z) := 0$ for other variables z occurring in g, h yields $Val(g) = Val(h) = 1$. Hence there exists $a \leq r$ with $\{x_{a,1}, \cdots, x_{a,t_a}\} \subseteq \{y_1, \cdots, y_l\}$. Moreover, for any variable evaluation we have $Val(g') \geq Val(g) = Val(h)$. Suppose there is a one satisfying $Val(g') = 1 > 0 = Val(h)$. Since $Val(h) = Val(g)$, in that case we can just as well assume $Val(x) = 1$ and $Val(y_1) = \cdots = Val(y_l) = 1$, which infers $Val(x_{a,1} \wedge \cdots \wedge x_{a,t_a}) = 1 = Val(h)$ - a contradiction. Hence $g' \sim h$, Q.E.D.

3. Follows from 1 and 2.
4. Follows from 2 and Lemma 1.

Remark 2. For arbitrary f, the conversion to $\texttt{BASE}(f)$ must not necessarily preserve boolean equivalence. This is because $\texttt{EXPANDCP}$ does not preserve $\sim$ in the whole polynomial domain. For example, $\texttt{EXPANDCP}(x \vee \overline{x}) = x \nsim x \vee \overline{x}$.

3 Borel space $\mathbb{B}_{\mathbf{0}}^{\boldsymbol{n^2}}$

3.1 Preliminaries

1. Let $n > 1$ and $\mathbf{n} := \{1, \cdots, n\}$. For any $i, j \in \mathbf{n}$ let $(i, j) := n(i-1) + j$. Clearly $(-,-)$ is an injective pairing of type $\mathbf{n} \times \mathbf{n} \to \mathbf{n}^2$.
2. Let $\langle\langle \mathbf{n} \rangle\rangle := \{\{(i,j),(k,l)\} \mid i,j,k,l \in \mathbf{n} \wedge j \neq l\}$
3. For any $I \subset \langle\langle \mathbf{n} \rangle\rangle$ let $I^+ \subset \langle\langle \mathbf{n} \rangle\rangle$ be the minimal closure of I satisfying the following anti-equivalence conditions (a)-(c):
   (a) $I \subset I^+$
   (b) if $\{\{u,v\},\{v,w\},\{w,z\}\} \subset I^+$ and $\{u,z\} \in \langle\langle \mathbf{n} \rangle\rangle$, then $\{u,z\} \in I^+$
   (c) if $\{\{u,v\},\{v,w\},\{w,u\}\} \subset I^+$ then $I^+ := \langle\langle \mathbf{n} \rangle\rangle$
4. $I \subset \langle\langle \mathbf{n} \rangle\rangle$ is called *basic* iff $I = I^+$.

3.2 Basic Borel sets

Definition 2. *Set* $\mathbb{R}_{\neq 0}^{n^2} := \left(\mathbb{R}_{\neq 0}\right)^{n^2}$. *For any* $\{u,v\} \in \langle\langle \mathbf{n} \rangle\rangle$, *Borel sets* $\mathcal{O}_{\{u,v\}} := \left\{\overrightarrow{x} \in \mathbb{R}_{\neq 0}^{n^2} \mid x_u + x_v = 0\right\}$ *and* $\overline{\mathcal{O}}_{\{u,v\}} := \left\{\overrightarrow{x} \in \mathbb{R}_{\neq 0}^{n^2} \mid x_u + x_v \neq 0\right\}$ *are called regular planes and coplanes (in* $\mathbb{R}_{\neq 0}^{n^2}$*), respectively; they both are also called* regular elements. *A set of regular elements* $\left\{\mathcal{O}_\imath, \overline{\mathcal{O}}_\jmath \mid \imath \in I, \jmath \in J\right\}$, $I, J \subset \langle\langle \mathbf{n} \rangle\rangle$, *is called consistent iff* $I^+ \cap J = \emptyset$. *Intersections of sets of regular planes and consistent sets of regular elements are called* regular Π_1*-sets and* regular Π_1*-expansions, respectively. For any regular* Π_1*-expansion* $E = \bigcap_{\imath \in I}\mathcal{O}_\imath \cap \bigcap_{\jmath \in J}\overline{\mathcal{O}}_\jmath$, *denote by* E^{p} *the regular* Π_1*-set* $\bigcap_{\imath \in I}\mathcal{O}_\imath$; *clearly* $E \subset E^{\mathrm{p}}$. *Unions of regular* Π_1*-sets are called* regular Σ_2*-sets.*

Lemma 2. *Disjunction property. For any regular Π_1-sets U, V and W, if $U \subset V \cup W$ then either $U \subset V$ or $U \subset W$.*

Proof. Regular Π_1-sets are convex. (Geometrically obvious.)

Lemma 3. *Absorption property. For any regular Π_1-expansion E and regular Σ_2-set Z, if $E \subset Z$ then $E^{\mathrm{p}} \subset Z$.*

Proof. Suppose $E \subset Z$ and $a \in E^{\mathrm{p}} - Z$. There is a sufficiently small $1 - \dim$ interval $[b, c]$ and a regular Π_1-set $U \subset Z$ such that $a \in [b, c]$ and $b, c \in U$. Since U is convex, it follows that $a \in U$, and hence $a \in Z$ - a contradiction. (Geometrically obvious; cf. also `[GK: Lemma 34]`).

Lemma 4. *Monotonicity.* $\bigcap\limits_{\imath \in I} \mathcal{O}_\imath = \bigcap\limits_{\imath \in I^+} \mathcal{O}_\imath$. *If* $\bigcap\limits_{\jmath \in J} \mathcal{O}_\imath \subset \bigcap\limits_{\imath \in I} \mathcal{O}_\imath$ *then* $I^+ \subset J^+$.

Proof. Easy linear algebra a/o the completeness of equational calculus.

3.3 Basic syntax and semantic

1. Vocabulary:
   (a) Variables $\mathfrak{v}_1, \cdots, \mathfrak{v}_{n^2}$ (abbr.: x, y, z, w - possibly indexed)
   (b) Binary operations $\vee$, $\wedge$ and atomic unary operation $^-$
   (c) Binary relation J
2. Literals: terms xJy and $x\overline{J}y$ (abbr.: ℓ - possibly indexed)
3. *Basic clauses*: $\bigwedge\limits_{i=1}^{s} \mathfrak{v}_{u_i} J \mathfrak{v}_{v_i}$ for basic sets $\{\{u_i, v_i\} \mid i = 1, \cdots, s\} \subset \langle\langle \mathbf{n} \rangle\rangle$
4. Borel polynomials (or just polynomials, or circuits): arbitrary terms according to 1 (we also adopt boolean notions of positive polynomials and PDNF, see above Chapter 2.1). The *size* $\#f$ of a given polynomial f is the number of all operation occurrences in f
5. BP (*basis polynomials*): PDNF $\bigvee\limits_{i=1}^{k} \bigwedge\limits_{j=1}^{s_i} x_{i,j} J y_{i,j}$ such that:
   (a) for every $i = 1, \cdots, k$, $\bigwedge\limits_{j=1}^{s_i} x_{i,j} J y_{i,j}$ is a basic clause
   (b) for any $1 \leq q < r \leq k$, the lists $X_q := [\{x_{q,j}, y_{q,j}\} \mid j = 1, \cdots, s_q]$, $X_r := [\{x_{r,j}, y_{r,j}\} \mid j = 1, \cdots, s_r]$ are sets which are mutually incomparable with respect to $\subset$
6. Boolean evaluation of polynomials in real domain: $Val\,(f)$ is defined as in boolean algebra via
   $Val\,(xJy) := \begin{cases} 1 \text{ if } & Val\,(x) + Val\,(y) = 0 \\ 0 \text{ else} \end{cases}$
   for $Val\,(x)$, $Val\,(y)$ ranging over $\mathbb{R}_{\neq 0}$
7. Equalities:
   (a) f and g are *semi-equivalent* (abbr.: $f \sim_0 g$), iff $Val\,(f) = Val\,(g)$ holds for all variable evaluations in $\mathbb{R}_{\neq 0}$

(b) Two BP $f = \bigvee_{i=1}^{k} \bigwedge_{j=1}^{s_i} x_{i,j} J y_{i,j}$ and $g = \bigvee_{i=1}^{l} \bigwedge_{j=1}^{t_i} x'_{i,j} J y'_{i,j}$ are *isomorphic* (abbr.: $f \cong g$) iff $k = l$ and $\mathrm{Var}(f) = \mathrm{Var}(g)$, where

(c) $\mathrm{Var}(f) = \{\{\{x_{i,j}, y_{i,j}\} \mid j = 1, \cdots, s_i\} \mid i = 1, \cdots, k\}$

8. *Semi-basic* polynomials: polynomials semi-equivalent to basic polynomials
9. For any polynomial f and vector $\overrightarrow{u} \in \mathbb{R}^{n^2}_{\neq 0}$, we let $f(\overrightarrow{u}) := Val(f)$ for $Val(\mathfrak{v}_i) := u_i$, $i = 1, \cdots, n^2$. $\mathrm{Set}(f) := \left\{ \overrightarrow{u} \in \mathbb{R}^{n^2}_{\neq 0} \mid f(\overrightarrow{u}) = 1 \right\}$ is called *regular Borel set* generated by f.

Lemma 5. *For any BP f and g, $f \sim_0 g$ iff $f \cong g$.*

Proof. Obviously $f \sim_0 g$ iff $\mathrm{Set}(f) = \mathrm{Set}(f)$. If f and g are BP, then $\mathrm{Set}(f)$ and $\mathrm{Set}(g)$ are regular Σ_2-sets. The conclusion now follows straightfoward by Lemmata 2, 4 and minimality of BP (cf. also proof of Lemma 1).

3.4 "School-algebra" conversions

Algorithm 2 : $\mathtt{BASE}_0$. *We take as input any Borel polynomial f and construct* $\mathtt{BASE}_0(f)$ *in five subsequent steps 1-5:*

1. *Apply the same rules as at Step 1 of* BASE *(see above Algorithm 1) and arrive at* $\mathtt{EXPAND}(f)$.
2. *Consider any clause C from* $\mathtt{EXPAND}(f)$ *and let X be the set of positive literals (i.e. variables) occurring in C. Replace X by X^+ (see above) and rewrite C to $C[X := X^+]$. Apply this rewriting rule $C \hookrightarrow C[X := X^+]$ to all clauses of* $\mathtt{EXPAND}(f)$. *Denote the output by* $\mathtt{EXPANDB}(f)$.
3. *Proceed as at Step 2 of* BASE *and delete all inconsistent clauses occurring in* $\mathtt{EXPANDB}(f)$. *Denote the output by* $\mathtt{EXPANDBC}(f)$.
4. *Proceed as at Step 3 of* BASE *and delete all negative literals occurring in clauses of* $\mathtt{EXPANDBC}(f)$. *Denote the output by* $\mathtt{EXPANDBCP}(f)$.
5. *Proceed as at Step 4 of* BASE *and delete all but minimal clauses, i.e. those whose sets of variables are proper extensions of other conjunctions' sets of variables.*
   Denote the output by $\mathtt{BASE}_0(f)$.

Theorem 2. *The following holds for any semi-basic polynomials f and g.*

1. $\mathtt{BASE}_0(f)$ *is a BP*
2. $f \sim_0 \mathtt{EXPAND}(f) \sim_0 \mathtt{EXPANDB}(f) \sim_0 \mathtt{EXPANDBC}(f) \sim_0 \mathtt{EXPANDBCP}(f) \sim_0 \mathtt{BASE}_0(f)$
3. *$f \sim_0 g$ infers* $\mathtt{BASE}_0(f) \cong \mathtt{BASE}_0(g)$, *and hence* $\mathtt{BASE}_0(f) = \mathtt{BASE}_0(g)$ *(mod permutation of clauses and literals within clauses)*

Proof. 1. Straightforward by definition.

2. It will suffice to verify that the passage from EXPANDBC to EXPANDBCP preserves $\sim_0$ (the rest is obvious). By the assumption there is a basic polynomial

g such that $g \sim_0 f \sim_0 \mathtt{EXPANDBC}\,(f)$. Let $X := \mathrm{Set}\,(\mathtt{EXPANDBC}\,(f)) = \bigcup\limits_{\kappa\in K} E_\kappa$, $Y := \mathrm{Set}\,(\mathtt{EXPANDBCP}\,(f)) = \bigcup\limits_{\kappa\in K} E^{\mathrm{p}}_\kappa$, $Z := \mathrm{Set}\,(g)$ and note that Y and Z are regular Σ_2-sets. Moreover, we have $X = Z \subset Y$. Furthermore, by Lemma 11, $E_\kappa \subset Z$ infers $E^{\mathrm{p}}_\kappa \subset Z$. Thus $X = Z$ infers $Y \subset Z = X$, and hence $X = Y$, from which we arrive at $\mathtt{EXPANDBC}\,(f) \sim_0 \mathtt{EXPANDBCP}\,(f)$, Q.E.D.

3. Follows from 2 and Lemma 5.

4 The link

Definition 3. *Denote by $\mathbb{D}_n$ the set of all DNF $\bigvee\limits_{j=1}^{n}\bigwedge\limits_{i=1}^{n} \ell_{i,j}$, for $\ell_{i,j}$ ranging over arbitrary literals from $\mathbb{B}^n$. Denote by Φ_n (also called Borel universal CNF) a positive CNF $\bigwedge\limits_{\xi:\mathbf{n}\to\mathbf{n}}\bigvee\limits_{i<j\in\mathbf{n}} \mathfrak{v}_{(\xi(i),i)} J \mathfrak{v}_{(\xi(j),j)}$, in $\mathbb{B}_0^{n^2}$.*

Lemma 6. *There exists a fixed bijective embedding*

$$\mathbb{D}_n \ni f \mapsto \mathrm{VAL}_f : \left\{\mathfrak{v}_\imath \mid \imath \in \mathbf{n}^2\right\} \to \mathbb{Z}_{\neq 0} \subset \mathbb{R}_{\neq 0}$$

such that for every $f \in \mathbb{D}_n$, f is valid in $\mathbb{B}^n$ iff $\mathrm{Val}\,(\Phi_n) = 1$ in $\mathbb{B}_0^{n^2}$, where Val *is the evaluation uniquely determined by* VAL_f.

Proof. For any $f = \bigvee\limits_{j=1}^{n}\bigwedge\limits_{i=1}^{n} \ell_{i,j}$, let $\mathrm{VAL}_f\left(\mathfrak{v}_{(i,j)}\right) := \begin{cases} k & \text{if } \ell_{i,j} = \mathfrak{v}_k \\ -k & \text{if } \ell_{i,j} = \overline{\mathfrak{v}_k} \end{cases}$.

Remark 3. The lemma shows that the whole $n-$dim boolean DNF family $\mathbb{D}_n$ is representable by one n^2-dim positive Borel polynomial Φ_n and/or the corresponding Borel set $\mathrm{Set}\,(\Phi_n)$. Moreover, the validity problem for $\mathbb{D}_n$ is reducible to the evaluation of Φ_n in $\mathbb{Z}_{\neq 0}$ by an algorithm whose Turing complexity is polynomial in n. Note that circuit complexity of Φ_n is exponential in n.

Theorem 3. *$\mathcal{NP} \subset \mathcal{P}/poly$ holds iff there exists $c \in \mathbb{N}$ such that for every $n \in \mathbb{N}$ there exists a n^2-dim Borel polynomial $f \sim_0 \Phi_n$, in $\mathbb{B}_0^{n^2}$, such that $\#f$ does not exceed n^c.*

Proof. Sufficiency easily follows from the last remark. Consider necessity. A binary relation $J \subset \mathbf{n}^2$ is called *anti-equivalence* iff the following holds:

$$\left(\forall x, y, z, u \in \mathbf{n}^2\right)\left(\begin{array}{c}(xJy \to yJx) \wedge (xJy \wedge yJz \wedge zJu \to xJu) \\ \wedge\, (xJy \wedge yJz \to \neg xJz)\end{array}\right)$$

Such J determines a disjoint partition $\mathbf{n}^2 = \bigcup\limits_{i=1}^{m} S_i \cup \bigcup\limits_{i=1}^{m} S'_i$, $m \leq n$, where

$$xJy \leftrightarrow (\exists i \leq m)\left((x \in S_i \wedge y \in S'_i) \vee (y \in S_i \wedge x \in S'_i)\right)$$

and a DNF $D_J = \bigvee\limits_{j=1}^{n}\bigwedge\limits_{i=1}^{n} \ell_{i,j} \in \mathbb{D}_n$ where $\ell_{i,j} := \begin{cases} \mathfrak{v}_k \text{ if } (i,j) \in S_k \\ \overline{\mathfrak{v}_k} \text{ if } (i,j) \in S'_k \end{cases}$

with the corresponding word $\mathsf{w}_J = a_1, \cdots, a_{n^4} \in \{0,1\}^{n^4}$ such that $a_\kappa = 1 \leftrightarrow \imath J \jmath$ holds for $\imath, \jmath \in \mathbf{n}^2$ with $\kappa = n^2(\imath - 1) + \jmath$. We observe that $\mathcal{L} = \left\{\mathsf{w} \in \{0,1\}^{n^4} \mid \mathsf{w} = \mathsf{w}_J \text{ where } J \text{ is anti-equivalence and } D_J \text{ is valid}\right\}$ is a coNP complete language. Hence $\mathcal{NP} \subset \mathcal{P}/poly$ iff $\mathcal{L}$ can be recognized in $\{0,1\}^{n^4}$ by a polynomial size boolean circuit in De Morgan normal form. From this we arrive at the required $n^2 - \dim$ Borel polynomial $f \sim_0 \Phi_n$ by rewriting circuit's sources $\mathfrak{v}_\kappa$ and $\overline{\mathfrak{v}_\kappa}$ to literals $\mathfrak{v}_\imath J \mathfrak{v}_\jmath$ and $\mathfrak{v}_\imath \overline{J} \mathfrak{v}_\jmath$, respectively, where $\kappa = n^2(\imath - 1) + \jmath$. [This proof is elaborated in [K] - it upgrades considerations from [G], [GK].]

Now by Theorem 2 (3), we replace $f \sim_0 \Phi_n$ by $\mathtt{BASE}_0(f) \cong \mathtt{BASE}_0(\Phi_n)$ and arrive, by contraposition, at the desired Conclusion 1 (see Chapter 1.2 above).

5 Structure of $\mathtt{BASE}_0(\Phi_n)$

Definition 4. *Consider any $n \times m$ matrix $M = (\ell_{i,j})_{n \times m}$, $m \leq n$, where $\ell_{i,j}$ are arbitrary literals from $\mathbb{B}^n$. M is called valid iff so is the correlated boolean DNF $\bigvee_{j=1}^{m} \bigwedge_{i=1}^{n} \ell_{i,j}$. Let $\partial(M) := \left\{\{(i,j),(k,l)\} \in \langle\langle \mathbf{n} \rangle\rangle \mid j,l \leq m \wedge \ell_{i,j} = \overline{\ell_{k,l}}\right\}$. A valid $n \times m$ matrix M is called basic DNF_n matrix iff for any $m' \leq n$ and a valid $n \times m'$ matrix M', $\partial(M')$ is not a proper subset of $\partial(M)$. Denote by Ω_n the set of $\partial(M)$ for M ranging over arbitrary basic DNF_n matrices.b*

Theorem 4. *Suppose $\Omega_n = \{\partial(M_\imath) \mid \imath = 1, \cdots, r\}$. For $\imath \in \mathbf{r}$, let $\partial(M_\imath) = \{\{\alpha_{\imath,\jmath}, \beta_{\imath,\jmath}\} \in \langle\langle \mathbf{n} \rangle\rangle \mid \jmath = 1, \cdots, s_\imath\}$. Then $\mathtt{BASE}_0(\Phi_n) \cong \bigvee_{\imath=1}^{r} \bigwedge_{\jmath=1}^{s_\imath} \mathfrak{v}_{\alpha_{\imath,\jmath}} J \mathfrak{v}_{\beta_{\imath,\jmath}}$.*

Proof. Let f_n be $\bigvee_{\imath=1}^{r} \bigwedge_{\jmath=1}^{s_\imath} \mathfrak{v}_{\alpha_{\imath,\jmath}} J \mathfrak{v}_{\beta_{\imath,\jmath}}$, as above. Clearly f_n is a BP and by Theorem 2 (1) so is $\mathtt{BASE}_0(\Phi_n)$. Now by Lemma 5 and Theorem 2 (2), $\mathtt{BASE}_0(\Phi_n) \cong f_n$ iff $\mathtt{BASE}_0(\Phi_n) \sim_0 f_n$ iff $\Phi_n \sim_0 f_n$. That $\Phi_n \sim_0 f_n$ holds true follows by standard boolean arguments. Hence $\mathtt{BASE}_0(\Phi_n) \cong f_n$, Q.E.D.

Example 2. There are two basic DNF_2 matrices

$$M_1 = \begin{pmatrix} x & \overline{x} \\ x & \overline{x} \end{pmatrix} \text{ and } M_2 = \begin{pmatrix} \overline{x} & x \\ \overline{x} & x \end{pmatrix}$$

which are mutually isomorphic, i.e. $\partial(M_1) = \partial(M_2)$. Hence

$$\Omega_2 = \{\{(1,1),(1,2)\},\{(1,1),(2,2)\},\{(2,1),(1,2)\},\{(2,1),(2,2)\}\}$$

and $\#\Omega_2 = 1$. There are 129 non-isomorphic basic DNF_3 matrices, and hence $\#\Omega_3 = 129$. $\#\Omega_4 = 1023486$.

Remark 4. By the last theorem, $\mathtt{BASE}_0(\Phi_n)$ is uniquely determined by the set Ω_n. Loosely speaking, $\mathcal{P} < \mathcal{NP}$ problem reduces to appropriate asymptotic combinatorial analysis of Ω_n as $n \to \infty$. Ω_n also admits explicit recursive definition that enables detailed investigations along these lines.

6 Appendix: Combinatorial translations

1. For any $X, Y \subset \langle\langle \mathbf{n} \rangle\rangle$ let $X \oslash Y := \begin{cases} X \text{ if } & X \cap Y = \emptyset \\ \emptyset & \text{else} \end{cases}$
2. Let $\mathcal{T}$ be a finite rooted binary tree whose gates are labeled by $\wedge$ or $\vee$, and whose sources x are labeled by elements $\langle \chi(x), \rho(x) \rangle$ of the ordinary Cartesian product $\mathbf{2} \times \langle\langle \mathbf{n} \rangle\rangle$. Denote by $\#\mathcal{T}$ the *weight* of $\mathcal{T}$, i.e. total number of all vertices occurring in $\mathcal{T}$.
   (a) A set of sources X is called $\mathcal{T}$-*conjunctive* iff any pair of distinct sources from X has the closest common ancestor $\wedge$.
   (b) Maximal $\mathcal{T}$-conjunctive sets of sources are called $\mathcal{T}$-*cuts*. Denote by $\mathrm{CUT}(\mathcal{T})$ the set of all $\mathcal{T}$-cuts.
   (c) For any $X \in \mathrm{CUT}(\mathcal{T})$ and $i \in \mathbf{2}$, let $X_{(i)} := \{\rho(x) \mid \chi(x) = i \wedge x \in X\}$.
   (d) Set $\Im(\mathcal{T}) := \left\{ X^{+}_{(1)} \oslash X_{(2)} \mid X \in \mathrm{CUT}(\mathcal{T}) \right\}$ (cf. Preliminaries 3.1).
3. For any labeled trees $\mathcal{T}$ and $\mathcal{T}'$, as above, let

$$\begin{aligned} \mathcal{T} \approx \mathcal{T}' :\Leftrightarrow & (\forall X \in \Im(\mathcal{T}))(\exists X' \in \Im(\mathcal{T}'))(X' \subset X) \\ & \wedge (\forall X' \in \Im(\mathcal{T}'))(\exists X \in \Im(\mathcal{T}))(X \subset X') \end{aligned}$$

4. Denote by $\mathcal{F}_n$ a labeled tree $\bigwedge\limits_{\xi:\mathbf{n}\to\mathbf{n}} \bigvee\limits_{i<j\in\mathbf{n}} \langle 1, \{(\xi(i), i), (\xi(j), j)\}\rangle$.
5. Denote by CS the following combinatorial sentence:
   $\mathcal{T} \approx \mathcal{F}_n$ *fails for sufficiently large* n *if* $\#\mathcal{T}$ *is merely polynomial in* n.

Theorem 5. *In Peano Arithmetic,* $\mathcal{P} < \mathcal{NP}$ *is derivable from* CS.

Proof. It is readily seen that CS is equivalent to C_0. Now proof of Theorem 3 (see above) can be formalized by standard encodings in Peano Arithmetic.

7 References

[C]: S. Cook, *The P versus NP Problem*,
http://www.claymath.org/millennium/P vs NP/
[F]: H. Friedman. *CLAY MILLENIUM PROBLEM:P = NP*
http://www.math.ohio-state.edu/%7Efriedman/manuscripts.html
[G]: L. Gordeev, *Proof-sketch: Why NP is not P*, 2004
http://www-ls.informatik.uni-tuebingen.de/gordeew/publikationen/Proof Sketch.pdf
[GK]: L. Gordeev, A. Krebs, *Elementary interpretations of P vs. NP*, 2003
http://www-ls.informatik.uni-tuebingen.de/gordeew/publikationen/e4.pdf
[K]: A. Krebs, *Turing machine transformation*, Preprint, 2005
[P]: E. Post, *Finite combinatory processes - formulation I*,
Journ. Symb. Logic 1 (1936), 103-105
[T]: A. Turing, *On computable numbers, with an application to the Entscheidungsproblem*, Proc. London Math. Soc. 42 (1937), 230-265

Models of Timing Abstraction in Simultaneous Multithreaded and Multi-Core Processors

Neal A. Harman*

Department of Computer Science,
University of Wales Swansea, Swansea SA2 8PP, UK
n.a.harman@swan.ac.uk

Abstract. This paper builds on a series examing algebraic models of microprocessors and their correctness. The modeling is based on using recursive functions on many-sorted algebras and the reasoning is based on the connection with algebraic specification and rewriting. Current models can accommodate *pipelined* and *superscalar* processors. However, these are no longer state-of-the-art: *Simultaneous Multithreaded* (SMT) and *Multi-core* microprocessors enable a single microprocessor implementation to present itself to the programmer as multiple (virtual in the case of SMT) processors with shared state. We extend the existing algebraic tools for modeling microprocessors and their correctness to SMT and Multi-Core systems. We outline how the *one-step theorems* for simplifying verification are modified for SMT and Multi-Core processors.
Keywords: Algebraic Models; Formal Verification; Microprocessors; Multi-Core; Simultaneous Multithreading

Formal approaches to the correctness of computer hardware are now well established and enjoy a degree of industrial adoption, with a range of impressive examples [1, 5, 12]. However, despite this success, there is little attention paid to (a) how hardware systems should be modelled, and (b) what it means for an implementation to be correct with respect to a specification. Modelling techniques are wide-ranging and correctness concepts are - sometimes subtly and sometimes wildly - different; often there is no clear statement of what 'correctness' means in a particular instance. For example, some correctness models consider only the passage of a single, isolated, instruction through an execution pipeline, and do not consider interaction between instructions. This paper forms part of a series that attempts to address these issues: focussing specifically on modeling and not verification. The recursive schemes we use to model different processors at different levels of abstraction are relatively simple theoretically. However, in comparing models we obtain more complex equations made from schemes with different data spaces and different timing abstractions.

Past work has addressed microprogrammed [10]; pipelined [9]; and superscalar [7] processors. The pipelined model has been used to verify the ARM6

* The author would like to thank Prof. J V Tucker of his valuable suggestions when revising this paper.

microprocessor: the first 'commercial' processor to be verified not explicitly developed with verification in mind [5, 6]. Here, the model is extended to *Simultaneous Multithreaded* and *Multi-Core* processors.Our models are algebra-based: however, they can be moved to other formalisms ([5, 6] uses HOL).

Microprocessor implementations are growing in complexity. The simplest and most obvious case is an implementation G of a processor specification F in which the execution of each machine instruction is completed before the next one starts: now restricted to simple embedded processors. The timing relationship between specification F and G in such a case is trivial: each clock cycle in F corresponds uniquely with one or more cycles in G. However, more complex timing relationships increase the complexity of the timing relationship between F and G.

- *Pipelined* implementations overlap the execution of machine instructions. Consequently the relationship between clock cycles in F and G is no longer unique.
- *Superscalar* implementations attempt to execute multiple instructions simultaneously, sometimes out of program order. In the (common) event of multiple instructions ending simultaneously (or out of order) there may be no identifiable cycle of G's clock S corresponding with a cycle of F's clock T.
- *Simultaneous Multithreaded (SMT)*[1] and *Multi-Core* implementations behave like multiple specification level processors F. In the case of SMT these processors are virtual, and in the case of multi-core they are real. In both cases, some of the implementation processor state is *shared* between specification level processors - more in the case of SMT than multi-core. Furthermore, SMT processors are by definition also superscalar; in principle, Multi-Core processors need not be, but currently-available examples are.

In this paper we extend a suite of existing models and correctness definitions [4, 7, 9, 10] to accommodate SMT and multi-core processors. The structure of the paper is as follows: first we briefly review the field; then we introduce the basic concepts of the model: clocks dividing up time; retimings relating clocks at different levels of abstraction and their extension to superscalar timing models; iterated map models of microprocessors; correctness models and the one-step theorems that simplify the verification process. Then we extend the existing iterated map model and correctness model to accommodate SMT and Multi-Core processors.

1 Related Work

There is a substantial body of work devoted to microprocessor verification and only a brief summary is possible here. A common characteristic of much of the work is the need to address a specific, usually complex[2] example. Neglecting

[1] called *Hyperthreading* by Intel.

[2] At least with reference to the state-of-the-art in processor verification at the time.

early work from the 1980s[3] and earlier, a landmark leading to much subsequent work is [2]. Key concepts like the relationship between time at different levels of abstraction, and how it can be addressed appear [13, 7]. Work appears on pipelined and superscalar models with parallels with our own: substantial differences are that although the concept of *timing abstraction* is present, formally the notion of *time itself* is often not. Also, in pipelined and superscalar processors are specification state components distributed in time in an implementation [16]; or (our position) are they *functions* of implementation state components from the *same* time? Consider a three-stage pipeline in which instructions are fetched three cycles ahead of execution[4]. Is the the program counter *pc* in the specification the value from the implementation three [specification] cycles earlier, or the current value less three[5]? This always enables use to separate timing and data abstraction maps. Recent work addresses larger and more complex examples: the VAMP project, [1], the ARM6 verification project at Cambridge[5, 6], Hunt's Group Austin,Texas[14], and the UV group at Utah [12].

2 Clocks and Basic Models of Computers

A clock T is an algebra $T = (\mathbf{N} \mid 0, t+1)$ denoting intervals of time called *clock cycles*. Time is defined in terms events and not vice versa: typically, clock cycles mark the beginning/end of 'interesting' events, and need not be equal in length.

Systems are modelled by *state algebras* $St = (A, T \mid F)$ where A is the state set and *iterated map* $F : T \times A \to A$. St is implemented in terms of *next-state algebra* $Nst = (A, T \mid init, next)$

$$F(0, a) = init(a),$$
$$F(t+1, a) = next(F(t, a))$$

where $init : A \to A$ and $next : A \to A$ are the initialization and next state functions, implemented in terms of a *machine algebra*: finite bit sequences and operations on bit sequences typical of those found in low-level hardware. State set A is a Cartesian product of simpler state components. In general, we expect machine algebra operations to be at most simultaneous primitive recursive functions over St [8]. Hence F is generally a simultaneous primitive recursive function. The rôle of *init* is not to initialize the processor, but to ensure that the execution trace of F starts in a legal state, since not all initial states $a \in A$ may be consistent with correct future execution: see section 4.

3 Correctness Models for Non-Pipelined, Pipelined and Superscalar Processors

Correctness models relate iterated maps $F : T \times A \to A$ and $G : S \times B \to B$. The majority of our effort is devoted to time: the relationship between state sets

[3] And also work on verifying processor *fragments*.
[4] Neglecting complications to do with branching and so on.
[5] More likely, some multiple of three.

A and B is a typically a straightforward projection $\pi : B \to A$. However, timing abstraction is complex.

We define a *retiming* $\lambda : S \to T$ to be a surjective (all specification times occur in the implementation) and monotonic (time does not go backwards) map. Retimings are typically parameterized by the implementation state, and since microprocessors are deterministic, λ is uniquely determined by the initial implementation state.

$$\lambda : B \to [S \to T]$$

In addition to retimings, we also need *immersions* $\overline{\lambda} : B \to [T \to S]$:

$$\overline{\lambda}(b)(t) = \text{least } s[\lambda(b)(s) \geq t]$$

and the *start* operator $start : [S \to T] \to [S \to S]$

$$start(\lambda)(s) = \overline{\lambda}\lambda(s).$$

Microprocessors can be modelled at different levels of abstraction. We are concerned with the lowest level accessible by a programmer: the *programmer's model* PM and the most abstract implementation level: the *abstract circuit model* AC. Clock cycles in PM models correspond with machine instructions: in AC models they are [some multiple of] system clock cycles. A non-pipelined microprocessor implementation G of AC is correct with respect to a specification F of PM if and only if the state of G under data abstraction map π is identical to the state of F for all times $s \in S$ corresponding with the start/end of cycles of T. That is, for all $s = start(\lambda)(s)$:

$$F(\overline{\lambda}(b)(s), \pi(b)) = \pi(G(s, b)).$$

In a pipelined processor, instruction execution overlaps, and *during instruction execution* we cannot uniquely relate cycles of S with cycles of T. However, instructions *terminate* at unique times: If instruction i terminates at time s_i, then no other instruction will terminate at time s_i. Provided retiming λ relates s to the time $t_i \in T$ corresponding with the end of instruction i and the start of $i + 1$, our correctness model above still applies [9].

Superscalar processors attempt to execute multiple [pipelined] instructions in parallel, and instructions are allowed to terminate simultaneously, or out of program order. We cannot uniquely associate cycles of S corresponding with the start/end of instructions with cycles of T. Our approach is based on the following: in the event that instructions i and $i + 1$ terminate simultaneously or out of order, it is not meaningful to ask 'is G correct with respect to F after i has terminated but before $i + 1$' because there is no such time.

We introduce a new *retirement clock* R marking the completion of one or more instructions, with retimings $\lambda_1 : T \to R$ and $\lambda_2 : S \to R$. Retiming λ_1 captures the relationship between the sequential 'one-at-a-time' execution model of the programmer and the actual order of instruction completion; λ_2 marks instruction

completion times with respect to the system clock. The non-surjective *adjunct retiming* $\rho : S \rightarrow T$ constructed by

$$\rho(s) = \overline{\lambda}_1 \lambda_2(s)$$

relates system clock times and the completion of machine instructions.

Our existing correctness model still applies if we replace retiming λ with adjunct retiming ρ [4].

4 One-Step Theorems

The obvious correctness proof for the models in section 3 is induction over clock S. However, we can eliminate induction by using the *one-step theorems*. We require two conditions.

- Iterated map G is *time-consistent*. That is:

  $$G(s, b) = init_G(G(s, b))$$

  for all $s = start(\lambda)(s)$ and where $init_G$ is the initialization function for iterated map G. Equivalently, $G(\overline{\lambda}(b)(t)+\overline{\lambda}(b)(t'), b) = G(\overline{\lambda}(b)(t), G(\overline{\lambda}(b)(t'), b))$.
- Retiming λ is *uniform*. That is for all $t \in T$

  $$\overline{\lambda}(b)(t+1) - \overline{\lambda}(b)(t) = dur(b),$$

  where $dur : B \rightarrow \mathbf{N}^+$ is a *duration function*.

Informally, microprocessors are functions only of their state (and possibly inputs) at any given time $s \in S$: not of the numerical value of s. The conditions above establish the independence of AC model G from the numerical value of s. As well as being necessary conditions for the application of the one-step theorems, time-consistency and uniformity are characteristics of real hardware, so models that did not possess them would be flawed.

Not all elements of a state set B will be consistent with correct execution of G. For example, consider an example with program counter pc, memory m and instruction register ir: we require $ir = m[pc]$. Time-consistency requires a carefully constructed initialization function *init* which leaves all 'legal' states unchanged: otherwise, for some state $b \in B$, $init(b) \neq b$ and hence time-consistency would not hold. In practice, implementations are so complex that identifying legal states can be difficult, and defining *init* problematic. A systematic mechanism is introduced in [9] that can be used whenever the contents of a pipeline are uniquely determined at time s by its contents at time s. This is not always the case - consider an example with two integer execution units, in which the unit chosen for a particular instructions is determined by which has the shortest queue: the contents of the queues may be a function of instructions that have already left the pipeline.

To establish that retiming λ is uniform, it is sufficient to define its immersion in terms of a duration function:

$$\overline{\lambda}(b)(0) = 0,$$
$$\overline{\lambda}(b)(t+1) = imm(b)(t) + dur(G(\overline{\lambda}(b)(t), b).$$

Because a typical implementation G is extremely complex, defining *dur* independently of G is usually prohibitively difficult. The usual definition is *non-constructive* of the form:

$$dur(b) = \text{least } s[end(G(s,b)),]$$

where $end : B \to \mathbf{B}$ is some function that identifies when one (or more) instructions have completed. Because G forms part of the definition of *dur* and λ, in this case our correctness model makes no statement about how long each instruction will take to execute.

Initially, it seems that establishing time-consistency requires induction. However, the first one-step theorem addresses this. Given iterated map $G : S \times B \to B$, and retiming $\lambda : S \to [S \to T]$ then to establish

$$G(s,b) = init_G(G(s,b))$$

for all $s = start(\lambda)(s)$, it is sufficient to show that:

$$G(0,b) = init_G(G(0,b)), \text{ and}$$
$$G(\overline{\lambda}(b)(1), b) = init(G(\overline{\lambda}(b)(1), b)).$$

The proof [4] is omitted.

The second one-step theorem can be used to establish correctness.Given time-consistent iterated maps $F : S \times A \to A$ and $G : S \times B \to B$, and uniform retiming $\lambda : S \to [S \to T]$ then to establish

$$F(\overline{\lambda}(b)(s), \pi(b)) = \pi(G(s,b))$$

for all $s = start(\lambda)(s)$, it is sufficient to show that:

$$F(0, \pi(b)) = \pi(G(0,b)), \text{ and}$$
$$F(1, \pi(b)) = \pi(G(\overline{\lambda}(b)(1), b)).$$

The proof [4] is omitted.

We omit discussion of the case when F and G are related by an adjunct retiming ρ other than to say that the one step theorems still hold [4, 9] .

5 VTM Model Definition

In this section, we consider how we can extend our existing microprocessor model to accommodate SMT. From the perspective of an operating system kernel programmer, an SMT/multi-core processor appears as multiple PM-level processors

in which some state is *shared*. These processor will be implemented, collectively, by a single *AC*-level model We will use the term *Virtual Thread Model* (VTM) to distinguish these processors from the conventional PM level.

The temporal relationship with *other* VTM processors is exposed via the shared state. This relationship is defined by state information that is *not* present in the VTM state, but is in the *implementation* (AC) state. For example, one implementation may choose to prioritize one thread at the expense of others as a function of state elements not visible at the VTM level, while another implementation *of the same* VTM *level model* may not. Consequently, VTM models must be defined over a PM state set extended by at least part of the corresponding AC state. In this paper we choose to use the complete AC state. However, there is a case for introducing a new, intermediate, level of abstraction [11].

5.1 VTM State Set

Consider a Simultaneous Multi-Threaded/Multi-Core processor that is able to execute n threads - that is, it appears to be n (virtual) processors. Each virtual processor F^i_{VTM}, $i \in \{1,\ldots, n\}$, will operate over its own clock T_i; the state of F^i_{VTM} will be composed of some parts that are *local* to F^i_{VTM} and some parts that are *shared* with $F^1_{\mathrm{VTM}},\ldots, F^{i-1}_{\mathrm{VTM}}, F^{i+1}_{\mathrm{VTM}},\ldots, F^n_{\mathrm{VTM}}$. We assume, without loss of generality, that the private state elements $priv \in \Sigma^{\mathrm{priv}}_{\mathrm{VTM}}$ precede the shared state elements $share \in \Sigma^{\mathrm{share}}_{\mathrm{VTM}}$ in the state vector:

$$\Sigma_{\mathrm{VTM}} = \Sigma^{\mathrm{priv}}_{\mathrm{VTM}} \times \Sigma^{\mathrm{share}}_{\mathrm{VTM}}$$

The state trace of each VTM processor will be a function of its own local and shared state, and the shared state of all other VTM processors. There is only one shared state in the AC level implementation. However each individual VTM model has its own copy of the shared state, from its own perspective: a conceit we wish to maintain. Consequently we need to merge the shared states of each VTM level processor.

Each VTM processor operates with its own clock: to correctly merge shared states from different VTM processors, we must match states at the appropriate times. We can relate times on different VTM processors using the [adjunct] retimings and corresponding immersions between VTM and AC level clocks: $\rho_i(b)\overline{\rho}_j(b)(t_j)$ is the time on clock T_i corresponding to time $t_j \in T_j$.

5.2 VTM Definition

Given clocks T_i, $i \in \{1,\ldots, n\}$; AC state set Σ_{AC}; VTM state set Σ_{VTM}; adjunct retimings $\rho_i : \Sigma_{\mathrm{AC}} \to [S \to T_i]$; AC state projection function $\pi^{\mathrm{i}} : \Sigma_{\mathrm{AC}} \to \Sigma_{\mathrm{VTM}}$; private state projection functions $\pi^i_{\mathrm{priv}} : \Sigma_{\mathrm{VTM}} \to \Sigma^{\mathrm{priv}}_{\mathrm{VTM}}$; *merge operators* $\tau_i : (\Sigma^{\mathrm{share}}_{\mathrm{VTM}})^n \to \Sigma^{\mathrm{share}}_{\mathrm{VTM}}$, for $i \in \{1,\ldots,n\}$; initialization function $init : \Sigma_{\mathrm{VTM}} \to$

Σ_{VTM}; and next-state function $next : \Sigma_{\mathrm{VTM}} \to \Sigma^{\mathrm{priv}}_{\mathrm{VTM}}$, we model an individual VTM level processor F^i_{VTM} as follows.

$$
\begin{aligned}
&F^i_{\mathrm{VTM}} : T_i \times \Sigma_{\mathrm{AC}} \to \Sigma_{\mathrm{VTM}} \\
&F^i_{\mathrm{VTM}}(0, \sigma_{\mathrm{AC}}) = init(\pi^{\mathrm{i}}(\sigma_{\mathrm{AC}})) \\
&F^i_{\mathrm{VTM}}(t+1, \sigma_{\mathrm{AC}}) = next[(\pi^i_{\mathrm{priv}}(F^i_{\mathrm{VTM}}(t, \sigma_{\mathrm{AC}}))), \\
&\qquad \tau_i(F^i_{\mathrm{VTM}}(t, \sigma_{\mathrm{AC}}), \\
&\qquad\quad F^1_{\mathrm{VTM}}(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_1(\sigma_{\mathrm{AC}})(t), \sigma_{\mathrm{AC}}), \\
&\qquad\quad \vdots \\
&\qquad\quad F^{i-1}_{\mathrm{VTM}}(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_{i-1}(\sigma_{\mathrm{AC}})(t), \sigma_{\mathrm{AC}}), \\
&\qquad\quad F^{i+1}_{\mathrm{VTM}}(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_{i+1}(\sigma_{\mathrm{AC}})(t), \sigma_{\mathrm{AC}}), \\
&\qquad\quad \vdots \\
&\qquad\quad F^n_{\mathrm{VTM}}(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_n(\sigma_{\mathrm{AC}})(t), \sigma_{\mathrm{AC}}))].
\end{aligned}
$$

Note that as well as containing the AC-level state Σ_{AC}, our VTM-level definition contains the state dependent adjunct retimings ρ_i. Recall that it is usual to define ρ_i in terms of some AC-level implementation (Section 3). Consequently, the AC implementation is deeply embedded in the definition of a VTM level model. We return to this issue in Section 6.

5.3 VTM Model Observations

A significant issue in our VTM level model is that next-state and initialization functions are taken from a [probably existing] PM level model. Since the definitions of *next* and *init* are by far the most complex part of model definition, it is important to be able to reuse them which is the case in the model above.

The definition above is the most general case: in some circumstances it can be simplified, depending on the requirements of the merge operators τ_i that unify the various shared state components of the n VTM processors. The definition of τ will depend on the precise nature of the shared state $\Sigma^{\mathrm{share}}_{\mathrm{VTM}}$; and the behaviour of the processor implementation - for example, if two VTM processors attempt to update the same state unit simultaneously. Commonly, the shared state will consist of the processor's main memory. In some circumstances, the definitions of τ_i will not be functions of the private state; and in others all the merge operators τ_i are identical: see [11].

5.4 Correctness of the VTM Model

We now consider what it means for a VTM level model to be correctly implemented by an AC level model. Note that because there are n VTM level processors corresponding to each AC level processor, there are n separate correctness statements.

AC model $G : S \times \Sigma_{\mathrm{AC}} \rightarrow \Sigma_{\mathrm{AC}}$ is said to be a *correct implementation* of VTM model $F^i_{\mathrm{VTM}} : T_i \times \Sigma_{\mathrm{AC}} \rightarrow \Sigma_{\mathrm{VTM}}$, $i \in \{1, \ldots, n\}$ if, given adjunct retimings $\rho_i : Ret('\Sigma_{\mathrm{AC}}, S, T_i)$ and surjective data abstraction map $\psi : \Sigma_{\mathrm{AC}} \rightarrow \Sigma_{\mathrm{VTM}}$, then for each clock T_i, for all $s = start(\rho_i(\sigma_{\mathrm{AC}}))(s)$ and $state \in \Sigma_{\mathrm{AC}}$, the following conditions hold for $i \in \{1, \ldots, n\}$

$$F^i_{\mathrm{VTM}}(\rho_i(\sigma_{\mathrm{AC}})(s), \sigma_{\mathrm{AC}}) = \psi(G(s, \sigma_{\mathrm{AC}}).$$

5.5 Extending the One-Step Theorems to the VTM Model

In considering how we can apply the one-step theorems to our VTM model, recall that the *AC*-level processor $G : S \times B \rightarrow B$ is essentially identical to any other AC processor model: that is, it represents the implementation of a more abstract specification. We are interested in it correctness with respect to some uniform adjunct retiming ρ at times s such that $s = start(\rho)(s)$. Hence the existing one-step theorems show how to establish that G is time-consistent. Clearly we can establish the uniformity of retiming ρ_i by construction in terms of a duration function (section 4). However, we must establish that the time-consistency of the collection of VTM level processors F_i, $i \in \{1 \ldots n\}$ can be determined by the first one-step theorem. Note we omit the straightforward (though tedious) proofs [11].

Let $F_i : T_i \times \Sigma_{\mathrm{AC}} \rightarrow \Sigma_{\mathrm{VTM}}$ be the i^{th} VTM component $F : S \times \Sigma_{\mathrm{AC}} \rightarrow \Sigma_{\mathrm{AC}}$. For all $t \in T_i$ and $i \in \{1,\ldots,n\}$:

$$F_i(t + t', \sigma_{\mathrm{AC}}) = F_i(t, F_i(t', \sigma_{\mathrm{AC}}))$$

if and only if

$$F_i(0, \sigma_{\mathrm{AC}}) = init(F_i(0, \sigma_{\mathrm{AC}}), \text{ and}$$
$$F_i(1, \sigma_{\mathrm{AC}}) = init(F_i(1, \sigma_{\mathrm{AC}})$$

The proof is by induction over t and requires the following lemma:

$$F_j(\rho_j(\sigma_{\mathrm{AC}})\overline{\rho}_i(\sigma_{\mathrm{AC}})(t + t'), \sigma_{\mathrm{AC}}) = F_j(\rho_j(\beta)\overline{\rho}_i(\beta)(t), \beta),$$

for each $F_j : T_j \times \Sigma_{\mathrm{AC}} \rightarrow \Sigma_{\mathrm{VTM}}$ $j = \{1,\ldots,i-1, i+1,\ldots,n\}$ and $t \in T_i$, where $\beta \in \Sigma_{\mathrm{AC}}$ is the *AC* system state at time $t' \in T_i$, defined as follows:

$$\begin{aligned}
\beta = &[\pi_{\mathrm{priv}}(F_1(\rho_1(\sigma_{\mathrm{AC}})\overline{\rho}_i(\sigma_{\mathrm{AC}})(t'), \sigma_{\mathrm{AC}})), \ldots, \\
&\pi_{\mathrm{priv}}(F_{i-1}(\rho_{i-1}(\sigma_{\mathrm{AC}})\overline{\rho}_i(\sigma_{\mathrm{AC}})(t'), \sigma_{\mathrm{AC}})), \\
&\pi_{\mathrm{priv}}(F_i(t', \sigma_{\mathrm{AC}})), \\
&\pi_{\mathrm{priv}}(F_{i+1}(\rho_{i+1}(\sigma_{\mathrm{AC}})\overline{\rho}_i(\sigma_{\mathrm{AC}})(t'), \sigma_{\mathrm{AC}})), \ldots, \\
&[\pi_{\mathrm{priv}}(F_n(\rho_n(\sigma_{\mathrm{AC}})\overline{\rho}_i(\sigma_{\mathrm{AC}})(t'), \sigma_{\mathrm{AC}}))]
\end{aligned}$$

Informally, $\rho_j(\sigma_{\mathrm{AC}})\overline{\rho}_i(\sigma_{\mathrm{AC}})(t + t') \in T_j$ corresponds to time $t + t' \in T_i$. Evaluating F_j at this time, from starting state $\sigma_{\mathrm{AC}} \in \Sigma_{\mathrm{AC}}$ is equivalent to first evaluating F_j at time $t' \in T_i$ (resulting in intermediate state $\sigma_{\mathrm{AC}}{}' \in \Sigma_{\mathrm{AC}}$), and then evaluating F_j at time $t \in T_i$ from starting state β. The proof is again by induction over $t \in T_i$.

6 Concluding Remarks

We have extended our existing models of microprocessors and their correctness to superscalar SMT and multi-core processor implementations, which represent the state-of-the-art in current commercial implementation. However, although we can successfully model such processors, and define what it means for them to be correct, practical verification of realistic examples would be a formidable undertaking. (This is generally the case: practical verifications of complete non-trivial processors are currently limited to non-superscalar pipelined processors.) There are some practical steps that can be taken to reduce complexity: we omit discussion here, but see [11]. Although helpful, these simplifications are unlikely to make realistic examples practical at the current time. Nonetheless, we feel that a considered approach to modeling processors and their correctness that runs ahead of actual application is useful: the modeling approaches to pipelined processors that were ultimately used to verify ARM6 [5, 6] were developed some years in advance of their practical use.

A point worthy of note is the presence of the definition of the AC level implementation in the definition of the VTM level model. This should not be surprising: in practice, the implementation of an SMT or multi-core processor does impact the behaviour seen by programmers; and the timing behaviour of all processors is a function of their implementation. This last fact is generally acknowledged in our model by the definition of retimings in terms of the AC-level model. There has been a general weakening of the long-established separation of processor architecture and implementation: good compilers for modern processors need to be aware of implementation details (e.g. how many functional units are there, and of what type) in order to generate high-quality code.

Finally, observe that a situation similar to SMT occurs with operating system kernels: a single physical processor presents as multiple virtual processors. The situation is somewhat different (in a kernel a privileged virtual processor at the higher level of abstraction, rather than the lower) but we believe the work here can be adapted to accommodate operating system kernels. Together with [15] on modelling high- and low-level languages and their relationships, this would produce a chain of fundamentally identical models from high-level languages to abstract hardware.

References

1. S Beyer, C Jacobi, D Kröning, D Leinenbach and W Paul. *Instantiating uninterpreted functional unit and memory system: Functional verification of VAMP* In D Geist and T Enrico, editors, *Correct Hardware Design and Verification Methods*, pages 51-65. Lecture Notes in Computer Science 2860, Springer-Verlag, 2003.
2. J Burch and D Dill. Automatic verification of pipelined microprocessor control. In D Dill, editor, *Proceedings of the 6th International Conference, CAV'94: Computer-Aided Verification*, pages 68 – 80. Lecture Notes in Computer Science 818, Springer-Verlag, 1994.

3. D Cyrluk, J Rushby, and M Srivas. Systematic formal verification of interpreters. In *IEEE international conference on formal engineering methods (ICFEM'97*, pages 140 – 149, 1997.
4. A C J Fox. *Algebraic Representation of Advanced Microprocessors.* PhD thesis, Department of Computer Science, University of Wales Swansea, 1998.
5. A C J Fox. *Formal specification and verification of ARM6.* In David Basin and Burkhart Wolff, editors, TPHOLs '03, volume 2758 of LNCS, pages 25-40. Springer-Verlag, 2003.
6. A C J Fox. *An algebraic framework for verifying the correctness of hardware with input and output: a formalization in HOL.*In J.L. Fiadeiro et al. (Eds.): CALCO 2005, LNCS 3629, pp. 157-174, 2005
7. A C J Fox and N A Harman. Algebraic models of superscalar microprocessor implementations: A case study. In B Möller and J V Tucker, editors, *Prospects for Hardware Foundations*, pages 138 – 183. Lecture Notes in Computer Science 1546, Springer-Verlag, 1998.
8. A C J Fox and N A Harman. Algebraic Models of Correctness for Microprocessors *Formal Aspects of Computing*, 12(4): 298–312, 2000.
9. A C J Fox and N A Harman. *Algebraic Models of Correctness for Abstract Pipelines. The Journal of Algebraic and Logic Programming*, 57 (2003), 71-107
10. N A Harman and J V Tucker. Algebraic models of microprocessors: Architecture and organisation. *Acta Informatica*, 33:421 – 456, 1996.
11. N A Harman. *Algebraic Models of Virtual Processor Implementations.* In preparation, 2005.
12. R Hosabettu, G Gopalakrishnan, and M Srivas, *Formal Verification of a Complex Pipelined Processor.* In Formal Methods in System Design, 23(2), 171–213, 2003.
13. S Miller and M Srivas. Formal verification of an avionics microprocessor. Technical report, SRI International Computer Science Laboratory CSL-95-04, 1995.
14. S Ray and W A Hunt .: *Deductive Verification of Pipelined Machines Using First-Order Quantification.* In Computer Aided Verification CAV 2004, pages 31–43, Springer-Verlag, Lecture Notes in Computer Science 3114, 2004.
15. K Stephenson. Algebraic specification of the Java virtual machine. In B Möller and J V Tucker, editors, *Prospects for Hardware Foundations.* Lecture Notes in Computer Science 1546, Springer-Verlag, 1998.
16. P Windley and J Burch. Mechanically checking a lemma used in an automatic verification tool. In A Camilleri and M Srivas, editors, *Formal methods in computer-aided design*, pages 362 – 376. Lecture Notes in Computer Science 1166, Springer-Verlag, 1996.

Finite Prediction of Recursive Real-Valued Functions

Eiju Hirowatari[1], Kouichi Hirata[2], and Tetsuhiro Miyahara[3]

[1] Center for Fundamental Education, The University of Kitakyushu
Kitakyushu 802-8577, Japan
eiju@kitakyu-u.ac.jp
[2] Department of Artificial Intelligence, Kyushu Institute of Technology
Iizuka 820-8502, Japan
hirata@ai.kyutech.ac.jp
[3] Faculty of Information Sciences, Hiroshima City University
Hiroshima 731-3194, Japan
miyahara@its.hiroshima-cu.ac.jp

Abstract. This paper concerns learning theory of *recursive real-valued functions* that are one of the formulations for the computable real function. Hirowatari *et al.* (2005) have introduced the *finite prediction* of recursive real-valued functions, which is based on a *finite prediction machine* that is a procedure to request *finite examples* of a recursive real-valued function f and a datum of a real number x, and to output a datum of a real number as the value of $f(x)$. In this paper, we newly establish the interaction of the criterion RealFP for finite prediction of recursive real-valued functions and the criteria RealEx, RealCons, RealFin and RealNum! for inductive inference of recursive real-valued functions.

1 Introduction

A *computable real function* [12, 13, 16], of which origin draws back into the classical work by Turing [15], is a model of the computations with continuous data like real numbers. Recently, the computable real function has been developed in a new research field of computational paradigm [4, 6] related to analysis, mathematical logic and computability. In this field, the computable real function is characterized from the logical viewpoints (*cf.*, [4, 6]).

A *recursive real-valued function* [8–10], which we mainly deal with in this paper, is one of the formulations for the computable real function. The recursive real-valued function is formulated as a function that maps a sequence of closed intervals which converges to a real number to a sequence of closed intervals which converges to another real number.

Inductive inference of recursive real-valued functions has been first introduced by Hirowatari and Arikawa [7] and developed by their co-authors [1, 8]. In their works, the criteria such as RealEx and RealNum! for inductive inference of recursive real-valued functions have been formulated as extensions of

Ex for *identification in the limit* and Num! for *identification by enumeration* [5], respectively, and their interaction has been widely studied (*cf.*, [3, 11]).

On the other hand, the *prediction* or *extrapolation* of recursive functions [2, 3] is to predict the n-th element of the sequence given the first $n-1$ elements by a prediction machine. The *prediction machine* is realized as simply an algorithmic device that accepts as input a finite (possibly empty) sequence of values and may output some value and halt, or may diverge.

Note that the examples in the prediction of recursive real-valued functions is not ordered, while the examples in the prediction of recursive functions implicitly assumed to be ordered, which is a reason why the criterion NV of the prediction is named after 'next value.' Then, Hirowatari *et al.* [9] have formulated *prediction* of recursive real-valued functions as *an infinite process* and introduced the criterion RealNV as similar as NV [2, 3].

In the criterion RealNV, it is necessary to assume that the prediction machine can receive infinitely many approximate values with any error bound of the target function, so RealNV is considered as a supervised learning process and is insufficient to capture the realistic prediction of functions. On the other hand, Hirowatari *et al.* [10] have formulated the *finite prediction*, that is, prediction of recursive real-valued functions from *finite examples*, and introduced another criterion RealFP.

The finite prediction is based on a *finite prediction machine* that is a procedure to request *finite examples* of a recursive real-valued function f and a datum of a real number x, and to output a datum of a real number as the value of $f(x)$. Since the criterion RealFP contains no infinite process and preserves *consistency* in the sense of RealCons [8], it is a more realistic model for prediction of recursive real-valued functions than RealNV.

Note here that this prediction model is concerned with our practical work on discovery of differential equations from numerical data [14]. However, in [10], there remains an open problem to compare the criterion RealFP with other criteria for inductive inference of recursive real-valued function given in [8].

Hence, in this paper, by comparing the criterion RealFP with other criteria RealEx, RealCons, RealFin and RealNum!, we newly establish the interaction of their criteria as in Fig. 1 (left). In particular, by focusing on the interaction between RealFP and RealCons, we show that RealFP and RealCons are incomparable, that is, RealCons$\setminus$RealFP $\neq \emptyset$ and RealFP$\setminus$RealCons $\neq \emptyset$.

2 Recursive Real-Valued Functions

In this section, we prepare some notions for a *recursive real-valued function*, which is one of the formulations for a computable real function [12, 13, 16]. Refer to papers [7–9] in more detail.

Let N, Q and R be the sets of all natural numbers, rational numbers and real numbers, respectively. By N^+ and Q^+ we denote the sets of all positive natural numbers and positive rational numbers, respectively. By $[a, b]$, we denote a closed

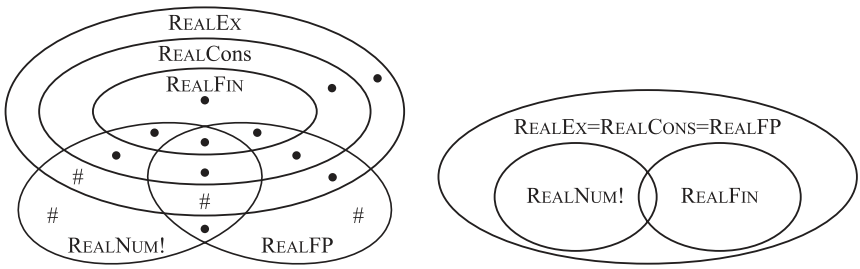


Fig. 1. The interaction of criteria for finite prediction and inductive inference of recursive real-valued functions (left) and one of recursive real-valued functions defined on a fixed rational closed interval $[8, 10]$ (right). We will show in this paper the existence of the functions in the place marked by $\bullet$.

interval, where $a, b \in R$ such that $a < b$. Furthermore, the length of a closed interval $[a, b]$ is defined as $b - a$.

Throughout of this paper, h is a real-valued function from S to R, where $S \subseteq R$. By $dom(h)$ we denote the domain of h, that is, $dom(h) = S$.

Definition 1. Let f and g be functions from N to Q and Q^+, respectively, and x a real number. We say that a pair $\langle f, g \rangle$ is an *approximate expression* of x if f and g satisfy the following conditions: (1) $\lim_{n\to\infty} g(n) = 0$, and (2) $|f(n) - x| \leq g(n)$ for each $n \in N$. Note here that $f(n)$ and $g(n)$ represent an *approximate value* of x and an *error bound* of x at point n, respectively. A real number x is *recursive* if there exists an approximate expression $\langle f, g \rangle$ of x such that f and g are recursive.

Definition 2. For $S \subseteq R$, a *rationalized domain* of S, denoted by RD_S, is a subset of $Q \times Q^+$ satisfying the following conditions:

(1) *Every interval in RD_S is contained in S.* For each $\langle p, \alpha \rangle \in RD_S$, it holds that $[p - \alpha, p + \alpha] \subseteq S$.
(2) *RD_S covers the whole S.* For each $x \in S$, there exists an element $\langle p, \alpha \rangle \in RD_S$ such that $x \in [p - \alpha, p + \alpha]$. In particular, if $x \in S$ is an interior point in S, then there exists an element $\langle p, \alpha \rangle \in RD_S$ such that $x \in (p - \alpha, p + \alpha)$.
(3) *RD_S is closed under subintervals.* For each $\langle p, \alpha \rangle \in RD_S$ and $\langle q, \beta \rangle \in Q \times Q^+$ such that $[q - \beta, q + \beta] \subseteq [p - \alpha, p + \alpha]$, it holds that $\langle q, \beta \rangle \in RD_S$.

For a real-valued function h, we denote $RD_{dom(h)}$ by RD_h simply.

Definition 3. Let h be a real-valued function. A *rationalized function* of h, denoted by $\mathcal{A}_h$, is a computable function from RD_h to $Q \times Q^+$ satisfying the following condition: For each $x \in dom(h)$, let $\langle f, g \rangle$ be an approximate expression of x. Then, there exists an approximate expression $\langle f_0, g_0 \rangle$ of $h(x)$ and it holds that $\mathcal{A}_h(\langle f(n), g(n) \rangle) = \langle f_0(n), g_0(n) \rangle$ for each $n \in N$ such that $\langle f(n), g(n) \rangle \in RD_h$.

For a real-valued function h, an *algorithm which computes* h means an algorithm which computes a rationaized function $\mathcal{A}_h$ of h.

Definition 4. A function h is a *recursive real-valued function* if there exists a rationalized function $\mathcal{A}_h : RD_h \to Q \times Q^+$ of h, where RD_h is a rationalized domain of $dom(h)$. We demand that $\mathcal{A}_h(\langle p, \alpha\rangle)$ does not halt for all $\langle p, \alpha\rangle \notin RD_h$. Furthermore, by $\mathcal{RRVF}$ we denote *the set of all recursive real-valued functions*.

For a set $S \subseteq R$, the set $Dom_S = \{\langle p, \alpha\rangle \in Q \times Q^+ \mid [p - \alpha, p + \alpha] \subseteq S\}$ is a rationalized domain of S. If a function $h : S \to R$ is a recursive real-valued function, then there exists a rationalized function $\mathcal{A}_h : Dom_S \to Q \times Q^+$ of h. Hence, without loss of generality, we can assume that a rationalized domain RD_h of $dom(h)$ is fixed to $\{\langle p, \alpha\rangle \in Q \times Q^+ \mid [p - \alpha, p + \alpha] \subseteq dom(h)\}$. Then, $\mathcal{A}_h(\langle p, \alpha\rangle)$ always halts for each $\langle p, \alpha\rangle$ such that $[p - \alpha, p + \alpha] \subseteq dom(h)$.

As an approximation of a real number x, we deal with a pair $\langle p, \alpha\rangle \in Q \times Q^+$ of rational numbers such that p is an approximate value of x and α is its error bound, i.e., $x \in [p - \alpha, p + \alpha]$. We call such a pair $\langle p, \alpha\rangle$ a *datum* of x.

Definition 5. An *example* of a recursive real-valued function h is a pair $\langle\langle p, \alpha\rangle, \langle q, \beta\rangle\rangle$ satisfying that there exists a real number $x \in dom(h)$ such that $\langle p, \alpha\rangle$ and $\langle q, \beta\rangle$ are data of x and $h(x)$, respectively.

Definition 6. A *presentation* of a recursive real-valued function h is an infinite sequence $\sigma = w_1, w_2, \ldots$ of examples of h in which, for each real number $x \in dom(h)$ and $\zeta > 0$, there exists an example $w_k = \langle\langle p_k, \alpha_k\rangle, \langle q_k, \beta_k\rangle\rangle$ such that $x \in [p_k - \alpha_k, p_k + \alpha_k]$, $h(x) \in [q_k - \beta_k, q_k + \beta_k]$, $\alpha_k \leq \zeta$ and $\beta_k \leq \zeta$.

By $\sigma[n]$ and $\sigma\{n\}$, we denote the initial segment of n examples in σ and the set of all examples in $\sigma[n]$, respectively.

An *inductive inference machine* (*IIM*, for short) is a procedure that requests inputs from time to time and produces algorithms, called *conjectures*, that compute recursive real-valued functions from time to time. Let σ be a presentation of a function. For $\sigma[n] = \langle w_1, w_2, \ldots, w_n\rangle$ and an IIM $\mathcal{M}$, by $\mathcal{M}(\sigma[n])$ we denote the last conjecture of $\mathcal{M}$ after requesting examples $w_1, w_2, \ldots, w_n$ as inputs.

Definition 7. Let σ be a presentation of a function and $\{\mathcal{M}(\sigma[n])\}_{n\geq 1}$ an infinite sequence of conjectures produced by an IIM $\mathcal{M}$. A sequence $\{\mathcal{M}(\sigma[n])\}_{n\geq 1}$ *converges* to an algorithm a_h if there exists a number $n_0 \in N$ such that $\mathcal{M}(\sigma[m])$ equals a_h for each $m \geq n_0$.

Definition 8. Let h be a recursive real-valued function and $\mathcal{T}$ a set of recursive real-valued functions. An IIM $\mathcal{M}$ RealEx-*infers* h, denoted by $h \in \text{RealEx}(\mathcal{M})$, if, for each presentation σ of h, the sequence $\{\mathcal{M}(\sigma[n])\}_{n\geq 1}$ converges to an algorithm that computes an extension of h.

Definition 9. Let h be a recursive real-valued function and $\mathcal{T}$ a set of recursive real-valued functions. An IIM $\mathcal{M}$ RealCons-*infers* h, denoted by $h \in \text{RealCons}(\mathcal{M})$, if it satisfies the following conditions.

1. $h \in \text{REALEX}(\mathcal{M})$.
2. For each presentation σ of h, conjecture $a_n = \mathcal{M}(\sigma[n])$ and $\langle\langle p, \alpha\rangle, \langle q, \beta\rangle\rangle \in \sigma[n]$ such that $[p-\alpha, p+\alpha] \subseteq S$, there exists an $x \in [p-\alpha, p+\alpha]$ such that $h_n(x) \in [q-2\beta, q+2\beta]$, where h_n is a recursive real-valued function and a_n is an algorithm which computes h_n.

Definition 10. Let h be a recursive real-valued function and $\mathcal{T}$ a set of recursive real-valued functions. An IIM $\mathcal{M}$ REALFIN-*infers* h, denoted by $h \in \text{REALFIN}(\mathcal{M})$, if for each presentation σ of h, $\mathcal{M}$ outputs a unique algorithm that computes an extension of h after some finite time from presented σ's data.

Let $\mathcal{T}$ be a set of recursive real-valued functions and $X \in \{\text{EX}, \text{CONS}, \text{FIN}\}$. Then, an IIM $\mathcal{M}$ REALX-*infers* $\mathcal{T}$ if $\mathcal{M}$ REALX-infers every $h \in \mathcal{T}$, and $\mathcal{T}$ is REALX-*inferable* if there exists an IIM that REALX-infers $\mathcal{T}$. By REALX, we denote *the class of all* REALX-*inferable sets of recursive real-valued functions*.

Definition 11. A set $\mathcal{T}$ of recursive real-valued functions is *recursively enumerable* if there exists a recursive function Ψ such that the set $\mathcal{T}$ is equal to the set of all functions computed by algorithms $\Psi(1), \Psi(2), \cdots$. By REALNUM!, we denote the class of all recursively enumerable sets of recursive real-valued functions. We also say that a set $\mathcal{T}$ of recursive real-valued functions is *recursively enumerable* if there exists a set $\mathcal{H}$ of recursive real-valued functions which is a set of extensions of functions in $\mathcal{T}$ and $\mathcal{H}$ is recursively enumerable.

We call functions x, $-x$, $\frac{1}{x}$, e^x, $\log x$, $\sin x$, $\arctan x$, $x^{\frac{1}{2}}$, $\arcsin x$ and the constant functions c_r for each recursive real number r *basic functions*. Here, $\frac{1}{x}$ for $x = 0$, $\log x$ for each $x \leq 0$, $x^{\frac{1}{2}}$ for each $x \leq 0$, and $\arcsin x$ for each $x \in R$ such that $|x| \geq 1$ are undefined as usual. By $\mathcal{BF}$ we denote the set of all basic functions.

Definition 12. By $\mathcal{EF}$ we denote the smallest set containing $\mathcal{BF}$ and satisfying the following condition: If $h_1, h_2 \in \mathcal{EF}$, then $h_1 + h_2, h_1 \times h_2, h_1 \circ h_2 \in \mathcal{EF}$. We say that a function in $\mathcal{EF}$ an *elementary function*.

We can show that every elementary function is a recursive real-valued function [7]. Hence, we can conclude that the class of recursive real-valued functions is rich enough to express the elementary functions with recursive real coefficient.

3 Finite Prediction Machine

In this section, we introduce a finite prediction machine of recursive real-valued functions.

Let $w = \langle\langle p, \alpha\rangle, \langle q, \beta\rangle\rangle$ be an example of h. Then, we can imagine an example w of h as a rectangle box $[p-\alpha, p+\alpha] \times [q-\beta, q+\beta]$. Then, a set $W = \{w_1, w_2, \ldots, w_n\}$ of such boxes is finite examples of h if each example contains a point $(x, h(x))$ on the graph of h.

A *finite prediction machine* (*FPM*, for short) is a procedure that requests finite examples of a recursive real-valued function and a datum of a real number, and that outputs a datum of a real number. For an FPM $\mathcal{P}$, finite examples W of a recursive real-valued function and a datum $\langle p, \alpha \rangle$ of a real number, by $\mathcal{P}(W, \langle p, \alpha \rangle)$, we denote the output of $\mathcal{P}$ after requesting W and $\langle p, \alpha \rangle$ as inputs. In this paper, we assume that $\mathcal{P}(W, \langle p, \alpha \rangle)$ is defined for each finite examples W of a recursive real-valued function and datum $\langle p, \alpha \rangle$ of a real number.

Definition 13. Let h be a recursive real-valued function, W finite examples of h, and $\langle p, \alpha \rangle$ a datum of a real number such that $[p-\alpha, p+\alpha] \cap dom(h) \neq \emptyset$. Then, we say that an FPM $\mathcal{P}$ *predicts* h *from* W *exactly* if the following conditions hold.

(1) $\langle \langle p, \alpha \rangle, \mathcal{P}(W, \langle p, \alpha \rangle) \rangle$ is an example of h.
(2) If $\langle p, \alpha \rangle \in RD_h$, then it holds that $h(x) \in [q - \beta, q + \beta]$ for each $x \in [p - \alpha, p + \alpha]$, where $\langle q, \beta \rangle = \mathcal{P}(W, \langle p, \alpha \rangle)$.
(3) For each $x \in dom(h)$ and each approximate expression $\langle f_x, g_x \rangle$ of x, there exists an approximate expression $\langle f_{h(x)}, g_{h(x)} \rangle$ of $h(x)$ satisfying that, for each $m \in N$ with $\langle f_x(m), g_x(m) \rangle \in RD_h$, it holds that $\langle f_{h(x)}(m), g_{h(x)}(m) \rangle = \mathcal{P}(W, \langle f_x(m), g_x(m) \rangle)$.

Now consider the set $\mathcal{T}$ of recursive real-valued functions. It is difficult that, for a target function $h \in \mathcal{T}$, an FPM predicts h from finite examples exactly. Then, we propose prediction from finite examples approximately which permits some error bound.

Definition 14. Let h be a recursive real-valued function, $\langle \langle p, \alpha \rangle, \langle q, \beta \rangle \rangle$ a datum of a recursive real-valued function and W finite examples of a recursive real-valued function. Then, we say that $\langle \langle p, \alpha \rangle, \langle q, \beta \rangle \rangle$ is *near to* h if there exists an $x \in [p - \alpha, p + \alpha]$ such that $h(x) \in [q - 2\beta, q + 2\beta]$. Furthermore, we say that W is *near to* h if there exists an $x \in [p - \alpha, p + \alpha]$ such that $h(x) \in [q - 2\beta, q + 2\beta]$ for each $\langle \langle p, \alpha \rangle, \langle q, \beta \rangle \rangle \in W$.

Definition 15. Let h be a recursive real-valued function, W finite examples of h, and $\langle p, \alpha \rangle$ a datum of a real number such that $[p - \alpha, p + \alpha] \cap dom(h) \neq \emptyset$. Then, we say that an FPM $\mathcal{P}$ *predicts* h *from* W *approximately* if there exists a recursive real-valued function h' such that the following conditions hold.

(1) W is near to h'.
(2) It holds that $dom(h) = dom(h')$.
(3) $\mathcal{P}$ predicts h' from W exactly.

For a target function h, by using a presentation σ of h, we introduce an FPM which finitely predicts h in the limit.

Definition 16. Let h be a recursive real-valued function. Then, we say that an FPM $\mathcal{P}$ *predicts* h *with limiting convergence* if, for each presentation σ of h, there exists a natural number $k \in N$ such that $\mathcal{P}(\sigma\{n\}, \langle p, \alpha \rangle) = \mathcal{P}(\sigma\{k\}, \langle p, \alpha \rangle)$ for each $n \in N$ such that $n \geq k$ and each $\langle p, \alpha \rangle \in Q \times Q^+$.

Definition 17. Let h be a recursive real-valued function. Then, we say that an FPM $\mathcal{P}$ *finitely predicts* h if the following conditions hold.

(1) $\mathcal{P}$ predicts h from W approximately, for each finite examples W of h.
(2) $\mathcal{P}$ predicts h with limiting convergence.

Also let $\mathcal{T}$ be a set of recursive real-valued functions. Then, we say that an FPM $\mathcal{P}$ *finitely predicts* $\mathcal{T}$ if $\mathcal{P}$ finitely predicts every $h \in \mathcal{T}$, and $\mathcal{T}$ is *finitely predictable* if there exists an FPM that finitely predicts $\mathcal{T}$. By RealFP, we denote *the class of all finitely predictable sets of recursive real-valued functions*.

4 Finite Prediction and Inductive Inference

In this section, we investigate the interaction of the criteria RealFP, RealEx, RealCons, RealFin and RealNum!. First we note that the following relations hold: RealFin $\subsetneq$ RealCons, RealCons $\subsetneq$ RealEx, RealFin$\cap$RealNum! $\neq \emptyset$, RealFin $\setminus$ RealNum! $\neq \emptyset$, and RealNum! $\setminus$ RealEx $\neq \emptyset$ [8].

Now let S be a subset of R such that there exist the following computable functions Com_S^1 and Com_S^2 from $Q \times Q^+$ to $\{0, 1\}$.

$$Com_S^1(\langle p, \alpha \rangle) = \begin{cases} 1 & \text{if } [p-\alpha, p+\alpha] \subseteq S, \\ 0 & \text{otherwise.} \end{cases}$$

$$Com_S^2(\langle p, \alpha \rangle) = \begin{cases} 1 & \text{if } [p-\alpha, p+\alpha] \cap S \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

For the set S, let $\mathcal{T}_S$ be a recursively enumerable set of recursive real-valued functions defined on a fixed domain S. Then, the following theorem holds.

Theorem 1. $\mathcal{T}_S \in$ RealFP $\cap$ RealCons.

Proof. Since $\mathcal{T}_S$ is recursively enumerable and the rationalized domain of S is recursive, there exists an IIM $\mathcal{M}$ which RealCons-infers $\mathcal{T}_S$.

For each function $h \in \mathcal{T}_S$, let W be finite examples of h, and $\langle p, \alpha \rangle$ a datum of a real number such that $[p-\alpha, p+\alpha] \cap S \neq \emptyset$. Furthermore, let σ be a presentation of h such that $\sigma\{n\} = W$ for an $n \in N$. Then, we can construct the following FPM $\mathcal{P}$.

$$\mathcal{P}(W, \langle p, \alpha \rangle) = \begin{cases} \mathcal{M}(\sigma[n])(\langle p, \alpha \rangle) & \text{if } [p-\alpha, p+\alpha] \subseteq S, \\ \mathcal{M}(\sigma[n])(\mathcal{B}_h(\langle p, \alpha \rangle)) & \text{if } [p-\alpha, p+\alpha] \not\subseteq S \\ & \quad \text{and } [p-\alpha, p+\alpha] \cap S \neq \emptyset, \\ \langle q, \beta \rangle & \text{otherwise,} \end{cases}$$

where $\langle q, \beta \rangle \in Q \times Q^+$. Here we can construct a computable function $\mathcal{B}_h$ which receives $\langle p, \alpha \rangle \in Q \times Q^+$ and outputs $\langle p_0, \alpha_0 \rangle \in Q \times Q^+$ such that $[p_0 - \alpha_0, p_0 + \alpha_0] \subseteq [p-2\alpha, p+2\alpha] \cap S$ and $[p_0 - \alpha_0, p_0 + \alpha_0] \cap [p-\alpha, p+\alpha] \neq \emptyset$. Hence, $\mathcal{P}$ finitely predicts h. $\square$

By φ_j we denote the partial recursive function from N to N computed by a program j. By $\mathcal{P}$ we denote the set $\{\varphi_0, \varphi_1, \varphi_2, \ldots\}$ of all partial recursive functions from N to N and by $\mathcal{R}$ the set of all recursive functions.

For the domain $S_j \subseteq N$ of $\varphi_j \in \mathcal{P}$ and $S = \bigcup_{i \in S_j} (i - \frac{1}{2}, i + \frac{1}{2})$, the following function $h_j : S \to R$ $(S \subseteq R)$ is called the *stair function* of φ_j: $h_j(x) = \varphi_j(i)\,(x \in (i - \frac{1}{2}, i + \frac{1}{2}), i \in S_0)$. For $\mathcal{S} \subseteq \mathcal{P}$, we call a stair function of a function in $\mathcal{S}$ a *stair function* in $\mathcal{S}$ simply.

Theorem 2. (1) $\textsc{RealFP} \cap \textsc{RealFin} \cap \textsc{RealNum}! \neq \emptyset$.
(2) $(\textsc{RealFP} \cap \textsc{RealCons} \cap \textsc{RealNum}!) \setminus \textsc{RealFin} \neq \emptyset$.
(3) $(\textsc{RealFP} \cap \textsc{RealFin}) \setminus \textsc{RealNum}! \neq \emptyset$.
(4) $(\textsc{RealFP} \cap \textsc{RealCons}) \setminus (\textsc{RealNum}! \cup \textsc{RealFin}) \neq \emptyset$.

Proof. (1) It is obvious that $\{h(x) = 0\} \in \textsc{RealFP} \cap \textsc{RealFin} \cap \textsc{RealNum}!$. (2) Let $\mathcal{C}_Q$ be the set of all constant functions $c_r : R \to Q$ such that $c_r(x) = r$ for each $r \in Q$. By Theorem 1, $\mathcal{C}_Q \in (\textsc{RealFP} \cup \textsc{RealCons} \cap \textsc{RealNum}!)$. Since $\mathcal{C}_Q \notin \textsc{RealFin}$, it holds that $\mathcal{C}_Q \in (\textsc{RealFP} \cap \textsc{RealCons} \cap \textsc{RealNum}!) \setminus \textsc{RealFin}$. (3) Let U be a subset of N which is not recursively enumerable and $\mathcal{C}_U$ the set of all constant functions $c_r : R \to U$ such that $c_r(x) = r$ for each $r \in U$. Then, it holds that $(\textsc{RealFP} \cap \textsc{RealFin}) \setminus \textsc{RealNum}! \neq \emptyset$. (4) Let S_0 be the set of all recursive functions f from N to N such that $\varphi_{f(0)} = f$ and $f(n) \leq 1$ for each $n \in N^+$, and $\mathcal{SF}_{S_0}$ the set of all stair functions in S_0. Since $\mathcal{SF}_{S_0}$ is the set of all stair functions in S_0, it holds that $\mathcal{SF}_{S_0} \in \textsc{RealFin} \setminus \textsc{RealNum}!$. Since $\textsc{RealFin} \subsetneq \textsc{RealCons}$, there exists an IIM $\mathcal{M}$ whose outputs are algorithms that computes a recursive real-valued functions defined on the domain $S_N = \bigcup_{i \in N} (i - \frac{1}{2}, i + \frac{1}{2})$, and then $\mathcal{M}$ $\textsc{RealCons}$-infers every $h \in \mathcal{SF}_{S_0}$.

For each function $h \in \mathcal{SF}_{S_0}$, let W be finite examples of h, and $\langle p, \alpha \rangle$ a datum of a real number such that $[p - \alpha, p + \alpha] \cap S_N \neq \emptyset$. Furthermore, let σ be a presentation of h such that $\sigma\{n\} = W$ for an $n \in N$. Then, we can construct the following FPM $\mathcal{P}$.

$$\mathcal{P}(W, \langle p, \alpha \rangle) = \begin{cases} \mathcal{M}(\sigma[n])(\langle p, \alpha \rangle) & \text{if } [p - \alpha, p + \alpha] \subseteq S_N, \\ \mathcal{M}(\sigma[n])(\mathcal{B}_0(\langle p, \alpha \rangle)) & \text{if } p - \alpha < -\frac{1}{2} \text{ and } |p + \alpha| < \frac{1}{2}, \\ \langle \frac{1}{2}, \frac{1}{2} \rangle & \text{otherwise.} \end{cases}$$

Here we can construct a computable function $\mathcal{B}_0$ which receives $\langle p, \alpha \rangle \in Q \times Q^+$ and outputs $\langle p_0, \alpha_0 \rangle \in Q \times Q^+$ such that $[p_0 - \alpha_0, p_0 + \alpha_0] \subseteq [p - 2\alpha, p + 2\alpha] \cap (-\frac{1}{2}, \frac{1}{2})$. Since $\mathcal{P}$ finitely predicts h, it holds that $\mathcal{SF}_{S_0} \in \textsc{RealFP}$.

Let $\mathcal{C}_Q^-$ be the set of all constant functions $c_r : (-\infty, -\frac{1}{2}) \to Q$ such that $c_r(x) = r$ for each $r \in Q$. Since $\mathcal{C}_Q^- \in (\textsc{RealFP} \cap \textsc{RealCons} \cap \textsc{RealNum}!) \setminus \textsc{RealFin}$, it holds that $\mathcal{SF}_{S_0} \cup \mathcal{C}_Q^- \in (\textsc{RealFP} \cap \textsc{RealCons}) \setminus (\textsc{RealNum}! \cup \textsc{RealFin})$. □

Theorem 3. (1) $\textsc{RealFin} \cap \textsc{RealNum}! \setminus \textsc{RealFP} \neq \emptyset$.
(2) $\textsc{RealFin} \setminus (\textsc{RealFP} \cup \textsc{RealNum}!) \neq \emptyset$.
(3) $(\textsc{RealCons} \cap \textsc{RealNum}!) \setminus (\textsc{RealFP} \cup \textsc{RealFin}) \neq \emptyset$.
(4) $\textsc{RealCons} \setminus (\textsc{RealFP} \cup \textsc{RealFin} \cup \textsc{RealNum}!) \neq \emptyset$.

Proof. Note that no FPM finitely predicts the following function $t(x)$ [9], where $\Phi_n(n)$ is a step counting function for computation of $\varphi_n(n)$.

$$t(x) = \begin{cases} |\tan \pi \left(n - \frac{1}{2^m} - \frac{1}{2}\right)| & \text{if } x \in [n - \frac{1}{2^m}, n + \frac{1}{2^m}] \text{ for } n, m \in N^+ \\ & \text{such that } \varphi_n(n) \text{ is defined and } \Phi_n(n) = m, \\ |\tan \pi(x - \frac{1}{2})| & \text{otherwise.} \end{cases}$$

Then, for the sets $\mathcal{C}_U$, $\mathcal{C}_Q$ and $\mathcal{SF}_{S_0} \cup \mathcal{C}_Q^-$ in the proof of Theorem 2, $\{t(x)\}$, $\mathcal{C}_U \cup \{t(x)\}$, $\mathcal{C}_Q \cup \{t(x)\}$ and $\mathcal{SF}_{S_0} \cup \mathcal{C}_Q^- \cup \{t(x)\}$ are the sets showing the statements, respectively. □

Theorem 4. (1) $(\text{RealFP} \cap \text{RealEx}) \setminus (\text{RealCons} \cup \text{RealNum!}) \neq \emptyset$.
(2) $\text{RealEx} \setminus (\text{RealFP} \cup \text{ReanCons} \cup \text{RealNum!}) \neq \emptyset$.

Proof. Let $\mathcal{R}_0$ be the set of all recursive function $f \in \mathcal{R}$ satisfying the following conditions: There exist $j \in N^+$ and $k \in N$ such that $f(k-1) = 0$, $f(k) = j$, and $f(n) \in \{1, 2\}$ for each $n \in N$ such that $n > k$. Furthermore, let $\mathcal{SF}_{\mathcal{R}_0}$ be the set of all stair functions in R_0. Then, $\mathcal{SF}_{\mathcal{R}_0} \in \text{RealEx} \setminus \text{RealCons}$.

(1) For each $h \in \mathcal{SF}_{\mathcal{R}_0}$, let W be finite examples of h, and $Point(W)$ a set of all $\langle x, y\rangle \in N \times N$ such that, for each $\langle\langle p_0, \alpha_0\rangle, \langle q_0, \beta_0\rangle\rangle \in W$, $|p_0 - x| < \min(\alpha_0, \frac{1}{2} - \alpha_0)$ and $|q_0 - y| < \min(\beta_0, \frac{1}{2} - \beta_0)$. Furthermore, let $\sigma_{Point(W)} = \langle x_1, y_1\rangle, \langle x_2, y_2\rangle, \ldots, \langle x_k, y_k\rangle$ be a finite sequence such that $Point(W) = \{\langle x_1, y_1\rangle, \langle x_2, y_2\rangle, \ldots, \langle x_k, y_k\rangle\}$ and $x_1 < x_2 < \ldots < x_k$. Then, there exists an algorithm Al that, after receiving finite examples W of h, behaves as follows: If there exist $j \in N^+$ and $t \in N$ such that $y_{t-1} = 0$, $y_t = j$, and $y_n \in \{1, 2\}$ for each $n \in N$ such that $t < n$ and $n \leq k$, then Al outputs $\langle t, j\rangle$, otherwise Al outputs 0, where $\sigma_{Point(W)} = \langle x_1, y_1\rangle, \langle x_2, y_2\rangle, \ldots, \langle x_k, y_k\rangle$.

By $Al(W)$, we denote the output of Al receiving W. Let π be a computable function from $N \times N$ to N such that $\pi(\langle x, y\rangle) = y$ for each $\langle x, y\rangle \in N \times N$.

For each $h \in \mathcal{SF}_{\mathcal{R}_0}$, h is defined on the domain $S_N = \bigcup_{i \in N}(i - \frac{1}{2}, i + \frac{1}{2})$. For each $\langle p, \alpha\rangle$ a datum of a real number such that $[p - \alpha, p + \alpha] \cap S_N \neq \emptyset$, we can construct the following FPM $\mathcal{P}$.

$$\mathcal{P}(W, \langle p, \alpha\rangle) = \begin{cases} \langle \varphi_{\pi(Al(W))}(x_*), \alpha\rangle) & \text{if } [p - \alpha, p + \alpha] \subseteq S_N, Al(W) \neq 0, \\ & \text{and } \Phi_{\pi(Al(W))}(x_*) \leq \frac{1}{\alpha} \\ \langle \frac{y_*}{2}, \frac{y_*}{2}\rangle & \text{otherwise,} \end{cases}$$

where $x_* \in N$ and $y_* \in Q$ such that $|x_* - p| < \frac{1}{2}$ and $y_* = \max\{p + \alpha \mid \langle\langle p, \alpha\rangle, \langle q, \beta\rangle\rangle \in W\}$. Here, $\Phi_n(m)$ is a step counting function for computation of $\varphi_n(m)$. Hence, $\mathcal{P}$ finitely predicts h.

(2) By using the set $\{t(x)\}$ in the proof of Theorem 3, it holds that $\mathcal{SF}_{\mathcal{R}_0} \cup \{t(x)\} \in \text{RealEx} \setminus (\text{RealFP} \cup \text{ReanCons} \cup \text{RealNum!})$. □

Theorem 5. $(\text{RealFP} \cap \text{RealNum!}) \setminus \text{RealEx} \neq \emptyset$.

Proof. Let $\hat{\mathcal{H}}$ be $\{\hat{h}, h_0, h_1, h_2, \cdots\}$, where h_i and $\hat{h}$ are the following recursive real-valued functions for each $i \in N$.

$$h_i(x) = \begin{cases} 1 & \text{if } x \leq 0, \\ 0 & \text{if } x > \frac{1}{2^i}. \end{cases} \qquad \hat{h}(x) = \begin{cases} 1 & \text{if } x < 0, \\ 0 & \text{if } x > 0. \end{cases}$$

Then, it holds that $\hat{\mathcal{H}} \in \text{RealFP} \cap \text{RealNum!} \setminus \text{RealEx}$. □

5 Conclusion

In this paper, we have compared the criterion RealFP for finite prediction of recursive real-valued functions with the criteria RealEx, RealCons, RealFin and RealNum! for inductive inference of recursive real-valued functions and shown the interaction as Fig. 1 (left), where it is open whether or not the set marked by # is not empty. Furthermore, we have shown the interaction of the criteria defined on a fixed rational closed interval as Fig. 1 (right) [8, 10]. While we have shown the interaction between RealFP and RealCons, we have not characterized the interaction between RealNV and RealCons. Note that the relationship RealFP $\subseteq$ RealNV follows from the definitions of RealFP and RealNV [9]. Hence, it is a future work to investigate the interaction between RealFP, RealNV and RealCons in more detail.

References

1. K. Apsītis, S. Arikawa, R. Freivalds, E. Hirowatari, C. H. Smith, *On the inductive inference of recursive real-valued functions*, Theoret. Comput. Sci. **219**, 3–17, 1999.
2. J. M. Bārzdiņš, R. V. Freivalds, *On the prediction of general recursive functions*, Soviet Mathematics Doklady **13**, 1224–1228, 1972.
3. J. Case, C. Smith, *Comparison of identification criteria for machine inductive inference*, Theoret. Comput. Sci. **25**, 193–220, 1983.
4. S. B. Cooper, B. Löwe, L. Torenvliet (eds.), *New computational paradigms, Proc. 1st International Conference on Computability in Europe*, LNCS **3526**, 2005.
5. E. M. Gold, *Language identification in the limit*, Inform. Control **10**, 447–474, 1967.
6. T. Grubba, P. Hertling, H. Tsuiki, K. Weihrauch (eds.), *Proc. 2nd International Conference on Computability and Complexity in Analysis*, 2005.
7. E. Hirowatari, S. Arikawa, *Inferability of recursive real-valued functions*, Proc. ALT'97, LNAI **1316**, 18–31, 1997.
8. E. Hirowatari, S. Arikawa, *A comparison of identification criteria for inductive inference of recursive real-valued functions*, Theoret. Comput. Sci. **268**, 351–366, 2001.
9. E. Hirowatari, K. Hirata, T. Miyahara, S. Arikawa, *On the prediction of recursive real-valued functions*, Proc. Computability in Europe 2005, 93-103, 2005.
10. E. Hirowatari, K. Hirata, T. Miyahara, *Prediction of recursive real-valued functions from Finite Examples*, Proc. JSAI 2005 Workshops, LNAI **4012**, 224–234, 2006.
11. S. Jain, D. Osherson, J. S. Royer, A. Sharma, *Systems that learn: An introduction to learning theory (2nd ed.)*, The MIT Press, 1999.
12. K. Ko. *Complexity theory of real functions*, Birkhäuser, 1991.
13. M. B. Pour-El, J. I. Richards, *Computability in analysis and physics*, Springer-Verlag, 1988.
14. K. Niijima, H. Uchida, E. Hirowatari, S. Arikawa, *Discovery of differential equations from numerical data*, Proc. DS'98, LNAI **1532**, 364–374, 1998.
15. A. M. Turing, *On computable numbers, with the application to the Entscheidungsproblem*, Proc. London Mathematical Society **42**, 230–265, 1936.
16. K. Weihrauch, *Computable analysis – An introduction*, Springer-Verlag, 2000.

The Dyment Reducibility on the Algebraic Structures and on the Families of Subsets of ω

Iskander Kalimullin[1]

Kazan State University, Kazan, 420008, Kremlevskaya str. 18, Russia, e-mail: Iskander.Kalimullin@ksu.ru

Abstract. In the paper a positive reducibility on the special noncomputable algebraic structures and the connected reducibilities on the families of subsets of ω are studied.

Keywords: algebraic structure, enumeration operator, partial computable function, computably enumerable sets, c.e. families of subsets of ω.

1 Introduction

Following Dyment [3] we will call any set P of partial functions on ω *partial mass problem*. Extending the Medvedev reducibility [6] on the total mass problems Dyment defined the corresponding reducibility for any partial mass problems. Namely $P \leq Q$ iff there is a enumeration operator Ψ such that for any $\psi \in Q$, $\Psi(\psi)$ is a partial function and $\Psi(\psi) \in P$.

In this paper we study the Dyment reducibility on partial mass problems of presentability of special algebraic structures. Given a countable structure $\mathcal{A}$ we consider the partial mass problem $P_D(\mathcal{A})$ of presentability of $\mathcal{A}$ containing all atomic diagrams $D(\mathcal{B})$, where $\mathcal{B} \cong \mathcal{A}$ and the universe of $\mathcal{B}$ is a subset of ω. If we require here that the universe of $\mathcal{B}$ is equal to ω then we get the Medvedev mass problem $P_M(\mathcal{A})$ of presentability of $\mathcal{A}$.

It is easy to check that $P_D(\mathcal{A}) \leq P_D(\mathcal{B})$ implies $P_M(\mathcal{A}) \leq P_M(\mathcal{B})$ and the reverse does not hold (see, e.g., Proposition 2). Also the reducibility $P_D(\mathcal{A}) \leq P_D(\mathcal{B})$ is equivalent to the computable embedding of the class $\{\mathcal{C} : \mathcal{C} \cong \mathcal{B}\}$ into the class $\{\mathcal{C} : \mathcal{C} \cong \mathcal{A}\}$ in the sense of the paper [1].

Let $\mathcal{S}$ be a family of subsets of ω. There are many natural ways of coding the family into an algebraic structure. For example, let $\mathcal{A}_\mathcal{S}$ be the structure with three binary predicates T^0, T^1, T^2 on the universe

$$\{a_j^{X,i} : X \in \mathcal{S} \ \& \ i \in \omega \ \& \ j \in \omega\} \cup \{b_j^{X,i} : X \in \mathcal{S} \ \& \ i \in \omega \ \& \ j \in X\}$$

such that

$$T^0_{\mathcal{A}_\mathcal{S}} = \{\langle a_0^{X,i}, a_j^{X,i}\rangle : X \in \mathcal{S} \ \& \ i \in \omega \ \& \ j \in \omega\},$$

$$T^1_{\mathcal{A}_\mathcal{S}} = \{\langle a_j^{X,i}, a_{j+1}^{X,i}\rangle : X \in \mathcal{S} \ \& \ i \in \omega \ \& \ j \in \omega\},$$

$$T^2_{\mathcal{A}_\mathcal{S}} = \{\langle a^{X,i}_j, b^{X,i}_j\rangle : X \in \mathcal{S} \,\&\, i \in \omega \,\&\, j \in X\}.$$

Then, for a countable family $\mathcal{S}$ the partial mass problem $P_D(\mathcal{A}_\mathcal{S})$ is equivalent to the partial mass problem $P_E(\mathcal{S})$ containing all *partial enumerations* of $\mathcal{S}$, i.e., all (semicharacteristic functions of) sets R such that

$$\{\emptyset\} \cup \mathcal{S} = \{R^{(x)} : x \in \omega\}, \text{ where } R^{(x)} = \{y \in \omega : \langle x, y\rangle \in R\}.$$

We will say that $\mathcal{S}_1$ *is Dyment reducible to* $\mathcal{S}_2$ (and write $\mathcal{S}_1 \leq \mathcal{S}_2$) for a countable families $\mathcal{S}_1$ and $\mathcal{S}_2$ iff $P_D(\mathcal{A}_{\mathcal{S}_1}) \leq P_D(\mathcal{A}_{\mathcal{S}_2})$, or equivalently $P_E(\mathcal{S}_1) \leq P_E(\mathcal{S}_2)$.

Under this reducibility the zero degree consists of all c.e. families (i.e., the families with c.e. partial enumerations), e.g., this contains the family CE of all c.e. sets.

Notations. For any $X \subseteq \omega$ and $x \in \omega$ we will write

$$X^{(x)} = \{y : \langle x, y\rangle \in X\};$$

$$X^{[x]} = \{\langle x, y\rangle : \langle x, y\rangle \in X\};$$

$$X^{[>x]} = \{\langle z, y\rangle : z > x \,\&\, \langle z, y\rangle \in X\};$$

$$X\lceil x = \{y : y < x \,\&\, y \in X\};$$

$$x + X = \{x + y : y \in X\};$$

$$\langle x, X\rangle = \{\langle x, y\rangle : y \in X\}.$$

The definitions and basic properties of the enumeration operators and the e-reducibility can be find in the monograph [7]. In particular, each e-operator Φ can be defined by the corresponding c.e. set of axioms $\langle x, F\rangle$, where $x \in \omega$ and F is a finite subset of ω, so that for any $X \subseteq \omega$

$$\Phi(X) = \{x \in \omega : (\exists F \subseteq X)[\langle x, F\rangle \in W]\}.$$

The jump $\mathbf{a}'$ of an e-degree $\mathbf{a}$ is the e-degree of the graph of the set $K(A) = \{x : x \in \Phi_x(A)\}$, where $A \in \mathbf{a}$ and $\{\Phi_x\}_{x\in\omega}$ is an acceptable numbering of all enumeration operators (see [2] for further information). In particular, $\mathbf{0}'_e$ is the e-degree of complements $\overline{K}$ of creative sets K.

2 Natural families of c.e. sets and the $\leq_\sigma$-reducibility

To illustrate the Dyment reducibility consider the following families:
$InfCE$—the family of infinite c.e. sets;
$InfCom$—the family of infinite computable sets;
$ComF$—the family of graphs of computable functions.

Note that these families are not c.e. (for a degree $\mathbf{a}$ these families have $\mathbf{a}$-c.e. enumerations iff $\mathbf{0}'' \leq \mathbf{a}'$, see [4]).

Proposition 1. *(Kalimullin, Puzarenko [5]).* $InfCE \equiv InfCom \equiv ComF \cup \{\omega\}$, *and hence* $InfCE \leq ComF$.

Proof. The reducibility $InfCE \leq InfCom$ follows from the fact that each infinite c.e. set contains an infinite computable set. Hence

$$InfCE = \{B \cup W_m : B \in InfComp \ \& \ m \in \omega\}.$$

Then, for any partial enumeration R of $InfCom$ the set

$$\{\langle\langle m+1, n\rangle, x\rangle : R^{(n)} \neq \emptyset \ \& \ x \in R^{(n)} \cup W_m\}$$

is a partial enumeration of $InfCE$.

To prove $InfCom \leq ComF \cup \{\omega\}$, we will use that the infinite computable sets coincides with the images of increasing computable functions. Define the e-operator Θ, such that $\Theta(B) = \omega$, if there are integers $x < y$ and $b \leq a$ such that $\langle x, a\rangle \in B$ and $\langle y, b\rangle \in B$, and otherwise $\Theta(B) = \{a : (\exists x)[\langle x, a\rangle \in B]\}$. Then $InfCom = \{\Theta(B) : B \in ComF \cup \{\omega\}\}$ and for any partial enumeration R of $ComF \cup \{\omega\}$ the set $\{\langle n, x\rangle : x \in \Theta(R^{(n)})\}$ is a partial enumeration of $InfCom$.

It remains to prove $ComF \cup \{\omega\} \leq InfCE$. For each partial computable function φ_m define the partial computable function $c_m(x) = (\mu s)[\varphi_{m,s}(x) \downarrow]$. Fix an injective effective listing $\{\tau_n\}_{n\in\omega}$ of all finite strings in the alphabet ω (which will be considered also as partial functions whose domains are the initial segments of ω).

Let L_m be the set consisting of all n such that $\tau_n(x) = c_m(x)$ for all $x < |\tau_n|$. Clearly, the binary predicate $n \in L_m$ is computable, and L_m is infinite iff φ_m is total. Then there is computable sequence of e-operators Θ_m, $m \in \omega$, such that $\Theta_m(B) = \text{graph } (\varphi_m)$, if $B \subseteq L_m$, and $\Theta_m(B) = \omega$ otherwise.

It follows, that $ComF \cup \{\omega\} = \{\Theta_m(B) : B \in InfCE \ \& \ m \in \omega\}$ and, therefore, for any partial enumeration R of $InfCE$ the set

$$\{\langle\langle m+1, n\rangle, y\rangle : R^{(n)} \neq \emptyset \ \& \ y \in \Theta_m(R^{(n)})\}$$

is a partial enumeration of $ComF \cup \{\omega\}$.

□

Proposition 2. *(Kalimullin, Puzarenko [5]). $ComF \not\leq InfCE$.*

Proof. Suppose that $P_E(ComF) \leq P_E(InfCE)$ via an e-operator Φ. Then the c.e. set $\{\langle x+1, y\rangle : y \in \Psi(\omega)^{(x)}\}$ will be a partial enumeration of $ComF$, but this is impossible since $ComF$ is not c.e.

Indeed, first note that for any finite set F there a partial enumeration $R \supseteq F$ of $InfCE$, since $\omega \in InfCE$. Then $\Psi(\omega)^{(x)}$ is a graph of a partial function for any $x \in \omega$. Moreover, if $\Psi(\omega)^{(x)} \neq \emptyset$ then $\Psi(\omega)^{(x)}$ is a graph of a total (and computable) function. Let f be a total computable function. Then for any partial enumeration R of $InfCE$ there is a column x such that $\Psi(R)^{(x)} = \text{graph } (f)$, hence for some x we have graph $(f) = \Psi(\omega)^{(x)}$ since the last is a graph of a function. □

Note that in contrast with the last result it is easy to check that

$$P_M(\mathcal{A}_{ComF}) \equiv P_M(\mathcal{A}_{InfCom}) \equiv P_M(\mathcal{A}_{InfCE})$$

and this gives an example when $P_M(\mathcal{A}) \leq P_M(\mathcal{B})$ and $P_D(\mathcal{A}) \not\leq P_D(\mathcal{B})$.

All reducibilities $\mathcal{S}_1 \leq \mathcal{S}_2$ from Proposition 1 are provided by computable sequences $\{\Phi_i\}_{i\in\omega}$ of e-operators (and in some cases by only one e-operator) such that

$$\mathcal{S}_1 = \{\Phi_i(X) : X \in \mathcal{S}_2 \ \& \ i \in \omega\}.$$

Kalimullin and Puzarenko [5] introduced and studied a generalization of this special case of Dyment reducibility.

Definition 1. *(Kalimullin, Puzarenko [5]). 1) Let $\mathcal{S}$ be a family of subsets of ω. Any set $\{\langle n, m\rangle\} \oplus (A_1 \oplus \ldots \oplus A_m)$, where $A_i \in \mathcal{S}$ for each i, $1 \leq i \leq m$, we will call an $\mathcal{S}$-cortege. The family of all $\mathcal{S}$-corteges we denote via $Q(\mathcal{S})$.*
2) We write $\mathcal{S}_1 \leq^0_\sigma \mathcal{S}_2$ if for some e-operator Φ we have

$$\{\emptyset\} \cup \mathcal{S}_1 = \{\Phi(B \oplus E(\mathcal{S}_2)) : B \in Q(\mathcal{S}_2)\},$$

where $E(\mathcal{S}_2)$ is the set of (canonical indices of) finite sets F such that $F \subseteq X$ for some $X \in \mathcal{S}_2$ (in fact, the set $E(\mathcal{S}_2)$ is e-equivalent to the $\exists$-theory of $\mathcal{A}_{\mathcal{S}_2}$).
2) We write $\mathcal{S}_1 \leq_\sigma \mathcal{S}_2$ if for some e-operator Φ and an $\mathcal{S}_2$-cortege A we have

$$\{\emptyset\} \cup \mathcal{S}_1 = \{\Phi(A \oplus B \oplus E(\mathcal{S}_2)) : B \in Q(\mathcal{S}_2)\}.$$

Note that if $\mathcal{S}_2$ is a countable family, then $\mathcal{S}_1 \leq^0_\sigma \mathcal{S}_2$ implies $\mathcal{S}_1 \leq \mathcal{S}_2$, and if $\mathcal{S}_2$ consists of c.e. sets (or even of sets e-reducible to $E(\mathcal{S}_2)$), then $\mathcal{S}_1 \leq_\sigma \mathcal{S}_2$ is equivalent to $\mathcal{S}_1 \leq^0_\sigma \mathcal{S}_2$.

In contrast with Dyment reducibility the $\leq_\sigma$- and $\leq^0_\sigma$-reducibilities can be considered on uncountable families. Also, these are tightly connected with Σ-definability on admissible sets.

Theorem 1. *(Kalimullin, Puzarenko [5]). 1) $\mathcal{S}_1 \leq^0_\sigma \mathcal{S}_2$ iff $\mathcal{S}_1$ is Σ-definable in $\mathbb{HF}(\mathcal{A}_{\mathcal{S}_2})$ without parameters.*
2) $\mathcal{S}_1 \leq_\sigma \mathcal{S}_2$ iff $\mathcal{S}_1$ is Σ-definable in $\mathbb{HF}(\mathcal{A}_{\mathcal{S}_2})$ with parameters.

Thus, for a fixed family $\mathcal{S}_2$ the families $\mathcal{S}_1 \leq_\sigma \mathcal{S}_2$ precisely are the c.e. families of subsets of ω in the hereditary finite superstructure over the structure $\mathcal{A}_{\mathcal{S}_2}$. For all simple examples of countable families we always have $\mathcal{S}_1 \leq \mathcal{S}_2$ iff $\mathcal{S}_1 \leq^0_\sigma \mathcal{S}_2$, so that it is natural to state the following problem.

Problem 1. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be countable families.
1) Is $\mathcal{S}_1 \leq^0_\sigma \mathcal{S}_2$ equivalent to $\mathcal{S}_1 \leq \mathcal{S}_2$?
2) Is $\mathcal{S}_1 \leq_\sigma \mathcal{S}_2$ equivalent to $P_D(\mathcal{A}_{\mathcal{S}_1}) \leq P_D(\mathcal{A}_{\mathcal{S}_2}, c_1, \ldots, c_m)$ for some $c_1, \ldots c_m \in A_{\mathcal{S}_2}$?

In the case of positive answer we get an arithmetical characterization of Dyment reducibily of two families $\mathcal{S}_1$ and $\mathcal{S}_2$ presented by their partial enumerations R_1 and R_2. Namely, the predicate $\{R_1^{(x)} : x \in \omega\} \leq_\sigma^0 \{R_2^{(x)} : x \in \omega\}$ is Σ_4^0 under R_1 and R_2 by the definition of $\leq_\sigma^0$. In contrast, for one-element families we have $\{A\} \leq \{B\}$ iff $A \leq_e B$ and the last condition is Σ_3^0 by the definition of enumeration reducibility.

The author still does not know an easy counterexample for the negative answer of this problem, but an existence of such counterexamples follows from a structural difference between $\leq$- and $\leq_\sigma$-reducibilities. In general, there is a feeling that $\leq_\sigma$ is more like Turing and enumeration reducibilities on subsets of ω. In particular, the following theorem can be proved.

Theorem 2. *If a family $\mathcal{S}$ is not c.e. then for any sequence of sets X_i, $i \in \omega$, there is a set A such that $\mathcal{S} \not\leq_\sigma \mathcal{F}(A)$ and $A \not\leq_e X_i$ for all $i \in \omega$, where $\mathcal{F}(A)$ is the family containing all singletons $\{x\}$ for $x \in A$ and the set ω.*

Proof. Since for all $A \subseteq \omega$ each element of $\mathcal{F}(A)$ is c.e. and $E(\mathcal{F}(A)) = \omega$ it suffices to construct a set A satisfying the requirements

$$P_{\langle i,j\rangle} : A \neq \Phi_j(X_i);$$

$$N_n : \mathcal{S}^0 \neq \{\Phi_n(C) : C \in Q(\mathcal{F}(A))\}$$

for every $i, j, n \in \omega$, where $\mathcal{S}^0 = \mathcal{S} \cup \{\emptyset\}$ and $\{\Phi_n\}_{n\in\omega}$ is a computable sequence of all e-operators.

At each stage s of the construction below we will define a finite set A_s such that $A_s \subseteq A_{s+1}$ and $A = \cup_s A_s$ will be the desired set. At each stage s we will also define an auxiliary finite set B_s such that $B_s \subseteq B_{s+1}$ and $A_t \cap B_s = \emptyset$ for all s and t.

Construction.

Stage $s = 0$. $A_0 = B_0 = \emptyset$.

Stage $s+1 = 2\langle i, j\rangle+1$. Let $x_{\langle i,j\rangle} = (\mu x)[x \notin A_s \cup B_s]$. If $x_{\langle i,j\rangle} \in \Phi_j(X_i)$ define $A_{s+1} = A_s$ and $B_{s+1} = B_s \cup \{x_{\langle i,j\rangle}\}$. If $x_{\langle i,j\rangle} \notin \Phi_j(X_i)$ define $A_{s+1} = A_s \cup \{x_{\langle i,j\rangle}\}$ and $B_{s+1} = B_s$.

Stage $s + 1 = 2n + 2$. Since $\mathcal{S}$ is not c.e. we have

$$\mathcal{S}^0 \neq \{\Phi_n(C) : C \in Q(\mathcal{F}(\overline{B_s}))\}.$$

Let Y_n be such set that

$$Y_n \in (\mathcal{S}^0 - \{\Phi_n(C) : C \in Q(\mathcal{F}(\overline{B_s}))\}) \cup (\{\Phi_n(C) : C \in Q(\mathcal{F}(\overline{B_s}))\} - \mathcal{S}^0).$$

If $Y_n \in \mathcal{S}^0 - \{\Phi_n(C) : C \in Q(\mathcal{F}(\overline{B_s}))\}$ then define $A_{s+1} = A_s$ and $B_{s+1} = B_s$.

Suppose that $Y_n \in \{\Phi_n(C) : C \in Q(\mathcal{F}(\overline{B_s}))\} - \mathcal{S}^0$. Then we can fix a finite set $G_n \subseteq \overline{B_s}$ such that $Y_n = \Phi_n(C_n)$ for some $C_n \in Q(\mathcal{F}(G_n))$. Define $A_{s+1} = A_s \cup G_n$ and $B_{s+1} = B_s$.

The description of the construction is finished.

Let $A = \cup_s A_s$. Since $A_s \cap B_t = \emptyset$ for all $s, t \in \omega$, we have $x_{\langle i,j\rangle} \in A \iff x_{\langle i,j\rangle} \notin \Phi_j(X)$ for each $i, j \in \omega$. Hence, all requirements $P_{\langle i,j\rangle}$ are satisfied and $A \not\leq_e X_i$ for each $i \in \omega$.

To show that all requirements N_n are satisfied fix arbitrary $n \in \omega$ and consider the stage $s + 1 = 2n + 2$ of the construction.

If at this stage we have $Y_n \in \{\Phi_n(C) : C \in Q(\mathcal{F}(\overline{B_s}))\} - \mathcal{S}^0$ then the construction provides

$$Y_n \in \{\Phi_n(C) : C \in Q(\mathcal{F}(A_{s+1}))\} \subseteq \{\Phi_n(C) : C \in Q(\mathcal{F}(A))\}.$$

Otherwise we have $Y_n \in \mathcal{S}^0 - \{\Phi_n(C) : C \in Q(\mathcal{F}(\overline{B_s}))\}$, and hence

$$Y_n \notin \{\Phi_n(C) : C \in Q(\mathcal{F}(A))\},$$

since $A \subseteq \overline{B}_s$ and $\{\Phi_n(C) : C \in Q(\mathcal{F}(A))\} \subseteq \{\Phi_n(C) : C \in Q(\mathcal{F}(\overline{B_s}))\}$.

Thus, $Y_n \in \mathcal{S}^0 \iff Y_n \notin \{\Phi_n(C) : C \in Q(\mathcal{F}(A))\}$. Therefore, each requirements N_n is satisfied. □

3 $\leq$ is not equivalent to $\leq^{\mathbf{0}}_{\boldsymbol{\sigma}}$

Note that Theorem 2 does not hold for the Medvedev reducibility on the families. Moreover, Wehner [8] found a non-c.e. family $\mathcal{C}$ of finite sets such that $P_M(\mathcal{A}_\mathcal{C}) \leq P_M(\mathcal{B})$ for any noncomputable countable structure $\mathcal{B}$.

Definition 2. *(Wehner [8]). 1) For each $i, n, s \in \omega$ let $\Omega^{(n)}_{i,s} = \{x \in \omega : \langle i, x\rangle \in W_{n,s}\}$ and $\Omega^{(n)}_i = \cup_s \Omega^{(n)}_{i,s} = \{x \in \omega : \langle i, x\rangle \in W_n\}$.*
2) For each $n \in \omega$ define the c.e. family $\mathcal{C}_n = \{\Omega^{(n)}_i : i \in \omega\}$. Note that each c.e. family $\mathcal{F}$ coincides with $\mathcal{C}_n$ for some $n \in \omega$.
3) Define the partial computable function $r(n) = ((\mu\langle i, x, s\rangle)[\langle n, x\rangle \in \Omega^{(n)}_{i,s}])_1$. It is easy to see that $(\Omega^{(n)}_{r(n)})^{[n]} \neq \emptyset$ if $r(n) \downarrow$.
4) Fix a computable function g such that $(\Omega^{(n)}_{r(n)})^{[n]} = \langle n, W_{g(n)}\rangle$ if $r(n) \downarrow$ and $W_{g(n)} = \emptyset$ otherwise.
5) Let $\mathcal{C}$ be the family defined by

$$\mathcal{C} = \{\langle n, V\rangle : n \in \omega \,\&\, V \text{ is finite} \,\&\, V \neq W_{g(n)}\}.$$

Theorem 3. *(Wehner [8]) 1) $\mathcal{C}$ is not c.e.*
2) There is a computable sequence of partial Turing functionals Δ_n, $n \in \omega$ such that for any noncomputable set X each function Δ^X_n is total and $\{0, 1\}$-valued, and

$$\mathcal{C} = \{C : (\exists n \in \omega)[\chi_C = \Delta^X_n]\}.$$

Hence, $P_M(\mathcal{A}_\mathcal{C}) \leq Q$ for any total mass problem Q without computable elements.

Unfortunately, it is impossible to strength this theorem for the Dyment reducibility. Moreover, for any computable sequence of e-operators Ψ_n, $n \in \omega$, there are continually many sets $X \subseteq \omega$ such that $\mathcal{C} \neq \{\Psi_n(X) : n \in \omega\}$. For this reason, we make a correction in the Wehner's family (allowing infinite but c.e. elements), so that at this new family the Dyment reducibility has no the same property as $\leq_\sigma$ in Theorem 2.

Definition 3. *Let $\mathcal{D}$ be the family defined by*

$$\mathcal{D} = \{\langle n, V\rangle : n \in \omega \ \&\ V \text{ is c.e. } \&\ V \neq W_{g(n)}\}.$$

Proposition 3. *$\mathcal{D}$ is not c.e.*

Proof. Suppose that $\mathcal{D} = \Omega^{(n)}$ for some $n \in \omega$. Then $r(n) \downarrow$ and $\Omega^{(n)}_{r(n)} \in \mathcal{D}$. Therefore, $\Omega^{(n)}_{r(n)} = (\Omega^{(n)}_{r(n)})^{[n]} = \langle n, W_{g(n)}\rangle$. This contradicts with the definition of $\mathcal{D}$. □

Proposition 4. $\emptyset \in \mathcal{D}$.

Proof. By Proposition 3 there is $n_0 \in \omega$ such that $W_{g(n_0)} \neq \emptyset$, since otherwise

$$\mathcal{D} = \{\langle n, W_m \cup \{k\}\rangle : n, m, k \in \omega\}.$$

Therefore, $\emptyset = \langle n_0, \emptyset\rangle \in \mathcal{D}$. □

Proposition 5. *There is a computable sequence of e-operators Θ_n, $n \in \omega$, such that for each $n \in \omega$ and $X \notin \Delta^0_2$ the set $\Theta_n(X)$ is computable and $\Theta_n(X) \neq W_n$.*

Proof. We define each e-operator Θ_n, $n \in \omega$, by induction on stages $s \in \omega$

Stage $s = 0$. $\Theta_{n,0} = \emptyset$.

Stage $s+1$. For a finite set F we say that an axiom $\langle\langle x, y\rangle, F\rangle$ is *axiom of type I at the stage $s+1$*, if $x < s, y < s$, $W_{n,s}\lceil x \subseteq \Theta_{n,s}(F)$ and $\Theta_{n,s}(F)\lceil x - W_{n,s} \neq \emptyset$. We say also that an axiom $\langle\langle x, 2y+1\rangle, F\rangle$ is *axiom of type II at the stage $s+1$*, if $x < s, y < s$, $W_{n,s}\lceil x \subseteq \Theta_{n,s}(F)$ and $x \in F$.

The e-operator $\Theta_{n,s+1}$ is produced from $\Theta_{n,s}$ by adding all axioms of types I and II at the stage $s+1$.

Set $\Theta_n = \cup_s \Theta_{n,s}$ for each $n \in \omega$. Fix $n \in \omega$. It is easy to see, that if an axiom $\langle\langle x, y\rangle, F\rangle$ is the axiom of type I at a stage s, then $\langle\langle x, y'\rangle, F\rangle$ is also the axiom of type I at the stage s for each $y' < y$. Analogously, if an axiom $\langle\langle x, 2y+1\rangle, F\rangle$ is the axiom of type II at a stage s, then $\langle\langle x, 2y'+1\rangle, F\rangle$ is also the axiom of type II at the stage s for each $y' < y$. It follows that for a set X and an integer x we have one of the following possibilities:
Case 1. $(\Theta_n(X))^{[x]} = \{\langle x, y\rangle : y \in \omega\}$;
Case 2. $(\Theta_n(X))^{[x]} = \{\langle x, 2y+1\rangle : y \in \omega\} \cup \{\langle x, y\rangle : y < t\}$ for some t;
Case 3. $(\Theta_n(X))^{[x]} = \{\langle x, 2y+1\rangle : y < s\} \cup \{\langle x, y\rangle : y < t\}$ for some s and t.

Thus, for each $x \in \omega$ and $X \subseteq \omega$ the set $(\Theta_n(X))^{[x]}$ is computable.

Note that Case 1 holds iff there are infinitely many y such that for some $F \subseteq X$ the axiom $\langle\langle x, y\rangle, F\rangle$ is the axiom of type I at some stage, and the last condition is equivalent to $\Theta_n(X)\lceil x - W_n \neq \emptyset$ and $W_n\lceil x \subseteq \Theta_n(X)$.

If Case 1 does not hold then Case 2 holds iff there are infinitely many y such that for some $F \subseteq X$ the axiom $\langle\langle x, 2y+1\rangle, F\rangle$ is the axiom of type II at some stage, and the last is equivalent to $x \in X$ and $W_n\lceil x \subseteq \Theta_n(X)$.

In the Case 3 we also note that if $x \notin X$ then there are no axioms of type II $\langle\langle x, 2y+1\rangle, F\rangle$ such that $F \subseteq X$, and hence $(\Theta_n(X))^{[x]} = \{\langle x, y\rangle : y < t\}$ for some t.

Fix now an arbitrary set X which is not Δ^0_2. Suppose that $\Theta_n(X) = W_n$. Then for each $x \in \omega$ the Case 2 holds if $x \in X$, otherwise (if $x \notin X$) the Case 3 holds, i.e., $(\Theta_n(X))^{[x]} = \{\langle x, y\rangle : y < t\}$ for some t. Thus, $x \in X$ iff $\langle x, (\mu y)[\langle x, y\rangle \notin W_n] + 1\rangle \in W_n$ for each $x \in \omega$, so that $X \in \Delta^0_2$. A contradiction.

Thus, $\Theta_n(X) \neq W_n$. To prove a computability of $\Theta_n(X)$, suppose at first that $W_n \subseteq \Theta_n(X)$. Then $x_0 \in \Theta_n(X) - W_n$ for some x_0, and hence for all $x > x_0$ we have the Case 1, i.e., $(\Theta_n(X))^{[>x_0]} = \omega^{[>x_0]}$ is computable.

Suppose now that $W_n - \Theta_n(X) \neq \emptyset$. Fix $t \in \omega$ such that $W_{n,t}\lceil t - \Theta_n(X) \neq \emptyset$. Then for each $x > t$ and $s > t$ there are no axioms $\langle\langle x, y\rangle, F\rangle$ of types I or II at the stage s, such that $F \subseteq X$. Also, for all axioms $\langle\langle x, y\rangle, F\rangle$ of one of types I,II at the stages $s \leq t$ we have necessarily $x < s$. Therefore, $(\Theta_n(X))^{[>t]} = \emptyset$.

Thus, in any case there is $x_0 \in \omega$ such that $(\Theta_n(X))^{[>x_0]}$ is computable, and hence $\Theta_n(X)$ is computable, since for each x the set $(\Theta_n(X))^{[x]}$ is also computable. □

Theorem 4. *There is a computable sequence of e-operators Ψ_k, $k \in \omega$, such that $\mathcal{D} = \{\Psi_k(X) : k \in \omega\}$ for each $X \notin \Delta^0_2$.*

Proof. Let Θ_n, $n \in \omega$, be the computable sequence of e-operators from Proposition 5.

By the s-m-n Theorem there is a computable function $h(n, t)$ such that for all n, $t \in \omega$

$$W_{h(n,t)} = \{z \in \omega : z + t \in W_n\}.$$

For each m, n, $x \in \omega$ let $\Psi_{\langle m,n,x\rangle}$ be the e-operator such that for each $X \subseteq \omega$

- $\Psi_{\langle m,n,x\rangle}(X) = \emptyset$, if $x \notin W_m \cup W_{g(n)}$;
- $\Psi_{\langle m,n,x\rangle}(X) = \langle n, W_m\rangle$, if $x \in (W_m - W_{g(n)}) \cup (x \in W_{g(n)} - W_m)$;
- $\Psi_{\langle m,n,x\rangle}(X) = \langle n, W_m\lceil t \cup (t + \Theta_{h(g(n),t)}(X))\rangle$ otherwise, where $t \in \omega$ is the least integer such that $x \in W_{m,t} \cap W_{g(n),t}$.

Suppose that $\langle n, W_m\rangle \in \mathcal{D}$. By the definition of $\mathcal{D}$ there is $x \in (W_{g(n)} - W_m) \cup (W_m - W_{g(n)})$. Then $\Psi_{\langle m,n,x\rangle}(X) = \langle n, W_m\rangle$ for all $X \subseteq \omega$.

Fix arbitrary $X \notin \Delta^0_2$. Since $\emptyset \in \mathcal{D}$, it remains to prove that $\Psi_{\langle m,n,x\rangle}(X) \in \mathcal{D}$ for all m, n, $x \in \omega$ such that $x \in W_m \cup W_{g(n)}$. Then either $x \in (W_m - W_{g(n)}) \cup (W_{g(n)} - W_m)$, or $x \in W_m \cap W_{g(n)}$.

In the first case we have $W_m \neq W_{g(n)}$ and $\Psi_{\langle m,n,x\rangle}(X) = \langle n, W_m\rangle$. Then $\Psi_{\langle m,n,x\rangle}(X) \in \mathcal{D}$.

In the last case we have

$$\Psi_{\langle m,n,x\rangle}(X) = \langle n, W_m\lceil t \cup (t + \Theta_{h(g(n),t)}(X))\rangle$$

for some $t \in \omega$. Suppose that

$$W_m\lceil t \cup (t + \Theta_{h(g(n),t)}(X)) = W_{g(n)}.$$

Then we have $\Theta_{h(g(n),t)}(X) = W_{h(g(n),t)}$ by the choice of h, and this contradicts Proposition 5. Thus,

$$W_m\lceil t \cup (t + \Theta_{h(g(n),t)}(X)) \neq W_{g(n)},$$

and hence $\Psi_{\langle m,n,x\rangle}(X) \in \mathcal{D}$, since $\Theta_{h(g(n),t)}(X)$ is computable by Proposition 5.

Thus, $\mathcal{D} = \{\Psi_k(X) : k \in \omega\}$. □

Corollary 1. *There is a computable sequence of e-operators $\widehat{\Psi}_k$, $k \in \omega$, such that $\mathcal{D} = \{\widehat{\Psi}_k(X) : k \in \omega\}$ for each X such that $(\deg_e(X))' > \mathbf{0}'_e$.*

Proof. Since $(\deg_e(X))' = \mathbf{0}'_e \iff K(X) \in \Delta^0_2$ we can set $\widehat{\Psi}_k = \Psi_k \circ K$. □

Corollary 2. *There are families of c.e. sets $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $\mathcal{S}_1 \leq \mathcal{S}_2$ and $\mathcal{S}_1 \not\leq_\sigma \mathcal{S}_2$.*

Proof. Let S_1 be the family $\mathcal{D}$ from Definition 3. Apply Theorem 2 with $\mathcal{S} = \mathcal{S}_1$ and $X_1 = \overline{K}$ to obtain a set $A \notin \Sigma^0_2$ such that $\mathcal{S}_1 \not\leq_\sigma \mathcal{F}(A) = \{\{x\} : x \in A\} \cup \{\omega\}$.

On the another hand, $\mathcal{F}(A)$ can not have a partial enumeration $R \in \Delta^0_2$, since otherwise $A \in \Sigma^0_2$. Thus, $\mathcal{S}_1 \leq \mathcal{F}(A)$ by Theorem 4. Set $\mathcal{S}_2 = \mathcal{F}(A)$. □

In fact, Theorems 2 and 4 give something more: the predicate $\leq$ is not isomorphic to the predicate $\leq^0_\sigma$ on the class of countable families and on the class of families of c.e. sets (since there are only countably many families with Δ^0_2 enumerations).

Note that Theorem 2 with $\mathcal{S} = \mathcal{D}$ and $X_1 = \overline{K}$ produces the $\mathbf{0}''$-computable set A, so that the family $\mathcal{S}_2$ in the proof of Corollary 2 has Σ^0_2 partial enumerations. For the set A consider the family $\mathcal{G}(A) = \{\{x\} : x \in A\} \cup \{\{x, y\} : x \neq y\}$. Then $\mathcal{G}(A) \equiv^0_\sigma \mathcal{F}(A)$ and $\mathcal{G}(A)$ has a Δ^0_2 partial enumeration. Thus, the families $\mathcal{S}_1$ and $\mathcal{S}_2$ in the Corollary 2 can be chosen with Δ^0_2 partial enumerations.

The paper finishes by the following open problems.

Problem 2. Give a direct proof for Corollary 2.

It is easy to check that for any family $\mathcal{S} \leq_\sigma ComF$ of c.e. sets the index set $\{x \in \omega : W_x \in \mathcal{S}\}$ is Σ^0_3. The previous problem will be solved if such property does not hold for Dyment reducibility.

Problem 3. Let $\mathcal{S}$ contains only c.e. sets and $S \leq ComF$. Must the index set $\{x \in \omega : W_x \in \mathcal{S}\}$ be Σ^0_3?

In particular, let $CoInf$ be the family of all c.e. sets with infinite complements, its index set is Π^0_3-complete so that $CoInf \not\leq_\sigma ComF$. It is possible to check that $P_M(\mathcal{A}_{CoInf}) \leq P_M(\mathcal{A}_{ComF})$, but for the Dyment reducibility it is not so clear.

Problem 4. Is $CoInf \leq ComF$?

Note that $CoInf$ can not be Dyment equivalent to $ComF$. Moreover, for degree $\mathbf{a}$ the family $CoInf$ has $\mathbf{a}$-c.e. partial enumeration iff $\mathbf{0}''' \leq \mathbf{a}''$, so that $InfCE \not\leq CoInfCE$.

The family $\mathcal{S}_2$ from the Corollary 2 has no partial enumerations $R \in \Delta^0_2$. The family $\mathcal{G}(A)$ can not have a partial enumeration R such that $(\deg_e(R))' = \mathbf{0}'_e$, if $A \notin \Sigma^0_2$. By this reason the next problem seems interesting.

Problem 5. Does $\mathcal{S}_1 \leq \mathcal{S}_2$ imply $\mathcal{S}_1 \leq^0_\sigma \mathcal{S}_2$ if the family $\mathcal{S}_2$ has a partial enumeration R such that $(\deg_e(R))' = \mathbf{0}'_e$?

Theorem 4 and even Corollary 1 can not help to build a counterexample to this problem.

The last problem is connected to Theorem 3. Does the analogue of this theorem hold for the Dyment reducibility?

Problem 6. Is there a noncomputable countable structure $\mathcal{B}$ such that $P_D(\mathcal{B}) \leq Q$ for any partial mass problem Q without partial computable elements?

A positive answer would essentially improve Theorem 3. Corollary 1 gives only a weak version for such result: we have $P_D(\mathcal{A}_\mathcal{D}) \leq Q$ for any partial mass problem Q without elements of low e-degrees.

References

1. Calvert, W., Cummins, D., Knight, J. F., Miller S.: Comparing classes of finite structures. Algebra and Logic, **43** (2004), 374–392.
2. Cooper, S. B.: Enumeration reducibility, nondeterministic computations and relative computability of partial functions. In K. Ambos-Spies, G. Müller, and G. E. Sacks, editors, *Recursion Theory Week, Oberwolfach 1989*, volume 1432 of *Lecture Notes in Mathematics*, pages 57–110, Heidelberg, 1990. Springer–Verlag.
3. Dyment, E.Z.: Certain properties of the Medevedev lattice, Mat. Sborn. **101 (143)** (1976), 360-379.
4. Jockusch, C. G., Jr.: Degrees in which the recursive sets are uniformly recursive. Canad. J. Math., **24** (1972), 1092–1099.
5. Kalimullin, I.Sh., Puzarenko V.G.: On a reducibility on the families. In preparation.
6. Medvedev, Yu.T.: Degrees of difficulty of the mass problems. Dokl. Akad. Nauk SSSR, , **104** (1955), 501–504.
7. Rogers, H. Jr.: *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, New York, 1967.
8. Wehner, S.: Enumerations, counatble structures and Turing degrees. Proceedings of the American Mathematical Society, July 1998.

Computing the Recursive Truth Predicate on Ordinal Register Machines

Peter Koepke[1] and Ryan Siders[2]

[1] University of Bonn, Mathematisches Institut,
Beringstraße 1, D 53115 Bonn, Germany
Koepke@Math.Uni-Bonn.de
[2] University of Helsinki, Department of Mathematics and Statistics
Gustaf Hällströminkatu 2b, Helsinki 00014, Finland
bissell@mappi.helsinki.fi
(corresponding author)

Abstract. We prove that any constructible set is computable from ordinal parameters by a wellfounded program on an infinite-time ordinal-storing register machine.
This brings us closer to "minimal" computation of set theoretic constructibility. To that end, we describe data types and well-founded programming to consider what can be cut from the machine or programming languge.
These machines were designed to define and study run-time complexity for hypercomputation. We solve one complexity problem: deciding the recursive truth predicate is ordinal-exponential time on a register machine, and ordinal-polynomial time on a Turing machine.

1 Register Machines

Ordinal Register Machines increment and erase a finite number of registers containing ordinals; the number of necessary registers can eventually be reduced to four. They exemplify abstract model-theoretic computation. The ordinals they store can refer to the timesteps of a hypercomputation.

We motivate these machines by first considering 1. finite-time register machines storing finite register values, and 2. finite-time register machines ordinal register machines storing ordinal register values with an oracle for ordinal arithmetic.

Definition 1. *([10]) A register machine stores natural numbers and runs for finite time. A register machine program is written using the three instructions: 1.* `Zero`(x)*, which sets x to zero; 2. $x++$ which increments x; and 3.* `If` $(x = y)$ `goto` i `else` j*, which switches the flow of control, depending on whether two registers are equal or not. Here, i and j name instructions in the program. When the program is interpreted in a machine, i and j label states of the machine.*

Definition 2. *On a register machine, a "*`For`*-program" uses* `goto` *switches only to define the commands: 3a.* `if` $(x = y)$ (`instructions`)*; 3b.* `for` $(x = y; x < z; x++)$ (`instructions`)*, where x is never set to zero within the loop.*

A "`While`*-program" has the extra instruction 4.* `Decrement`(x)*; but only uses* `goto` *switches to define: 3a.* `if` $(x = 0)$ (`instructions`)*; 3b.* `while` $(x > 0; x - -)$ (`instructions`)*, where x is never incremented within the loop.*

We call finite register machine `For-` *and* `While`*-programs "well-structured."*

Definition 3. `for` x `from` 0 `to` y *means* `Zero` (x)*;* `for` $(x = x; x < y; x++)$. `if`$(a = b)$ `(instructions)` *means* `Zero`(x); `for`$(c = a; c < b; c++)(x++)$; `for`$(c = b; c < a; c++)(x++)$; `if`$(x = 0)$`(instructions)`.

Theorem 1. *([9] p. 205)* 5*-variable* `While`*-programs simulate Turing machines.*

Theorem 2. *([9] pp. 255-8)* `While`*-programs using* 2 *variables can simulate all* `While`*-programs.* `FOR`*-programs using* 3 *variables can simulate all* `While`*-programs.*

In this paper, we have used the word "model" in the sense of "a model of computation," and in the sense of "model theory." We try to never say a machine A "models" the behaviour of machine B, but rather that it "simulates" B. Simulating one machine on another, and building a model of one theory inside a model of another are such similar concepts, that the idea of computing over abstract models seems ripe for self-reference.

Definition 4. *(see the survey [11]) For M any model, an M-register machine stores (has some variables x_i refering to) elements a_i of M, and runs for finite time. Its programming language assigns variables to elements of M with the following two commands:*

1. assign x_i to a_j (duplicate);

2. assign x_i to a_j; assign x_j to a_i; (swap);

The constants, functions, and relations in M are assumed to be computable, so the programming language includes the following three commands

3. $a_i = f^M(a_1...a_n)$;

4. $a_i = c^M$;

5. If $R^M(a_1...a_n)$, then i, else j.

The storage of elements of M in the register machine is like the assignment of variables to elements of M in a finite-variable pebble game. The M-register machine can set a register to any constant in M, set a register a_1 to $f(a_1...a_n)$, for any n-ary function f, and switch its state, depending on whether $M \models R(a_1...a_n)$ or not.

Analyzing the flow of control in a computer program, without looking at what is in the registers, was pursued in [3], in which it is proved that "any Goto program is equivalent to a While program." Such analyses are fruitful too in "abstract computability theory," surveyed by [11].

1.1 Pairing and Stacking

Research on abstract-model recursion theory since [2] suggests that the ordinary theorems of recursion theory will lift, if equality is decidable, pairing is

computable, and the domain is finitely generated. Fortunately, ordinals have an efficient notion of pairing.

Definition 5. *Let Γ be the pairing function taking (a, b) to the wellorder of pairs $(c, d) <^2 (a, b)$, where $(c, d) <^2 (a, b)$ iff $\max(c, d) < \max(a, b)$, or $\max(c, d) = \max(a, b)$ and $c < a$, or $\max(c, d) = \max(a, b)$ and $c = a$ and $d < b$.*

Lemma 1. *$\Gamma(a, a) \cong a$ iff a is a $\times$-closed ordinal. (so Γ is "efficient.")*

Remark 1. ([11] pp. 485,486) show that recursion theory lifts to M-register machines if M has counting and stacking. Our main result depends heavily on using the following binary stack to prove that ordinals have stacking, and that the stack is robust, as the ordinal register machine runs over a limit-ordinal time.

Definition 6. *A "binary stack" codes a finite, monotonically decreasing sequence $(\beta_i : i < n)$ as $\sum_i 2^{\beta_i+1} = 2^{\beta_0+1} + 2^{\beta_1+1} + ... + 2^{\beta_{n-1}+1}$, where 2^α is ordinal exponentiation.*

The reason for $+1$ in the exponent is that when β_{i+1} limits up to β_i, and β_i is a limit ordinal and false (it codes ϕ_{β_i}, a falsehood, so for T the Recursive Truth Predicate of definition 13, $T(\beta) =$"false") then we need to be able to check whether β_i being false witnesses the truth of β_{i-1}, before incrementing β_i. The $+1$ in the exponent allows us to identify a limit ordinal appearing as a term in the $\sum$ which is the stack as a limit of earlier stack elements and not a stack element itself.

Definition 7. *Let "Seq" be the set of finite, descending sequences of ordinals, all less than $\$_-$, ordered by their first difference.*

Lemma 2. *$(\beta_i) \mapsto \sum 2^{\beta_i}$ is an isomorphism between Seq and $2^{\$_-}$.*

Lemma 3. *If the supremum of T is α, then the supremum of $\{2^\beta : \beta \in T\}$ is 2^α.*

We will define Pop, Push, and IsEmpty for this stack in section 3, after we have learned the natural data types and definitions for ordinal computers and are comfortable writing longer routines.

1.2 A model of infinitely-long computation

Definition 8. *An Infinite-time Ordinal-storing Register Machine is a register machine storing ordinal values and runing for ordinal time, with a programming language including the three instructions: 1. Zero(x); 2. $x++$; 3. if $x = y$ goto i else j; in which the registers' values at limit times obeys the following three rules:*

R1. If the command "Zero(x)" is called at each time $t \in T$, then x is 0 at time $\sup T$, too.

R2. At limit times, command passes to the liminf of the commands which have been active cofinally often.

R3. Until it is zero'd, a register's value increases continuously.

We'll also call this model of computation an Ordinal Computer, or *OC*.

Remark 2. The study of continuous computation or abstract register machine computation over an infinite model motivates the study of hypercomputation.

Our rule 2. is the same as that in [6]. Other definitions of limit time use the lim-sup rule ([4]) or require wellfounded programs ([1], see definition 9) so at a limit time, the machine only decides whether to keep looping (see remark 3 below). These approaches are equivalent.

Definition 9. *An OC program is "well-structured" if* `goto` *switches are only used to model the following two commands: 3a.* `if`$(x_i = 0)$ *(instructions); 3b.* `for`$(x = y; x < z; x++)$ *(instructions); where neither* `Zero`(x) *nor* `Zero`(z) *is among the instructions in the loop. A* `for` *loop tests* $x < z$*, then executes.*

Remark 3. At any limit time λ during the run of a well-structued program, there is a unique instruction `for`$(x = y; x < z; x{+}{+})$`(loop)` for which we have checked whether $x \geq z$ at times T a cub subset of λ, but $x < z$ was true each time. Rule 2. of definition 8 requies that at a limit time, control returns to the start of the loop; the start of a `for` loop is its test, so equivalently, Rule 2'. At a limit time λ, repeat the outermost loop which has been active cofinally often. That is, after checking $x < z$ infinitely often and finding it always true, check again.

Rule 2' makes a very reasonable, but nontrivial, requirement of the machine's state at a limit time: that if the program says to loop states until $x \geq z$, then the machine does not stop looping states simply because it reaches a limit time, but only when $x \geq z$.

Theorem 3. *For well-structured computations, Rule 1 can be simplified, so as to require nothing of the machine's limiting behaviour, but only require something syntactical, about how well-structured programs are formulated: Rule 1': In a well-founded program, immediately before a switch* `if`$(x = 0)$ *is called, x was changed one last time (and not an infinite, unbounded set of times, before the switch is called).*

Then any register, at any time, is defined in terms of other variables, each of which was defined one last time beforehand, such that there is a finite tree of variables and times, on whose definition x's value at the time of the switch depends, the leaves of which are variables which are never zero'd, during the computation. So *OC* programs can be written so that switches well-foundedly depend on monotonic variables.

1.3 Discovering data types

Lemma 4. *1. If all registers are set to* 0 *repeatedly (after any time t, each register is again set to* 0 *at some time $t' > t$), then there is a time at which all registers are simultaneously zero. 2. Any active loop index is equal to the clock at all times ϵ_α.*

As a result, we find that there are fundamentally different natural data types.

Remark 4. In a well-structured computation we can identify three types of registers: 1. registers which are zero'd infinitely often, 2. registers which are never zero'd, and so are cub-often equal to the clock, and 3. registers which are neither incremented nor zero'd.

Registers that are zero'd infinitely often are the indices to short loops and "active memory." Registers that are frequently close to the clock are really just marking time (and storing type 1 variables between each other). Registers that do not change during the computation are the parameters which the computation (seen as a subroutine) was given from the outside.

Definition 10. *We call type 1. variables ORD, type 2. variables MON, and type 3 variables STO, for "ordinal," "monotone," and "static storage." In long programs, all variables will have for scope only the subroutine they are defined in, and we will declare their type before using them, for clarity.*

Lemma 5. *All well-structured ordinal computer programs halt.*

Proof: by induction on the depth of `For` loops: the maximum value ρ of the registers is a normal function in time, which has arbitrarily large fixed points. At exp-closed ordinal times, the loop-index is time, as well. At these times, the loop index and bound are equal, and the loop terminates. □

But other programs need not halt:

Example 1. A. `For`-programs halt when they reach a fixed point. B. some non-well-structured programs do not halt at all

A. `for(`$b = a + 1$`;`$a < b$`;`a`++) (`$b++$`)` halts at the least limit ordinal $> b$.

B. $b = a$`; 1. if (`$b = a$`) (`$b++$`);` $b++$`;` $a++$`; if (`$b \neq a$`) (goto 1);` never halts since at limit times line-control passes to its lim-inf (def'n 8, R2).

Because of the intuitive variable types found in lemma 4 and remark 4 and the simple program-flow described in lemma 5 and remark 3, we will restrict our attention to well-structured ordinal computer programs.

1.4 Reflection of few registers

How many registers are needed to simulate an infinitary Turing machine on an ordinal computer? How many registers are clearly trivial? Four registers are universal, and three registers are fairly trivial. In the next subsection we will prove the reflection of up to three registers. Note that the program computing the universal truth predicate uses 6 variables outside any subroutine, and the longest subroutine, `Pop`, uses 6 variables. So twelve registers are enough to compute any element of the constructible universe, and hence any ordinal computer with more than twelve registers. A reduction to four registers is simply technical, using oscilating stacks as in 2, repeating all finite intervals as many times as there are stack elements, and using the last variable to store a single variable,

just as it appears on the stack, and store it, after the stack limits and is erased, very high on the stack, where it won't be erased by the varying and limiting of values lower on the stack. However, we will not prove rigorously in this paper that four register suffice.

Definition 11. *Let OC^n be the set of n-register well-structured ordinal computer programs (obeying rules 1 and 2'). Say $\rho : Ord^n \to \alpha^n$ reflects OC^n if for each P in OC^n, the function f_P which takes the inputs to P to the output of P, commutes with ρ. Let L_n be the vocabulary with a function for each n-register program: $L_n = \{Ord, <, =\} \cup \{f_P : P \in OC^n\}$, and let $FO^k(L)$ be the first order formulas in the language L, to quantifier depth k.*

Definition 12. *Let ρ_0 be the function $\rho_0(\alpha) = \alpha \mod \omega$.*

Let ρ_1 be the identity below ω, and be $\omega + \rho_0$ above ω.

Let ρ_2 be the identity below $\omega \times 2$, and be $\omega \times 2 + \rho_0$ above $\omega \times 2$.

Let $\rho_3(\alpha) = \alpha \mod \omega^\omega$.

Let ρ_4 be the identity below ω^ω, and be $\omega^\omega + \rho_3$ above ω^ω.

Let $\rho_5(\alpha, \beta)$ be the pair $(\rho_4(\alpha), \rho_4(\alpha) + \rho_4(\beta - \alpha))$ if $\alpha \leq \beta$ and be undefined if $\alpha > \beta$.

Lemma 6. *$\rho_1 : Ord \to \omega \times 2$ reflects OC^1, is the minimal reflection preserving $FO^1(L_1)$, and preserves even $FO^2(L_1)$. ρ_2 preserves $FO^3(L_1)$.*

Corollary 1. *$FO^3(L_1)$ is much less expressive than the 4-quantifier theory of linear order, $FO^4(<)$. Indeed, $FO^4(<)$ can define every predicate definable in $FO^3(L_1)$.*

Lemma 7. *$\rho_5 : Ord^2 \to (\omega^\omega \times 3)^2$ reflects OC^2, is minimal such that it preserves $FO^2(L_2)$, and preserves $FO(L_2)$.*

Remark 5. Suppose that the rule 3b in definition 9 were relaxed, and only the index were not allowed to be zero'd. Then $y{+}{+}$; `for(`$x = 0; x < y; x{+}{+}$`)` `(Zero(`y`);` `for(`$y = y; y < x; y{+}{+})(x{+}{+}); y{+}{+}$`)` halts with register values ω^ω).

Corollary 2. *$FO(L_2)$ computes $x \mapsto x \times \omega$ and $x \mapsto n \times x$ (for finite n), but it is weaker than $FO(Ord, <, c_0, c_1)$, where c_0 and c_1 are constants naming any two $\times$-closed ordinals; this is much weaker than $FO(Ord, +)$. So an ORM with two variables can not compute the $+$ of two input values.*

By theorem 2, OC^3 can simulate a finite turing machine. But OC^3 still reflects into a small ordinal, and as a result, we find that stacking ordinals requires more registers than stacking finite numbers. Moving an infinite ordinal onto a stack by incrementing the stack once every few time-steps requires infinite time. So the stacking operation with which OC^n simulates a finite-time $(\omega_1, +, \times, a \to \omega^a)$-register machine (an abstract register machine as in definition 4) must limit continuously without losing any information.

Lemma 8. *Ord^3 reflects below $\epsilon_{\omega \times 4}$, and not lower.*

2 Recursive Truth Predicate

Our main theorem is that infinite ordinal register machines can decide all sets of ordinals which are elements of the constructible hierarchy, L, i.e.: For every set of ordinals S which exists in L, there is an ordinal computer program P and a single ordinal input, the Γ-stack $\Gamma(...\Gamma(\alpha_0, \alpha_1)..., \alpha_n)$ of $(\alpha_0...\alpha_{n-1})$, which program decides S.

Conversely, the definition of the program P exists within L, so OC computation reflects into L. That is, anything OC-computed from finite ordinal parameters $a_0...a_n \in L$ is thereby constructed in L.

Theorem 4. *(analogous to Theorem 5 of [8]) A set S of ordinals is ordinal computable from some finite set of ordinal parameters if and only if it is an element of the constructible universe L.*

We prove the theorem, that everything in L can be computed by an ordinal computer (from some ordinal parameters), by computing the "recursive truth predicate" described in [8].

The recursive truth predicate is a recursive characteristic function on the ordinals, coding all constructible sets of ordinals. It is defined as

Definition 13. *Let T be the recursive truth predicate, defined by: $T(\alpha) = True$ if and only if $(\alpha, <, \Gamma, T \restriction \alpha) \models \phi_\alpha$, where Γ is the ordinal pairing function in definition 5, where the sentence ϕ_α is coded by α, has a finite number of ordinal parameters.*

Definition 14. *(F from H): $F(\alpha) = True \iff \exists \beta < \alpha \; H(\alpha, \beta, F(\beta)) = True$:*

```
for β from 0 to α (
    if (F(β) = False and H(α, β, False) = True) (return True);
    if (F(β) = True and H(α, β, True) = True) (return True);
);
Return False
```

That program is written in a language which allows a subroutine to call itself. First, we show that from this recursive routine, set-theoretical constructibility can be carried out.

Theorem 5. *If an OC can simulate the recursive routine in definition 14, then theorem 4 holds.*

Now we simulate the program in definition 14 using a wellordered stack of formulas on an OC.

3 Stack, Pop, Push, IsEmpty

Definition 15. `Pop`*, taking two parameters* `(Stack, Threshold)` *and referencing the global variable* `$_` *in which the program has received as its input a formula*

whose truth is to be witnessed, (and which serves as an upper bound to all formulas and all searches) is the following routine:

```
MON SmallStack := 0;
MON TempStack := 0;
for ϵ from 0 to $_ (
   for α from 0 to Stack (
      if (α + 2^ϵ+SmallStack = Stack) (
         if (ϵ > Threshold) (return ϵ);
         for TempStack to Smallstack ();
         for Smallstack to 2^ϵ ();
         for κ from 0 to TempStack (Smallstack++)
      )
   )
)
```

Pop doesn't really change the stack. It just reads the next element, past a certain threshold.

Lemma 9. Pop *reads least element* 2^ϵ *of the* Stack*, such that* ϵ *is at least as large as the parameter* Threshold*.*

Definition 16. Push*, a program taking two parameters* (Stack, β)*, is the following routine:*

```
Stack ++;
for ι from 0 to 2^(β+1) (
   if (¬ (2^(β+1) divides Stack)) (Stack ++)
)
```

where

β divides α *is the routine:*

```
MON γ = 0
for ι from 0 to α (
  for κ from 0 to β (γ + +)
  if (γ = α) (Return Yes);
  if (γ > α) (Return No)
);
Return No
```

Push(β onto Stack) erases all stack values less than β.

Lemma 10. Push(β onto Stack) *increases the Stack to the next full multiple of* $2^{\beta+1}$*.*

Definition 17. IsEmpty*, taking the single input* Stack*, is the routine:*

```
ORD α = Pop(Stack,0);
if (2^α = Stack) (return "True")
else (return "False")
```

IsEmpty(Stack) returns the value "True" when the stack is a singleton, $2^{\$}$-, i.e., the initial value, the truth of which we would like to determine.

Definition 18. β `is the largest element on the stack` *is the program*

```
for ι < 2^β (if (2^β + ι = Stack) (Return Yes));
Return No
```

Clearly, this halts before ι exhausts β iff β is indeed the largest stack value. On the other hand, $2^\beta + \iota =$ `Stack` never holds if β is larger than the largest stack value, nor if β is less than the largest stack value.

3.1 The Recursive Truth Predicate OC Program

Theorem 6. *The recursive truth predicate F defined in 14 is equivalent to the following program:*

```
Determining the Truth Value of ($_):
ORD α = 0;
ORD β = 0;
ORD ν = 0;
MON Stack = 0;
ORD TruthValue = Unknown;
Push($_ onto Stack);
for ι from 0 to 2^$- (
    β = Pop(Stack,0);
    If (β is a limit) (TruthValue = Unknown);
    If IsEmpty(Stack) (Stack ++); # That is, "if Stack = 2^β."
    if TruthValue is Unknown (
       if β is a successor ordinal (Stack ++;β = 0);
       α = Pop(Stack,β + 1);
       if β is not a successor ordinal and α = β + 1 (
          β = α;TruthValue = False;
       );
       if β is not a successor ordinal and α ≠ β + 1 (
          Push(β onto Stack);
       )
    );
    while TruthValue is Known (
       if β is the largest element on the stack (return TruthValue);
       α = Pop(Stack,β + 1);
       Let ν = H(α − 1, β − 1, TruthValue);
       if (ν = True)(β = α;TruthValue= ν);
       if (ν = False and α = β + 1)(β = α;TruthValue= ν);
       if (ν = False and α ≠ β + 1)(
          TruthValue=Unknown;
          Push(β onto Stack)
       )
    )
)
```

Definition 19. *Call $(\alpha_i : i \leq k)$ a witnessing sequence if, for each $i < k - 1$, $\alpha_i - 1$ is true and $H(\alpha_i - 1, \alpha_{i+1} - 1, F(\alpha_{i+1} - 1)) =$ True, and α_{k-1} is false and $\alpha_k = \alpha_{k-1} - 1$. Call $WIT(\gamma)$ the least witnessing sequence $(\alpha_i : i < k)$ such*

that $\gamma = \alpha_0$ *and* $\alpha_{k-1} - 1$ *is false and a limit. Call* $W(\gamma) = \sum_{\alpha \in W(\gamma)} 2^\alpha$ *the witnessing series.*

Lemma 11. *a) If* $Stack = \sum_{i<n} 2^{\gamma_i+1} + W(\gamma_n)$*, where* γ_n *does not witness* γ_{n-1} *(i.e.: it is not the case that* γ_n *is true and* $H(\gamma_{n-1}, \gamma_n, 1) = 1$*, nor is it the case that* γ_n *is the predecessor of* γ_{n-1} *and* γ_{n-1} *is false), and* $W(\gamma_n)$ *is a series with* m *terms, then in the first part of the loop TruthValue becomes known and after* m *iterations of the while loop, the program passes command to* `Push(`γ_{n-1} `onto Stack)`*.*

b) If $Stack = W(\gamma_n)$*, a series with* m *terms, then in the first part of the loop TruthValue becomes known and after* m *iterations of the while loop, the program halts, returning the truth value of* $\gamma_0 - 1$*.*

Now we prove theorem 6 by induction on the following:

Lemma 12. *Intention Lemma:*

a. If Stack is $2^{\$\_+1} + \sigma + 2^\gamma$*, where* $2^\gamma \times 2$ *divides* σ*,* γ *is a successor, and TruthValue is Unknown, then the Stack will reach* $2^{\$\_+1} + \sigma + W(\gamma)$*, when* $\iota \leq (\sigma \times 1/2) + 2^{\gamma-1}$*, with* $\beta = \gamma$ *and TruthValue known.*

b. If Stack is $2^{\$\_+1} + \sigma + 2^\gamma$*, where* $2^\gamma \times 2$ *divides* σ*, and* $\gamma - 1$ *is a false limit then the stack will be* $\geq 2^{\$\_+1} + \sigma + 2^\gamma + 2^\delta$*, for each successor* $\delta < \gamma - 1$*, at some time* $\iota \leq (\sigma \times 1/2) + 2^{\gamma-1} + 2^{\delta-1}$*.*

c. if the Stack is $2^{\$\_+1} = 2^\gamma$*, and TruthValue is Unknown, then the Stack will reach* $W(\gamma)$*, when* $\iota \leq 2^{\gamma-1}$*, with* $\beta = \gamma$ *and TruthValue known, i.e., P halts on input* $\$\_$ *after at most* $2^{\$\_}$ *loops through the main loop, and returns the value* $F(\$\_)$*.*

References

1. R. Bissell-Siders, *Ordinal computers.* math.LO/9804076 at arXiv.org, 1998.
2. H. Friedman, *Algorithmic procedures, generalized Turing algorithms, and elementary recursion theory.* Logic Colloquium '69 (Proc. Summer School and Colloq., Manchester, 1969), pp. 361–389. North-Holland, Amsterdam, 1971.
3. G. Jacopini and C. Böhm, *Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules.* Comm ACM, 9,5 May 1966.
4. J. Hamkins and A. Lewis, *Infinite Time Turing Machines.* J. Symbolic Logic, 65(2): 567-604, 2000.
5. P. Koepke, *Infinite Time Register Machines.* Submitted to Computing In Europe 2006; this volume. 11(3): 377-397, 2005.
6. P. Koepke, *Turing Computations on Ordinals.* Bulletin of Symbolic Logic, 11(3): 377-397, 2005.
7. P. Koepke and M. Koerwien, *The Theory of Sets of Ordinals.* math.LO/0502265 at the e-print archive arXiv.org, 2005
8. P. Koepke and M. Koerwien, *Computing a Model of Set Theory.* CIE 2005.
9. M. Minsky, *Computation: Finite and Infinite Machines.* Prentice-Hall, 1967.
10. J. Shepherdson and H. Sturgis, *Computability of recursive functions*, J. Assoc. Comput. Mach. 10 217–255, 1963.
11. J. Tucker and J. Zucker, *Computable functions and semicomputable sets on many sorted algebras*, in S. Abramsky, D. Gabbay and T Maibaum (eds.) *Handbook of Logic for Computer Science*, Volume V, Oxford University Press, 317-523.

Logic Programs with Uncertainty: Neural Computation and Automated Reasoning

Ekaterina Komendantskaya[1] and Anthony Seda[2]

[1] Department of Mathematics, University College Cork, Cork, Ireland
e.komendantskaya@mars.ucc.ie
[2] Department of Mathematics, University College Cork, Cork, Ireland
a.seda@ucc.ie ⋆ ⋆⋆

Abstract. Bilattice-based annotated logic programs (BAPs) form a very general class of programs which can handle uncertainty and conflicting information. We use BAPs to integrate two alternative paradigms of computation: specifically, we build learning artificial neural networks which can model iterations of the semantic operator associated with each BAP and introduce sound and complete SLD-resolution for this class of programs.
Key words: Logic programs, artificial neural networks, SLD-resolution

1 Introduction

The problem of reasoning with uncertainty and conflicting sources of information has been a subject of research for quite a long time, see [1, 5, 6, 9], for example. In [7], we defined very general annotated (first-order) logic programs (BAPs) based on infinite bilattices. These logic programs can process information about facts whilst incorporating conflicting or incomplete information about them. The semantic operator $\mathcal{T}_P$ defined in [7] for BAPs reflects some remarkable properties of the least Herbrand model for BAPs. In this paper, we show that the computation of $\mathcal{T}_P$ by connectionist neural networks requires the introduction of learning functions into the structure of the networks. In this sense, we believe that BAPs provide a suitable formalism for integrating the pure logical deduction of connectionism and the properties of spontaneous learning manifested by artificial neural networks (ANNs) thought of as nature-inspired models of computation. We propose an SLD-resolution for BAPs which is the first sound and complete proof procedure we know of for logic programs based on infinite (bi)lattices.

The structure of the paper is as follows. In §2, we summarise all the results obtained in [7] in relation to the computation of the least Herbrand model for BAPs. In §3, we build learning ANNs which are able to compute the least Herbrand model for BAPs and prove this fact. In particular, we describe in §3 how

⋆ The authors thank the Boole Centre for Research in Informatics (BCRI) at University College Cork for substantial support in the preparation of this paper.

⋆⋆ The authors are grateful to three anonymous referees for their useful suggestions concerning a preliminary version of the paper. We also thank D. Woods for providing us with interesting examples of reasoning with uncertainty in complexity theory.

first-order fragments of BAPs can be approximated by ANNs. In §4, we introduce sound and complete SLD-resolution for BAPs. In §5, we conclude by giving a summary of our results.

2 Bilattice-Based Logic Programming

In this section, we survey some basic definitions and results obtained in [7]. We use the well-known definition of bilattices due to Ginsberg, see [1].

Definition 1. *A* bilattice $\mathbf{B}$ *is a sextuple* $(\mathbf{B}, \vee, \wedge, \oplus, \otimes, \neg)$ *such that* $(\mathbf{B}, \vee, \wedge)$ *and* $(\mathbf{B}, \oplus, \otimes)$ *are both complete lattices, and* $\neg : \mathbf{B} \to \mathbf{B}$ *is a mapping satisfying the following three properties:* $\neg^2 = Id_{\mathbf{B}}$, $\neg$ *is a dual lattice homomorphism from* $(\mathbf{B}, \vee, \wedge)$ *to* $(\mathbf{B}, \wedge, \vee)$, *and* $\neg$ *is the identity mapping on* $(\mathbf{B}, \oplus, \otimes)$.

We use here the fact that each distributive bilattice can be regarded as a product of two lattices, see [1]. Therefore, we consider only logic programs over distributive bilattices and regard the underlying bilattice of any program as a product of two lattices. Moreover, we always treat each bilattice we work with as isomorphic to some subset of $\mathbf{B} = L_1 \times L_2 = ([0,1], \leq) \times ([0,1], \leq)$, where $[0,1]$ is the unit interval of reals with the linear ordering defined on it.[3] Throughout this paper, we use $\mathbf{B}$ to denote the underlying bilattice of a given language.

In fact, we can use bilattice structures to formalize hypothetical and uncertain reasoning of the sort humans beings are capable of carrying out.

Example 1. Hypothetical reasoning is natural when, for example, scientists face an unsolvable problem which is, however, very important for their subject. Consider, for example the "P $\neq$ NP?" problem. Imagine two bright scientists Dr. N and Dr. M employed by a university to solve it. The scientists consider some related problems which may lead to the final proof of "P $\neq$ NP"; for example, "NP $\neq$ coNP?" and "NC$\neq$ NP?". After a while, both scientists give proofs (both are very long and need to be checked by someone): Dr. M has proven that "NP$\neq$ coNP", and Dr. N has proven that "NP=coNP". If a two-valued logic program receives the data, it will report a contradiction. Humans might, for example, make the following conclusions. If "NP$\neq$ coNP" and then "NP=coNP" were proven, than no conclusions yet can be made about the "P $\neq$ NP" problem. If next Dr. M reports that "P $\neq$ NP" and Dr. N reports that "P=NP", it will be possible to derive the evidence foe and against the statement "It is proven that "NC$\neq$ NP".

We need to introduce some formalism to reason in situations of the sort described in Example 1. We define an annotated bilattice-based language $\mathcal{L}$

[3] Elements of such a bilattice are pairs: the first element of each pair denotes evidence for a fact, and the second element denotes evidence against it. Thus, $\langle 1, 0 \rangle$ is the analogue of "truth" and is maximal with respect to the truth ordering, while $\langle 1, 1 \rangle$ may be seen as "contradiction" (or "both") and is maximal with respect to the knowledge ordering.

to consist of individual variables, constants, functions and predicate symbols together with annotation terms which can consist of variables, constants and/or functions over a bilattice. We allow six connectives and two quantifiers, as follows: $\oplus, \otimes, \vee, \wedge, \neg, \sim, \Sigma, \Pi$.

An *annotated formula* is defined inductively as follows: if R is an n-ary predicate symbol, $t_1, \ldots, t_n$ are terms, and (μ, ν) is an annotation term, then $R(t_1, \ldots, t_n) : (\mu, \nu)$ is an *annotated formula* (called an *annotated atom*). Annotated atoms can be combined to form complex formulae using the connectives and quantifiers.

A *bilattice-based annotated logic program (BAP)* P consists of a finite set of (annotated) *program clauses* of the form

$$A : (\mu, \nu) \leftarrow L_1 : (\mu_1, \nu_1), \ldots, L_n : (\mu_n, \nu_n),$$

where $A : (\mu, \nu)$ denotes an annotated atom called the *head* of the clause, and $L_1 : (\mu_1, \nu_1), \ldots, L_n : (\mu_n, \nu_n)$ denotes $L_1 : (\mu_1, \nu_1) \otimes \ldots \otimes L_n : (\mu_n, \nu_n)$ and is called the *body* of the clause; each $L_i : (\mu_i, \nu_i)$ is an annotated literal called an *annotated body literal* of the clause. Individual and annotation variables in the body are thought of as being existentially quantified using Σ.

In [7], we showed how the remaining connectives $\oplus, \vee, \wedge$ can be introduced into BAPs. The definitions of the terms *unit clause*, *program goal* and *pre-interpretation* are standard, see [8].

Let D, v, and J denote respectively a domain of (pre-)interpretation, a variable assignment and a pre-interpretation for a given language, see [8]. An interpretation I for $\mathcal{L}$ consists of J together with the following mappings. The first mapping $\mathcal{I}$ assigns $|R|_{\mathcal{I},v} : D^n \longrightarrow \mathbf{B}$ to each n-ary predicate symbol R in $\mathcal{L}$. Further, for each element $\langle \alpha, \beta \rangle$ of $\mathbf{B}$, we define a mapping $\chi_{\langle \alpha, \beta \rangle} : \mathbf{B} \longrightarrow \mathbf{B}$, where $\chi_{\langle \alpha, \beta \rangle}(\langle \alpha', \beta' \rangle) = \langle 1, 0 \rangle$ if $\langle \alpha, \beta \rangle \leq_k \langle \alpha', \beta' \rangle$ and $\chi_{\langle \alpha, \beta \rangle}(\langle \alpha', \beta' \rangle) = \langle 0, 1 \rangle$ otherwise. The mapping χ is used to evaluate annotated formulae. Thus, if F is an annotated atom $R(t_1, \ldots, t_n) : (\mu, \nu)$, then the value of F is given by $I(F) = \chi_{\langle \mu, \nu \rangle}(|R|_{\mathcal{I},v}(|t_1|_v, \ldots, |t_n|_v))$. Furthermore, using χ we can proceed to give interpretation to complex annotated formulae in the standard way, see [7] (or [6] for lattice-based interpretations of annotated logic programs). All the connectives of the language are put into correspondence with bilattice operations, and in particular quantifiers correspond to infinite bilattice operations. We call the composition of the two mappings $\mathcal{I}$ and χ an *interpretation* for the bilattice-based annotated language $\mathcal{L}$ and for simplicity of notation denote it by I. Indeed, the interpretations of BAPs possess some remarkable properties which make the study of BAPs worthwhile, as follows.

Proposition 1. *[7]*

1. *Let F be a formula, and fix the value $I(F)$. If $I(F : (\alpha, \beta)) = \langle 1, 0 \rangle$, then $I(F : (\alpha', \beta')) = \langle 1, 0 \rangle$ for all $\langle \alpha', \beta' \rangle \leq_k \langle \alpha, \beta \rangle$.*
2. *$I(F_1 : (\mu_1, \nu_1) \otimes \ldots \otimes F_k : (\mu_k, \nu_k)) = \langle 1, 0 \rangle \iff I(F_1 : (\mu_1, \nu_1) \oplus \ldots \oplus F_k : (\mu_k, \nu_k)) = \langle 1, 0 \rangle \iff I(F_1 : (\mu_1, \nu_1) \wedge \ldots \wedge F_k : (\mu_k, \nu_k)) = \langle 1, 0 \rangle \iff$ each $I(F_i : (\mu_i, \nu_i)) = \langle 1, 0 \rangle$, where $i \in \{1, \ldots, k\}$.*

3. *If* $I(F_1 : (\mu_1, \nu_1) \odot \ldots \odot F_k : (\mu_k, \nu_k)) = \langle 1, 0\rangle$, *then* $I((F_1 \odot \ldots \odot F_k) : ((\mu_1, \nu_1) \odot \ldots \odot (\mu_k, \nu_k))) = \langle 1, 0\rangle$, *where* $\odot$ *is any one of the connectives* $\otimes, \oplus, \wedge$.
4. *For every formula* F, $I(F : (0, 0)) = \langle 1, 0\rangle$.

These properties influence models for BAPs. In particular, we introduced in [7] a semantic operator which shows how all the logical consequences of each program can be computed.

Let I be an interpretation for $\mathcal{L}$ and let F be a closed annotated formula of $\mathcal{L}$. Then I is a *model* for F if $I(F) = \langle 1, 0\rangle$. We say that I is a model for a set S of annotated formulae if I is a model for each annotated formula of S. We say that F *is a logical consequence of* S if, for every interpretation I of $\mathcal{L}$, I is a model for S implies I is a model for F.

Let B_P and U_P denote an annotation Herbrand base respectively Herbrand universe for a program P, see [7] for further explanations. An *annotation Herbrand interpretation* HI *for* P consists of the Herbrand pre-interpretation HJ (see [8]) with domain HD of $\mathcal{L}$ together with the following: for each n-ary predicate symbol in $\mathcal{L}$, the assignment of a mapping from $U_{\mathcal{L}}^n$ into $\mathbf{B}$. In common with conventional logic programming, each Herbrand interpretation HI for P can be identified with the subset $\{R(t_1, \ldots, t_k) : (\alpha, \beta) \in B_P | R(t_1, \ldots, t_k) : (\alpha, \beta)$ receives the value $\langle 1, 0\rangle$ with respect to $HI\}$ of B_P it determines, where $R(t_1, \ldots, t_k) : (\alpha, \beta)$ denotes a typical element of B_P. This set constitutes an *annotation Herbrand model for* P. Finally, we let $\mathrm{HI}_{P,\mathbf{B}}$ denote the set of all annotation Herbrand interpretations for P.

In [7], we introduced a semantic operator $\mathcal{T}_P$ for BAPs, proved its continuity and showed that it computes the least Herbrand model for a given BAP. Indeed, we define $\mathcal{T}_P$ next.

Definition 2. *We define the mapping* $\mathcal{T}_P : \mathrm{HI}_{P,\mathbf{B}} \to \mathrm{HI}_{P,\mathbf{B}}$ *as follows:* $\mathcal{T}_P(\mathrm{HI})$ *denotes the set of all* $A : (\mu, \nu) \in B_P$ *such that either*

1. *There is a strictly ground instance of a clause* $A : (\mu, \nu) \leftarrow L_1 : (\mu_1, \nu_1), \ldots, L_n : (\mu_n, \nu_n)$ *in* P *such that there exist annotations* $(\mu'_1, \nu'_1), \ldots, (\mu'_n, \nu'_n)$ *satisfying* $\{L_1 : (\mu'_1, \nu'_1), \ldots, L_n : (\mu'_n, \nu'_n)\} \subseteq \mathrm{HI}$, *and one of the following conditions holds for each* (μ'_i, ν'_i)*:*
   (a) $(\mu'_i, \nu'_i) \geq_k (\mu_i, \nu_i)$,
   (b) $(\mu'_i, \nu'_i) \geq_k \oplus_{j \in J_i} (\mu_j, \nu_j)$, *where* J_i *is the finite set of those indices such that* $L_j = L_i$

or

2. *there are annotated strictly ground atoms* $A : (\mu^*_1, \nu^*_1), \ldots, A : (\mu^*_k, \nu^*_k) \in \mathrm{HI}$ *such that* $\langle \mu, \nu\rangle \leq_k \langle \mu^*_1, \nu^*_1\rangle \oplus \ldots \oplus \langle \mu^*_k, \nu^*_k\rangle$.[4]

Semantic operators defined for many logic programs as in the papers of Fitting and Van Emden (and other authors) use only some form of item 1.*a* from Definition 2. However, this condition is not sufficient for computation of the Herbrand models for (bi)lattice-based logic programs.

[4] Note that whenever $F : (\mu, \nu) \in HI$ and $(\mu', \nu') \leq_k (\mu, \nu)$, then $F : (\mu', \nu') \in HI$. Also, for each formula F, $F : (0, 0) \in HI$.

Example 2. Consider the logic program: $B : (0,1) \leftarrow$, $B : (1,0) \leftarrow$, $A : (0,0) \leftarrow B : (1,1)$, $C : (1,1) \leftarrow A : (1,0), A : (0,1)$. We can regard this program as formalizing Example 1. Let B stand for "NP $\neq$ coNP", A stand for "P $\neq$ NP" and C stand for "It is proven that "NC $\neq$ NP", annotations $(0,0), (0,1), (1,0), (1,1)$ express respectively "no proof/refutation is given", "proven", "proven the opposite" and "contradictory, or proven both the statement and the opposite". The least fixed point of $\mathcal{T}_P$ is $\mathcal{T}_P \uparrow 3 = \{B : (0,1), B : (1,0), B : (1,1), A : (0,0), C : (1,1)\}$, precisely the conclusions we mentioned in Example 1. However, the item 1.a (corresponding to the classical semantic operator) would allow us to compute only $\mathcal{T}_P \uparrow 1 = \{B : (0,1), B : (1,0)\}$, that is, only explicit consequences of a program, which then leads to a contradiction in the two-valued case. In the same way, the properties stated in Proposition 1 suggest that there can be some implicit logical consequences which can be derived if we take into consideration the underlying (bi)lattice structure of the program.

3 Neural Networks for Reasoning with Uncertainty

3.1 Connectionist Networks: Some Basic Definitions

In this subsection, we follow closely [3] and [4]. A *connectionist network* is a directed graph. A *unit* k in this graph is characterized, at time t, by its *input vector* $(i_{k1}(t), \ldots i_{kn_k}(t))$, its potential $p_k(t) \in \mathbb{R}$, its *threshold* $\Theta_k \in \mathbb{R}$, and its *value* $v_k(t)$. Units are connected via a set of directed and weighted connections. If there is a connection from unit j to unit k, then $w_{kj} \in \mathbb{R}$ denotes the *weight* associated with this connection, and $i_{kj}(t) = w_{kj}v_j(t)$ is the *input* received by k from j at time t. The units are updated synchronously. In each update, the potential and value of a unit are computed with respect to an *activation* and an *output function* respectively. All units considered in this paper compute their potential as the weighted sum of their inputs minus their threshold:

$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) \right) - \Theta_k.$$

The units are updated synchronously, time becomes $t + \Delta t$, and the output value for k, $v_k(t + \Delta t)$ is calculated from $p_k(t)$ by means of a given *output function* ψ, that is, $v_k(t + \Delta t) = \psi(p_k(t))$. The output function ψ we use in this paper is the binary threshold function H, that is, $v_k(t + \Delta t) = H(p_k(t))$, where $H(p_k(t)) = 1$ if $p_k(t) > 0$ and 0 otherwise. Units of this type are called *binary threshold units*.

In this paper, we will only consider connectionist networks where the units can be organized in layers. A *layer* is a vector of units. An n-layer *feedforward network* $\mathcal{F}$ consists of the *input* layer, $n-2$ *hidden* layers, and the *output* layer, where $n \geq 2$. Each unit occurring in the i-th layer is connected to each unit occurring in the $(i+1)$-st layer, $1 \leq i < n$. Let r and s be the number of units occurring in the input and output layers, respectively. A connectionist network $\mathcal{F}$ is called a *multilayer feedforward network* if it is an n-layer feedforward network

for some n. A miltilayer feedforward network $\mathcal{F}$ computes a function $f_{\mathcal{F}} : \mathbb{R}^r \to \mathbb{R}^s$ as follows. The input vector (the argument of $f_{\mathcal{F}}$) is presented to the input layer at time t_0 and propagated through the hidden layer to the output layer. At each time point, all units update their potential and value. At time $t_0+(n-1)\Delta t$, the output vector (the image under $f_{\mathcal{F}}$ of the input layer) is read off the output layer.

3.2 Neural Networks and Propositional BAPs.

Hölldobler et al. defined in [4] ANNs which are capable of computing the immediate consequence operator T_P for classical propositional logic programs. However, these ANNs cannot "learn" new information, that is, they cannot change their weights either with supervision or without it.

We extend this approach to learning ANNs which can compute logical consequences of BAPs. This will allow us to introduce hypothetical and uncertain reasoning into the framework of neural-symbolic computation. Bilattice-based logic programs can work with conflicting sources of information and inconsistent databases. Therefore, ANNs corresponding to these logic programs should reflect this facility as well, and this is why we introduce some forms of learning into ANNs. These forms of leaning can be seen as corresponding to a sort of unsupervised Hebbian learning, which is commonly used in the context of ANNs. The general idea behind Hebbian learning is that positively correlated activities of two neurons strengthen the weight of the connection between them and that uncorrelated or negatively correlated activities weaken the weight of the connection (the latter form is known as Anti-Hebbian learning).

The general conventional definition of Hebbian learning is given as follows, see [2] for example. Let k and j denote two neurons and w_{kj} denote a weight of the connection from j to k. We denote the value of j at time t as $v_j(t)$ and the potential of k at time t as $p_k(t)$. Then the rate of change in the weight between j and k is expressed in the form

$$\Delta w_{kj}(t) = F(v_j(t), p_k(t)),$$

where F is some function. As a special case of this formula, it is common to write

$$\Delta w_{kj}(t) = \eta(v_j(t))(p_k(t)),$$

where η is a constant that determines the *rate of learning* and is positive in case of Hebbian learning and negative in case of Anti-Hebbian learning. In this section, we will compare the two learning functions we introduce with this conventional definition of Hebbian learning.

First, we prove a theorem establishing a relationship between learning ANNs and bilattice-based annotated logic programs with no function symbols occurring either in the predicate symbols or in the annotations. (Since the Herbrand base for these programs is finite, they can equivalently be seen as propositional bilattice-based logic programs with no functions allowed in the annotations.) In the next subsection, we will extend the result to first-order BAPs with functions in individual and annotation terms.

Theorem 1. *For each function-free BAP P, there exists a 3-layer feedforward learning ANN which computes $\mathcal{T}_P$.*

Proof. Let m and n be the number of strictly ground annotated atoms from the annotation Herbrand base $\mathcal{B}_P$ and the number of clauses occurring in P respectively. Without loss of generality, we may assume that the annotated atoms are ordered. The network associated with P can now be constructed by the following translation algorithm.

1. The input and output layers are vectors of binary threshold units of length k, where the i-th unit in the input and output layers represents the i-th strictly ground annotated atom, $1 \leq k \leq m$. The threshold of each unit occurring in the input or output layer is set to 0.5.
2. For each clause of the form $A : (\alpha, \beta) \leftarrow B_1 : (\alpha_1, \beta_1), \ldots, B_m : (\alpha_m, \beta_m)$, $m \geq 0$, in P do the following.
   2.1 Add a binary threshold unit c to the hidden layer.
   2.2 Connect c to the unit representing $A : (\alpha, \beta)$ in the output layer with weight 1. We will call connections of this type 1-*connections*.
   2.3 For each atom $B_j : (\alpha_j, \beta_j)$ in the input layer, connect the unit representing $B_j : (\alpha_j, \beta_j)$ to c and set the weight to 1. (We will call these connections 1-connections also.)
   2.4 Set the threshold θ_c of c to $l - 0.5$, where l is the number of atoms in $B_1 : (\alpha_1, \beta_1), \ldots, B_m : (\alpha_m, \beta_m)$.
   2.5 If an input unit representing $B : (\alpha, \beta)$ is connected to a hidden unit c, connect each of the input units representing annotated atoms $B_i : (\alpha_i, \beta_i), \ldots, B_j : (\alpha_j, \beta_j)$, where $(B_i = B), \ldots, (B_j = B)$, to c. These connections will be called $\otimes$-*connections*. The weights of these connections will depend on a learning function. If the function is inactive, set the weight of each $\otimes$-*connection* to 0.
3. If there are units representing atoms of the form $B_i : (\alpha_i, \beta_i), \ldots, B_j : (\alpha_j, \beta_j)$, where $B_i = \ldots = B_j$ in input and output layers, correlate them as follows. For each $B_i : (\alpha_i, \beta_i)$, connect the unit representing $B_i : (\alpha_i, \beta_i)$ in the input layer to each of the units representing $B_i : (\alpha_i, \beta_i), \ldots, B_j : (\alpha_j, \beta_j)$ in the output layer. These connections will be called the $\oplus$-*connections*. If an $\oplus$-connection is set between two atoms with different annotations, we consider them as being connected via hidden units with thresholds 0. If an $\oplus$-connection is set between input and output units representing the same annotated atom $B : (\alpha, \beta)$, we set the threshold of the hidden unit connecting them to -0.5, and we will call them $\oplus$-hidden units, so as to distinguish the hidden units of this type. The weights of all these $\oplus$-connections will depend on a learning function. If the function is inactive, set the weight of each $\oplus$-*connection* to 0.
4. Set all the weights which are not covered by these rules to 0.

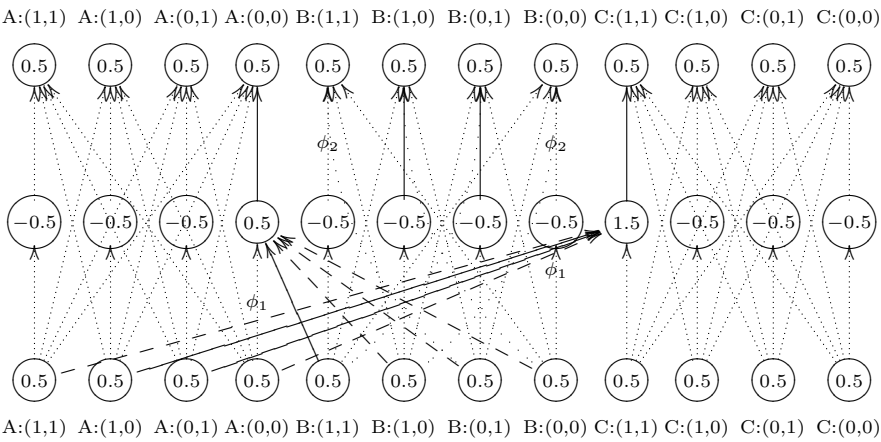
Allow two learning functions to be embedded into the $\otimes$ -connections and the $\oplus$ -connections. We let v_i denote the value of the neuron representing $B_i : (\alpha_i, \beta_i)$ and p_c denote the potential of the unit c.

Let a unit representing $B_i : (\alpha_i, \beta_i)$ in the input layer be denoted by i. If i is connected to a hidden unit c via an $\otimes$ -connection, then a learning function ϕ_1 is associated to this connection. We let $\phi_1 = \Delta w_{ci}(t) = (v_i(t))(-p_c(t) + 0.5)$ become active and change the weight of the $\otimes$-connection from i to c at time t if units representing atoms $B_j : (\alpha_j, \beta_j), \ldots, B_k : (\alpha_k, \beta_k)$ $(B_i = B_j = \ldots = B_k)$ became activated at time $t - \Delta t$, they are connected to c via 1-connections and $\langle \alpha_i, \beta_i \rangle \geq_k (\langle \alpha_j, \beta_j \rangle \otimes \ldots \otimes \langle \alpha_k, \beta_k \rangle)$.

Function ϕ_2 is embedded only into connections of type $\oplus$, namely, into $\oplus$-connections between hidden and output layers. Let o be an output unit representing an annotated atom $B_i : (\alpha_i, \beta_i)$. Activate $\phi_2 = \Delta w_{oc}(t) = (v_c(t))(p_o(t)+1.5)$ at time t if it is embedded into an $\oplus$-connection from the $\oplus$- hidden unit c to o and there are output units representing annotated atoms $B_j : (\alpha_j, \beta_j), \ldots, B_k : (\alpha_k, \beta_k)$, where $(B_i = B_j), \ldots, (B_i = B_k)$, which are connected to the unit o via $\oplus$-connections, these output units became activated at time $t - 2\Delta t$ and $\langle \alpha_i, \beta_i \rangle \leq_k (\langle \alpha_j, \beta_j \rangle \oplus \ldots \oplus \langle \alpha_k, \beta_k \rangle)$.

Each interpretation I for P can be represented by a binary vector $(v_1, \ldots, v_m)$. Such an interpretation is given as an input to the network by externally activating corresponding units of the input layer at time t_0. It remains to show that $A : (\alpha, \beta) \in \mathcal{T}_P \uparrow n$ for some n if and only if the unit representing $A : (\alpha, \beta)$ becomes active at time $t_0 + 2\Delta t$, for some Δt. The proof that this is so proceeds by routine induction.

Example 3. The following diagram displays the neural network which computes $\mathcal{T}_P \uparrow 3$ from Example 2. Without functions ϕ_1, ϕ_2 the ANN will compute only $\mathcal{T}_P \uparrow 1 = \{B : (0,1), B : (1,0)\}$, explicit logical consequences of the program. Note that arrows $\longrightarrow$, $- - \succ$, $\cdots\cdots\succ$ denote respectively 1-connections, $\otimes$-connections and $\oplus$-connections, and we have marked by ϕ_1, ϕ_2 the connections which are activated by the learning functions.[5]



[5] According to the conventional definition of feedforward ANNs, each output neuron denoting some atom is in turn connected to the input neuron which denotes the same atom via a 1-connection and thus forms a loop. We do not draw these connections here.

We can make several conclusions from the construction of Theorem 1.

- Neurons representing annotated atoms with identical first-order (or propositional) components are joined into multineurons in which neurons are correlated using $\oplus$- and $\otimes$-connections.
- The learning function ϕ_2 roughly corresponds to Hebbian learning, with the rate of learning $\eta_2 = 1$, the learning function ϕ_1 corresponds to Anti-Hebbian learning with the rate of learning 1, and we can regard η_1 as negative because the factor p_c in the formula for ϕ_1 is multiplied by (-1).
- The main problem Hebbian learning causes is that the weights of connections with embedded learning functions tend to grow exponentially, which cannot fit the model of biological neurons. This is why traditionally some functions are introduced to bound the growth. In the ANNs we have built some of the weights may grow with iterations, but the growth will be very slow, because of the activation functions, namely, binary threshold functions, used in the computation of each v_i.

3.3 Neural Networks and First-Order BAPs

Since ANNs were proven to compute least fixed points of the semantic operator defined for propositional logic programs, many attempts have been made to extend this result to first-order logic programs. See, for example, [3], [10]. We extend here the result obtained by Seda [10] for two-valued first-order logic programs and their representations by ANNs to first-order BAPs.

Let $l : B_P \to \mathbb{N}$ be a *level mapping* with the property that, given $n \in \mathbb{N}$, we can effectively find the set of all $A : (\mu, \nu) \in B_P$ such that $l(A : (\mu, \nu)) = n$. The following definition is due to Fitting, and for further explanation see [10] or [7].

Definition 3. *Let* $\mathrm{HI}_{P,\mathbf{B}}$ *be the set of all interpretations* $B_P \to \mathbf{B}$*. We define the ultrametric* $d : \mathrm{HI}_{P,\mathbf{B}} \times \mathrm{HI}_{P,\mathbf{B}} \to \mathbb{R}$ *as follows: if* $\mathrm{HI}_1 = \mathrm{HI}_2$*, we set* $d(\mathrm{HI}_1, \mathrm{HI}_2) = 0$*, and if* $\mathrm{HI}_1 \neq \mathrm{HI}_2$*, we set* $d(\mathrm{HI}_1, \mathrm{HI}_2) = 2^{-N}$*, where* N *is such that* HI_1 *and* HI_2 *differ on some ground atom of level* N *and agree on all atoms of level less then* N*.*

Fix an interpretation HI from elements of the Herbrand base for a given program P to the set of values $\{\langle 1, 0\rangle, \langle 0, 1\rangle\}$. We assume further that $\langle 1, 0\rangle$ is encoded by 1 and $\langle 0, 1\rangle$ is encoded by 0. Let HI_P denote the set of all such interpretations, and take the semantic operator $\mathcal{T}_P$ as in Definition 2. Let $\mathcal{F}$ denote a 3-layer feedforward learning ANN with m units in the input and output layers. The input-output mapping $f_{\mathcal{F}}$ is a mapping $f_{\mathcal{F}} : HI_P \to HI_P$ defined as follows. Given $HI \in HI_P$, we present the vector $(HI(B_1 : (\alpha_1, \beta_1)), \ldots, HI(B_m : (\alpha_m, \beta_m)))$, to the input layer; after propagation through the network, we determine $f_{\mathcal{F}}(HI)$ by taking the value of $f_{\mathcal{F}}(HI)(A_j : (\alpha_j, \beta_j))$ to be the value in the jth unit in the output layer, $j = 1, \ldots, m$, and taking all other values of $f_{\mathcal{F}}(HI)(A_j : (\alpha_j, \beta_j))$ to be 0.

Suppose that M is a fixed point of $\mathcal{T}_P$. Following [10], we say that a family $\mathcal{F} = \{\mathcal{F}_i : i \in \mathcal{I}\}$ of 3-layer feedforward learning network $\mathcal{F}_i$ computes M if

there exists $HI \in HI_P$ such that the following holds: given any $\varepsilon > 0$, there is an index $i \in \mathcal{I}$ and a natural number m_i such that for all $m \geq m_i$ we have $d(f_i^m(HI), M) < \varepsilon$, where f_i denotes $f_{\mathcal{F}_i}$ and $f_i^m(HI)$ denotes the mth iterate of f_i applied to HI.

Theorem 2. *Let P be an arbitrary annotated program, let* HI *denote the least fixed point of $\mathcal{T}_P$ and suppose that we are given $\varepsilon > 0$. Then there exists a finite program $\overline{P} = \overline{P}(\varepsilon)$ (a finite subset of ground(P)) such that $d(\overline{\mathrm{HI}}, \mathrm{HI}) < \varepsilon$, where $\overline{\mathrm{HI}}$ denotes the least fixed point of $\mathcal{T}_{\overline{P}}$. Therefore, the family $\{\mathcal{F}_n | n \in \mathbb{N}\}$ computes* HI*, where $\mathcal{F}_n$ denotes the neural network obtained by applying the algorithm of Theorem 1 to P_n, and P_n denotes $\overline{P}(\varepsilon)$ with ε taken as 2^{-n} for $n = 1, 2, 3, \ldots$.*

This theorem clearly contains two results corresponding to the two separate statements made in it. The first concerns finite approximation of $\mathcal{T}_P$, and is a straightforward generalization of a theorem established in [10]. The second is an immediate consequence of the first conclusion and Theorem 1. Thus, we have shown that the learning ANNs we have built can approximate the least fixed point of the semantic operator defined for first-order BAPs.

4 SLD-Resolution for BAPs

We propose a sound and complete proof procedure for BAPs as an alternative computational paradigm to ANNs. It can be particularly useful for programs whose annotation Herbrand Base is infinite, because in this case it may be problematical to build ANNs approximating the least fixed point of $\mathcal{T}_P$. As far as we know, this is the first sound and complete proof procedure for first-order infinitely interpreted (bi)lattice-based annotated logic programs. Compare, for example, our results with those obtained for constrained resolution for GAPs, which was shown to be incomplete, see [6], or with sound and complete (SLD)-resolutions for finitely-interpreted annotated logic programs (these logic programs do not contain annotation variables and annotation functions), see, for example, [5, 9]. We proceed with the definition of our proof procedure for BAPs.

We adopt the following terminology. Let P be a BAP and let G be a goal $\leftarrow A_1 : (\mu_1, \nu_1), \ldots, A_k : (\mu_k, \nu_k)$. An *answer* for $P \cup \{G\}$ is a substitution $\theta\lambda$ for individual and annotation variables of G. We say that $\theta\lambda$ is a *correct answer* for $P \cup \{G\}$ if $\Pi((A_1 : (\mu_1, \nu_1), \ldots, A_k : (\mu_k, \nu_k))\theta\lambda)$ is a logical consequence of P.

Definition 4 (SLD-derivation). *Let G_i be the annotated goal $\leftarrow A_1 : (\mu_1, \nu_1), \ldots, A_k : (\mu_k, \nu_k)$, and let $C, C_1^*, \ldots, C_l^*$ be the annotated clauses $A : (\mu, \nu) \leftarrow B_1 : (\mu_1', \nu_1'), \ldots, B_q : (\mu_q', \nu_q')$, $A_1^* : (\mu_1^*, \nu_1^*) \leftarrow body_1^*, \ldots, A_l^* : (\mu_l^*, \nu_l^*) \leftarrow body_l^*$. Then the set of goals $G_{i+1}^1, \ldots, G_{i+1}^m$ is* derived *from G_i and C (and $C_1^*, \ldots, C_l^*$) using mgu[6] $\theta\lambda$ if the following conditions hold.*

1. *$A_m : (\mu_m, \nu_m)$ is an annotated atom, called the selected atom, in G.*

[6] Throughout this section, *mgu* stands for "most general unifier".

2. *θ is an* mgu *of A_m and A, and one of the following conditions holds:*
   (a) *λ is an* mgu *of (μ_m, ν_m) and (μ, ν);*
   (b) *$(\mu_m, \nu_m)\lambda$ and $(\mu, \nu)\lambda$ are constants and $(\mu, \nu)\lambda \geq_k (\mu_m, \nu_m)\lambda$;*
   (c) *there are clauses $C_1^*, \ldots, C_l^*$ of the form $A_1^* : (\mu_1^*, \nu_1^*) \leftarrow body_1^*, \ldots, A_l^* : (\mu_l^*, \nu_l^*) \leftarrow body_l^*$ in P, such that θ is an* mgu *of A, A_m and $A_1^*, \ldots, A_l^*$, λ is an* mgu *of (μ_m, ν_m), (μ, ν) and $(\mu_1^*, \nu_1^*), \ldots, (\mu_l^*, \nu_l^*)$ or $(\mu_m, \nu_m)\lambda$, $(\mu, \nu)\lambda$ and $(\mu^*, \nu^*)\lambda, \ldots, (\mu_l^*, \nu_l^*)\lambda$ are constants such that $(\mu_m, \nu_m)\lambda \leq_k ((\mu, \nu)\lambda \oplus (\mu_1^*, \nu_1^*)\lambda \oplus \ldots \oplus (\mu_l^*, \nu_l^*)\lambda)$.*
3. *in case 2(a), 2(b), $G_{i+1} = (\leftarrow A_1 : (\mu_1, \nu_1), \ldots, A_{m-1} : (\mu_{m-1}, \nu_{m-1}), B_1 : (\mu_1', \nu_1'), \ldots, B_q : (\mu_q', \nu_q'), A_{m+1} : (\mu_{m+1}, \nu_{m+1}), \ldots, A_k : (\mu_k, \nu_k))\theta\lambda$.*
4. *in case 2(c), $G_{i+1} = (\leftarrow A_1 : (\mu_1, \nu_1), \ldots, A_{m-1} : (\mu_{m-1}, \nu_{m-1}), B_1 : (\mu_1', \nu_1'), \ldots, B_q : (\mu_q', \nu_q'), body_1^*, \ldots, body_l^*, A_{m+1} : (\mu_{m+1}, \nu_{m+1}), \ldots, A_k : (\mu_k, \nu_k))\theta\lambda$.*
   In this case, G_{i+1} is said to be derived from G_i, C and $C_1^*, \ldots, C_l^*$ using $\theta\lambda$.
5. *The goals $G_{i+1}^1, \ldots, G_{i+1}^m$ can be obtained using the following rules: in case there are atomic formulae $F_i : (\mu_i, \nu_i), F_{i+1} : (\mu_{i+1}, \nu_{i+1}), \ldots, F_j : (\mu_j, \nu_j)$ in G_i such that $F_i\theta = F_{i+1}\theta = \ldots = F_j\theta$, form the next goal $G_{i+1}^1 = F_i\theta : ((\mu_i, \nu_i) \otimes (\mu_{i+1}, \nu_{i+1})), \ldots, F_j : (\mu_j, \nu_j)$, then $G_{i+1}^2 = F_i : (\mu_i, \nu_i), F_i\theta : ((\mu_{i+1}, \nu_{i+1}) \otimes (\mu_{i+2}, \nu_{i+2})), \ldots, F_j : (\mu_j, \nu_j)$ and so on for all possible combinations of these replacements. Form the set of goals $G_{i+1}^1, \ldots, G_{i+1}^m$, which is always finite and can be effectively enumerated by, for example, enumerating goals according to their leftmost replacements and then according to the number of replacements.*
6. *Whenever a goal G_j^i contains a formula of the form $F : (0, 0)$, then remove $F : (0, 0)$ from the goal and form the next goal G_{j+1}^i.*

Definition 5. *Suppose that P is a BAP and G_0 is a goal. An* SLD-derivation *of $P \cup \{G_0\}$ consists of a sequence $G_0, G_1^i, G_2^j \ldots$ of BAP goals, a sequence $C_1, C_2, \ldots$ of BAP clauses and a sequence $\theta_1\lambda_1, \theta_2\lambda_2, \ldots$ of* mgus *such that each G_{i+1}^k is derived from G_i^j and C_{i+1} using $\theta_{i+1}\lambda_{i+1}$.*

An SLD-refutation *of $P \cup \{G_0\}$ is a finite SLD-derivation of $P \cup \{G\}$ which has the empty clause $\square$ as the last goal of the derivation. If $G_n^i = \square$, we say that the refutation has length n.*

The *success set* of P is the set of all $A : (\mu, \nu) \in B_P$ such that $P \cup \{\sim A\}$ has an SLD-refutation.

Theorem 3 (Soundness and completeness of SLD-resolution). *The success set of P is equal to its least annotation Herbrand model. Alternatively, soundness and completeness can be stated as follows. Every computed answer for $P \cup \{G\}$ is a correct answer for $P \cup \{G\}$, and for every correct answer $\theta\lambda$ for $P \cup \{G\}$, there exist a computed answer $\theta^*\lambda^*$ for $P \cup \{G\}$ and substitutions φ, ψ such that $\theta = \theta^*\varphi$ and $\lambda = \lambda^*\psi$.*

5 Conclusions and Further Work

We have shown that the logical consequences of the BAPs introduced in [7] can be computed by artificial neural networks with learning functions. Certain

constructions in the ANNs we have built for BAPs appear to be novel, and the question concerning the relationship between quantitative (bi)lattice-based logic programming and learning neural networks is itself quite novel. The BAPs were shown to be a very general formalism for reasoning about uncertainty and conflicting sources of information. In [7], we showed that implication-based logic programs á la van Emden and the annotation-free bilattice-based logic programs of [1] can be translated into the language of BAPs, and iterations of the semantic operators usually associated with these logic programs were shown to correspond to iterations of $\mathcal{T}_P$. These results extend bounds for further implementation of the ANNs we have introduced in the paper. The sound and complete SLD-resolution we have introduced for BAPs will serve as a complementary technique when working with BAPs having infinite annotation Herbrand base.

References

1. M. C. Fitting. Bilattices in logic programming. In G. Epstein, editor, *The twentieth International Symposium on Multiple-Valued Logic*, pages 238–246. IEEE, 1990.
2. S. Haykin. *Neural Networks. A comprehensive Foundation.* Macmillan College Publishing Company, 1994.
3. P. Hitzler, S. Hölldobler, and A. K. Seda. Logic programs and connectionist networks. *Journal of Applied Logic*, 2(3):245–272, 2004.
4. S. Hölldobler, Y. Kalinke, and H. P. Storr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11:45–58, 1999.
5. M. Kifer and E. L. Lozinskii. Ri: A logic for reasoning with inconsistency. In *Proceedings of the 4th IEEE Symposium on Logic in Computer Science (LICS)*, pages 253–262, Asilomar, 1989. IEEE Computer Press.
6. M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of logic programming*, 12:335–367, 1991.
7. E. Komendantskaya, A. K. Seda, and V. Komendantsky. On approximation of the semantic operators determined by bilattice-based logic programs. In *Proceedings of the Seventh International Workshop on First-Order Theorem Proving (FTP'05)*, pages 112–130, Koblenz, Germany, September 15–17 2005.
8. J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 2nd edition, 1987.
9. J. J. Lu, N. V. Murray, and E. Rosental. Deduction and search strategies for regular multiple-valued logics. *Journal of Multiple-valued logic and soft computing*, 11:375–406, 2005.
10. A. K. Seda. On the integration of connectionist and logic-based systems. In T. Hurley, M. Mac an Airchinnigh, M. Schellekens, A. K. Seda, and G. Strong, editors, *Proceedings of MFCSIT2004, Trinity College Dublin, July, 2004*, Electronic Notes in Theoretical Computer Science. Elsevier, 2005. To appear.

Clocking Type-2 Computation in the Unit Cost Model

Chung-Chih Li

School of Information Technology
Illinois State University, Normal, IL 61790, USA

Abstract. In [12] we defined a class of functions called Type-2 Time Bounds (henceforth $\mathbf{T_2TB}$) for clocking the Oracle Turing Machine (OTM) in order to capture the long missing notion of complexity classes at type-2. In the present paper we further advance the type-2 complexity theory under our notion of type-2 complexity classes. We have learned that the theory is highly sensitive to how the oracle answers are handled. We present a reasonable alternative called *unit cost* model, and examine how this model shapes the outlook of the type-2 complexity theory. Under the unit cost model we prove two theorems opposite to the classical union theorem and gap theorem. We also investigate some properties of $\mathbf{T_2TB}$ including a very useful theorem stating that there is an effective operator to convert any $\beta \in \mathbf{T_2TB}$ into an equivalent one that is *locking-detectable*. The existence of such operator allows us to simplify many proofs without loss of generality.[1]

1 Introduction

Let $\langle \varphi_i \rangle_{i \in \mathbf{N}}$ be an *acceptable programming system* and $\langle \Phi_i \rangle_{i \in \mathbf{N}}$ be a *complexity measure* associated to $\langle \varphi_i \rangle_{i \in \mathbf{N}}$, where $\mathbf{N}$ is the set of natural numbers. Simply put, one may consider φ_i as the function computed by the i^{th} Turing machine. A formal definition for an *acceptable programming system* can be found in [16, 15]. For the complexity measure, one may consider $\Phi_i(x)$ as the amount of resource needed to compute φ_i on x. We use $\varphi_i(x) \downarrow = y$ to denote that the computation of φ_i on x is converged and its value is y. Similarly, $\Phi_i(x) \downarrow = m$ means that the cost of computing φ_i on x is converged to m. In [7] Hartmanis and Stearns gave a precise definition for complexity classes as follows: $C(t) = \{f \mid \exists i[\varphi_i = f \text{ and } \Phi_i \leq^* t]\}$, where $\Phi_i \leq^* t$ means that the relation, $\Phi_i(x) \leq t(x)$, holds on *all but finitely many* values of x. Within two years, Blum proposed two axioms in [1] as the basic requirements for any reasonable dynamic complexity measures to meet. The two requirements are straightforward: for any $i, x, m \in \mathbf{N}$, we require (i) $\varphi_i(x) \downarrow$ if and only if $\Phi_i(x) \downarrow$, and (ii) $\Phi_i(x) = m$ is effectively decidable. The two axioms had successfully lifted the study of complexity theory to an abstract level with rich results that are independent from

[1] We've omitted all detailed proofs due to the space constraints. A full version is available at `http://galaxy.cs.lamar.edu/~chungli/type2/UnitCost.pdf`.

any specific machine models. These two landmark papers initiated an important study now known as *abstract complexity theory* in theoretical computer science.

It is obvious that the complexity theory should be extended into type-2 (a.k.a. second-ordered) computation. This inquiry can be traced back to Constable's 1973 paper [5] in which he asked what should a type-2 complexity theory look like? However, only a few scattered works had been done in the past three decades due to the difficulty of having a generally accepted abstraction for type-2 computation. In particular, we face a fundamental problem that there is no Church-Turing thesis at type-2. As a result, the notion of asymptotical behavior at type-2 and the way of clocking whatever type-2 computing devices become not quite as intuitive as ordinary type-1 computation. Recently, we introduced a notion of type-2 asymptotical behavior in [11] to catch the idea of its type-1 counterpart – *for all but finitely many*. Using this notion and the clocking scheme with type-2 time bounds proposed in [12], we describe a natural notion of type-2 complexity classes that seems to be a solid ground for type-2 complexity theory to take off. In the present paper, we further study the properties of our type-2 time bounds and point out that the type-2 complexity theory is highly sensitive to the actual *cost model* used in the clocking scheme. We believe that our investigation initiates a sound framework for theorists to further speculate a more complete machine-independent complexity theory for type-2 computation.

Notations: We first fix some necessary notations. By convention, natural numbers are taken as type-0 objects and functions over natural numbers are type-1 objects. Type-2 objects are *functionals* that take as inputs and produce as outputs type-1 objects. Let type-0 $\subset$ type-1 $\subset$ type-2. We are interested only in total functions of type $\mathbf{N} \to \mathbf{N}$ when they are taken as inputs of type-2 functionals. For convenience, let $\mathcal{T}$ denote the set of total functions of type $\mathbf{N} \to \mathbf{N}$ and $\mathcal{P}$ the set of partial functions of type $\mathbf{N} \rightharpoonup \mathbf{N}$. Also, let $\mathcal{F}$ denote the set of *finite* functions, i.e., $\mathcal{F} \subset \mathcal{P}$ and $\sigma \in \mathcal{F}$ if and only if $\mathsf{dom}(\sigma) \subset \mathbf{N}$ and $\mathsf{card}(\sigma) \in \mathbf{N}$. We fix a canonical indexing for $\mathcal{F}$ so we can treat any function in $\mathcal{F}$ as a natural number when it is taken as the input of some type-1 function. Let $\langle \cdot, \cdot \rangle$ be the standard pairing function defined in [15]. Thus, for every $\sigma \in \mathcal{F}$ and $x \in \mathbf{N}$, there is a unique $\langle \sigma, x \rangle \in \mathbf{N}$. Let $|n|$ be the length of the presentation of $n \in \mathbf{N}$. Unless stated otherwise, we let a, b, x, y, z range over $\mathbf{N}$, f, g, h range over $\mathcal{T}$, and F, G, H range over type-2 functionals. Without loss of generality we restrict type-2 functionals to our standard type $\mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$. Thus, we can follow the tradition by using the OTM as our standard formalism for type-2 computation. We also fix some necessary conventions for OTM's in the following paragraph.

Oracle Turing Machines: In addition to the standard I/O tape of a TM, an OTM has two extra tapes called query tape and answer tape. The type-0 numerical input is prepared at the beginning of the I/O tape and the type-1 functional input is prepared as a function oracle attached to the machine before the computation begins. During the course of computation, if the OTM needs some value from the function oracle, the OTM have to place the quetion to the query tape and then transit to a special state called query state. Then, the oracle will place the

answer to the answer tape in one step; no matter how big the answer might be. As for the classical complexity theory, we fix a programming system $\langle\widehat{\varphi}_i\rangle_{i\in\mathbf{N}}$ associated with a complexity measure $\langle\widehat{\Phi}_i\rangle_{i\in\mathbf{N}}$ for our OTM's. Conventionally, we take the number of steps an OTM performed as our time complexity measure. Note that the steps for the OTM to prepare the query and read the answer are counted as a part of the computational cost.

2 Type-2 Complexity Classes and Time Bounds

Seth followed Hartmanis and Stearns's notion to define type-2 complexity classes in [18] where he proposed two alternatives:

1. Given recursive $t : \mathbf{N} \to \mathbf{N}$, let $DTIME(t)$ denote the set of type-2 functionals such that, for every functional $F \in DTIME(t)$, F is total and there is an OTM $\widehat{M}_e$ that computes F and, on *every* $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\widehat{M}_e$ halts within $t(m)$ steps, where $m = |\mathsf{max}(\{x\}\cup Q)|$ and Q is the set of all answers returned from the oracle during the course of the computation.
2. Given computable $H : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, let $DTIME(H)$ denote the set of type-2 functionals such that, for every functional $F \in DTIME(H)$, F is total and there is an OTM $\widehat{M}_e$ that computes F and, on *every* $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\widehat{M}_e$ halts within $H(f, x)$ steps.

The key idea behind Seth's complexity classes is directly lifted from [7]. In fact, the same idea can also be found in other works such as [8, 17] along the line of machine characterizations. However, we face some problems that do not exist in type-1 computation. For example, in one of Seth's definitions, the resource bound is determined by the set of all answers returned from the oracle; but this set in general is not computable and hence it can't be available before the computation completes. Alternatively, we should update the resource bound dynamically upon each answer returned from the oracle during the course of the computation. But if we do so, there is no guarantee that a clocked OTM must be total. For example, Cook's POTM [6] is an OTM bounded by a polynomial in this manner but a POTM may run forever. Kapron and Cook's proposed their remedies in the context of feasible functionals in [8] and gave a very neat characterizations of type-2 Basic Feasible Functionals (BFF), where the so-called second-ordered polynomial is used as the resource bound. We may adapt all these ideas with our $\leq^*_2$ defined in [11] and extend the second-ordered polynomial to a general type-2 computable functional to have the following complexity class:

$$DTIME(H) = \{F \mid \exists e[\widehat{\varphi}_e = F \text{ and } \widehat{\Phi}_e \leq^*_2 H]\}. \tag{1}$$

$DTIME(H)$ seems to be a perfect analog of classical $DTIME$. However, we do not think using a type-2 functional as a resource bound is proper because the bound should not depend on information that is irrelevant to the computation. In other words, we prefer the clocking scheme of POTM. To avoid the problem of POTM we mentioned, we give a class of functions called Type-2 Time Bounds denoted by $\mathbf{T_2TB}$ in order to properly clock OTMs [12].

Definition 1 (Type-2 Time Bounds). *Let $\beta : \mathcal{F} \times \mathbf{N} \to \mathbf{N}$. We say that:*

1. *β is nontrivial, if for every $(\sigma, a) \in \mathcal{F} \times \mathbf{N}$, $\beta(\sigma, a) \geq |a| + 1$;*
2. *β is bounded, if for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\sigma \in \mathcal{F}$, and $\sigma \subset f$, we have $\beta(\sigma, x) \leq \lim_{\tau \to f} \beta(\tau, x)$;*
3. *β is convergent, if for every $(f, a) \in \mathcal{T} \times \mathbf{N}$, there exists $\sigma \in \mathcal{F}$ with $\sigma \subset f$ such that, for all τ with $\sigma \subseteq \tau$, we have $\beta(\sigma, a) = \beta(\tau, a)$;*
4. *β is $\mathcal{F}$-monotone, if for every $a \in \mathbf{N}$ and $\sigma, \tau \in \mathcal{F}$ with $\sigma \subseteq \tau$, we have $\beta(\sigma, a) \leq \beta(\tau, a)$.*

If β is computable, nontrivial, bounded, and convergent, we say that β is a type-2 time bound. Moreover, if β is $\mathcal{F}$-monotone, we say that β is strong.[2]

The properties listed in Definition 1 are formulated so to catch our intuition about what a resource bound should be in clocking the OTM (see the full version for details). By a standard diagonalization, one can prove that $\mathbf{T_2TB}$ is not *recursively enumerable*. This indeed is an uneasy fact, since being able to enumerate $\mathbf{T_2TB}$ is a property that can make many proofs possible or easier. Let $\beta(\sigma, x) \downarrow$ denote the situation that $\forall \tau \supseteq \sigma[\beta(\sigma, a) = \beta(\tau, a)]$. If $\beta(\sigma, x) \downarrow$, we say that (σ, x) is a *locking fragment* of β.

Definition 2 (Locking Detectors). *Let $\beta \in \mathbf{T_2TB}$. We say that ℓ is a locking detector of β if $\ell : \mathcal{F} \times \mathbf{N} \to \{0, 1\}$ is computable and (i) ℓ is $\mathcal{F}$-monotone, (ii) $\forall(\sigma, x) \in \mathcal{F} \times \mathbf{N}[(\ell(\sigma, x) = 1) \Rightarrow \beta(\sigma, x) \downarrow]$, and (iii) $\forall(f, x) \in \mathcal{T} \times \mathbf{N}[\lim_{\sigma \to f} \ell(\sigma, x) = 1]$.*

If $\beta \in \mathbf{T_2TB}$ has a locking detector ℓ, we say that β is locking detectable. If $\ell(\sigma, x) = 1$, then (σ, x) is a *locking fragment* of β. It is clear that whether a given β on some (σ, x) has converged is undecidable. Thus, we cannot simply assume that every type-2 time bound is locking detectable. Nevertheless, we have a very positive theorem allowing us to make that assumption without loss of generality (see Theorem 9 in Section 5).

3 A Clocking Scheme and Two Cost Models

We present a clocking scheme using our $\mathbf{T_2TB}$. This scheme is used implicitly in some works such as Kapron and Cook's [8], Seth's [18], and Royer's [17].

Definition 3 (Clocked OTM). *Let $\beta \in \mathbf{T_2TB}$ and $\widehat{M}_e$ be an OTM with index e. We say that $\widehat{M}_e$ is clocked by β if $\widehat{M}_e$ is simulated by the procedure shown in Figure 1. Such a clocked OTM is denoted by $\widehat{M}_{e,\beta}$ and the functional computed by $\widehat{M}_{e,\beta}$ is denoted by $\widehat{\varphi}_{e,\beta}$.*

Consider the procedure in Figure 1. The *budget* provided by β is computed upon every answer returned from the oracle during the course of the simulation

[2] In [12] we used WB to denote the set of type-2 time bounds and SB to denote the set of strong type-2 time bounds. Clearly, SB $\subset$ WB and SB $\neq$ WB.

```
Program for Clocked OTM M̂_{e,β} :
    input (f, x) ∈ T × N;
    var σ ∈ F; q, y, expense, budget ∈ N;          /* variable declaration */
    σ ⟵ ∅; expense ⟵ 0; budget ⟵ β(σ, x);          /* initialization */
    Simulate M̂_e on (f, x) step by step and upon each step completed do:
        expense ⟵ expense + 1;
        if (expense > budget)                       /* check budge */
            then output ⊥ and stop;        /* ⊥ is the bottom symbol. (⇑) */
        if (M̂_e halts with the output y)
            then output y and stop;          /* simulation completed. (⇓) */
        if (the step just simulated completes an oracle query)
            then do
                q ⟵ current query;
                σ ⟵ σ ∪ {(q, f(q))};          /* update query-answer set */
                budget ⟵ β(σ, x);                    /* update budget */
            end-do;
    Resume the simulation;
End program
```

Fig. 1. A Clocking Scheme for OTM's

of $\widehat{M}_e$ on (f, x). If the simulation has overrun the budget, then the simulation will be terminated at the line marked ($\Uparrow$). In this case we say that $\widehat{M}_e$ is *clipped down* by β on (f, a) denoted by $\widehat{\varphi}_{e,\beta}(f, a) \Uparrow$. On the other hand, if the simulation reaches the line marked ($\Downarrow$), which means that the simulation of $\widehat{M}_e$ on (f, a) is successfully completed, then we say that $\widehat{\varphi}_{e,\beta}(f, a)$ converges to value $\widehat{\varphi}_e(f, a)$. We denote this situation by $\widehat{\varphi}_{e,\beta}(f, a) \Downarrow$. Since β is convergent, it follows that the simulation of $\widehat{M}_e$ on (f, a) will either complete or eventually be clipped down by the clock. Therefore, for any $\beta \in \mathbf{T_2TB}$, $\widehat{\varphi}_{e,\beta}$ is a total computable functional of type $\mathcal{T} \times \mathbf{N} \to \mathbf{N}$. This removes the problem of POTM.

Theorem 1. *Given any computable $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ and $\beta \in \mathbf{T_2TB}$, there is a $\widehat{\varphi}$-program e for F such that $\widehat{\varphi}_e \neq \widehat{\varphi}_{e,\beta}$.*

The theorem above show that arbitrarily complex $\widehat{\varphi}$-programs exist. The proof is an easy application of classical recursion theory. Although the locking fragment of β in general is undecidable (see Section 5), we need not to know the value of β in the limit in order to construct an arbitrarily complex $\widehat{\varphi}$-program for any given computable type-2 functional. This is similar to the ordinary type-1 computation. A slightly more involved theorem is the type-2 version of Rabin Theorem [14] stating that, given any $\beta \in \mathbf{T_2TB}$, there is a 0-1 valued computable functional that cannot be computed by any $\widehat{\varphi}_{e,\beta}$. We prove a versions in [11] where the bound function is simply a computable type-2 functional. The proof is perfectly valid under the present clocking scheme with $\mathbf{T_2TB}$.

Unfortunately, the properties of $\beta \in \mathbf{T_2TB}$ and our intuitive clocking scheme are not sufficient to standardize a framework for type-2 complexity theory. The

way an OTM handles the oracle answers does matter. We have the following two conventions under our clocking scheme.

Definition 4 (Two Cost Models for OTM's).

1. *Answer-Length Cost Model: Whenever the oracle returns an answer to the oracle query, the machine is required to read every bit of the answer.*
2. *Unit Cost Model: The machine needs not to read any bit of the oracle answer unless the machine decides to do so.*

In other words, the cost for each answer returned from the oracle under the answer-length cost model is one unit step plus the length of the answer, while the other model is one. The underlying cost model used in [8, 17, 18] are the answer-length cost model. Also, the outline of a type-2 complexity theory given in [12] is also based on the answer-length cost model. The answer-length cost model from many aspects is more manageable. Nevertheless, we do not think the unit cost model is merely a peculiar convention. On the contrary, the unit cost model is rather reasonable in real computation. For example, only the first bit of the answer is needed to decide whether it is odd or even. However, the controversial part is that, under the unit cost model, the computation can aggressively gain some budget by just querying the oracle without reading the answers. This trick makes the complexity theory under the unit cost model much flatter than the theory under the other model. For example, there exist certain versions of Union Theorems [12] and Gap Theorems [10] under the answer-length cost model, but the theorems fail to hold under the unit cost model.

4 Complexity Theory under The Unit Cost Model

We explicitly make the notations for the unit cost model different by using a superscript u as follows: $\widehat{M}^u_e$, $\widehat{\varphi}^u_e$, $\widehat{\Phi}^u_e$, $\widehat{M}^u_{e,\beta}$, $\widehat{\varphi}^u_{e,\beta}$, and $\widehat{\Phi}^u_{e,\beta}$. For example, $\widehat{M}^u_e$ is the unit cost model OTM with index e, and $\widehat{\varphi}^u_{e,\beta}$ denotes the functional computed by $\widehat{M}^u_e$ with clock β. Since the two models do not differ in computability, we have $\widehat{\varphi}_e = \widehat{\varphi}^u_e$ for every e. On the other hand, $\widehat{\Phi}_e \neq \widehat{\Phi}^u_e$, and hence $\widehat{\varphi}_{e,\beta} \neq \widehat{\varphi}^u_{e,\beta}$ in general. Unless stated otherwise, if the superscript is omitted from the statement of some theorem, we mean that the theorem holds under both cost models. We adapt the notion of type-2 complexity classes proposed in [10–12] and alter the cost model to the unit cost model. The *exception set*, $E^u_{e,\beta}$ is defined as follows:

$$E^u_{e,\beta} = \left\{ (f,x) \in \mathcal{T} \times \mathbf{N} \;\middle|\; \widehat{\varphi}^u_{e,\beta}(f,x) \Uparrow \right\}.$$

We also adopt the topology introduced in [11], i.e., for every continuous functional $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, $\mathbb{T}(F)$ is the topology obtained by taking the set of total extensions of every *minimum* locking fragment of F as a basic open set. Since each functional $\widehat{\varphi}_e$ is continuous, the topology $\mathbb{T}(\widehat{\varphi}_e)$ is well defined. It's also clear that such $\mathbb{T}(F)$ is induced from the Baire topology. Note that, we require σ to be the minimum locking fragment, otherwise $\mathbb{T}(F)$ will inflate to the Baire topology.

Definition 5 (Type-2 Complexity Classes). *Let $\beta \in \mathbf{T_2TB}$. Define the set of computable type-2 functionals $\mathbf{C}^u(\beta)$ as*

$$\mathbf{C}^u(\beta) = \left\{ \widehat{\varphi}_e : \mathcal{T} \times \mathbf{N} \to \mathbf{N} \;\middle|\; e \in \mathbf{N} \text{ and } E^u_{e,\beta} \text{ is compact in } \mathbb{T}(\widehat{\varphi}_e) \right\}.$$

Inclusion property: In [11] we pointed out that it doesn't seem likely to have a reasonable notion for type-2 asymptotic relation, $\leq^*_{\mathbf{2}}$, that is transitive due to the topological constraints. Thus, if a type-2 complexity class is defined by some type-2 functional in the classical manner such as (1), a bigger resource bound does not always promise a bigger complexity class. Our clocking scheme and type-2 time bounds can easily fix this problem. This adds another reason to why we do not think using type-2 functionals as resource bounds is appropriate. Note that the theorem below holds under both cost models.

Theorem 2. *For every $\beta_1, \beta_2 \in \mathbf{T_2TB}$, $[\beta_1 \leq \beta_2] \implies [\mathbf{C}(\beta_1) \subseteq \mathbf{C}(\beta_2)]$.*

Since the value of a continuous functional on a compact set is bounded, it follows that, intuitively, if $\widehat{\varphi}_e \in \mathbf{C}(\beta)$ then we need only some constant extra budget to let $\widehat{\varphi}_e$ finish its computation on every points in $E_{e,\beta}$. This intuition indeed is correct under the unit cost model, i.e., if $F \in \mathbf{C}^u(\beta)$, then there exist $c, e \in \mathbf{N}$ such that $\widehat{\varphi}^u_e = F$ and $E^u_{e,\beta+c} = \emptyset$. However, under the answer-length cost model, we need more than a constant extra budge as shown in [10]: If $F \in \mathbf{C}(\beta)$, then there exist $c, e \in \mathbf{N}$ such that $\widehat{\varphi}^u_e = F$ and $E_{e,2\beta+c} = \emptyset$ under the answer-length cost model.

Enumerability: It is easy to show that the finite invariant closure of a type-1 complexity class is *recursively enumerable* [2]. However, not every complexity class itself can be recursively enumerated. When the cost bound function t is too small, the complexity class determined by t is unlikely to be recursively enumerable [2, 9]. On the other hand, if t is big enough to bound all *finite support functions*[3] almost everywhere, then the complexity class determined by t is recursively enumerable. In particular, if $t(x) \geq |x| + 1$ for all $x \in \mathbf{N}$, then all finite support functions are contained in the complexity class determined by t (see [3], section 9.4). Although we required every $\beta \in \mathbf{T_2TB}$ to be nontrivial, this requirement is not sufficient for enumerating $\mathbf{C}(\beta)$. The difficulty is that, given $(\sigma, x) \in \mathcal{F} \times \mathbf{N}$, we may not be able to test if it is the case that $\sigma \subset f$ for any input $f \in \mathcal{T}$ under the the answer-length cost model, since querying the oracle outside the domain of the locking fragment of β is dangerous, which may cause a huge returned answer and the OTM will use up its budget in scanning the entire answer as required under the answer-length cost model. In [10] we imposed two rather strong conditions to have $\mathbf{C}(\beta)$ being recursively enumerable. We also conjecture that there exists $\beta \in \mathbf{T_2TB}$ such that $\mathbf{C}(\beta)$ is not recursively enumerable. On the contrary, the cost of querying the oracle is more manageable under the unit cost model. As a result, we have the following theorem without any extra condition on β needed.

Theorem 3. *For every $\beta \in \mathbf{T_2TB}$, $\mathbf{C}^u(\beta)$ is recursively enumerable.*

[3] A function f is finite support if the value of f is 0 almost everywhere.

Non-Union Theorem: The Union Theorem [13] is one of the most fascinating theorems in classical complexity theory. Not just because the technique used in the proof then was new to complexity theorists, but also the theorem told us that most *natural* complexity classes have clear boundaries in terms of the bounds that determine Hartmanis and Stearns's complexity classes. In other words, we can use one computable function to exactly bound any given natural complexity class. Although any arbitrary union of computable functions is not necessarily a complexity class in general, we only need a very weak condition to have the following theorem known as the union theorem.

Theorem 4 (McCreight & Meyer [13]). *Let the sequence of recursive functions, f_0, f_1, f_2, ..., be recursive and $f_i(x) \leq f_{i+1}(x)$ for all $i, x \in \mathbf{N}$. Then, there is a recursive function g such that $C(g) = \bigcup_{i \in \mathbf{N}} C(f_i)$.*

According to the theorem, a complexity class such as $PTIME$, $PSPACE$, etc., each can be exactly determined by one recursive function; same to the set of computable functions bounded by computable functions in $O(f)$ (the big-O notation). At type-2, the union theorem seems to break down. For example, the class of type-2 basic feasible functionals is not a complexity class [10]. Nevertheless, in [12, 10] we imposed some quite strong but yet reasonable conditions on the sequence of type-2 time bounds to have a type-2 union theorem under the answer-length cost model. As a result, if we define a type-2 big-O notation as $\mathbf{O}(\beta) = \bigcup_{a,b \in \mathbf{N}} \mathbf{C}(a\beta + b)$, then for each $\beta \in \mathbf{T_2TB}$ there exists $\gamma \in \mathbf{T_2TB}$ such that $\mathbf{C}(\gamma) = \mathbf{O}(\beta)$ under the answer-length cost model. However, the conditions are not sufficient under the union cost model. The union theorem does not hold under the unit cost mode unless the sequence of the type-2 time bounds tends to trivial.

Theorem 5 (Non-Union Theorem). *For any $\beta \in \mathbf{T_2TB}$, there is no $\alpha \in \mathbf{T_2TB}$ such that $\mathbf{C}^u(\alpha) = \bigcup_{c \in \mathbf{N}} \mathbf{C}^u(c\beta)$.*

Thus, $\mathbf{O}(\beta)$ is not a complexity class under the unit cost model. Note that the sequence we proposed above, $\beta, 2\beta, 3\beta, \ldots$, is very conservative in a sense that the sequence is uniformly convergent, i.e., every one in the sequence converges at the same locking fragment, which is a very strong condition. Thus, the condition must be further strengthened if we want to sustain the union theorem under the unit cost model. For example, we may require the value of the sequence to be bounded on any (f, x), i.e., $\lim_{(i \to \infty, \sigma \to f)} \beta_i(\sigma, x) \in \mathbf{N}$. However, we consider a union theorem under such strong condition trivial.

Anti-Gap Theorem: When people tried to find an effective operation to enlarge a type-1 complexity class, the gap phenomena were discovered [2, 4]. We have learned that it is impossible to have such effective operation unless some "nice" property is assumed. We state a stronger version of gap theorems known as the Operator Gap Theorem in the following.

Theorem 6 (Constable [4] & Young [19]). *For any total effective operator Θ, we can effectively find an arbitrarily large recursive function t such that $\mathcal{C}(t) = \mathcal{C}(\Theta(t))$.*

In other words, we can always find resource bound t such that the given effective operator fails to enlarge the complexity class determined by t. Some properties such as *time-constructibility* and *honesty* are those commonly mentioned "nice" ones to dismiss the gap phenomena. Three major theorems in classical complexity theory – Compression theorem, Gap theorem, and Honesty theorem – form a wonderful trilogy telling a full story along this line.

In [10] we gave a preliminary idea for type-2 time-constructibility, but we still do not fully understand what should be the proper meaning of *type-2 honest functionals*. Under the answer-length cost model, the gap phenomena are inherited from the type-1 computation, i.e., the gap phenomena are caused by the type-1 part of the computation. We observe that no "pure" type-2 computation is possible under the answer-length cost model because every oracle query must be followed by an inevitable type-1 computation (i.e., reading the answer that can go arbitrarily huge). On the other hand, under the unit cost model, the type-2 computation becomes "purer" and the gap phenomena disappear. We see this as a positive result because we can uniformly enlarge a complexity class. The only condition is that, β has to be strong.

Theorem 7 (Anti-Gap Theorem). *Suppose $g : \mathbf{N} \to \mathbf{N}$ is recursive and, for every $x \in \mathbf{N}$, $g(x) \geq 3x$. Then, for every strong $\beta \in \mathbf{T_2TB}$, $\mathbf{C}^u(\beta) \subset \mathbf{C}^u(g \circ \beta)$.*

Note that Theorem 7 above does not hold if β is not strong (i.e., not $\mathcal{F}$-monotone). An intuitive explanation is that, if β is not strong, then it can shrink the budget to the bottom (i.e., $|x| + 1$) until it receives a locking fragment that is too long to be seen under the budget provided by $g \circ \beta$.

5 Properties of Type-2 Time Bounds

In this section we study the relation between type-2 time bounds and type-2 functionals. It is clear that each type-2 time bound determines a limit functional as follows. (More details about limit functionals can be found in Rogers' [15]).

Definition 6. *Given any $\beta \in \mathbf{T_2TB}$, define $F_\beta = \lambda f, x.(\lim_{\sigma \to f} \beta(\sigma, x))$.*

Some obvious properties of this limit functional, F_β, come directly from the properties of β. For example, F_β is a continuous functional and total on $\mathcal{T} \times \mathbf{N}$. Taking the type-2 almost everywhere relation, $\leq^*_{\mathbf{2}}$, defined in [11], we can prove that, $\widehat{\varphi}_e \in \mathbf{C}(\beta) \Rightarrow \widehat{\Phi}_e \leq^*_{\mathbf{2}} F_\beta$. However, the converse is false because the history of requesting budget from β does matter. Thus, $F_{\beta_1} = F_{\beta_2}$ does not imply that $\mathbf{C}(\beta_1) = \mathbf{C}(\beta_2)$, which means that the budget provided by β in the limit may not be useful for the computation. This causes the major difference between complexity classes defined by $\mathbf{T_2TB}$ and type-2 functionals. However, we may want to have a certain computable operation on β's to force the budget in the limit to be used earlier during the computation; in such a way all computations with complexity bounded by F_β can be finished. We argue that such an effective operation is impossible. We state this in the following theorem.

Theorem 8. *There is no recursive operator* $\Theta : \mathbf{T_2TB} \to \mathbf{T_2TB}$ *such that, for any* $\beta \in \mathbf{T_2TB}$ *and* $\widehat{\varphi}_e : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, $\widehat{\Phi}_e(f,x) \leq F_\beta(f,x) \Leftrightarrow \widehat{\varphi}_{e,\Theta(\beta)}(f,x) \Downarrow$.

We know that not every limit functional corresponds to a recursive functional that is total on recursive functions [15]. It is clear that if β is locking-detectable, then F_β is computable. Thus, there is a $\beta \in \mathbf{T_2TB}$ such that F_β is total but not computable. We obtain the following corollary.

Corollary 1. *There is* $\beta \in \mathbf{T_2TB}$ *that is not locking detectable.*

Locking Detectable Type-2 Time Bounds: Locking detectability probably is the most useful property we want to have in our proofs. For example, the proof of Theorem 3 makes such assumption. We argue that making this seemingly strong assumption in fact does not lose the generality of our proofs under both cost models. We say that β and α are equivalent if the two determine the same complexity class, i.e., $\mathbf{C}(\beta) = \mathbf{C}(\alpha)$. Our approach is to construct an effective operator that converts any $\beta \in \mathbf{T_2TB}$ to an equivalent locking detectable one. We state the theorem as follows.

Theorem 9. *There is an effective operator* $\Theta_L : \mathbf{T_2TB} \to \mathbf{T_2TB}$ *such that, for every* $\beta \in \mathbf{T_2TB}$, $\Theta_L(\beta)$ *is a locking detectable type-2 time bound equivalent to* β. *Moreover, if* β *is strong, then so is* $\Theta_L(\beta)$.

6 Conclusion and Futures

Type-2 computation to some extent is a better model for many real-world computations. Just to name some: real computation, real time (interactive) computation, mass database inquiries, machine learning, Web search engine, and so on, where we do not use all available information just as the OTM does not use the entire knowledge of the oracle. But as a matter of fact, type-2 complexity theory is a highly underdeveloped area mainly because a tiny difference between computation models can easily cause manifest discrepancy in the notion of computability. The situation becomes even worse when computational complexity is concerned. Even with the OTM, a widely accepted standard for type-2 computation, the way we treat the answers returned from the oracle radically shapes the outlook of the complexity theory at type-2. The answer-length cost is the most common cost model assumed in the literatures. This obviously is not the only choice (we do not read every entry returned from the Google search engine, do we?). We thus propose a reasonable alternative model called *unit cost model*. We have learned that even under the same clocking scheme, this cost model gives us a very different type-2 complexity structure. The complexity theory is much more fragile under the unit cost model.

There are apparently many questions yet to be answered along the line of classical complexity theorems. For example, is there any reasonable version of the Speedup theorem? Hierarchy theorem? What is the meaning of complete-problems at type-2? Is the classical notion of *honesty* necessarily trivial at type-2?

If yes, what should it be and what problem it is meant to fix? BFF does not fit our notion of complexity classes, but are there any intuitively feasible classes that do? Etc, etc. All these questions should be answered and we speculate that the clocking scheme together with our type-2 time bounds has provided an accessible approach to explore this long missing piece of a more general complexity theory.

References

1. Manuel Blum. A machine-independent theory of the complexity of recursive functions. *Journal of the ACM*, 14(2):322–336, 1967.
2. A. Borodin. Computational complexity and the existence of complexity gaps. *Journal of the ACM*, 19(1):158–174, 1972.
3. Walter S. Brainerd and Landweber Lawrance H. *Theory of Computation*. John Wiley & Sons, New York, 1974.
4. Robert L. Constable. The operator gap. *Journal of the ACM*, 19:175–183, 1972.
5. Robert L. Constable. Type two computational complexity. In *Proceedings of the 5^{th} ACM Symposium on the Theory of Computing*, pages 108–122, 1973.
6. Stephen A. Cook. Computability and complexity of higher type functions. In Y. N. Mpschovakis, editor, *Logic from Computer Science*, pages 51–72. Springer-Verlag, 1991.
7. J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transitions of the American Mathematics Society*, pages 285–306, May 1965.
8. Bruce M. Kapron and Stephen A. Cook. A new characterization of type 2 feasibility. *SIAM Journal on Computing*, 25:117–132, 1996.
9. L.H. Landweber and E.R. Robertson. Recursive properties of abstract complexity classes. *ACM Symposium on the Theory of Complexity*, May 1970.
10. Chung-Chih Li. Type-2 complexity theory. Ph.d. dissertation, Syracuse University, New York, 2001.
11. Chung-Chih Li. Asymptotic behaviors of type-2 algorithms and induced baire topologies. *Proceedings of the Third International Conference on Theoretical Computer Science*, pages 471–484, August 2004.
12. Chung-Chih Li and James S. Royer. On type-2 complexity classes: Preliminary report. *Proceedings of the Third International Workshop on Implicit Computational Complexity*, pages 123–138, May 2001.
13. E. McCreight and A. R. Meyer. Classes of computable functions defined by bounds on computation. *Proceedings of the First ACM Symposium on the Theory of Computing*, pages 79–88, 1969.
14. M.O. Rabin. Degree of difficulty of computing a function and a partial ordering of recursive sets. Technical Report 2, Hebrew University, 1960.
15. Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1967. First paperback edition published by MIT Press in 1987.
16. James Royer and John Case. *Subrecursive Programming Systems: Complexity & Succinctness*. Birkhäuser, 1994.
17. James S. Royer. Semantics vs. syntax vs. computations: Machine models of type-2 polynomial-time bounded functionals. *Journal of Computer and System Science*, 54:424–436, 1997.
18. Anil Seth. Complexity theory of higher type functionals. Ph.d. dissertation, University of Bombay, 1994.
19. Paul Young. Easy construction in complexity theory: Gap and speed-up theorems. *Proceedings of the American Mathematical Society*, 37(2):555–563, February 1973.

On the Calculating Power of Laplace's Demon

John Longley

LFCS, School of Informatics, University of Edinburgh, UK.
Email: jrl@inf.ed.ac.uk

Abstract. We discuss some of the choices that arise when one tries to make the idea of physical determinism more precise. Broadly speaking, 'ontological' notions of determinism are parameterized by one's choice of mathematical ideology, whilst 'epistemological' notions of determinism are parameterized by the choice of an appropriate notion of computability. We present some simple examples to show that these choices can indeed make a difference to whether a given physical theory is 'deterministic' or not.

Keywords: Laplace's demon, physical determinism, philosophy of mathematics, notions of computability.

1 Introduction

> *Given for one instant an intelligence which could comprehend all the forces by which nature is animated and the respective situations of the beings who compose it — an intelligence sufficiently vast to submit these data to analysis — it would embrace in the same formula the movements of the greatest bodies and those of the lightest atom; for it, nothing would be uncertain and the future, as the past, would be present to its eyes.* [19, chapter II]

In these now famous words, Laplace articulated his vision of an orderly, mechanistic universe whose history unfolds like clockwork according to fixed deterministic laws. In essence, this vision may be traced back at least to Democritus, and it has remained enormously influential down to the present time (see e.g. [25] for a modern incarnation). Philosophers still argue over whether or not the issue of physical determinism has any bearing on the problem of free will (see e.g. [6]). It is therefore very natural to ask how well Laplace's claim holds up in the light of our present-day understanding of science and mathematics.

Broadly speaking, the answer to this question will depend on two kinds of considerations. Firstly, it clearly depends on what the 'laws of physics' actually are: for example, some proposed formulations of quantum theory appear to allow for some kind of indeterminacy at the interface between 'quantum' and 'classical' levels, whilst others do not. Issues of this kind are clearly a matter for the physicists. Secondly, and less obviously, one can ask how exactly the Laplacian concept of determinism is to be made precise. It is this latter question that I wish to consider in this paper.

I will argue, drawing on ideas from computability theory and mathematical logic, that there are a whole range of different ways in which the idea of determinism might be understood. Some of the choices involved are purely technical in nature, whilst others touch on deeper philosophical issues. I will show, moreover that these choices can sometimes radically affect whether a physical theory is 'deterministic' or not, even in the case of very simple theories.

Laplace's imagery of a hypothetical predictive 'intelligence' (nowadays known as *Laplace's demon*) provides a valuable prop for the imagination. Roughly speaking, we will be asking exactly how the instantaneous state of the universe is supposed to be 'presented' to the demon (that is, what kinds of raw facts about this state he has access to), and exactly what kinds of 'analysis' — particularly what kinds of *infinitary* operations — he is supposed to be able to perform on this data. I hope to show how these considerations can make interesting and perhaps surprising differences to the conclusions that can be drawn.

1.1 Ontological versus epistemological determinism

As a first stab, we may broadly distinguish between two ways of interpreting Laplace's claim, which we call the *ontological* and the *epistemological* interpretation. The ontological version would say that given the present state of the universe (or of some closed physical system), there is, in fact, only one possible course of history starting from this state in which the laws of physics are upheld (whether or not we have any way of knowing what that history is). By contrast, an epistemological version would say that given knowledge of the present state, there is some way of *knowing* or 'working out' how the future will unfold.

The ontological and epistemological notions of determinism may be understood with reference to the mathematical notions of *truth* and *computability* respectively. Schematically, if histories are represented by mathematical functions from *Times* to *States*, ontological determinism claims that some sentence of the form

$$\forall s : States,\ t : Times.\ \exists! h : Times \to States.\ h(t) = s \wedge Laws\_Of\_Physics(h) \quad (\dagger)$$

is *true*, where $Laws\_Of\_Physics(h)$ might say (for instance) that certain differential equations are satisfied at every point in space and time, and s specifies a boundary condition.[1] By contrast, epistemological determinism would claim that there is some kind of computable operation

$$\Phi : States \to (Times \to States)$$

[1] Some care is needed over the status of the sentence (†). If it is understood purely as a mathematical statement about some *model* of physics, it does not succeed in saying anything about how the actual universe behaves. On the other hand, if it is understood as referring directly to physically real entities, it does not say what we want: since there is only one *actual* course of history, the uniqueness assertion becomes vacuous — we would really like h to range over all *mathematically* possible history functions. Our proposed solution is to understand (†) as a mathematical assertion, and to supplement it with the following statement, in which T, T' range

such that for any state s, the history $\Phi(s)$ correctly represents the evolution of the physical system starting from s. In connection with the ontological claim, one might imagine a demon so powerful that he can magically survey *all possible history functions* and pick out the one with the required property; for the epistemological version, one might imagine a more modest demon who makes predictions by following some kind of algorithmic procedure, perhaps involving idealized 'measurements' on the state s.

The *a priori* possibility that the behaviour of physical systems might be mathematically deterministic but not algorithmically computable in nature has been highlighted by Penrose [22, 23], who has furthermore suggested that physical laws of this kind might play an essential role in the science of consciousness.

1.2 Drawing finer distinctions

These two varieties of determinism presuppose, respectively, a notion of *mathematical truth* and a notion of *computability*. Discussions of determinism often implicitly assume that these are both unambiguous and unproblematic notions: surely in mathematics the notion of truth is absolute, and Church and Turing have provided us with the definitive notion of computability. However, an acquaintance with mathematical logic and computability theory would tend to suggest that things are not quite so simple.

On the one hand, the concept of mathematical truth certainly touches on deep philosophical issues. And since the idea of ontological determinism is itself of such philosophical interest, it is surely natural to ask what philosophical presuppositions this idea rests on. What metaphysical status does an assertion such as (†) about 'possible histories' really have? Different philosophies of mathematics would answer this question in very different ways. For example:[2]

- *Platonism* (the 'classical' view of mathematics) maintains that mathematical sentences like (†) do indeed have a definite truth-value independently of whether we can know what it is. This view involves a metaphysical commitment to a notion of truth not grounded in empirical or sensory experience.
- *Semi-constructivism* would subscribe only to a much more limited version of this idea: if $\phi(n)$ has a definite truth-value for each $n \in \mathbb{N}$, it is accepted that $\exists n.\phi(n)$ has a definite truth value. (This idea is embodied in the so-called 'Limited Principle of Omniscience'.) There is still a metaphysical commitment here, though it is more moderate than in the case of Platonism.

---

over actual points in time and $S(T)$ is the actual state of the universe at time T.

$$\forall h : \mathit{Times} \to \mathit{States}.\ (\forall t, T.\ \mathit{Models}(t, T) \Rightarrow \mathit{Models}(h(t), S(T)) \ \Rightarrow\ \mathit{Laws\_Of\_Physics}(h)$$

This at least isolates the *mathematical* content of determinism in our assertion (†).

[2] For a discussion of the main philosophical issues at stake (from an intuitionist perspective), see [7, chapter 7].

– *Constructivism* regards mathematical statements purely as expressions of what we can actually do or calculate; there is no reference to any independent notion of truth. Nothing entitles us to say $\exists x.\phi(x)$ other than *knowing* some suitable value of x. Under a constructivist reading of (†), our ontological notion of determinism might closely resemble an epistemological one.

Many further subdivisions and intermediate positions might be mentioned. We thus obtain a whole spectrum of interpretations of 'ontological determinism', involving varying levels of metaphysical commitment.

Regarding the question of *computability*, the familiar Turing notion is indeed generally accepted as the definitive notion for computations involving *natural numbers* or other finite entities that can be effectively coded by them. But what do we mean by computation where *infinite* entities are concerned, such as real numbers or continuous functions on the reals? Typically, many different answers to this question can be given, leading to several plausible but distinct notions of computability for such entities. Many of the issues, and possible choices, are discussed in [20], which focuses on computability at higher types over $\mathbb{N}$, an arena of particular interest within computer science. In other settings (e.g. higher types over $\mathbb{R}$, or the classical spaces of functional analysis), several computability notions have been proposed and studied (see e.g. [32, 36]), but we are still some way from seeing the overall picture. The crucial point here is that, in our characterization of epistemological determinism, the demand for a 'computable' operation Φ might be interpreted in many different ways.

It will be clear by now that there is considerable overlap between the ontological and epistemological notions. On the one hand, ontological determinism from a constructive standpoint can often be closely related to epistemological determinism based on some 'finitary' computability notion. Indeed, there is an extensive body of metamathematical work on using computability notions to model constructive formal systems (see e.g. [35]). On the other hand, ontological determinism from a non-constructive standpoint can often be related to epistemological determinism based on 'infinitary' computability notions. For instance, what 'exists' from a semi-constructivist standpoint is closely related to what is 'computable' in the presence of the existential quantifier $\exists : (\mathbb{N} \to \mathbb{B}) \to \mathbb{B}$ where $\mathbb{B} = \{\mathit{true}, \mathit{false}\}$ (see [15]) — that is, what would be visible to a demon who could 'see all the natural numbers at once'.[3] In the presence of even more powerful infinitary operations, computability would approach classical notions of truth (see e.g. [29]).

All these considerations might seem rather arcane, and the suggestion that any of them might be of relevance to real physical theories might at first seem rather far-fetched. My purpose in the remainder of this paper is to present a selection of examples to show that these considerations really do make a difference to the question of determinism, even for simple physical theories.

[3] It is interesting to note that critiques of the Limited Principle of Omniscience sometimes take the form that it comes precariously close to presupposing the existence of such a 'demon'. See for example Wittgenstein [38, §352].

1.3 Infinities in the physical universe?

The questions of mathematical ideology discussed above, as well as the computability considerations we have mentioned, are closely bound up with the mathematical idea of 'the infinite'. Consequently, many of the issues we are discussing would trivialize if, in fact, the universe could be completely modelled by some discrete, finite mathematical structure. A brief discussion of this possibility is therefore in order; see also [24, chapters 3,33], [30, II.D], and [5].

The vast majority of successful physical theories in use today rely heavily on the calculus, which presupposes the mathematical idea of an infinitely subdivisible *continuum*. However, we do not know whether genuine continua — or infinities of any kind — actually occur anywhere in the physical world, and physicists have sometimes expressed unease at the seeming ontological extravagance of this assumption (see e.g. [11, pp. 57–8]). There have been several interesting attempts to put physics onto a more 'discrete' footing, but it would seem that this is not so easy to do, and most leading-edge physical theories still make extensive use of the mathematical continuum.

In view of this, it seems to us that it is interesting to explore the implications of the supposition that physical continua do exist. Even if, in the end, such an investigation served only to convince us of the implausibility of this supposition, this would still be a valuable outcome. Of course, investigations of this kind are perhaps of academic interest for physical theories that are already known not to hold 'all the way down', but they acquire an added dimension of significance for theories which are proposed as candidates for an 'ultimate' description of physical reality.

In this short article, my intention is not to map out a coherent programme of research, nor to consider current leading-edge physical theories in detail, but merely to collect together a few observations, based on known mathematical results, in order to illustrate the *kinds* of issues that can arise, and thus perhaps to indicate that the general area merits further exploration.

2 Determinism and the constructive continuum

First, I would like to explore some implications of adopting a strictly constructivist mathematical stance *à la* Bishop [3], by focusing on a childishly simple problem in Newtonian physics. A particle in one dimension is initially at rest, and no forces act on it: what happens to it?

Suppose we model this problem using a physical theory such as the following:[4]

$$States = \mathbb{R}, \quad Times = \mathbb{R}, \quad Laws\_Of\_Physics(h) \equiv \forall t.\, \dot{h}(t) = 0,$$
$$s = c \text{ (a constant)}, \quad t_0 = 0$$

[4] The standard Newtonian formulation of this problem would of course involve the second time derivative, but even the simplified version we give here will serve to illustrate our point.

The existence of suitable solutions, and even their computability (given c) is of course unproblematic; the issue is with the uniqueness part of (†). The problem is that, assuming $h(0) = c$ and $\mathit{Laws\_Of\_Physics}(h)$, we cannot *constructively* conclude that $h(t) = c$ for all t. This is because the Fundamental Theorem of the Calculus, saying that any continuous function has a unique antiderivative modulo an added constant, is not constructively valid.

One can understand the problem better by considering an alternative function h satisfying the above conditions within the universe of effective mathematics (which provides one possible model for constructive mathematics — see e.g. [14]). Such a solution is easy to construct using the well-known *Kleene tree* [15, §LII]. Looked at from a *classical* perspective, h is at most points a locally constant function, but one whose value jumps around, with discontinuities at a set of points homeomorphic to Cantor space. However, the values of t at which h is discontinuous are all non-computable reals — so seen from within the effective universe, h is continuous and has derivative 0 everywhere! Such a pathological history function is clearly ludicrous in physical terms, but the point is to ask how *precisely* we intend our theory to rule out such a possibility.

One might suppose that the problem could be fixed simply by strengthening our *Laws_Of_Physics* predicate in some way. But the same problem will beset *any* proposed predicate $\mathit{Laws\_Of\_Physics}(h)$ which is *local* in character:[5] that is, any predicate of the form $\forall t.\, L(h,t)$, where L satisfies

$$\forall h, h', t.\ (h, h' \text{ agree on some neighbourhood of } t) \ \Rightarrow\ (L(h,t) \Leftrightarrow L(h',t))$$

This is because the pathological solution above is locally just fine: at every point, it agrees locally with some globally constant function which we *do* want to allow.

There are, of course, many possible responses to this problem, e.g.:

1. Accept the non-determinism. (This would be silly: any theory that fails to predict that everyday objects do not jump about in this erratic fashion must be judged sorely deficient.)
2. Abandon the continuous model of time.
3. Abandon the locality principle. For example, we might postulate an additional physical law saying that h had to be *uniformly* continuous on any compact interval, a non-local property.[6]
4. Abandon strict constructivism, and admit some additional mathematical principle that allows us to deduce the uniqueness of h. One minimal such principle would be a weak (double-negation sanitized) version of König's Lemma for binary trees (cf. [31, Chapter IV]), which can certainly be justified on the philosophical premises of semi-constructivism.[7]

[5] The idea that the laws of physics ought to be local in character seems quite deeply ingrained in the informal conception of a 'mechanistic' universe. For an illuminating discussion of the 'locality principle' from a physicist's perspective, see [11, Chapter 2].

[6] This is in fact the notion of continuity adopted by Bishop-style constructivists in order to obtain a viable theory.

[7] Or, for that matter, on the premises of Brouwerian intuitionism.

One impression that emerges from this situation is that there is some kind of trade-off between the strength of the *physical* assumptions (as in 3) and that of the *mathematical* (or metamathematical) assumptions (as in 4) needed to conclude determinism. It does not pay to be too parsimonious on both fronts.

3 Finite dimensional systems

From a semi-constructivist or Platonist standpoint, it seems that for physical systems with $States = \mathbb{R}^n$ governed by ordinary differential equations (e.g. n-body problems), both ontological and epistemological determinism are relatively unproblematic. Indeed, there is a now well-established canonical computability notion for total functions $f : \mathbb{R}^{n+1} \to \mathbb{R}^n$ (first introduced by Lacombe [17] and Grzegorczyk [13]), and this notion appears to suffice for predicting behaviour in all cases of physical interest.

The essence of this notion of computability is that one can compute the output $f(\mathbf{x})$ to within any desired $\epsilon > 0$ if one knows the input $\mathbf{x}$ to within some $\delta > 0$ dependent on ϵ and $\mathbf{x}$. (In particular, all computable functions are continuous). We can therefore think in terms of a demon equipped with an infinite sequence of measuring devices of increasing resolving power, who is able to make affirmative observations on the state corresponding to *open subsets* of $\mathbb{R}^n$, and follows some effective procedure for processing the results of such observations. The set of all observationally affirmable properties that hold at time 1 (say) can then be computably determined from the set of all affirmable properties at time 0.

However, even in this relatively unproblematic setting, a couple of caveats need to be made. The first concerns physical systems whose behaviour exhibits singularities — whose state at time 1 can be discontinuous in the state at time 0. One example (described in [22, chapter 5]) is the collision problem for three elastic billiard balls: the classical physical theory does not determine what happens if the three balls collide at *exactly* the same point in time. At the very least, one should here modify one's claim of determinism to a conditional statement involving a computable *partial* function $\mathbb{R}^{n+1} \rightharpoonup \mathbb{R}^n$. It is fair to add here that the relationships between candidate definitions of computability for partial functions on the reals remain to be fully clarified.

A second caveat is a rather technical one concerning the notion of effective procedure involved: the demon had better not be following a program in a deterministic, *sequential* programming language. More specifically, imagine that once the demon has decided to test for some property like '$x > \frac{1}{2}$', he is committed to obtaining an answer before he can proceed with anything else. If indeed $x > \frac{1}{2}$, the demon will eventually discover this by means of a sufficiently precise measurement; but if $x = \frac{1}{2}$, he will be side-tracked into making measurements of increasing precision forever.[8] The problem can be overcome if the demon is allowed to use a 'parallel conditional' operator of the kind used in [9].

[8] This is reminiscent of the fact that even functions as simple as addition are not computable in the sequential version of Real PCF [10].

4 Infinite dimensional systems

For physical systems with infinite state spaces (e.g. continuously varying fields governed by partial differential equations), the picture gets considerably more interesting. As an example, we will consider the *wave equation* for a scalar field $\psi(\mathbf{x}, t)$ in three space dimensions: $\nabla^2\psi = \ddot{\psi}$. We take the set *States* to be some space of functions $s : \mathbb{R}^3 \to \mathbb{R}^2$: the first component of $s(\mathbf{x})$ gives the instantaneous value of ψ at $\mathbf{x}$, while the second component gives its time-derivative $\dot{\psi}$. We are therefore interested in the computability or otherwise of an operation of type $[\mathbb{R}^3 \to \mathbb{R}^2] \to [\mathbb{R}^4 \to \mathbb{R}^2]$.

The theory of second-order computability over the reals is far from trivial, and there are various choices available to us. First, let us briefly consider one choice that will *not* work well here: namely, Kleene-style higher-type computability in the spirit of [15]. In this approach, functions are treated as *oracles*, so that the only way to extract information about a function $[\mathbb{R}^3 \to \mathbb{R}^2]$ would be to apply it to particular values $\mathbf{x}$. Suppose then that we consider the style of computation envisaged in Section 3 augmented with such oracle calls. This would mean that a computation of the value of $\psi(\mathbf{0}, 1)$ (say) to within some $\epsilon > 0$ would only be able to interrogate the initial state s at finitely many points before returning an answer. It is easy to see that this is not enough, since the true value of $\psi(\mathbf{0}, 1)$ cannot be determined even up to ϵ from such a finite sample.

Our demon must therefore somehow be able to observe properties of s that pertain to whole *regions* of $\mathbb{R}^3$, rather than simply its value at particular points. One could of course endow the demon with the ability to survey regions by adding a suitable quantifier such as $\exists_{\mathbb{R}}$ to its repertoire, but this would seem like overkill. It seems more interesting to regard certain kinds of observations over regions as 'atomic', and ask whether some finitary style of computation involving these observations will suffice.

4.1 The Pour-El/Richards approach

A more suitable approach involves the theory of computability for Banach spaces developed by Pour-El and Richards [27, 28], who have moreover made a particular study of the wave equation in their setting. In this theory, one first axiomatizes the notion of a *computability structure* on a Banach space X, consisting of a set of *computable sequences* $\mathbb{N} \to X$ with certain closure properties. An element $x \in X$ is considered computable if $x, x, x, \ldots$ is a computable sequence.

Although computability structures are defined axiomatically, it turns out for quite general reasons that, for all naturally occurring spaces X, there is only one reasonable choice of computability structure. This gives the theory the attractive feature that the notion of computability is *intrinsic* for such spaces. However, this computability notion is sensitive to the choice of *norm* on the space: thus, for instance, there exist continuous functions $s : \mathbb{R}^3 \to \mathbb{R}^2$ with compact support which are non-computable if regarded as elements of $C(\mathbb{R}^3, \mathbb{R}^2)$ (the space of bounded continuous functions with the supremum norm), but computable if regarded as elements of $L_2(\mathbb{R}^3, \mathbb{R}^2)$. Insofar as the choice of norm is up to us

rather than given 'by nature', we therefore have a range of possible computability notions even at the level of individual states.[9]

Next, consider linear maps $T : X \to Y$ between Banach spaces with computability structures. A major theorem of [28] asserts that, under modest conditions, T maps computable sequences to computable sequences iff T is *bounded*. In fact it is very reasonable to regard such maps T as the *computable maps* $X \to Y$, as is shown by results of [32, 33]. Specifically, the set of computable elements of a Banach space with computability structure can naturally be endowed with a *computable representation* in the sense of Weihrauch [36] (or alternatively with an effective domain representation in the sense of Scott and Ershov). It can then be shown that the linear maps that satisfy the Pour-El/Richards conditions are precisely those whose action on computable elements is computable in the Weihrauch sense (or alternatively in the sense of effective domain theory).

In the case of the supremum norm, these notions have especially good credentials. A natural and robust notion of second order computability over the reals has been studied in [2], where several non-trivially equivalent characterizations are given. Here one is able to perform computations on first order functions f by making use of 'compact-open observations' about them: '$f(x) \in U$ for all $x \in K$', where U is open and K is compact. This is one way to make precise the idea of 'atomic observations over regions' mentioned earlier. Now let X be the Banach space $C(\mathbb{I}^3, \mathbb{R}^2)$ (we here replace $\mathbb{R}$ by the unit interval $\mathbb{I}$ to avoid some annoying technical complications). Then the computable maps $X \to X$ turn out to be precisely the linear maps that are computable in the sense of [2].

Given the good credentials of this notion, it is perhaps all the more surprising that the solution operators for the wave equation are *not* computable in this sense. For simplicity, let $X = C(\mathbb{R}^3, \mathbb{R}^2)$, and let $T : X \to X$ be the operator that maps a wave state at time 0 to the wave state at time 1 that results from it. It was shown in [26] that $T : X \to X$ is not computable — in fact, that there is even a computable state s such that the first component of $T(s)(\mathbf{0})$ is a non-computable real. An important aspect of the example given in [26] is that though the first component of s is computable, its space-derivative is non-computable.

What about the computability notions arising from spaces with other norms? One important example considered in [28] is the *energy norm* $\| - \|_E$ given by:

$$\| \langle s_0, s_1 \rangle \|_E^2 \;=\; \iiint_{\mathbb{R}^3} |\nabla s_0|^2 + s_1^2$$

In physical terms, this corresponds to the total amount of energy present in a wave state, and it satisfies the axioms required for a norm. The fact that energy is conserved as the wave propagates says that the operator T is bounded by 1, and is therefore a computable map. The wave equation thus provides a good example of a physical theory for which the question of epistemological determinism is sensitive to the computability notion adopted.

[9] Interestingly, in the case of *quantum mechanical* systems this issue seems not to arise, since the physical theory itself makes essential use of an inner product which gives us a preferred choice of norm. See [1].

The precise conditions under which the solution operator is or is not computable have been investigated in some detail by Weihrauch and Zhong [37]. As these authors point out, different choices of norm (or topology) correspond to different notions of 'possible affirmative observation' on states, and it is a matter of physics to investigate whether any of these choices correspond to what is observable by means of 'idealized physical measurements' of some kind.

Let us suppose that the class of 'physically realizable' affirmative observations (in some idealized sense) corresponded precisely to the topology induced by some norm $\| - \|_{obs}$. The effective domain of closed balls for this norm (say) would then constitute an *epistemic* (i.e. information-theoretic) model of the physical system, and states (considered as functions $\mathbb{R}^3 \to \mathbb{R}^2$) would appear as maximal consistent sets of affirmable observations. However, a more minimalist ontology might then reject these 'ideal' elements in favour of a purely finitary theory of possible observations, in which idealized mathematical points would be abandoned in favour of a theory of intervals or regions, somewhat in the spirit of [8]. Indeed, one can conceive that this might offer one way of putting the whole physical theory on a more 'constructive' or 'finitary' footing as suggested in [30].

If it turned out that T were computable with respect to $\| - \|_{obs}$, we would be in the very pleasing situation that the set of potentially observable facts at time 1 was computably determined by the set of such facts at time 0 — and this would also systematically explain why the ontologically underlying non-computable values of the wave function ψ (on computable arguments, for computable initial states) can never surface at the level of what is measurable. However, if (as the author suspects is more likely), T were not computable with respect to $\| - \|_{obs}$ we would face a dilemma similar to the one encountered at the end of Section 2: either certain kinds of state information must be considered as ontologically real though they are not observable even by approximation, or the future state cannot be effectively predicted from the present state.

Conclusion and acknowledgements

In this short paper we have barely scratched the surface of our proposed area of investigation, and many of the issues we have highlighted demand much more detailed discussion than we have provided. Nevertheless, we hope that our selection of examples has persuaded the reader that the task of elucidating different notions of determinism is interesting and worthwhile, and that this may prove to be a fruitful area of interaction between physics and computability theory.

I am grateful to Gordon Plotkin, Matthias Schröder, Alex Simpson and Alan Smaill for helpful comments and discussions, and to the anonymous referees for valuable suggestions and pointers. Barry Cooper's papers [4, 5] provided much of the initial stimulus for my reflections, as well as a helpful guide to the existing literature on determinism and computability.

APPENDIX: A thought experiment concerning non-computability in the physical world

We here present a simple thought experiment in connection with the question of whether non-computability ever manifests itself in the physical world. Though this stands somewhat apart from our present inquiry into possible interpretations of determinism, it is clearly relevant to whether the laws of physics *are* deterministic in a strong epistemological sense, and provides another example of the application of ideas from computability theory to such questions.

It is tempting to imagine that — because non-computability is an infinitary property that cannot be detected from any finite sample of data, and because every continuous function on $[0, 1]$ can be approximated arbitrarily closely by a computable one, and so on — it cannot possibly make any kind of observable difference whether the laws of physics are computable or not. The following experiment suggests one possible sense in which this is not the case.

The experiment involves the *Lacombe tree* [18], a variation on the Kleene tree which deserves to be better known. The Lacombe tree is a computable binary tree T (that is, a decidable prefix-closed set of finite sequences over $\{0, 1\}$) such that

- every *computable* infinite sequence over $\{0, 1\}$ eventually exits from T; but
- classically, the set of infinite sequences that eventually exit from T has some measure $m < \frac{1}{2}$ within $\{0, 1\}^{\mathbb{N}}$.

One may therefore imagine setting up 100 instances of the following apparatus. Some physical device not known to have computable behaviour (perhaps a Geiger counter) is set up to generate a stream of binary digits, which are fed into a computer. At each stage, the computer tests whether the finite sequence received so far is a node of T. If, at some stage, the sequence is found to have exited from T, a light is turned on.

After allowing the experiment to run for some finite period of time, we return and count how many of the lights have come on. If the answer is fewer than $m.100$, we cannot conclude anything — perhaps we have just not waited long enough for the other sequences to exit yet. However, if the answer is 95 (say), this provides good evidence that the infinite sequences generated by our devices are *not* being drawn at random from the classical set $\{0, 1\}^{\mathbb{N}}$, but from some more restricted set, perhaps the set of computable sequences. If one supposes that the choice is between 'all sequences' and 'the computable sequences', we thus have a probabilistic semi-decision test for whether the behaviour of our physical devices is computable. (Interesting further questions arise if we also consider the possibility of other sets, such as the set of *hyperarithmetic* sequences.)

We hasten to add that this experiment would be of no use in practice, in view of the hyper-astronomical length of time one would have to wait for a significant number of lights to come on.

References

1. Baez, J.C., *Recursivity in quantum mechanics*, Trans. Amer. Math. Soc. **208(1)** (1983), 339–350.
2. Bauer, A., M.H. Escardó, and A. Simpson, *Comparing functional paradigms for exact real-number computation*, in Proc. ICALP 2002, Springer LNCS **2380** (2002), 488–500.
3. Bishop, E., and D. Bridges, "Constructive Analysis", Springer-Verlag (1985). Revised and expanded version of E. Bishop, "Foundations of Constructive Analysis", McGraw-Hill (1967).
4. Cooper, S.B., *Clockwork or Turing U/universe? — Remarks on causal determinism and computability*, in S.B. Cooper and J.K. Truss, editors, "Models and Computability", Cambridge University Press (1999), 63–116.
5. Cooper, S.B., and P. Odifreddi, *Incomputability in nature*, in S.B. Cooper and S.S. Goncharov, editors, "Computability and Models: Perspectives East and West", Kluwer/Plenum (2003), 137–160.
6. Dennett, D.C., "Freedom evolves", Penguin Books (2003).
7. Dummett, M., "Elements of intuitionism", Clarendon Press, Oxford (1977).
8. Edalat, A., *Domain theory and integration*, Theoretical Computer Science **151** (1995), 163–193.
9. Escardó, M.H., *PCF extended with real numbers*, Theoretical Computer Science **162(1)** (1996), 79–115.
10. Escardó, M.H., M. Hofmann and T. Streicher, "On the non-sequential nature of the interval-domain model of exact real-number computation." Mathematical Structures in Computer Science **14(6)** (2004), 803–814.
11. Feynman, R., "The character of physical law", Penguin Books (1992).
12. Geroch, R., and J.B. Hartle, *Computability and physical theories*, Foundations of Physics **16** (1986), 533–550.
13. Grzegorczyk, A., *On the definitions of computable real continuous functions*, Fundamenta Mathematicae **44** (1957), 61–71.
14. Hyland, J.M.E., *The effective topos*, in Troelstra, A.S. and D. van Dalen, editors, "The L.E.J. Brouwer Centenary Symposium", NorthHolland (1982), 165–216.
15. Kleene, S.C., *Recursive functionals and quantifiers of finite types I*, Transactions of the American Mathematical Society **91** (1959), 1–52.
16. Kreisel, G., *A notion of mechanistic theory*, Synthese **29** (1974), 11–26.
17. Lacombe, D., *Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles I, II, III*, Comptes Rendus de l'Académie des Sciences **240** (1955), 2478–2480, and **241** (1955), 13–14, 151–153.
18. Lacombe, D., *Remarques sur les opérateurs récursifs et sur les fonctions récursives d'une variable réelle*, Comptes Rendus de l'Académie des Sciences **241** (1955), 151–153.
19. Laplace, P. S. de, *Essai philosophique sur les probabilités* (1819). English translation by F.W. Truscott and F.L. Emory, Dover, New York (1951).
20. Longley, J.R., *Notions of computability at higher types I*, in R. Cori, A. Razborov, S.Todorčević and C. Wood, editors, "Logic Colloquium 2000: Proceedings of the ASL meeting held in Paris", Lecture Notes in Logic 200, ASL (2005), 32–142.
21. Odifreddi, P.G., *Kreisel's Church*, in P.G. Odifreddi, editor, "Kreiseliana: About and Around Georg Kreisel", A.K. Peters (1996).
22. Penrose, R., "The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics", Oxford University Press (1989).

23. Penrose, R., "Shadows of the Mind: A Search for the Missing Science of Consciousness", Oxford University Press (1994).
24. Penrose, R., "The Road to Reality", Jonathan Cape (2004).
25. Poundstone, W., "The Recursive Universe: Cosmic Complexity and the Limits of Scientific Knowledge", New York: Morrow (1985).
26. Pour-El, M.B., and J.I. Richards, *The wave equation with computable initial data such that its unique solution is not computable*, Advances in Mathematics **39** (1981), 215–239.
27. Pour-El, M.B., and J.I. Richards, *Noncomputability in analysis and physics*, Advances in Mathematics **48** (1983), 44–74.
28. Pour-El, M.B., and J.I. Richards, "Computability in Analysis and Physics", Springer-Verlag (1989).
29. Sacks, G.E., "Higher Recursion Theory", Perspectives in Mathematical Logic, Springer-Verlag (1990).
30. Shipman, J., *Aspects of computability in physics*, in Proc. 1992 Workshop on Physics and Computation, IEEE (1993).
31. Simpson, S.G., "Subsystems of second order arithmetic", Perspectives in Mathematical Logic, Springer (1998).
32. Stoltenberg-Hansen, V., and J.V. Tucker, *Concrete models of computation for topological algebras*, Theoretical Computer Science **219** (1999), 347–378.
33. Stoltenberg-Hansen, V., and J.V. Tucker, *Computable and continuous partial homomorphisms on metric partial algebras*, Bulletin of Symbolic Logic **9(3)** (2003), 299–334.
34. Svozil, K., *The Church-Turing thesis as a guiding principle for physics*, in C.S. Calude, J. Casti and M.J. Dinnen, editors, "Unconventional Models of Computation", Springer (1998), 371–385.
35. Troelstra, A.S., "Metamathematical Investigation of Intuitionistic Arithmetic and Analysis," Lecture Notes in Mathematics **344**, Springer-Verlag, 1973.
36. Weihrauch, K., "Computable Analysis", Springer (2000).
37. Weihrauch, K., and N. Zhong, *Is wave propagation computable or can wave computers beat the Turing machine?*, Proc. London Mathematical Society **85(2)** (2002), 312-332.
38. Wittgenstein, L., "Philosophical Investigations", Blackwell (1953). Third edition, with English translation by G.E.M. Anscombe, Blackwell (2001).

Scaled Dimension of Individual Strings

María López-Valdés

Departamento de Informática e Ing. de Sistemas, María de Luna 1, Universidad de Zaragoza. 50018 Zaragoza, SPAIN. marlopez@unizar.es *

Abstract. We define a new discrete version of scaled dimension and we find connections between the scaled dimension of a string and its Kolmogorov complexity and predictability. We give a new characterization of constructive scaled dimension by Kolmogorov complexity, and prove a new result about scaled dimension and prediction.

1 Introduction

Effective fractal dimension, defined by Lutz (2003) [10], allows us to study the fractal structure of many sets of interest in computational complexity. Furthermore, many connections have been found between effective fractal dimension and other topics in computational complexity like Kolmogorov complexity [12], [11] and prediction [2], [3].

In 2004, Hitchcock et al [5] introduced the scaled dimension as a natural hierarchy of "rescaled" effective fractal dimensions. Their main objective was to overcome some limitations of the effective fractal dimension for investigating complexity classes. For example classes such as the Boolean circuit-size complexity classes SIZE($2^{\alpha n}$) and SIZE($2^{n^{\alpha}}$) have trivial dimensions, and the definition of scaled dimension made possible to quantify the difference between those classes. Connections between Kolmogorov Complexity and scaled dimension were found in [6].

The definition of effective fractal dimension is based on a characterization of the classical Hausdorff dimension in the Cantor space $\mathbf{C}$ in terms of gales (s-gales). Intuitively, we regard an s-gale d as an strategy for betting on the successive bits of a sequence $S \in \mathbf{C}$ and the parameter s gives us an idea about the fairness on the gambling game. Scaled dimension is defined using scaled gales (s^g-gales), intuitively, d is an strategy for betting on a sequence but the fairness on the gambling game depends on the s and on the scale g.

In [11], Lutz used supertermgales, which are supergale-like functions that bet on the terminations of (finite, binary) strings as well on their successive bits, to define a discrete version of constructive dimension (an special case of effective fractal dimension). Lutz then characterized the dimension of a finite string in terms of its Kolmogorov complexity. We generalize those results by defining a new discrete version of constructive scaled dimension (section 3).

* This research was supported by Spanish Government MEC project TIN2005-08832-C03-02

The main result of this section states that the scaled dimension of an infinite sequence is characterized by the scaled dimension of its prefixes (Theorem 4). As a consequence, when we obtain characterizations of the scaled dimension of individual strings in terms of Kolmogorov complexity or prediction (section 4), we can obtain results in constructive scaled dimension, just by applying the results to the prefixes of a sequence.

With this method, we obtain a new characterization of the i^{th} -order scaled constructive dimension in terms of Kolmogorov Complexity extending the results in [6].

Finally, we define the concept of termpredictor by adding the ability to predict the end of an unknown finite string to the standard on-line prediction algorithms. That is, a termpredictor guesses the next character as well as the termination point of a finite string.

We show that the scaled constructive dimension of sets of sequences can be bounded in terms of the log-loss of constructive termpredictors. This extends partially the characterization that Hitchcock obtained in [3] for resource-bounded dimension to the cases of scaled and constructive dimension.

2 Preliminaries

A string is a finite, binary string $w \in \{0,1\}^*$. We write $|w|$ for the length of a string and λ for the empty string. The Cantor space $\mathbf{C}$ is the set of all infinite binary sequences. If $w \in \{0,1\}^*$ and $x \in \{0,1\}^* \cup \mathbf{C}$, $w \sqsubseteq x$ means that w is a prefix of x. For $0 \leq i \leq j$, we write $x[i \dots j]$ for the string consisting of the i-th through the j-th bits of x.
The set of all terminated binary strings and prefixes thereof is the set

$$\mathcal{T} = \{0,1\}^* \cup \{0,1\}^*\square,$$

where we use the symbol $\square$ to mark the end of a string.

Definition 1. *Let $f : D \to \mathbb{R}$ be a function where D is some discrete domain such as $\mathbb{N}, \{0,1\}^*, \mathcal{T}$, etc.*

1. *f is computable if there is a computable function $\hat{f} : D \times \mathbb{N} \to \mathbb{Q}$ such that for all $(w,n) \in D \times \mathbb{N}$, $|\hat{f}(w,n) - f(w)| \leq 2^{-n}$.*
2. *f is lower semicomputable if there is a computable function $\hat{f} : D \times \mathbb{N} \to \mathbb{Q}$ such that*
   (a) for all $(w,n) \in D \times \mathbb{N}$, $\hat{f}(w,n) \leq \hat{f}(w,n+1) < f(w)$, and
   (b) for all $w \in D$, $\lim_{n\to\infty} \hat{f}(w,n) = f(w)$.

Definition 2. *1. A subprobability measure on $\{0,1\}^*$ is a function $p : \{0,1\}^* \to [0,1]$ such that*

$$\sum_{w \in \{0,1\}^*} p(w) \leq 1.$$

2. *A subprobability measure on* $\{0,1\}^*$ *is constructive if it is lower semicomputable.*
3. *A subprobability measure* p *on* $\{0,1\}^*$ *is optimal constructive if for every constructive subprobability measure* p' *there is a real constant* $\alpha > 0$ *such that, for all* $w \in \{0,1\}^*$, $p(w) > \alpha p'(w)$.

Theorem 1. *(Levin [13]) There exists an optimal constructive subprobability measure* $\mathbf{m}$ *on* $\{0,1\}^*$.

The following theorem is the well-know characterization by Levin [7], [8] and Chaitin [1] of Kolmogorov complexity in terms of $\mathbf{m}$. Further details may be found in [9].

Theorem 2. *There is a constant* $c \in \mathbb{N}$ *such that for all* $w \in \{0,1\}^*$,

$$\left| K(w) - \log \frac{1}{\mathbf{m}(w)} \right| \leq c.$$

Definition 3. *A scale is a continuous function* $g : H \times [0,\infty) \to \mathbb{R}$ *with the following properties.*

1. $H = (a,\infty)$ *for some* $a \in \mathbb{R} \cup \{-\infty\}$.
2. $g(m,1) = m$ *for all* $m \in H$.
3. $g(m,0) = g(m',0) \geq 0$ *for all* $m, m' \in H$.
4. *For every sufficiently large* $m \in H$, *the function* $s \mapsto g(m,s)$ *is nonnegative and strictly increasing.*
5. *For all* $s' > s \geq 0$, $\lim_{m\to\infty}[g(m,s') - g(m,s)] = \infty$.

For each scale $g : H \times [0,\infty) \to \mathbb{R}$, we define $\Delta g : H \times [0,\infty) \to \mathbb{R}$ by

$$\Delta g(m,s) = g(m+1,s) - g(m,s).$$

Definition 4. *A smooth scale is a computable scale function* $g : H \times [0,\infty) \to \mathbb{R}$ *such that verifies*

1. g *is differentiable in the second coordinate and* $\frac{\partial g}{\partial s}(m,\cdot)$ *are strictly increasing for all* $m \in H$.
2. $\frac{\partial g}{\partial s}(m,0) \to \infty$ *as* $m \to \infty$.
3. $\triangle g(m,s') - \triangle g(m,s) > 0$ *for all* $m \in H,\ s' > s$.

The following important family of smooth scales is used in the definition of the i^{th}-order dimension.

Definition 5. *We define* $g_i : H_i \times [0,\infty) \to \mathbb{R}$ *by the recursion on* $i \in \mathbb{N}$ *as follows:*

$g_0(m,s) = ms$.
$g_{i+1}(m,s) = 2^{g_i(\log m, s)}$.

The domain of g_i *is of the form* $H_i = (a_i,\infty)$, *where* $a_0 = -\infty$ *and* $a_{i+1} = 2^{a_i}$.

Definition 6. *Let $g : H \times [0,\infty)$ be a scale function. Denote by $f^m : [g(m,0),\infty] \to [0,\infty)$ the inverse of $g(m,.)$, that is the function defined as $f^m(x) = y$ if $g(m,y) = x$. This function is well define since $g(m,.)$ is strictly increasing. For the family $\{g_i\}$ we denote by f_i^m the inverse of $g_i(m,.)$ and*

$$f_i^m(x) = \frac{\log(\log(.\overset{i}{.}.\log(x)..))}{\log(\log(.\overset{i}{.}.(\log(m+1))..))}.$$

3 Scaled Dimension of Individual Strings

In this section we first introduce scaled termgales and scaled supertermgales, which are a generalization of the termgales introduced by Lutz in [11]. Next, we show the existence of optimal constructive scaled supertermgales that allows us to give a universal definition of the scaled dimension of a string.

Definition 7. *For $s \in [0,\infty)$ and $g : H \times [0,\infty) \to [0,\infty)$ a scale function,*

1. *An s^g-supertermgale is a function $d_g : \mathcal{T} \to [0,\infty)$ such that*
   (a) $d_g(w) \leq 1$ for $|w| \notin H$.
   (b) For all $w \in \{0,1\}^*$ with $|w| \in H$,

$$d_g(w) \geq 2^{-\triangle g(|w|,s)}[d_g(w0) + d_g(w1) + d_g(w\square)]. \tag{1}$$

2. *An s^g-termgale is an s^g-supertermgale that satisfies (1) with equality for all $w \in \{0,1\}^*$ with $|w| \in H$.*

An s^g-termgale is a strategy for betting on the successive bits of a binary string and also on the point at which the string terminates. The fairness of the gambling game depends on the s and on the scale function g.

Remark 1. Let $g : H \times [0,\infty) \to \mathbb{R}$ be a scale, $d_g, d'_g : \mathcal{T} \to [0,\infty)$ and $s, s' \in [0,\infty)$. If

$$2^{-g(|w|,s)} d_g(w) = 2^{-g(|w|,s')} d'_g(w)$$

for all $w \in \mathcal{T}$ with $|w| \in H$, then d_g is an s^g-supertermgale (s^g-termgale) if and only if d'_g is an $s'^g - supertermgale$ (s'^g-termgale).

Thanks to this remark, a 0^g-supertermgale (termgale) determines a whole family of s^g-supertermgales (termgales).

Definition 8. *For $g : H \times [0,\infty) \to [0,\infty)$ a constructive scale function,*

1. *A g-supertermgale is a family $d_g = \{d_g^s \mid s \in [0,\infty)\}$ such that each d_g^s is an s^g-supertermgale and*

$$2^{-g(|w|,s)} d_g^s(w) = 2^{-g(|w|,s')} d_g^{s'}(w)$$

   for all $s, s' \in [0,\infty)$, $w \in \mathcal{T}, |w| \in H$.
2. *A g-termgale is a g-supertermgale where each d_g^s is an s^g-termgale for all $s \in [0,\infty)$.*

3. *A g-supertermgale d_g is constructive if d_g^0 is constructive.*
4. *A constructive g-supertermgale $\tilde{d_g}$ is optimal if for every constructive g-supertermgale d_g there is a constant $\alpha > 0$ such that for all $s \in [0, \infty)$ and $w \in \{0,1\}^*$ with $|w| \in H$,*

$$\tilde{d_g}^s(w\square) > \alpha\, d_g^s(w\square).$$

5. *The g-supertermgale induced by a subprobability measure p on $\{0,1\}^*$ is the family $d_g[p] = \{d_g^s[p] \mid s \in [0,\infty)\}$, where each $d_g^s[p]$ is defined by*

$$d_g^s[p](w) = 2^{g(|w|,s)} \sum_{\substack{x \in \{0,1\}^* \\ w \sqsubseteq x\square}} p(x)$$

for all $w \in \mathcal{T}$ with $|w| \in H$.

Theorem 3. *If $\tilde{p}$ is an optimal constructive subprobability measure on $\{0,1\}^*$ and $g : H \times [0,\infty) \to [0,\infty)$ is a constructive scale function then $d_g[\tilde{p}]$ is an optimal constructive g-supertermgale.*

Corollary 1. *For every $g : H \times [0,\infty] \to [0,\infty)$ constructive scale function, there exists an optimal constructive g-supertermgale.*

Definition 9. *Let $g : H \times [0,\infty] \to [0,\infty)$ be a scale function and $w \in \{0,1\}^*$ with $|w| \in H$. If d_g is a constructive g-supertermgale, then the g-dimension of w relative to d_g is*

$$\dim_{d_g}(w) = \inf\{s \in [0,\infty) \mid d_g^s(w\square) > 1\}.$$

The next two results prepare the definition of g-dimension of a string.

Proposition 1. *Let $g : H \times [0,\infty) \to [0,\infty)$ be an smooth scale function. If $\tilde{d_g}$ is an optimal constructive g-supertermgale and d_g is a constructive g-supertermgale, there exists $C > 0$ such that*

$$\dim_{\tilde{d_g}}(w) \leq \dim_{d_g}(w) + \frac{C}{\frac{\partial g}{\partial s}(|w|+1, \dim_{d_g}(w))}$$

for all $|w| \in \{0,1\}^*$ ($|w|$ large enough).

Corollary 2. *Let $g : H \times [0,\infty) \to [0,\infty)$ be an smooth scale function. If $\tilde{d}_{g_1}$ and $\tilde{d}_{g_2}$ are optimal constructive g-supertermgales, then there is a constant $C > 0$ such that for all $w \in \{0,1\}^*$ ($|w|$ large enough),*

$$|\dim_{\tilde{d}_{g_1}}(w) - \dim_{\tilde{d}_{g_2}}(w)| \leq \frac{C}{\frac{\partial g}{\partial s}(|w|+1, s_0)}$$

where $s_0 = \min\{\dim_{\tilde{d}_{g_1}}(w), \dim_{\tilde{d}_{g_2}}(w)\}$.

As g is an smooth scale function, $\frac{\partial g}{\partial s}(m,0) \to +\infty$ as $m \to \infty$, and Corollary 2 says that if we base our definition of g-dimension on an optimal constructive g-supertermgale $\tilde{d}_g$, then the particular choice of $\tilde{d}_g$ has negligible impact on the dimension $\dim_{\tilde{d}_g}(w)$.
We fix an optimal constructive g-supertermgale $d_{g\square}$ and define the g-dimensions of finite strings as follows.

Definition 10. *Let $g : H \times [0,\infty) \to [0,\infty)$ be an smooth scale function. The g-dimension of a string $w \in \{0,1\}^*$ with $|w| \in H$ is*

$$\dim_g(w) = \dim_{d_{g\square}}(w).$$

3.1 Scaled dimension of strings and sequences

Resource-bounded scaled dimension of sequences in the Cantor space was defined in [5] as a generalization of resource-bounded dimension. In that definition scaled gales were used.

Definition 11. *Let $g : H \times [0,\infty) \to \mathbb{R}$ be a scale function, and let $s \in [0,\infty)$.*

1. *An s^g-supergale is a function $d : \{0,1\}^* \to [0,\infty)$ such that for all $w \in \{0,1\}^*$ with $|w| \in H$,*

$$d(w) \geq 2^{-\Delta g(|w|,s)}[d(w0) + d(w1)].$$

2. *We say that an s^g-supergale d succeeds on a sequence $S \in \mathbf{C}$ if*

$$\limsup_n d(S[0 \ldots n-1]) = \infty.$$

3. *The success set of an s^g-supergale d is $S^\infty[d] = \{S \in \mathbf{C} \mid d \text{ succeeds on } S\}$.*

Definition 12. *Let g be a scale function and $X \subseteq C$*

1. *$\widehat{\mathcal{G}}(X)$ is the set of all $s \in [0,\infty)$ such that there is an s^g-supergale d for which $X \subseteq S^\infty[d]$.*
2. *$\widehat{\mathcal{G}}_{\mathrm{constr}}(X)$ is the set of all $s \in [0,\infty)$ such that there is a lower semicomputable s^g-supergale d for which $X \subseteq S^\infty[d]$.*
3. *The constructive g-scaled dimension of X is $\mathrm{cdim}_g(X) = \inf \widehat{\mathcal{G}}_{\mathrm{constr}}(X)$.*
4. *The constructive g-scaled dimension of a sequence $S \in \mathbf{C}$ is $\dim_g(S) = \mathrm{cdim}_g(\{S\})$.*

The main result of this section states that the constructive scaled dimension of a sequence is characterized by the scaled dimension of its prefixes in the following way,

Theorem 4. *Let $g : H \times [0,\infty) \to [0,\infty)$ be an smooth scale function and $S \in C$,*

$$\dim_g(S) = \liminf_n \dim_g(S[0 \ldots n-1]).$$

In [4] Hitchcock showed that constructive gales and constructive supergales are interchangeable in order to define constructive Hausdorff dimension. In this spirit, the next lemma relates constructive scaled dimension of finite strings $\dim_g(w)$, that uses optimal constructive supertermgales, and constructive scaled dimension with just constructive termgales involved.

Lemma 1. *Let g be an smooth scale function and $w \in \{0,1\}^*$, then*

$$\dim_g(w) \geq \inf\{\dim_d(w) \mid d \quad \textit{constructive } g\textit{-termgale}\}.$$

Such inequality has a remarkable aplication for infinite strings, namely the following characterization of constructive scaled dimension just using constructive termgales.

Corollary 3. *Let $S \in \mathbf{C}$ and g smooth scale function,*

$$\dim_g(S) = \liminf_n \mathcal{D}_g(S[0 \ldots n-1]).$$

where $\mathcal{D}_g(w) = \inf\{\dim_d(w) \mid d \textit{ constructive } g\textit{-termgale}\}$.

4 Kolmogorov Complexity and Log-loss prediction

4.1 Scaled dimension and Kolmogorov Complexity

In [6] the authors give an exact characterization of computable and space-bounded scaled dimension of a sequence in terms of (time and space-bounded) Kolmogorov complexity .

Theorem 5. *[6]. Let $S \in \mathbf{C}$*

1. *For all $i \in \mathbb{N}$*

$$\dim^{(i)}_{\text{comp}}(S) = \inf_{t \in \text{comp}} \liminf_n f_i^n(K^{t(n)}(S[0 \ldots n-1])).$$

2. *For all $i, j \in \mathbb{N}$ with $i < j$*

$$\dim^{(i)}_{\text{p}_j\text{space}}(S) = \inf_{t \in \text{p}_j\text{space}} \liminf_n f_i^n(KS^{t(n)}(S[0 \ldots n-1])).$$

In this section we obtain the relationship between the scaled dimension of a finite string and its Kolmogorov complexity, and this result allow us to give a new characterization for constructive scaled dimension of an infinite sequence, extending theorem 5.

Theorem 6. *Let $g : H \times [0,\infty) \to [0,\infty)$ be an smooth scale function. Then there exists a constant $c > 0$ such that for all $w \in \{0,1\}^*$ ($|w|$ large enough),*

$$\left| f^{|w|+1}(K(w)) - dim_g(w) \right| \leq \frac{c}{\frac{\partial g}{\partial s}(|w|+1, 0)}$$

Corollary 4. *Let $S \in \mathbf{C}$ and g smooth scale function,*

$$\dim_g(S[0..n-1]) = \lim_n f^{n+1}(K(S[0..n-1])).$$

Example 1. For the family $g_i,\ i \in \mathbb{N}$,

$$\dim^{(i)}(S) = \liminf_n f_i^n(K(S[0\ldots n-1])).$$

In the particular case of $i = 0$ we have the result of constructive dimension obtained by Mayordomo in [12].

$$\dim(S) = \liminf_n \frac{K(S[0..n-1])}{n+1}.$$

4.2 Scaled dimension and Prediction.

Consider predicting the symbols of an unknown finite string. Then, given a prefix of this string, the next character could be 0, 1 or may be, the string doesn't have any more characters. A termpredictor Π gives us an estimation of the probability of each of these cases.

Definition 13. *A function $\Pi : \{0,1\}^* \times \{0,1,\square\} \to [0,1]$ is a termpredictor if*

$$\Pi(w,0) + \Pi(w,1) + \Pi(w,\square) = 1.$$

We interpret $\Pi(w,a)$ as the Π's estimation of the likehood that there is a bit a following the string (if $a = 0$ or 1) or there is no bit following the string (if $a = \square$).

The next lemma establishes a correspondence between termpredictors and g-termgales.

Lemma 2. *Let g be an smooth scale function.*

1. *Let Π be a termpredictor, define $\forall s \in [0,\infty)$, $d^s_{\Pi,g} : \mathcal{T} \to [0,\infty)$ by*

$$d^s_{\Pi,g}(w) = 1 \qquad \text{if } |w| \notin H.$$

$$d^s_{\Pi,g}(w) = 2^{g(|w|,s)} \prod_{i=0}^{|w|-1} \Pi(w[0\ldots i-1], w[i]) \qquad \text{if } |w| \in H.$$

   Then, $d_{\Pi,g}$ is a g-termgale.
2. *Let d_g be a g-termgale, then for $s \in [0,\infty)$ define $\Pi_{d_g} : \{0,1\}^* \times \{0,1,\square\} \to [0,1]$ by*

$$\Pi_{d_g}(w,a) = 2^{-\Delta g(|w|,s)} \frac{d_g(wa)}{d_g(w)} \qquad \text{if } d_g(w) \neq 0.$$

$$\Pi_{d_g}(w,a) = \frac{1}{3} \qquad \text{if } d_g(w) = 0.$$

   Π_{d_g} is a termpredictor and this definition doesn't depends on s.

3. $d_{\Pi_{d_g},g} = d_g$ *and* $\Pi_{d_{\Pi,g}} = \Pi$.

In order to define the performance of a termpredictor, we will consider (as in [3]) the sum of its "loss" on each individual symbol (including $\square$).

Definition. For $w \in \mathcal{T}$ and Π termpredictor we define the log-loss

$$\mathcal{L}^{\log}_{\Pi}(w) = \sum_{i=0}^{|w|-1} \log \frac{1}{\Pi(w[0\ldots i-1], w[i])}.$$

Theorem 7. *Let g be an smooth scale function, let d_g be a constructive g-termgale and $w \in \{0,1\}^*$ with $|w| \in H$ then,*

$$\dim_{d_g}(w) = f^{|w|+1}(\mathcal{L}^{\log}_{\Pi_{d_g}}(w\square)).$$

In particular if d is a simple termgale and $w \in \{0,1\}^*$ then

$$\dim_d(w) = \frac{\mathcal{L}^{\log}_{\Pi_d}(w\square)}{|w|+1}.$$

Unfortunately, there are no existence of optimal constructive termgales (or optimal constructive termpredictors) and we can not prove an equality of this kind for the definition of scaled dimension of a string. But we have the following result for infinite sequences as a consequence of Proposition 1 and Theorem 7.

Theorem 8. *Let g be an smooth scale function and $S \in \mathbf{C}$,*

$$\dim^g(S) \leq \inf\{\mathcal{L}^{\log}_{\Pi,g}(S) \mid \Pi \text{ is a constructive termpredictor }\}$$

where

$$\mathcal{L}^{\log}_{\Pi,g}(S) = \liminf_n f^{n+1}(\mathcal{L}^{\log}_{\Pi}(S[0\ldots n-1]\square)).$$

The next result partially extends the characterization that Hitchcock obtained in [3] for resource-bounded dimension to the cases of scaled and constructive dimension.

Theorem 9. *Let $S \in \mathbf{C}$ and let g be an smooth scale function,*

$$\dim_g(S) \leq \mathcal{L}^{\log}_g(S)$$

where,

$$\mathcal{L}^{\log}_g(S) = \inf\{\mathcal{L}^{\log}_{\Pi,g}(S) \mid \Pi \text{ is a constructive predictor }\}$$

and

$$\mathcal{L}^{\log}_{\Pi,g}(S) = \liminf_n f^{n+1}(\mathcal{L}^{\log}_{\Pi}(S[0\ldots n-1])).$$

For the particular case of constructive dimension

$$\dim(S) \leq \inf\{\mathcal{L}^{\log}_{\Pi}(S) \mid \Pi \text{ is a constructive predictor }\}$$

where

$$\mathcal{L}^{\log}_{\Pi}(S) = \liminf_n \frac{\mathcal{L}^{\log}_{\Pi}(S[0\ldots n-1])}{n}.$$

The other inequality seems to be closely related to the open question of whether constructive prediction and constructive gales are equivalent.

Acknowledgement

The author thanks Elvira Mayordomo and Julio Bernues for very helpful discussions, comments and suggestions.

References

1. G. J. Chaitin. A theory of program size formally identical to information theory. *Journal of the Association for Computing Machinery*, 22:329–340, 1975.
2. L. Fortnow and J. H. Lutz. Prediction and dimension. *Journal of Computer and System Sciences*. To appear. Preliminary version appeared in *Proceedings of the 15th Annual Conference on Computational Learning Theory*, LNCS 2375, pages 380–395, 2002.
3. J. M. Hitchcock. Fractal dimension and logarithmic loss unpredictability. *Theoretical Computer Science*, 304(1–3):431–441, 2003.
4. J. M. Hitchcock. Gales suffice for constructive dimension. *Information Processing Letters*, 86(1):9–12, 2003.
5. J. M. Hitchcock, J. H. Lutz, and E. Mayordomo. Scaled dimension and nonuniform complexity. *Journal of Computer and System Sciences*, 69:97–122, 2004.
6. J.M. Hitchcock, M. López-Valdés, and E. Mayordomo. Scaled dimension and the Kolmogorov complexity of Turing-hard sets. In *Proceedings of the 29th International Symposium on Mathematical Foundations of Computer Science*, volume 3153 of *Lecture Notes in Computer Science*, pages 476–487. Springer-Verlag, 2004.
7. L. A. Levin. On the notion of a random sequence. *Soviet Mathematics Doklady*, 14:1413–1416, 1973.
8. L. A. Levin. Laws of information conservation (nongrowth) and aspects of the foundation of probability theory. *Problems of Information Transmission*, 10:206–210, 1974.
9. M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer-Verlag, Berlin, 1997. Second Edition.
10. J. H. Lutz. Dimension in complexity classes. *SIAM Journal on Computing*, 32:1236–1259, 2003.
11. J. H. Lutz. The dimensions of individual strings and sequences. *Information and Computation*, 187:49–79, 2003.
12. E. Mayordomo. A Kolmogorov complexity characterization of constructive Hausdorff dimension. *Information Processing Letters*, 84(1):1–3, 2002.
13. A. K. Zvonkin and L. A. Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Mathematical Surveys*, 25:83–124, 1970.

Solving a PSPACE-Complete Problem by a Linear Interval-Valued Computation

Benedek Nagy[1,2] and Sándor Vályi[1]

1: Department of Computer Science, Faculty of Informatics, University of Debrecen
Hungary H-4010 Debrecen, P.O. Box 12
{nbenedek|valyis}@inf.unideb.hu
2: Research Group on Mathematical Linguistics, Rovira i Virgili University
Tarragona, Spain

Abstract. An interval-valued computing approach was presented in [Nagy 2005b]. The computation is executed on interval-valued bytes, the bits of which indexed by the interval [0,1] rather than by a finite set. This device was presented there as a new possible model of analogue computers. The question of which complexity is needed to solve PSPACE-complete problems in this paradigm was also raised in [Nagy 2005b]. An answer to this question is provided now, namely, we show that the problem of validity of quantified propositional formulae is decidable by a linear interval-valued computation.

Keywords: Interval-valued Logic, Computing with Interval-values, Non-standard Computing, Massively Parallel Computing

1 Introduction

It is widely acknowledged in the literature that all theoretical models for general computing devices involve some abstraction of the concrete possibilities of the available technical devices. For example, a Turing machine can access an unlimited number of tape units; complexity-theoretical aspects of Turing machines are usually examined with oracles of high complexity (see [Rogers 1987]). There is an extensive literature on infinite size and infinite time computations on Turing machines (e.g. [Lenzi and Monteleone 2004], [Hamkins and Seabold 2001], [Welch 2000]). In [Blum et al. 1989] (see also [Blum et al. 1998]), a complexity theory of calculations with real numbers is introduced. It is a possible approach to model analogue (in the sense of non-digital) computations. Another model appeared in [Nagy 2005b], where a new generalization of classical digital computers was proposed. The unit of information processing remains a byte which is in this case not a finite amount of bits but a full sequence of bits indexed by the interval [0,1]. The model in question has been based on a version of interval-valued logic proposed in [Nagy 2005a], which is a natural extension of Boolean logic to interval-values.

Motivation for this model is given in [Nagy 2005b]. The physical possibility of full implementation of these kinds of computation is of course a question left for future investigations. Some limited approximations were mentioned already in [Nagy 2005b]. The present article deals with a restricted model which is closer to an implementable version by hardware available today. It may also be of a mathematical interest to determine, what class of functions can be computed with the help of this paradigm and how its complexity hierarchy relates to other hierarchies of complexity. In [Nagy 2005b] this research program was started; more specifically, it was shown, that the problem SAT – which is a basic example for NP-complete problems – can be solved by a linear interval-valued computation.

In this paper we contribute to the development of complexity issues of this computing paradigm. [Nagy 2005b] raised the question whether there exists a PSPACE-complete problem solvable by an interval-valued computation of similar simple complexity. We show that the problem $QSAT$ – i.e. whether a quantified propositional formula is true, which problem is PSPACE-complete (see e.g. [Papadimitriou 1994]) – can be solved by a linear interval-valued computation, as well.

2 Interval-valued computations

In this section we recall the interval-valued computing system of [Nagy 2005b]. First the notion of interval-values is defined. After this, the allowed operations which can be used to build expressions are presented and the method computing the values of these expressions is also given. The selection of operators is natural but we do not claim that this is the only natural possibility. It is clear that the complexity hierarchy depends on this selection. It is left for possible future research to carry out investigations in this direction.

2.1 The allowed interval-values

In our model of computation, the computing device is similar to the processing unit of a traditional computer, but it operates on interval-values instead of bytes. We propose an alternative, but equivalent definition compared to that of [Nagy 2005b], which is more concise.

Any of the following objects is called a *subinterval*: $[a, b], [a, b), (a, b], (a, b), [a, a]$ where $0 \leqslant a < b \leqslant 1$. The *allowed class of interval values* (denoted by $\mathbb{V}$) coincides with the set of finite unions of subintervals.

It is important to note that in this paper a restricted set of interval-values is used. We allow only finite unions of subintervals as values. This selection limits the computational power but puts the system closer to those implementable by technical devices which are available today.

The set of finite unions includes the empty set, that is, $\emptyset$ is an allowed interval-value, too. $\mathbb{V}$ forms a Boolean set algebra with the usual set-theoretical opera-

tions of intersection (as conjunction), union (as disjunction) and complementation (as negation). Other logical operators (implication, nand etc.) are expressible using these basic operators in the usual way.

2.2 Non-logical operators on interval-values

A traditional computer has not only logical (above called set-theoretical) operations, accordingly we need to allow other operations, too, namely two shift operators and the "fractalian" product. This makes our system a general computing device, as was shown already in [Nagy 2005b].

First of all, we introduce the first-length function $Flength$ which will be useful in the forthcoming definitions. This function assigns a real number to each interval-value, namely the length of its first "component". More precisely, if we denote loosely both the interval-value and its characteristic function by the same letters, say $A, B, \ldots$ (we will use this notation through the rest of this article) then $Flength(A) = b - a$, if A contains the open interval (a, b) and A does not contain any interval (a, c) where $c > b$, moreover, the difference of A and $[a, b]$ does not contain any point x where $x < a$. Otherwise $Flength(A) = 0$.

With the help of $Flength$ we turn now to the definition of shift operators. The *left-shift* operator will shift the first interval-value by the first-length of the second operand to the left, and will remove the part which is shifted out of the interval $[0, 1]$. As opposed to that, the *right-shift* operator is defined in a circular way, i.e. the parts shifted above 1 will appear at the lower end of $[0, 1]$. If A, B are two interval-values then interval-values $Lshift(A, B)$ and $Rshift(A, B)$ are defined with the help of their characteristic functions.

$Lshift(A, B)(x) = A(x + Flength(B))$ if $0 \leqslant x + Flength(B) \leqslant 1$, and
$Lshift(A, B)(x) = 0$ otherwise.
$Rshift(A, B)(x) = A(frac(x - Flength(B)))$ if $x < 1$, and
$Rshift(A, B)(1) = Rshift(A, B)(0)$.

Here the function *frac* gives the fractional part of a real number, i.e. $frac(x) = x - int(x)$, where $int(x)$ is the greatest integer which is not greater than x.

Figure 1 illustrates both operations $Rshift$ and $Lshift$. The second (ancillary-) operands are shown in grey for better visualization, but they do not form real parts of the resulting interval-values.

Let us explain now the fractalian product on intervals. If A contains k interval components with end-points $a_{i,1}, a_{i,2}$ $(1 \leqslant i \leqslant k)$ and B contains l components with end-points $b_{i,1}, b_{i,2}$ $(1 \leqslant i \leqslant l)$, then we determine the value of $C = A * B$ as follows: we set the number of components of C is $k \cdot l$. During this process we can use double indices for the components of C. The starting point and the ending point of the ij-th component are $a_{i1} + b_{j1}(a_{i2} - a_{i1})$ and $a_{i1} + b_{j2}(a_{i2} - a_{i1})$, respectively. An end-point belongs to the interval if and only if the original endpoints belong to the interval-values of the original interval-values A and B.

In Figure 2 the fractalian product is illustrated.

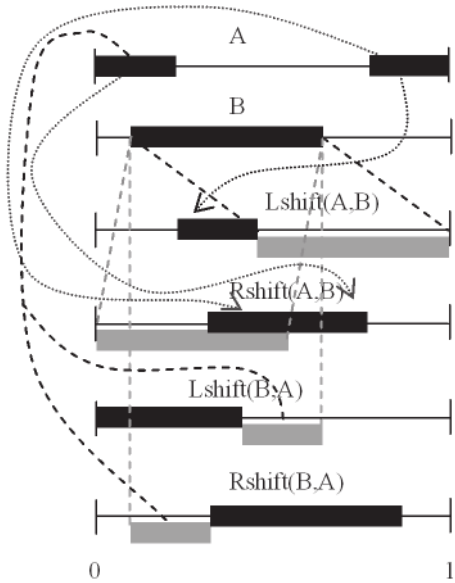


Fig. 1. Examples of shift operators with interval-values

2.3 Computations as evaluation of expressions

Several variations of ways of computing can be considered depending on the one hand, on the allowed constants and on the other hand, on the method checking the result. In this paper we use only one interval-valued constant and we allow to check emptiness and non-emptiness of the resulted interval-value. It is obvious that the concepts of computability and complexity depend on the variation used. Now we will describe the version used in this paper.

An *interval-valued expression* means a term composed from the constant $FIRSTHALF$ and from expression labels, by a finite number of applications of the two-argument operators defined above (the two shifts and the fractalian product) and the set-theoretical operators. No recursion, not even implicit recursion by a reference cycle, is permitted when expression labels are used. The *operational complexity* of an expression E means the number of operation symbols used while building up E from the constant and the labels. It is an interesting question left for future research to determine to which class of cyclic operator definitions one can provide adequate fixpoint semantics.

We define now the *interval-value* of an expression E, denoted by $\|E\|$. We fix $\|FIRSTHALF\|$ as $[0, \frac{1}{2}]$ while the value of the label of an already evaluated

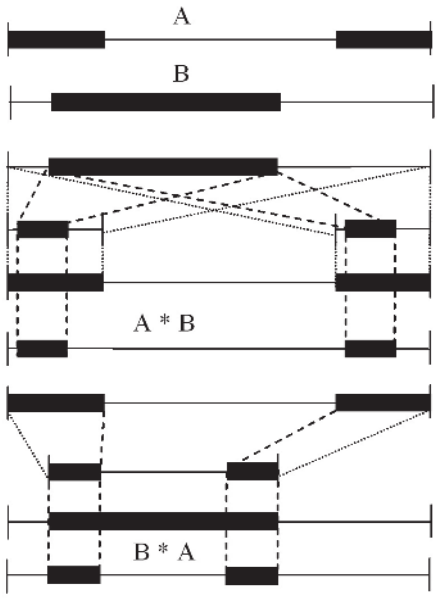


Fig. 2. Examples of product of interval-values

expression is just the value of the expression itself. Because of lack of recursion, the interval-value is well-defined, but the evaluation of a referred expression has to precede the evaluation of its referring expression. $\|Lshift(A,B)\|$, $\|Rshift(A,B)\|$ and $\|A*B\|$ have been defined in the preceding subsection like the logical operators, i.e. as interval-values of union, intersection and complementation of interval-values.

2.4 Decidability

Let Σ be a finite alphabet. We say that a language $L \subset \Sigma^*$ is *decidable by an interval-valued computation* if there is a logarithmic space algorithm A that for each input word $w \in \Sigma^*$ constructs an appropriate expression $A(w)$ such that $w \in L$ if and only if $\|A(w)\|$ is nonempty. Further, we consider in this case $\overline{L}$ also decidable. (This last remark makes possible to test emptiness and also $\|A(w)\| = [0,1]$ by set-theoretical operators.)

We say that a language $L \subset \Sigma^*$ is *decidable by a linear interval-valued computation* if and only if there is a positive constant c and an algorithm A with the following properties. For each input word $w \in \Sigma^*$ A constructs an appropriate list S of expressions with the last element $A(w)$ such that the length of S is not greater than $c \cdot |w|$, and the operational complexity of each member of S is less than c, moreover, $w \in L$ if and only if $\|A(w)\|$ is nonempty. Again, $\overline{L}$ can be decided instead of L itself.

Our motivation to define linear interval-valued computations in this way was to make explicit in what sense [Nagy 2005b] stated that a linear computation exists to decide SAT.

3 A linear interval-valued computation to decide $QSAT$

3.1 The language of the true quantified propositional formulae

We recall now the definition of (a suitable variation of) the language $QSAT$ of the *true quantified propositional formulae*. It is a subset of the satisfiable propositional formulae, built from the propositional variables $\{x_1, x_2, x_3, \ldots\}$. The quantifier prefix is not added syntactically to the propositional formulae, only the definition of semantics is given in this way. The variables of odd index are meant to quantify universally, while the ones of even index to quantify existentially. It can be shown by variable renaming and using of the fictive quantifiers that this variation is equally PSPACE-complete as the original $QSAT$ ([Papadimitriou 1994]).

Before we continue the definition, we have to arrange some notations. A *valuation* is a function with range $\{0,1\}$ on the domain $\{x_1, \ldots, x_n\}$ for some positive integer n. If $t_1, \ldots, t_n$ are truth values then we write $(t_1, \ldots, t_n)$ for the valuation v that $v(x_1) = t_1, \ldots, v(x_n) = t_n$ and $dom(v) = \{x_1, \ldots, x_n\}$. For a quantifier-free formula ϕ, $|\phi v|$ denotes the truth value of ϕ by the valuation v. For any positive integer i, the quantifier Q_i is $\forall$ if i is odd otherwise it is $\exists$.

So, for any formula ϕ, ϕ belongs to our variation of $QSAT$ if and only if there exists a positive integer n such that the propositional variables occurring in ϕ are exactly $x_1, \ldots, x_n$, and $(\forall t_1 \in \{0,1\})(\exists t_2 \in \{0,1\}) \ldots (Q_n t_n \in \{0,1\}) : |\phi(t_1, \ldots, t_n)| = 1$.

3.2 The result

Theorem 1. *$QSAT$ is decidable by a linear interval-valued computation.*

Let us choose c to be 9. We give an algorithm to construct expressions $A_1, \ldots, A_n$, $B_1, \ldots, B_m, C_0, \ldots, C_n, D$ for any input formula ϕ that contains exactly the variables $x_1, \ldots, x_n$ and the number of its subformulae is m. Clearly, the length of this list is less than $4 \cdot |\phi|$. Our algorithm constructs the mentioned list in such a way that $\|D\|$ will be empty if and only if $\phi \in QSAT$, moreover the operational complexity of each of its members is less than 9.

In [Nagy 2005b], a proof was given for the decidability of SAT by a linear interval-valued computation. This proof essentially gives an algorithm that for any formula ϕ, having m subformulae and built from $x_1, \ldots, x_n$, constructs an expression list $A_1, \ldots, A_n, B_1, \ldots, B_m$ of length less than $2 * |\phi|$ and the requirement on the operational complexity is also satisfied. We can realize that these expressions satisfy the following conditions:

(1) $\|A_1\| = [0, 1/2]$,

(2) generally, for every $j \in \{1, \ldots, n\}$, $\|A_j\| = \bigcup_{k=0}^{2^j-2} \left[\frac{2k}{2^j}, \frac{2k+1}{2^j}\right]$, further,

(3) $\|B_m\| = \{r \in [0, 1] : |\phi(r \in \|A_1\|, \ldots, r \in \|A_n\|)| = 1\}$.

Furthermore, for any $r \in [0, 1], r \neq \frac{k}{2^n} (k \in \mathbb{Z})$ hold

(4) if $r \in \|A_j\|$ then $r + \frac{1}{2^j} \in \|A_i\|$ if and only if $r \in \|A_i\|$ for all $i < j$,
(5) if $r \notin \|A_j\|$ then $r - \frac{1}{2^j} \in \|A_i\|$ if and only if $r \in \|A_i\|$ for all $i < j$,
(6) $r + \frac{1}{2^j} \in \|A_j\|$ if and only if $r \notin \|A_j\|$.

Our algorithm continues the work of the original algorithm and constructs a new expression list $C_0, C_1, \ldots, C_n, D$ to append to $A_1, \ldots, A_n, B_1, \ldots, B_m$. The older expressions are cited as labels in the new ones. The expressions $C_0, \ldots, C_n$ are determined by an inductive way. In this way we avoid cyclic recursions. C_0 is just a copy of B_m. For $j \in \{0, \ldots, n-1\}$, let C_{j+1} be

$((Lshift(C_j, A_{n-j}) \wedge A_{n-j}) \vee C_j) \vee ((Rshift(C_j, A_{n-j}) \wedge \neg A_{n-j}) \vee C_j)$, if $n - j$ is even,
$(Lshift(C_j, A_{n-j}) \wedge A_{n-j} \wedge C_j) \vee (Rshift(C_j, A_{n-j}) \wedge \neg A_{n-j} \wedge C_j)$ in the other case.

The following lemma is essential to finish our proof. To make it more readable, we assume that the variables from $\{t_1, t_2, \ldots\}$ range over the truth values without any further comment. We recall the definition of quantifier sequence $Q_1, Q_2, Q_3, \ldots$ as $\forall, \exists, \forall, \ldots$, respectively.

Lemma 1. *For all integer j between 0 and n and for all $r \in \left([0, 1] \setminus \bigcup_{k=0}^{2^n} \{\frac{k}{2^n}\}\right)$: $r \in \|C_j\|$ if and only if*
$Q_{n-j+1} t_{n-j+1} \ldots Q_n t_n \ |\phi(r \in \|A_1\|, \ldots, r \in \|A_{n-j}\|, t_{n-j+1}, \ldots, t_n)| = 1$.

Proof. It is an induction on j from 0 to n. For $j = 0$, condition (3) implies the needed equivalence, namely
$r \in \|C_0\|$ if and only if $|\phi(r \in \|A_1\|, \ldots, r \in \|A_n\|)| = 1$.

Assume that for any r that is in $\left([0, 1] \setminus \bigcup_{k=0}^{2^n} \{\frac{k}{2^n}\}\right)$, $r \in \|C_j\|$ if and only if $Q_{n-j+1} t_{n-j+1} \ldots Q_n t_n \ |\phi(r \in \|A_1\|, \ldots, r \in \|A_{n-j}\|, t_{n-j+1}, \ldots, t_n)| = 1$. This is the induction hypotheses.

We have to check whether $r \in \|C_{j+1}\|$ if and only if
$Q_{n-j} t_{n-j} \ldots Q_n t_n \ |\phi(r \in \|A_1\|, \ldots, r \in \|A_{n-(j+1)}\|, t_{n-j}, \ldots, t_n)| = 1$,
for arbitrary r from $\left([0, 1] \setminus \bigcup_{k=0}^{2^n} \{\frac{k}{2^n}\}\right)$.

We write a sequence of equivalent conditions starting with $r \in \|C_{j+1}\|$ and closing with the right side of the equivalence to prove. We prove the case when

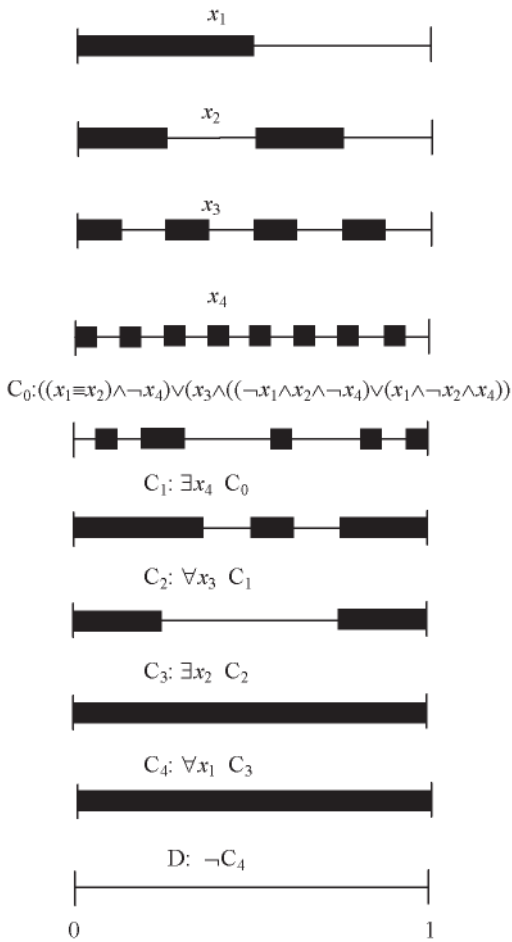


Fig. 3. $\forall x_1 \exists x_2 \forall x_3 \exists x_4 (((x_1 \equiv x_2) \wedge \neg x_4)) \vee (x_3 \wedge ((\neg x_1 \wedge x_2 \wedge \neg x_4) \vee (x_1 \wedge \neg x_2 \wedge x_4)))$ is true.

$n-j$ is even and Q_{n-j} is $\exists$, the proof in the case of an odd $n-j$ can be constructed analogously.

(i) $r \in \|C_{j+1}\|$,

(ii) $r \in \|C_j\|$ or $\big(r \in \|Lshift(C_j, A_{n-j})\| \wedge r \in \|A_{n-j}\|\big)$ or $\big(r \in \|Rshift(C_j, A_{n-j})\| \wedge r \in \neg\|A_{n-j}\|\big)$,

(iii) $\forall t_{n-j+1} \dots Q_n t_n \ |\phi(r \in \|A_1\|, \dots, r \in \|A_{n-j}\|, t_{n-j+1}, \dots, t_n)| = 1$ or $r \in \|A_{n-j}\| \wedge$ $\forall t_{n-j+1} \dots Q_n t_n \ |\phi(r+\frac{1}{2^{n-j}} \in \|A_1\|, \dots, r+\frac{1}{2^{n-j}} \in \|A_{n-j}\|, t_{n-j+1}, \dots, t_n)| = 1$, or $r \notin \|A_{n-j}\| \wedge$ $\forall t_{n-j+1} \dots Q_n t_n \ |\phi(r-\frac{1}{2^{n-j}} \in \|A_1\|, \dots, r-\frac{1}{2^{n-j}} \in \|A_{n-j}\|, t_{n-j+1}, \dots, t_n)| = 1$,

(iv) $\forall t_{n-j+1} \dots Q_n t_n \ |\phi(r \in \|A_1\|, \dots, r \in \|A_{n-j}\|, t_{n-j+1}, \dots, t_n)| = 1$ or $\forall t_{n-j+1} \dots Q_n t_n \ |\phi(r \in \|A_1\|, \dots, r \in \|A_{n-j-1}\|, r \notin \|A_{n-j}\|, t_{n-j+1}, \dots, t_n)| = 1$,

(v) $\exists t_{n-j} \forall t_{n-j+1} \dots Q_n |\phi(r \in \|A_1\|, \dots, r \in \|A_{n-j-1}\|, t_{n-j}, \dots, t_n)| = 1$

The equivalence of (i) and (ii) is validated by the definitions of C_{j+1} and the operators, while that of (ii), (iii) and (iv) by the properties (3)–(6). Finally, the equivalence of (iv) and (v) can be shown considering that only two possible truth values exist.
Q.E.D.

Now, we are ready for

Proof of Theorem 1. The above lemma ensures by letting $j = n$ that $r \in \|C_n\|$ if and only if $Q_1 t_1 \dots Q_n t_n \ : |B_m(t_1, \dots, t_n)| = 1$ and this holds for any r in [0,1] maybe except some of the values $\frac{k}{2^n}$ where $0 \leqslant k \leqslant 2^n$. By taking $D = \neg(Lshift(C_n, A_n * A_1) \vee C_n \vee Rshift(C_n, A_n * A_1))$ we get an expression satisfying the condition that $\|D\|$ is empty if and only if $\phi \in QSAT$.
Q.E.D.

Figure 3 shows an example of evaluating of a formula. The formula $(((x_1 \equiv x_2) \wedge \neg x_4)) \vee (x_3 \wedge ((\neg x_1 \wedge x_2 \wedge \neg x_4) \vee (x_1 \wedge \neg x_2 \wedge x_4)))$ is shown to be in $QSAT$.

We can observe that due to the special choice of the semantics of the expressions, the parallelism of our computations is finite (but unbounded). In fact, for any of our computations there is an integer n such that the occurring interval values are finite unions of intervals of the form $[\frac{k}{2^n}, \frac{k+1}{2^n}](0 \leqslant k < n)$ and their open or half open versions. It is obvious that these limitations restrict the computational power of the model.

4 Conclusion and final remarks

We proved the solvability of a $PSPACE$-complete problem by a linear interval-valued computation. It seems to be trivial that a constant amount of operators is not enough to decide even SAT, at least if we do not allow other operators on

interval-values. To find the upper side of limitations of this paradigm one should search for conditions describing linear or general computability of problems by interval-valued computation. These conditions may differ for different choices of initial interval values and of operators. For example, one may ask whether the language of true arithmetics is decidable by an interval-valued computation, of course without the restrictions used in this paper. We do not see any straightforward argumentation against this possibility.

Our definition for computability has a Boolean-network style in some sense. One could consider a computational framework in a more imperative manner and relate the hierarchies arising in this way.

Acknowledgements

The authors are grateful to the referee for her or his valuable remarks and to Enikő Tóth for her help concerning the language. This research was partly supported by the Hungarian Research Foundation for Scientific Research (OTKA), grant no. F043090, T049409 and T043242.

References

[Blum et al. 1989] Blum, L., Shub, M. and Smale, S. "On a theory of computation and complexity over the real numbers", Bull. AMS (New Series) 21(1989), no. 1, pp. 1–46.

[Blum et al. 1998] Blum, L., Cucker, F., Shub, M. and Smale, S. "Complexitiy and real computation", Springer, 1998

[Hamkins and Seabold 2001] Hamkins, J. D. and Seabold, D. E., "Infinite time Turing machines with only one tape", Math. Logic Quarterly 47(2001), no. 2, pp. 271–287.

[Lenzi and Monteleone 2004] Lenzi, G. and Monteleone, E., "On fixpoint arithmetic and infinite time Turing machines", Inform. Process. Letters 91(2004), no. 3, pp. 121–128.

[Nagy 2005a] Nagy, B., "A general fuzzy logic using intervals", 6th International Symposium of Hungarian Researchers on Computational Intelligence, Budapest, Hungary, pp. 613–624. (earlier version: Interval-valued logic, University of Debrecen, 1998, thesis, in Hungarian).

[Nagy 2005b] Nagy, B., "An Interval-valued Computing Device", in: "Computability in Europe 2005: New Computational Paradigms", (eds. S. B. Cooper, B. Löwe, L. Torenvliet), ILLC Publications X-2005-01, Amsterdam, pp. 166–177.

[Papadimitriou 1994] Papadimitriou, C. H., "Computational Complexity", 1994, Addison-Wesley.

[Rogers 1987] Rogers, H., "Theory of Recursive Functions and Effective Computability", 1987, MIT Press.

[Welch 2000] Welch, P. D., "Eventually infinite time Turing machine degrees: infinite time decidable reals", J. of Symb. Logic 65(2000), no. 3, pp. 1193–1203.

Hypercomputing the Mandelbrot Set?

Petrus H. Potgieter

Department of Decision Sciences, University of South Africa (Pretoria), PO Box 392, UNISA, 0003, Republic of South Africa, potgiph@unisa.ac.za

Abstract. The Mandelbrot set is an extremely well-known mathematical object that can be described in a quite simple way but has very interesting and non-trivial properties. This paper surveys some results that are known concerning the (non-)computability of the set. It considers two models of decidability over the reals (which are treated much more thoroughly and technically in [1], [2], [3] and [4] among others), two over the computable reals (the Russian school and hypercomputation) and a model over the rationals.

Keywords: Mandelbrot set, Zeno machines, hypercomputation.

1 Introduction

In theoretical computer science it is no surprise that the halting problem for Turing machines is the favourite target for solution by non-conventional models of computation. However, the decidability of sets of reals and the computability of functions in ordinary real analysis is a topic of great interest to the broader mathematical community and a potential area of application for so-called hypercomputation. This paper surveys some of the most important results and gives an extremely simple example of the application of accelerated Turing machines to the question of decidability of the Mandelbrot set. This very amusing problem was raised by Roger Penrose [5] and—like many good questions—implies considerable work on the definitions, in this case what exactly is meant by *decidable* for a subset of the plane.

In 1979 Benoît Mandelbrot used a computer to plot[1] a beautiful approximation of the subset

$$M = \{c \in \mathbb{C} \mid \text{for all } n \geq 1,\ |f_c^n(0)| \leq 2\} \qquad \text{where } f_c(x) = x^2 + c \qquad (1)$$

of the complex plane $\mathbb{C}$ (where f^n denotes n-th iteration of f). This set was originally described by Pierre Fatou in 1905 but after the appearance of a colourful plot of the set in Mandelbrot's book [6] and—especially—on the cover of Scientific American and in an accompanying column [7] in August 1985, the Mandelbrot set has become one of the greatest celebrities of popular mathematics.

[1] Mandelbrot actually plotted a *mirror image* of M.

Plots can easily be generated using a plethora of software freely available, including normal LaTeX code[2]. The relatively concise but non-optimised Octave[3] code snippet

```
n=1000;                              # For an nxn grid
m=50;                                # Number of iterations
c=meshgrid(linspace(-2,2,n))\        # Set up grid
  +i*meshgrid(linspace(2,-2,n))';
x=zeros(n,n);                        # Initial value on grid
for i=1:m
   x=x.^2+c;                         # Iterate the mapping
endfor
imagesc(min(abs(x),2.1))             # Plot monochrome, absolute
                                     # value of 2.1 is escape
```

suffices, for example, to plot the Mandelbrot image in Figure 1.

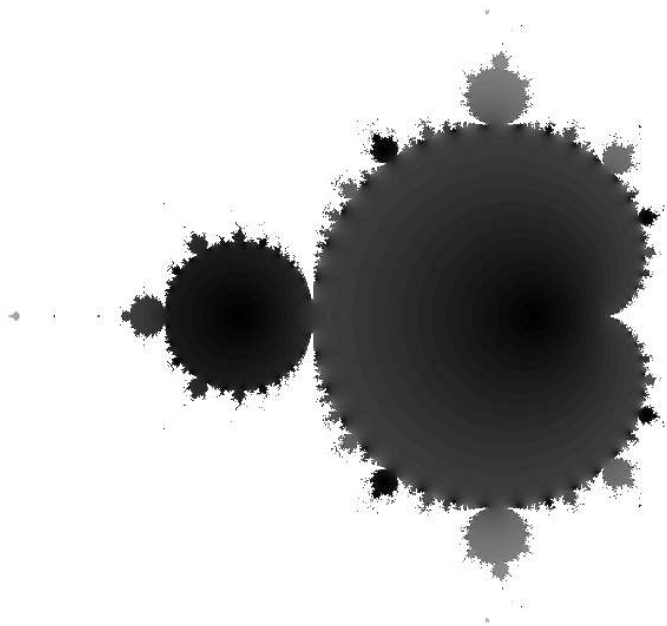


Fig. 1. Membership candidates for the Mandelbrot set

[2] http://www.thole.org/manfred/apfel/apfel.tex [accessed 2005-12-30] by Manfred Thole. The Mandelbrot set is approximated by the white area at the centre of this plot generated by a correctly compiled apfel.tex.

[3] Octave is a free and open-source high-level language for numerical computation with implementations on many platforms and largely compatible with MATLAB®. See http://www.octave.org/.

The image in Figure 1 shows an expanse of background (white) points which were shown during the execution of the code to have left the closed disk of radius 2—therefore not to belong to the Mandelbrot set—and grey and black points which are *candidates* for membership of M. In this plot the points c have a lighter colour when they have approached relatively closely to the boundary of the disk after exactly 50 iterations of the map $f_c(0)$. This shading scheme emphasises that the usual routines for generating Mandelbrot plots, like this one, clearly can identify only elements of the complement of M and not of the set itself. Scientists are well-aware this fact (that the complement of M is only known to be recursively enumerable) but it is usually not overemphasised in the more popular writing. Incidentally, M is compact and simply connected [8] which one cannot, and should not expect to, see in Figure 1. Figure 2 shows a plot of the Mandelbrot set candidates as one more usually sees it.

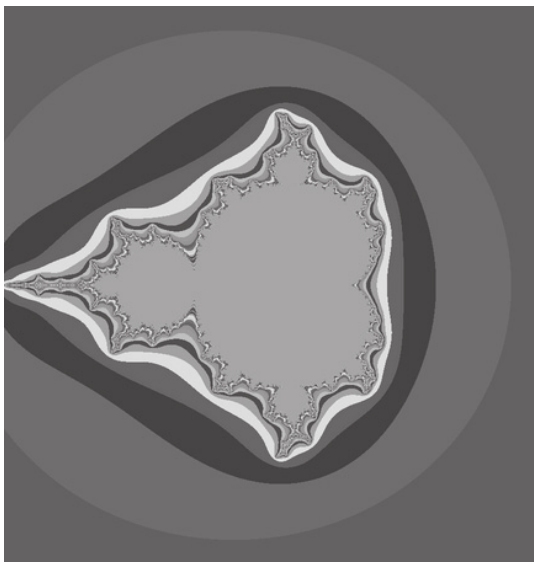


Fig. 2. The Mandelbrot plot as we too often see it

The central part is, of course, the same as the set of grey or black points in Figure 1—those points who have not (yet) escaped the disk after m iterations in the Octave code fragment

```
n=1000;                          # For an nxn grid
m=60;                            # Number of iterations
c=meshgrid(linspace(-2,2,n))\    # Set up grid
  +i*meshgrid(linspace(2,-2,n))';
x=zeros(n,n);y=ones(n,n);        # Initial values on grid
```

```
                                  # y counts number of iterations
                                  # without escape from the disk
for i=1:m
   x=x.^2+c;                      # Usual iteration
   y=y.+(.5+sign(3-abs(x))./2);# Add one if still in radius 3
   x=x.*(min(abs(x),3)\           # Scale back points far away to
     ./(abs(x)+!abs(x)));         # speed up and avoid overflow
endfor
imshow(imagesc(y),rand(m+1,3)) # Plot y with random colours
```

which now keeps track of when a point first leaves the disk (if at all) using the matrix y. The union of the outer coloured bands represents $\mathbb{R} \setminus M$ on the computational grid, or what we could discover of it in the number of iterations executed. Although pretty, these bands show the stages of construction of M's complement and disguise the fact that we know about M not much more than that it is somewhere in the central coloured section! Incidentally, John Ewing and Glenn Schober have made [9] laborious numerical estimates of the area[4] of the Mandelbrot set using two different plotting methods and arrived at answers of respectively 1.52 and 1.72. This is for a set which everybody believes that they have seen!

In a very readable article [11] Lenore Blum has described the gulf between numerical analysis and computer science. Blum calls the classical theory of (Turing) computability

> fundamentally inadequate for providing such a foundation for modern scientific computation, in which most algorithms–with origins in Newton, Euler, Gauss, et al.–are real number algorithms

as a partial justification of the Blum-Shub-Smale model of computation over the real numbers (mentioned in 3.1, below). The present author would be inclined to the opinion that 'modern scientific computation' is rather inadequately founded in the classical theory of computability and that the logical and foundational problems which arise in this regard (some of which are illustrated in this paper) could be much deeper and connected to our model of the real numbers and their representation; and to the finite/infinite duality which is drilled into mathematical recruits in a dingy room in the Hilbert Grand Hotel.

2 M in the Recursive Realm

First, let us consider only computable points in the Mandelbrot set M. By this we mean points that can be finitely specified and communicated in a consistent way, which one can take to mean *represented by a program for computing approximating rationals*. This is an intuitive idea used in all versions of computable

[4] A tighter estimate has recently been computed by Yuval Fisher and Jay Hill [10].

analysis and which can be readily grasped. Fix a universal Turing machine U as well as a (recursive, surjective) encoding $\phi : \mathbb{N} \to \{0,1\} \times \mathbb{N} \times \mathbb{N}$ with

$$\phi : n \mapsto (\phi_1(n), \phi_2(n), \phi_3(n))$$

so that every rational number is of the form

$$(-1)^{\phi_1(n)} \frac{\phi_2(n) - 1}{\phi_3(n)}$$

for some n. If $x \in \mathbb{R}$ then we say that x belongs to the set $\mathbb{R}_c$ of computable reals if there exists a program[5] i_x for U so that, if f_{i_x} is the function computed by i_x then

$$\left| x \; - \; (-1)^{\phi_1(f_{i_x}(m))} \frac{\phi_2(f_{i_x}(m)) - 1}{\phi_3(f_{i_x}(m))} \right| < 2^{-m}$$

for every $m \in \mathbb{N}$. Penrose's question in this section should be:

> Given a program-description of a computable complex number, can we algorithmically determine whether it belongs to M, or not?

In other words, does there exist a (partial) function $G :\subseteq \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, computable in some sense, such that whenever i_x and i_y are programs for $x, y \in \mathbb{R}_c$ respectively then

$$G(i_x, i_y) = \chi_M(x + iy). \tag{2}$$

This simply means that G can be used to determine membership of M for the recursive points in the plane. Identify $\mathbb{C}$ with $\mathbb{R}^2$ in the usual way and consider $M_c = M \cap \mathbb{R}_c^2$, the computable points in the plane that belong to the Mandelbrot set. The unsuspecting reader of popular scientific literature could reasonably assume that given a full description of a point (x, y), by a program pair (i_x, i_y) one supposes there exists a procedure for deciding membership of M_c. The existence of such a G has been implicitly suggested to the general public for two decades by the pretty pictures we have had our computers draw, but it turns out that we do not have such a procedure in classical computability theory at all.

2.1 Markov computability

In the Russian school of constructive mathematics, pioneered by Andrei Markov, a function $f : \mathbb{R}_c \to \mathbb{R}_c$ is considered to be computable (or, constructive) if there exists a Turing machine computable $G : \mathbb{N} \to \mathbb{N}$ such that whenever i_x is a U-program for x then $G(i_x)$ is a U-program for $f(x)$—and $f(x)$ is defined if and only if $G(i_x)$ is. A set is computable or decidable in this setting when its characteristic function is computable. However, every computable function is continuous [12] and therefore the only computable sets will be closed (and open). This is a rather

[5] We shall assume that the *program* and *input* for U (actually both simply inputs) are natural numbers.

ironic development as, for example, the computable interval $[0,1] \cap \mathbb{IR}_c$ will not be a computable set in this sense, being closed but not open in $\mathbb{IR}_c$. The same is true of the unit circle and the disk in the plane and in both cases the difficulty springs from the fact that there is not general procedure that, given i_x and i_y, can decide whether $x = y$ or not. The Mandelbrot set is, however, closed in $\mathbb{IR}$ (and hence closed in $\mathbb{IR}_c$), so could it be computable in this sense? No, since $-2 \in M_c$ and -2 is a cluster point in $\mathbb{IR}_c$ of $\mathbb{IR}_c \setminus M_c$ and hence M_c is not open in $\mathbb{IR}_c$. Consequently its characteristic function cannot be Markov computable[6].

Although apparently intuitive, the notion of computable set used in this subsection is clearly very, very bad from the point of view of real analysis. Nevertheless it corresponds in some sense exactly to what a programmer would regard as computable: given a procedure (subroutine) for an x, having a program that outputs 1 in finite time if $x \in M_c$ and zero otherwise. The situation here is in contrast to that of subsets of the natural numbers, for which the notion of decidability is very natural and well-established (*pace* the entire field of 'super-Turing' hypercomputation).

2.2 Zeno Machine Computability

Consider what may be called a *Zeno machine* (ZM) [13] or Accelerated Turing Machine [14, 15]. With this kind of speed-up of the computing device, one can solve the halting problem for Turing machines in finite time. Many hypercomputational schemes tend to be proposals for somehow accomplishing infinitely many computational steps in a finite time (see also [16], for example). Without loss of generality, we shall assume a ZM to be identical to a Turing machine with one input tape, one output tape and a storage tape *except* that the ZM takes $\frac{1}{2}$ hour to execute the first transition, $\frac{1}{4}$ hour for the second, $\frac{1}{8}$ hour for the third etc. After one hour the ZM will have finished its operation and one will perhaps find the answer to some tantalising question on the output tape. On a putative ZM one could implement an Octave interpreter that would execute the code

```
n=1000;                          # For an nxn grid
c=meshgrid(linspace(-2,2,n))\    # Set up grid
+i*meshgrid(linspace(2,-2,n))';
x=zeros(n,n);                    # Initial value on grid
do
   x=x.^2+c;                     # Usual iteration
   x=x.*(min(abs(x),3)\          # Scale back points far away to
    ./(abs(x)+!abs(x)));         # speed up and avoid overflow
                                 # and infinite values
until (1==0)                     # Repeat a lot
imagesc(min(abs(x),2.1))         # Plot x, 2.1 counts as escape
```

[6] Since the unit circle is also not computable here, this should not come as a great surprise.

in finite time. This would provide an *exact* plot of the Mandelbrot set on the grid points! With a small modification (adding a procedure/subroutine for approximating the computable real) a ZM can decide membership of M_c for any computable real. For example, the Octave code immediately above can be rewritten for an (ordinary, Turing computable) Octave function $c(i)$ (instead of a matrix c) where $c(i)$ gives a rational approximation of c to within 2^{-i} and a scalar x, initially zero. In the i-th iteration of the loop we then recompute[7] x using $c(i)$. A similar calculation could be done relative to an oracle for the halting problem. The ZM as used here does not *apparently* present any of the problems with respect to defining the terminal configuration of the devices described in [13] since the matrix x is always bounded. This stability is however illusory: in executing the code described here one needs to continually reset x back to zero and therefore the variable x will, for every $x \in \mathbb{R}_c \setminus M_c$, have values alternatively 0 and with absolute value 2.1 arbitrarily close to the end of execution time[8].

2.3 A Rational Refuge?

Is there some relief from these problems if we restrict our attention to the points with rational coordinates only? Consider again the (recursive, surjective) encoding $\phi : \mathbb{N} \to \{0,1\} \times \mathbb{N}^2$ of the rational numbers used earlier. A set $A \subseteq \mathbb{Q}$ can be defined as computable whenever a Turing machine computable function $F : \mathbb{N} \to \mathbb{N}$ exists such that

$$F|_{\phi^{-1}(A)} \equiv \chi_A \circ \phi.$$

In this sense, now, the rational points on the unit circle do constitute a computable set since the condition $x^2 + y^2 = 1$ can be checked by a Turing machine for rational x and y. Is $M \cap \mathbb{Q}$ computable in *this* sense?

This kind of computability over the rationals is very different from Markov-computability over the computable reals. Consider for example the function $f : \mathbb{Q} \to \{0,1\}$ such that $f(q) = 1$ if the reduced improper fraction representation of q has an even denominator and $f(q) = 0$ otherwise. This function is not continuous on $\mathbb{Q}$ and therefore not the restriction of a Markov-computable function to the rationals. It is therefore conceivable[9] that a Turing machine could compute the characteristic function of $M \cap \mathbb{Q}$ with respect to the representation ϕ of the rationals, $F|_{\phi^{-1}(M \cap \mathbb{Q})}$.

The rational points are perhaps a bad basis for developing a general theory of computability of subsets of $\mathbb{R}^n$ however. It could say nothing much about the curve $x^3 + y^3 = 1$. In fact, decidability with respect to the rationals suffers from a general failure to take the boundary into account (as in the example below). Nevertheless, for connected and compact sets with non-empty interior,

[7] This would require recomputing the previous iterates, of course, but there are only finitely many to do each time.

[8] Thomson's Lamp showing the way...

[9] Although it strikes the present author as unlikely that M will be computable in this sense, a proof is called for.

computability in this sense seems a relatively natural and quite desirable property. It would, for instance, allow one to plot the set with a computer using test points on a rational grid. Consider also that the set $\{(x, y) \in \mathbb{Q}^2 \mid y \geq e^x\}$ is computable with respect to the representation ϕ: if $e_1(q, m)$ is a computable function approximation of e^q from below and $e_2(q, m)$ from above such that $\lim_{m\to\infty} e_i(q, m) = e^q$ then an enumeration of the values $e_1(q, m)$ and $e_2(q, m)$ for $m = 1, 2, \ldots$ will after finitely many steps reveal whether any given rational lies below e^q or above it (since e^q is irrational for all rational $q \neq 0$ and the case $q = 0$ can be checked separately).

3 *M* in Real Space

Let us return to the standard real numbers and consider M as a subset of the standard plane. In pursuing an answer to Penrose's question, the formulation of an appropriate notion of decidability of subsets is again required. Most of the work in this regard has its roots in the Polish school of computable analysis, starting with Andrzej Gregorczyk and Daniel Lacombe. The Blum-Shub-Smale system on the other hand, has a rather algebraic flavour.

3.1 The Blum-Smale Result

Blum, Mike Shub and Steven Smale (BSS) have introduced [2, 11] a model of computation over arbitrary commutative rings which is based on machines that operate using the elements of the ring R in lieu of a finite alphabet. There exist universal machines in this model. In the case where the ring is $\mathbb{Z}_2$ the classical computability theory is recovered. BSS-computable functions over R are functions computed by such a machine and they call a set computable or decidable whenever its characteristic function is BSS-computable. Blum and Smale have shown that the Mandelbrot set is NOT computable in this framework [17] which is at least a partial answer to Penrose's question. However, Vasco Brattka has shown [3] that in the BSS scheme over the field of standard real numbers the set $\{(x, y) \in \mathbb{R}^2 \mid y \geq e^x\}$ is not computable either. Brattka's result reflects unfavourably on the claim that BSS computability provides a natural notion of decidable set—at least, for real analysis.

3.2 Computable Real Analysis

Computable real analysis in the Polish school, as developed in [18] and elsewhere, is based on the definition of a function as computable whenever it maps every computable sequence of points (in $\mathbb{R}^n$) to a computable sequence of points and has a recursive modulus of continuity (defined on $\{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots\}$) on every compact subset. The appropriate definition [3] in this context is for a set A to be *computable* whenever its distance function $d_A : \mathbb{R}^n \to \mathbb{R}$ is a computable function in this sense. Computable (in the classical sense) subsets of $\mathbb{N}$, viewed as subsets of $\mathbb{R}$, remain computable in this sense [1] and the notion is therefore a

true generalisation of the notion of classical Turing computability. Peter Hertling has recently shown [1] that if the Mandelbrot set M is locally connected[10], then its distance function has to be computable and hence Penrose's question would be answered in the affirmative.

As described in [4], in this notion of computability the closed decidable subsets of $\mathbb{R}^n$ are *exactly* those sets which can in principle be plotted with arbitrary accuracy on a computer screen[11]. It is therefore at this point not yet known whether the Mandelbrot set can be accurately drawn by a computer!

4 Conclusion

The table below summarises the results and the two open problems mentioned in this survey.

	Circle	$y \geq e^x$	M'brot
Markov-computability over $\mathbb{R}_c$	×	×	×
Blum-Shub-Smale over $\mathbb{R}$	✓	×	×
Turing-computability over $\mathbb{Q}$	✓	✓	**?**
Computable analysis (Brattka e.a.)	✓	✓	**?**
Zeno-computability over $\mathbb{R}_c$	✓	✓	✓

Although the classical computability of the rational Mandelbrot set $M \cap \mathbb{Q}^2$ is an obvious question which one would cautiously expect to be answered in the negative, the author is not aware of a current result implying this. Among the models of decidability of sets in $\mathbb{R}^n$, the approach 3.2 studied by Brattka, Weihrauch e.a. appears the most reasonable and a demonstration of the computability of M in this setting would be extremely interesting both in itself as well providing strong support for the intuitiveness of their approach. In the model over $\mathbb{Q}$ the Mandelbrot set is of course decidable with respect to an oracle for the halting problem but it seems unlikely the converse is true, so if $M \cap \mathbb{Q}^2$ is not Turing-decidable then it could be interesting[12] to study problems that can be solved relative to an oracle for $M \cap \mathbb{Q}^2$ (or an M-oracle in any model in which M is not decidable). Until such time as at least one of the question marks in the table have been decided, a Zeno machine (or any hypercomputing model capable of solving the halting problem for Turing machines) remains—alas!—the best way of imagining that we can actually decide membership of the Mandelbrot set.

[10] Actually, Hertling proved a stronger result: that the hyperbolic conjecture (which would be implied by local connectedness) would be sufficient to prove imply computability of the distance function.

[11] Using an approach which switches a pixel *on* when the set is *close* to the centre of the pixel. This approach requires some obvious assumptions about the scale.

[12] This question was also raised by Klaus Meer and Martin Ziegler in slides for a talk, http://www.upb.de/cs/ag-madh/WWW/ziegler/LUEBECK2.pdf [accessed 2006-01-04].

References

1. Hertling, P.: Is the Mandelbrot set computable? MLQ Math. Log. Q. **51**(1) (2005) 5–18
2. Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and real computation. Springer-Verlag, New York (1998) With a foreword by Richard M. Karp.
3. Brattka, V.: The emperor's new recursiveness: the epigraph of the exponential function in two models of computability. In: Words, languages & combinatorics, III (Kyoto, 2000). World Sci. Publishing, River Edge, NJ (2003) 63–72
4. Brattka, V., Weihrauch, K.: Computability on subsets of Euclidean space. I. Closed and compact subsets. Theoret. Comput. Sci. **219**(1-2) (1999) 65–93 Computability and complexity in analysis (Castle Dagstuhl, 1997).
5. Penrose, R.: The emperor's new mind. The Clarendon Press Oxford University Press, New York (1989) Concerning computers, minds, and the laws of physics, With a foreword by Martin Gardner.
6. Mandelbrot, B.B.: The fractal geometry of nature. W. H. Freeman and Co., San Francisco, Calif. (1982) Schriftenreihe für den Referenten. [Series for the Referee].
7. Dewdney, A.K.: Computer Recreations: A computer microscope zooms in for a look at the most complex object in mathematics. Scientific American **253**(2) (1985) 17–21, 24
8. Branner, B.: The Mandelbrot set. In: Chaos and fractals (Providence, RI, 1988). Volume 39 of Proc. Sympos. Appl. Math. Amer. Math. Soc., Providence, RI (1989) 75–105
9. Ewing, J.: Can we see the mandelbrot set? The College Mathematics Journal (1995)
10. Fisher, Y., Hill, J.: Bounding the Area of the Mandelbrot Set (date unknown) Available online:- http://citeseer.ist.psu.edu/35134.html [accessed 2006-01-02].
11. Blum, L.: Computing over the reals: where Turing meets Newton. Notices Amer. Math. Soc. **51**(9) (2004) 1024–1034
12. Uspensky, V., Semenov, A.: Algorithms: main ideas and applications. Volume 251 of Mathematics and its Applications. Kluwer Academic Publishers Group, Dordrecht (1993) Translated from the Russian by A. Shen'.
13. Potgieter, P.H.: Zeno machines and hypercomputation. Theoretical Computer Science (accepted for publication) arXiv:cs/0412022.
14. Copeland, B.: Hypercomputation: philosophical issues. Theoretical Computer Science **317** (2004) 251–267
15. Boolos, G., Jeffrey, R.C.: Computability and logic. Cambridge University Press (1980)
16. Calude, C.S., Păun, G.: Bio-steps beyond Turing. Biosystems **77**(1–3) (2004) 175–194
17. Blum, L., Smale, S.: The Gödel incompleteness theorem and decidability over a ring. In: From Topology to Computation: Proceedings of the Smalefest (Berkeley, CA, 1990), New York, Springer (1993) 321–339
18. Pour-El, M.B., Richards, J.I.: Computability in analysis and physics. Perspectives in Mathematical Logic. Springer-Verlag, Berlin (1989)

Definability of the Field of Reals in Admissible Sets*

Vadim Puzarenko

Sobolev Mathematical Institute of SD RAS
Novosibirsk
vagrig@math.nsc.ru

Abstract. Admissible sets are used in Definability, Computability and Model theories. Last time they are actively used in studying of Computable Models. Here we study several problems on representability of field of real numbers in Admissible Sets. Furthermore, some computational methods on admissible sets are developed. Problems of definability of classical structures are studied since 1985[5]. In [5] some positive and negative answers of definability of fields $\mathbb{C}$ and $\mathbb{R}$ in hereditarily finite sets over sets in empty language and over dense linear ordering was got. In this paper we give full description of the property of $\mathbb{R}$ to be definable in $\mathbb{A}$.

We consider $\mathbb{R}$ as the field $\langle R, +, \cdot, 0, 1, \leq\rangle$ of reals. We propose that all the structures are considered in finite languages. Recall some notions from Admissible Sets theory [1, 2].

Let τ be a finite signature such that $\tau \supseteq \{U^1, \in^2, \varnothing\}$. Then theory KPU_τ (K, P, U mean Kripke, Platek, Urelements respectively) contains the following axioms: extensionality, pair, union, existence of empty set, set structure (any urelement is not a set), Δ_0 Separation and Δ_0 Collection. Here U, $\in$, $\varnothing$ are interpreted as set of urelements, membership relation and empty set respectively. It can be considered as some weak fragment of Zermelo-Fraenkel theory ZFU_τ with urelements. As in ZF, we can define notions of transitive sets and ordinals, transitive closure, ordered pair and so on. The set of all ordinals has the same properties except of well foundedness. This set is called the *ordinal* or the *height* of a considered structure. A structure $\mathbb{A}$ of KPU_τ is called admissible if its ordinal is well ordered. We denote the height of $\mathbb{A}$ as $\mathrm{Ord}(\mathbb{A})$.

Class of Σ formulas is the least one containing Δ_0 formulas and closed under $\vee$, $\wedge$, $\forall x \in y$, $\exists x \in y$ and $\exists x$. $S \subseteq \mathrm{dom}(\mathbb{A})^k$, $k \geqslant 1$, is called Σ *predicate* on $\mathbb{A}$ if it is definable by some Σ formula (possibly with parameters) on $\mathbb{A}$. $S \subseteq \mathrm{dom}(\mathbb{A})^k$, $k \geqslant 1$, is called Δ *predicate* on $\mathbb{A}$ if S, $\mathrm{dom}(\mathbb{A})^k \setminus S$ are Σ on $\mathbb{A}$. A partial function $f :\subseteq \mathrm{dom}(\mathbb{A})^k \to \mathrm{dom}(\mathbb{A})$, $k \geqslant 1$, is called Σ *function* on $\mathbb{A}$ if its graph is Σ on $\mathbb{A}$. This function is called *total* if its domain is $\mathrm{dom}(\mathbb{A})^k$. Given $a \in \mathrm{dom}(\mathbb{A})$, we denote the transitive closure and the support of a as $\mathrm{TC}(a)$ and $\mathrm{sp}(a) = \{u \in \mathrm{TC}(a) \mid U(u)\}$ respectively.

* This paper is partially supported by INTAS 05-109-4919

Notice that the set of all natural ordinals ω is Δ on any admissible structure. Moreover, either $\omega = \mathrm{Ord}(\mathbb{A})$ or $\omega \in \mathrm{dom}(\mathbb{A})$.

Let $\mathfrak{M}$ be a structure in signature σ. An admissible set is called a *hereditarily finite set over* $\mathfrak{M}$ if it is the least under inclusion admissible set in signature $\sigma \cup \{U, \in, \varnothing\}$ with urelements $\mathrm{dom}(\mathfrak{M})$. It is denoted as $\mathbb{HF}(\mathfrak{M})$. The domain of this admissible set can be defined constructively in the following way: $\mathrm{HF}_0(\mathrm{dom}(\mathfrak{M})) = \mathrm{dom}(\mathfrak{M})$; $\mathrm{HF}_{n+1}(\mathrm{dom}(\mathfrak{M})) = \mathrm{HF}_n(\mathrm{dom}(\mathfrak{M})) \cup \mathcal{P}_{fin}(\mathrm{HF}_n(\mathrm{dom}(\mathfrak{M})))$; $\mathrm{HF}(\mathrm{dom}(\mathfrak{M})) = \cup_{n<\omega}\mathrm{HF}_n(\mathrm{dom}(\mathfrak{M}))$ where $\mathcal{P}_{fin}(X)$ is collection of all finite subset of X.

Definition 1. Let $\mathbb{A}$, $\mathbb{B}$ be admissible sets and let $\mathfrak{M}$ be an arbitrary structure in signature σ.

- We say that $\mathfrak{M}$ is Σ-*definable* in $\mathbb{A}$ ($\mathfrak{M} \leqslant_\Sigma \mathbb{A}$) if there exists a surjection map $\nu : \mathrm{dom}(\mathbb{A}) \to \mathrm{dom}(\mathfrak{M})$ such that $\nu^{-1}(=)$ and $\nu^{-1}(P)$ are Δ on $\mathbb{A}$, for any $P \in \sigma$;
- We say that $\mathfrak{M}$ is *definable* in $\mathbb{A}$ ($\mathfrak{M} \leqslant \mathbb{A}$) if there exists a surjection map $\nu : \mathrm{dom}(\mathbb{A}) \to \mathrm{dom}(\mathfrak{M})$ such that $\nu^{-1}(=)$ and $\nu^{-1}(P)$ are definable by some formulas on $\mathbb{A}$, for any $P \in \sigma$;
- We say that $\mathbb{B}$ is Σ *reducible* to $\mathbb{A}$ ($\mathbb{B} \sqsubseteq_\Sigma \mathbb{A}$) if there exists a surjection map $\nu : \mathrm{dom}(\mathbb{A}) \to \mathrm{dom}(\mathbb{B})$ such that $\nu^{-1}(C)$ is Σ on $\mathbb{A}$, for any Σ predicate C on $\mathbb{B}$;
- A collection $S \subseteq \mathcal{P}(\mathrm{dom}(\mathbb{A}))$ is called *computable* in $\mathbb{A}$ if there exists a surjection map $\nu : \mathrm{dom}(\mathbb{A}) \to S$ such that $\{\langle a, b\rangle \mid b \in \nu(a)\}$ is Σ on $\mathbb{A}$;
- A collection $S \subseteq \mathcal{P}(\mathrm{dom}(\mathbb{A}))$ is called *definable* in $\mathbb{A}$ if there exists a surjection map $\nu : \mathrm{dom}(\mathbb{A}) \to S$ such that $\{\langle a, b\rangle \mid b \in \nu(a)\}$ is definable in $\mathbb{A}$.

All the notions except of Σ reducibilty were introduced by Yu.L.Ershov. Of course, in [2] it is proposed that ν can be a partial surjection map with a domain which is Σ on $\mathbb{A}$ but it is easy to get from the following assumption that the notions introduced here are equivalent to the original definitions.

Proposition 2. *Let $\mathbb{A}$ be a KPU_τ-model, for some signature τ. Then $B \subseteq \mathrm{dom}(\mathbb{A})$ is Σ subset iff $B = \varnothing$ or there exists a total Σ function f with range B.*

Proof. Let B be a nonempty Σ subset. Fix some $a_0 \in B$ and some Σ formula $\Phi(x)$ which defines B. By Σ Reflection Principle[1], there exists a Δ_0 formula Φ_0 such that $\mathrm{KPU}_\tau \vdash \Phi(x) \equiv \exists u \Phi_0(x, u)$. Then

$$f(x) = \begin{cases} z & \text{if } x = \langle z, u\rangle \text{ for some } u, \mathbb{A} \models \Phi_0(z, u); \\ a_0 & \text{otherwise} \end{cases}$$

has desired property.

The following assumption can be got as corollary of complicated constructions from [2]. Here we give direct proof.

Proposition 3. *Let $\mathbb{A}$ be an admissible set and $\mathfrak{M}$ be a structure in some finite signature. Then $\mathfrak{M} \leqslant_\Sigma \mathbb{A}$ iff $\mathbb{HF}(\mathfrak{M}) \sqsubseteq_\Sigma \mathbb{A}$.*

Proof. We assume that $\mathfrak{M}$ is infinite otherwise it is trivial. First we describe how to construct a representation of $\mathbb{HF}(\mathfrak{M})$ with domain $\subseteq \omega \times (\cup_{n<\omega}\mathrm{dom}(\mathbb{A})^n)$. Consider $\mathbb{HF}(\mathfrak{N})$ where $\mathfrak{N}$ is the standard model of arithmetics. It has a computable representation ν with the following additional property: there exists a computable function f s.t. $f(u)$ is Δ_0 index of some finite set $\{m \mid \nu(m) \in \nu(u)\}$. Now we associate with every $\varkappa \in \mathrm{HF}(\omega)$ some term $t_\varkappa$ in signature $\{\{\,\}^1, \cup^2, \varnothing\}$ such that $\langle \mathrm{TC}(\{\varkappa\}), U, \in, \varnothing\rangle \simeq \langle \mathrm{TC}(\{t_\varkappa(\overline{u})\}), U, \in, \varnothing\rangle$ for any tuple $\overline{u} \in \mathrm{dom}(\mathfrak{M})^{<\omega}$ of distinct elements of length $\mathrm{card}(\mathrm{sp}(\varkappa))$. The associated numbering of such terms is also computable. We correspond $\langle \varkappa, \overline{u}\rangle$ to $t_\varkappa(\overline{u})$. To complete proof it remains to apply proposition 1 and representation of Σ subsets by computable sequences of existentional formulas in signature of $\mathfrak{M}$ (Theorem 1[4]).

Indeed Σ reducibility preserves computable collections of sets of natural numbers.

Lemma 4. *Let $\mathbb{A}$, $\mathbb{B}$ be admissible sets that $\mathbb{A} \sqsubseteq_\Sigma \mathbb{B}$ and let ν be a map from definition. Then $R_0 = \{\langle x, n\rangle \mid x \in \mathrm{dom}(\mathbb{B}),\ n \in \omega,\ \nu(x) = n\}$ is Δ on $\mathbb{B}$.*

Proof. We define this predicate by Σ recursion:
$\langle x, 0\rangle \in R$ iff $\nu(x) = \varnothing$ iff $\neg(\nu(x) \neq \varnothing)$;
$\langle x, n+1\rangle \in R_0$ iff $\exists y[\langle y, n\rangle \in R_0 \wedge (\nu(x) = \nu(y) + 1)]$;
$\langle x, n+1\rangle \notin R_0$ iff $\neg\mathrm{Nat}(\nu(x)) \vee (\nu(x) = \varnothing) \vee \exists y(\langle y, n\rangle \notin R_0 \wedge (\nu(x) = \nu(y)+1))$
where Nat is set of all natural ordinals.

Proposition 5. *Let $\mathbb{A}$, $\mathbb{B}$ be admissible sets that $\mathbb{A} \sqsubseteq_\Sigma \mathbb{B}$. Then any computable in $\mathbb{A}$ collection $\subseteq \mathcal{P}(\omega)$ is also computable in $\mathbb{B}$.*

Proof. Let $\mathbb{A}$ be Σ reducible to $\mathbb{B}$ via ν. Let $\varnothing \neq S \subseteq \mathcal{P}(\omega)$ be computable in $\mathbb{A}$ and let Q be Σ on $\mathbb{A}$ that $S = \{\{n \mid Q(a, n)\} \mid a \in \mathrm{dom}(\mathbb{A})\}$. Then $\nu^{-1}(Q)$ is Σ on $\mathbb{B}$. It is easy to verify that $S = \{\{n \mid \exists y(R_0(y, n) \wedge Q(\nu(x), \nu(y)))\} \mid x \in \mathrm{dom}(\mathbb{B})\}$. Thus S is computable in $\mathbb{B}$.

Similarly, we can get familiar results for definable structures and collections.

Proposition 6. *1. Let $\mathbb{A}$ be admissible and let $\mathfrak{M}$ be a structure. Then $\mathfrak{M} \leqslant \mathbb{A}$ iff $\mathbb{HF}(\mathfrak{M}) \leqslant \mathbb{A}$.*

2. Let $\mathbb{A}$, $\mathbb{B}$ be admissible sets that $\mathbb{A} \leqslant \mathbb{B}$ and let $S \subseteq \mathcal{P}(\omega)$ be definable in $\mathbb{A}$. Then S is definable in $\mathbb{B}$.

This technique allows to describe all the admissible sets in which the field of real numbers is definable. $\mathbb{R}$ is Archimedian field so the set of natural number $N \subseteq \mathrm{dom}(\mathbb{R})$ is Δ on $\mathbb{HF}(\mathbb{R})$. E.g., $N \subseteq \mathrm{dom}(\mathbb{C})$ is Σ but is not Δ on hereditarily finite set $\mathbb{HF}(\mathbb{C})$ over field of complex numbers. Furthermore, the natural correspondence between $\mathrm{Ord}(\mathbb{HF}(\mathbb{R}))$ and N is Σ function on $\mathbb{HF}(\mathbb{R})$.

Theorem 7. *[3] Let $\mathbb{A}$ be an admissible set. Then $\mathbb{R}$ is definable in $\mathbb{A}$ iff $\mathcal{P}(\omega)$ is definable in $\mathbb{A}$.*

PROOF. First we show that collection of all coinfinite sets of natural numbers is computable in $\mathbb{HF}(\mathbb{R})$. For any $x \in [0;2)$ we let $A_x = \{n \in N \mid [2 \cdot \{2^{n-1} \cdot x\}] = 1\}$ where $[x]$ and $\{x\}$ are integer and fractional parts of x respectively. It is evident that the collection $\{A_x \mid x \in [0;2)\}$ contains exactly coinfinite sets of natural numbers. Furthermore, collection of all cofinite sets of all natural numbers is also computable in $\mathbb{HF}(\mathbb{R})$. Thus if $\mathbb{R} \leqslant \mathbb{A}$ then $\mathcal{P}(\omega)$ is definable in $\mathbb{A}$, by Proposition 6. Conversely, if $\mathcal{P}(\omega)$ is definable in $\mathbb{A}$ then the collection of all coinfinite sets of natural numbers is also definable in $\mathbb{A}$. We will represent real numbers by cut of ordered ring $\mathcal{K} = \{\frac{n}{2^m} \mid n \in Z, m \in N\}$ (here Z is set of all integer numbers). We fix some its constructivization ν. Now we can define *cuts* of α as follows:

$B_\alpha = \{n \mid \nu(n) \leqslant \alpha\}$; $C_\alpha = \{n \mid \nu(n) > \alpha\}$;

$B^1_\alpha = \{n \mid \nu(n) < \alpha\}$; $C^1_\alpha = \{n \mid \nu(n) \geqslant \alpha\}$.

Then $(B_\alpha | C_\alpha) = (B^1_\alpha | C^1_\alpha)$ iff $\alpha \notin \mathcal{K}$. Furthermore, A_x is finite iff $x \in \mathcal{K}$. It allows to define addition and multiplication operations uniformly in terms of cuts. □

Corollary 8. *Let $\mathfrak{M}$ be a structure of categorical in some infinite power theory. Then $\mathbb{R}$ is not definable in $\mathbb{HF}(\mathfrak{M})$.*

Proof. Any countable structure of such theory T has a computable copy relative to oracle T. Hence the collection of all possible definable subsets of natural numbers in $\mathbb{HF}(\mathfrak{M})$ is countable.

Corollary 9. *[2] Let $\mathfrak{M}$ be a structure of a theory from the following list: dense linear order, sets without any structures, algebraically closed fields. Then $\mathbb{R}$ is not definable in $\mathbb{HF}(\mathfrak{M})$.*

Corollary 10. *[3] Let $\mathbb{A}$, $\mathbb{A}'$ be admissible sets such that $\mathbb{A} \preccurlyeq \mathbb{A}'$ and $\mathbb{R}$ is definable in $\mathbb{A}$. Then $\mathbb{R}$ is definable in $\mathbb{A}'$.*

The problem of Σ-definability of $\mathbb{R}$ in admissible sets is more complicated. Here we give a necessary condition for $\mathbb{R}$ to be Σ-definable in $\mathbb{A}$. A surjection map $\nu : \mathrm{dom}(\mathbb{A}) \to S$ is called *decidable in* $\mathbb{A}$ if $\{\langle a, b\rangle \mid \nu(a) = \nu(b)\}$ is Δ on $\mathbb{A}$. It is easy to verify that if $\mathbb{A} \sqsubseteq_\Sigma \mathbb{B}$ via μ and ν is decidable in $\mathbb{A}$ then $\nu \circ \mu$ is decidable in $\mathbb{B}$.

Proposition 11. *Let $\mathbb{A}$ be an admissible set such that $\mathbb{R}$ is Σ-definable in $\mathbb{A}$. Then $\mathbb{A}$ has a decidable map ν with range $\{A \subseteq \omega \mid \mathrm{card}(\omega \setminus A) = \infty\}$ such that $\{\langle x, n\rangle \mid n \in \nu(x), x \in \mathrm{dom}(\mathbb{A})\}$ is Δ on $\mathbb{A}$.*

Proof. This implies from Propositions 3,5 and remarks above.

In [6] an admissible set $\mathbb{A}$ is constructed with the following properties:

1) $\mathcal{P}(\omega)$ is computable in $\mathbb{A}$;

2) it has no decidable map for $\mathcal{P}(\omega)$ with additional condition from proposition.

Thus $\mathbb{R}$ is definable but is not Σ-definable in $\mathbb{A}$.

Now we give the full description of Σ-definability of $\mathbb{R}$ when the height of admissible sets is greater than ω. Indeed a height of such admissible set is $\geqslant \omega_1$.

Proposition 12. *There exists the least under inclusion admissible set* $\mathbb{C}_\omega$ *with the property* $\mathcal{P}(\omega) \subseteq \mathrm{dom}(\mathbb{C}_\omega)$. *Moreover, if an admissible* $\mathbb{A}$ *contains* $\mathcal{P}(\omega)$ *as subset then* $\mathbb{C}_\omega \leqslant_{\mathrm{end}} \mathbb{A}$ *and* $\mathbb{C}_\omega \sqsubseteq_\Sigma \mathbb{A}$.

Proof. Let $\mathbb{C}_\omega = \cup_{A \subseteq \omega} \mathbb{L}(\omega_1, A)$ where $\mathbb{L}(\alpha, a)$ is the set containing exactly elements which are constructible from a to step α. Then it is admissible. Furthermore, for any $\alpha < \omega_1$ there exists $A \subseteq \omega$ such that the least admissible ordinal of structure of kind $\mathbb{L}(\gamma, A)$ is greater than α.

Theorem 13. *Let* $\mathbb{A}$ *be an admissible set such that* $\omega \in \mathrm{Ord}(\mathbb{A})$. *Then* $\mathbb{R}$ *is* Σ*-definable in* $\mathbb{A}$ *iff* $\mathcal{P}(\omega) \subseteq \mathrm{dom}(\mathbb{A})$.

Proof. If $\mathbb{R}$ is Σ-definable in $\mathbb{A}$ and $\omega \in \mathrm{dom}(\mathbb{A})$ then for any $A \subseteq \omega$, $A \in \mathrm{dom}(\mathbb{A})$, by Δ Separation[1] and Proposition 5. Conversely, functionals defining operations in proof of Theorem 1 have quantifiers acting only on natural numbers. They can be restricted by ω.

Let $\aleph$ be an arbitrary cardinal. We denote $\mathcal{H}_\aleph(\mathfrak{M})$ as the admissible set over $\mathfrak{M}$ consisting of elements x such that $|\mathrm{TC}(x)| < \aleph$. Let $\mathcal{H}_\aleph$ be its pure part. In the case when $\aleph = \omega_1$ is the first uncountable cardinal, $\mathcal{H}_\aleph(\mathfrak{M})$ is called hereditarily countable set over $\mathfrak{M}$ and is denoted as $\mathbb{HC}(\mathfrak{M})$. Recall that $\mathcal{H}_\omega(\mathfrak{M}) = \mathbb{HF}(\mathfrak{M})$.

Corollary 14 (2). *Let* $\aleph > \omega$ *be a cardinal. Then* $\mathbb{R}$ *is* Σ*-definable in* $\mathcal{H}_\aleph(\mathfrak{M})$, *for any structure* $\mathfrak{M}$.

The description of admissible sets $\mathbb{A}$ such that $\omega = \mathrm{Ord}(\mathbb{A})$ and $\mathbb{R} \leqslant_\Sigma \mathbb{A}$ is open problem. It is possible that it has set theoretical nature.

References

1. J. Barwise. Admissible Sets and Structures. Springer–Verlag, Berlin, Göttingen, Heidelberg, 1975.
2. Yu.L.Ershov. Definability and Computability, New York, NY: Consultants Bureau. 1996.
3. V.G.Puzarenko. Generalized Numberings and Definability of the Field $\mathbb{R}$ in Admissible Sets, Vestnik NGU, vol. **3**(2), 2003, 107 – 117 (in Russian).
4. V.G.Puzarenko. On computability over models of decidable theories, Algebra and logic, vol. **39**, 2(2000), 98 – 113.
5. 4. Yu.L.Ershov. Σ-definability in Admissible Sets. Dokl. AN SSSR, **285**, 4(1985), 792 – 795.
6. V.G.Puzarenko. Computability in special models. Siberian Mathematical Journal, **46**, 1(2005), 148 – 165.

The Algebra of Labeled Forests Modulo Homomorphic Equivalence

Victor L. Selivanov *

A.P.Ershov Institute of Informatics Systems
Siberian Division of the Russian Academyof Sciences
vseliv@nspu.ru

Abstract. We introduce and study some natural operations on the homomorphic quasiorder of finite labeled forests which is of central interest for extending the difference hierarchy to the case of partitions. It is shown that the corresponding algebra is the simplest nontrivial semilattice with discrete closures. The algebra is also characterized as a free algebra in some quasivariety. Some of results are generalized to countable labeled forests without infinite chains.
Keywords. Tree, forest, labeled tree, homomorphic quasiorder, lattice, well quasiorder.

1 Introduction

In [Se04] we studied the structure $(\mathcal{F}_k;\leq)$, $k<\omega$, of finite k-labeled forests with the homomorphic quasiorder (earlier the structure was used as a tool for investigation of the discontinuity degrees of functions [H96]). The structure is interesting in its own right since the homomorphic quasiorder is one in a series of relations on words, trees and forests relevant to computer science (see [Ku06] and references therein). Our original interest to this structure was motivated by its close relationship to the Boolean hierarchy of k-partitions which we now recall briefly. Throughout this paper, k denotes an arbitrary integer, $k\geq 2$, which is identified with the set $\{0,\dots,k-1\}$.

Let M be a set, $P(M)$ be the class of subsets of M, and $\mathcal{L}\subseteq P(M)$ be a class of subsets closed under $\cup,\cap$ and containing $\emptyset, M$; for the sake of brevity, we call such a class $\mathcal{L}$ *a base*. As is well-known, there is a natural classification (called the Boolean hierarchy over $\mathcal{L}$) of elements of the Boolean algebra generated by $\mathcal{L}$ inside $P(M)$. The class $P(M)$ is in a natural bijective correspondence with the set 2^M of all functions $\nu: M\to 2=\{0,1\}$. In [Ko00,KW00,Ko05], a so called refined *Boolean hierarchy of k-partitions over* $\mathcal{L}$ $BH_k^*(\mathcal{L})$ was introduced that extends the Boolean hierarchy of sets to the case of k-partitions of M (i.e., pairwise disjoint sets $A_0,\dots,A_{k-1}$ with $A_0\cup\dots\cup A_{k-1}=M$). In [Ko00,KW00] it was also shown that the structure $(BH_k^*(\mathcal{L});\subseteq)$ is always a homomorphic image of the structure $(\mathcal{P}_k;\leq)$ of finite k-labeled posets with the homomorphic quasiorder. In

* Partially supported by a DAAD project within the program "Ostpartnerschaften".

[Se04] we have shown that for some important bases $\mathcal{L}$ the structure $(BH_k^*(\mathcal{L}); \subseteq)$ is actually equivalent to $(\mathcal{F}_k; \leq)$.

In this paper, we investigate the algebraic structure of $\mathcal{F}_k$ enriched by some natural operations. We will show that the corresponding algebra occupies a remarkable place among the so called semilattices with discrete closures introduced in our previous work on complete numberings and fine hierarchies of the arithmetical sets and functions [Se82,Se83].

In Section 2 we cite necessary definitions and results from [Se04] and make a couple of additional remarks. In Section 3 we define the operations on $\mathcal{F}_k$ and establish main properties of the corresponding algebra. In Section 4 we prove that the algebra is in a sense the simplest semilattice with discrete closures, while in Section 5 we show that it is an initial object in a suitable quasivariety. In Section 6 we generalize some results to countable forests without infinite chains and show that the corresponding quasiorder extends the classical Hausdorff difference hierarchy of sets in the Baire space to the case of k-partitions. We conclude in Section 7 with mentioning a possible future work.

2 $\mathcal{F}_k$ As a Lattice

Here we recall some necessary definitions and results from [Se04] and make a couple of remarks. We use some standard notation and terminology on posets which may be found in any book on the subject, see e.g. [DP94]. We will not be very cautious when applying notions about posets also to quasiorders (known also as preorders); in such cases we mean the corresponding quotient-poset of the quasiorder.

A poset $(P; \leq)$ will be often shorter denoted just by P (this applies also to structures of other signatures in place of $\{\leq\}$). Any subset of P may be considered as a poset with the induced partial ordering. In particular, this applies to the "cones" $\check{x} = \{y \in P | x \leq y\}$ and $\hat{x} = \{y \in P | y \leq x\}$ defined by any $x \in P$.

By *a forest* we mean a poset in which every upper cone $\check{x}$ is a chain. *A tree* is a forest having the greatest element (called *the root* of the tree). Note that any finite forest is uniquely representable as a disjoint union of trees, the roots of the trees being the maximal elements of the forest. *A proper forest* is a forest which is not a tree. Notice that our trees and forests "grow top down", contrary to the natural ones. The reason is that this is a bit more comfortable when dealing with the Boolean hierarchies of partitions.

A *k-labeled poset* (or just a k-poset) is an object $(P; , \leq, c)$ consisting of a poset $(P; \leq)$ and a labeling $c : P \to k$. Sometimes we simplify notation of a k-poset to (P, c) or even to P. A *morphism* $f : (P; \leq, c) \to (P'; \leq', c')$ between k-posets is a monotone function $f : (P; \leq) \to (P'; \leq')$ respecting the labelings, i.e. satisfying $c = c' \circ f$.

Let $\mathcal{P}_k$, $\mathcal{F}_k$ and $\mathcal{T}_k$ be the classes of all finite k-posets, finite k-forests and finite k-trees, respectively. Define [Ko00,KW00] a quasiorder $\leq$ on $\mathcal{P}_k$ as follows: $(P, c) \leq (P', c')$, if there is a morphism from (P, c) to (P', c'). By $\equiv$ we denote the equivalence relation on $\mathcal{P}_k$ induced by $\leq$. For technical reasons we consider

also the empty k-forest $\emptyset$ (which is not assumed to be a tree) assuming that $\emptyset \leq P$ for any $P \in \mathcal{P}_k$. Note that in this paper (contrary to [Se04]) we assume that $\emptyset \in \mathcal{F}_k$. In [Se04] we have given a description of finite minimal k-forests, i.e. k-forests not equivalent (under $\equiv$) to a k-forest of lesser cardinality; this description will be once used below.

Recall [Kr60,Kr72] that a quasiorder is called *a well quasiorder (wqo)* if it has neither infinite descending chains nor infinite antichains. Any wqo P has *a rank* $r(P)$ which is the greatest ordinal isomorphically embeddable into P. With any quasiorder we associate also its *width* $w(P)$ defined as follows: if P has antichains with any finite number of elements, then $w(P) = \omega$, otherwise $w(P)$ is the greatest natural number n for which P has an antichain with n elements.

For arbitrary finite k-trees $T_0, \ldots, T_n$, let $F = T_0 \sqcup \cdots \sqcup T_n$ be their join, i.e. the disjoint union. Then F is a k-forest whose trees are exactly $T_0, \ldots, T_n$. Of course, every finite k-forest is (equivalent to) the join of its trees. Note that the join operation applies also to finite k-forests, and the join of any two k-forests is clearly their supremum under $\leq$. Hence, $(\mathcal{F}_k; \leq)$ is an upper semilattice. The next result cites some more facts established in [Se04].

Proposition 1. *(i) For any $k \geq 2$, $(\mathcal{F}_k; \leq)$ is a wqo with $r(\mathcal{F}_k) = \omega$.*

(ii) $w(\mathcal{F}_2) = 2$ and $w(\mathcal{F}_k) = \omega$ for $k > 2$.

(iii) For any $k \geq 2$, the quotient structure of $(\mathcal{F}_k; \leq)$ is a distributive lattice.

(iv) The finite k-trees define exactly the non-empty join-irreducible elements of the lattice $(\mathcal{F}_k; \leq)$.

For any $i < k$, let $\mathcal{T}_k^i$ be the set of finite k-trees the roots of which carry the label i. Let us formulate some more properties of the introduced sets.

Proposition 2. *(i) $(\mathcal{T}_k^0, \ldots, \mathcal{T}_k^{k-1})$ is a partition of $\mathcal{T}_k$ modulo $\equiv$.*

(ii) For all $i, j < k$, $(\mathcal{T}_k^i; \leq)$ is isomorphic to $(\mathcal{T}_k^j; \leq)$. Moreover, there is an automorphism of $(\mathcal{F}_k; \leq)$ sending $\mathcal{T}_k^i$ onto $\mathcal{T}_k^j$.

Proof. (i) We have to show that if $(S, c) \in \mathcal{T}_k^i$, $(T, d) \in \mathcal{T}_k^j$ and $i \neq j$ then $S \not\equiv T$. Suppose the contrary, then there are morphisms $\phi : (S, c) \to (T, d)$ and $\psi : (T, d) \to (S, c)$. From $i \neq j$ it easily follows that $s > \psi\phi(s) > \psi\phi\psi\phi(s) > \cdots$, where s is the root of S. This contradicts to finiteness of S.

(ii) Let $\pi : k \to k$ be a permutation of k such that $\pi(i) = j$. The function $(P; \leq, c) \mapsto (P; \leq, \pi \circ c)$ induces an automorphism of $(\mathcal{F}_k; \leq)$ (and of $(\mathcal{P}_k; \leq)$) sending $\mathcal{T}_k^i$ onto $\mathcal{T}_k^j$. This completes the proof.

From the last proposition we immediately obtain the following

Corollary 1. *For every $i < k$, $\mathcal{T}_k^i$ is not definable in $(\mathcal{T}_k; \leq)$ (as well as in $(\mathcal{F}_k; \leq)$ and $(\mathcal{P}_k; \leq)$) by a formula without parameters.*

Our interest to the sets $\mathcal{F}_k$, $\mathcal{T}_k$ and $\mathcal{T}_k^i$ is explained by the above-mentioned relation to the Boolean hierarchy of k-partitions. Namely, for $k = 2$ the set $\mathcal{T}_k^0$ ($\mathcal{T}_k^1$, $\mathcal{F}_k \setminus \mathcal{T}_k$) corresponds to the collection of Σ-levels (respectively of Π- and Δ-levels) of the Boolean hierarchy of sets. For $k > 2$ and $i < k$ the sets $\mathcal{T}_k^i$ and $\mathcal{F}_k \setminus \mathcal{T}_k$ correspond to generalizations of Σ- (and Π-) levels and of Δ-levels of the Boolean hierarchy of k-partitions.

3 $\mathcal{F}_k$ As a *dc*-Semilattice

In this section, we enrich the set $\mathcal{F}_k$ by natural unary operations which make of $\mathcal{F}_k$ a *dc*-semilattice in the sence of the following notion introduced in [Se82].

Definition 1. *By a semilattice with discrete closures of rank k (a dc-semilattice for short) we mean a structure $(S;\cup,p_0,\dots,k_{k-1})$ satisfying the following axioms:*

1) $(S;\cup)$ is an upper semilattice, i.e. it satisfies $(x\cup y)\cup z = x\cup(y\cup z)$, $x\cup y = y\cup x$ and $x\cup x = x$; as usual, by $\leq$ we denote the induced partial order on S defined by $x\leq y$ iff $x\cup y = y$.

2) Every p_i, $i<k$, is a closure operation on $(S;\leq)$, i.e. it satisfies $x\leq p_i(x)$, $x\leq y\to p_i(x)\leq p_i(y)$ and $p_i(p_i(x))\leq p_i(x)$.

3) The operations p_i have the following discreteness property: for all distinct $i,j<k$, $p_i(x)\leq p_j(y)\to p_i(x)\leq y$.

4) Every $p_i(x)$ is join-irreducible, i.e. $p_i(x)\leq y\cup z\to(p_i(x)\leq y\vee p_i(x)\leq z)$.

The main observation of this section is the following

Theorem 1. *There exist unary operations $p_0\dots,p_{k-1}$ on $\mathcal{F}_k$ such that the quotient structure of $(\mathcal{F}_k;\sqcup,p_0\dots,p_{k-1})$ is a dc-semilattice.*

Proof. For every finite k-forest F and every $i<k$, let $p_i(F)$ be the k-tree obtained from F by joining a new greatest element and assigning the label i to this element. In particular, $p_i(\emptyset)$ will be the singleton tree carrying the label i. It is straightforward to check that the operations $p_0\dots,p_{k-1}$ respect the homomorphic equivalence on $\mathcal{F}_k$, and the corresponding operations on the quotient structure (which are denoted for simplicity again by $p_0\dots,p_{k-1}$) make it a *dc*-semilattice. This completes the proof.

Remarks. 1. The operations $\sqcup,p_0\dots,p_{k-1}$ extended in the natural way to the set $\mathcal{P}_k$ witness that the quotient structure $(\mathcal{P}_k;\sqcup,p_0\dots,p_{k-1})$ is a *dc*-semilattice as well.

2. Corollary 1 implies that the operations $p_0,\dots,p_{k-1}$ are not first order definable without parameters in $(\mathcal{F}_k;\leq)$.

3. The algebra $(\mathcal{F}_k;\sqcup,p_0\dots,p_{k-1})$ is generated by the empty tree $\emptyset$.

Next we use Theorem 1 to obtain additional information on the sets $\mathcal{T}_k^i$ introduced in the previous section.

Theorem 2. *(i) For every $i<k$, the quotient structure of $(\mathcal{T}_k^i;\leq)$ is a distributive lattice the non-zero join-irreducible elements of which are exactly the elements $p_ip_j(x)$, where $x\in\mathcal{F}_k$ and $j<k$, $j\neq i$.*

(ii) For all $k>1$ and $i<k$, $r(\mathcal{T}_k^i)=\omega$.

(iii) $w(\mathcal{T}_2^0)=1$ and $w(\mathcal{T}_k^i)=\omega$ for all $k>2$ and $i<k$.

Proof. The assertions (ii) and (iii) are easy, so we check only (i). From Theorem 1 it follows that for all $x,y\in\mathcal{T}_k^i$ the elements $p_i(x\sqcup y)$ and $p_i(x\sqcap y)$ (where $\sqcap$ is the infimum operation in $\mathcal{F}_k$) are respectively the supemum and infimum of x,y in $\mathcal{T}_k^i$.

As is well known, for proving distributivity it suffices to show that for all $x, y, z \in \mathcal{T}_k^i$ if $x \leq p_i(y \sqcup z)$ then there are $y', z' \in \mathcal{T}_k^i$ with $y' \leq y$, $z' \leq z$ and $x = p_i(y' \sqcup z')$. Let x be defined by a minimal tree $T \in \mathcal{T}_k^i$, then $T = p_i(T_0 \sqcup \cdots \sqcup T_n)$ for some $T_0 \sqcup \cdots \sqcup T_n \in \mathcal{T}_k \setminus \mathcal{T}_k^i$ (w.l.o.g. we assume that x is distinct from the least element $p_i(\emptyset)$ of $\mathcal{T}_k^i$). Then for every $j \leq n$ we have $T_j \leq p_i(y \sqcup z)$, hence $T_j \leq y \sqcup z$ and therefore $T_j \leq y$ or $T_j \leq z$. Then the trees $y' = p_i(\sqcup\{T_j | j \leq n, T_j \leq y\})$ and $z' = p_i(\sqcup\{T_j | j \leq n, T_j \leq z\})$ have the desired properties.

Now we check that elements of the form $p_i p_j(x)$, where $x \in \mathcal{F}_k$ and $j < k$, $j \neq i$, are join-irreducible in $\mathcal{T}_k^i$. Let $p_i p_j(x) \leq p_i(y \sqcup z)$, where $y, z \in \mathcal{T}_k^i$ (and so $y = p_i(y)$ and $z = p_i(z)$). Then $p_j(x) \leq p_i(y \sqcup z)$, hence $p_j(x) \leq y \sqcup z$. Then $p_j(x) \leq y$ or $p_j(x) \leq z$ and therefore $p_i p_j(x) \leq p_i(y) = y$ or $p_i p_j(x) \leq z$. Thus, $p_i p_j(x)$ is join-irreducible in $\mathcal{T}_k^i$.

Conversely, let x be a non-zero join-irreducible element of $(\mathcal{T}_k^i; \leq)$, so $x \not\equiv p_i(\emptyset)$. We have to show that x is of the form $p_i p_j(y)$ for some y and $j \neq i$. Let T be a minimal tree equivalent to x, then $T = p_i(F)$ for a non-empty minimal forest F (see description of minimal sets and forests in [Se04]), and therefore $F = T_0 \sqcup \cdots \sqcup T_n$ for some pairwise incomparable minimal trees $T_0, \ldots, T_n \in \mathcal{T}_k \setminus \mathcal{T}_k^i$. The case $n > 0$ is actually impossible because in this case the elements $p_i(T_0), \ldots, p_i(T_n)$ were pairwise incomparable and $x = p_i(p_i(T_0) \sqcup \cdots \sqcup p_i(T_n))$ were join-reducible in $(\mathcal{T}_k^i; \leq)$. By Proposition 2(i), $T_0 = p_j(y)$ for some $j < k$, $j \neq i$, and $y \in \mathcal{F}_k$. Therefore, $x = p_i p_j(y)$ which completes the proof of the theorem.

Remark. It is easy to see that the lattices $\mathcal{T}_k^i$ and $\mathcal{F}_l$ are not isomorphic (even not elementarily equivalent) for all $k \geq 2$ and $l \geq 1$.

Next we show that, similar to Theorem 1, it is possible to equip $\mathcal{T}_k^i$ with the structure of a *dc*-semilattice. To simplify notation a bit, we consider only the case $i = 0$. By Proposition 2(ii), this causes no loss of generality. The supremum operation in $\mathcal{T}_k^0$ is denoted by $\cup$.

Theorem 3. *For every $k \geq 2$, there exist unary operations $q_1, \ldots, q_{k-1}$ on $\mathcal{T}_k^0$ such that the quotient structure of $(\mathcal{T}_k^0; \cup, q_1, \ldots, q_{k-1})$ (where $\cup$ denotes the supremum operation on $\mathcal{T}_k^0$) is a dc-semilattice of rank $k-1$.*

Proof. For every finite k-tree $T \in \mathcal{T}_k^0$ and every $i, 0 < i < k$, let $q_i(T) = p_0 p_i(T')$, where T' is the k-subforest of $(T; \leq)$ obtained from T by removing all elements $x \in T$ such that every $y \geq x$ carries the label 0. In particular, $q_i(p_0(\emptyset)) = p_0 p_i(\emptyset)$. It is straightforward to check that the operations $q_1 \ldots, q_{k-1}$ respect the homomorphic equivalence on $\mathcal{T}_k^0$, and the corresponding operations on the quotient structure make it a *dc*-semilattice. This completes the proof.

We conclude this section with stating an interesting property of the structure $(\mathcal{T}_k; \leq)$ in terms of a notion introduced in [Se79,Se82].

Corollary 2. *The quotient structure of $(\mathcal{T}_k; \leq)$ is a discrete weak semilattice of rank k. This means that for every finite sequence $x_0, \ldots, x_n \in \mathcal{T}_k$ there exist $u_0, \ldots, u_{k-1} \in \mathcal{T}_k$ with the following properties:*

(i) $\forall i \leq n \forall j < k (x_i \leq u_j)$;

(ii) for every $x \in \mathcal{T}_k$, $\forall i \leq n(x_i \leq x) \to \exists j < k(u_j \leq x)$;
(iii) for every $x \in \mathcal{T}_k$, $\forall j < k(x \leq u_j) \to \exists i \leq n(x \leq x_i)$.

To see this it suffices to set $u_j = p_j(x_0 \sqcup \cdots \sqcup x_n)$, $j < k$, and apply Theorem 1. Note that if there is no greatest element in $(\{x_0, \ldots, x_n\}; \leq)$ then the elements $u_0, \ldots, u_{k-1}$ are pairwise incomparable.

Properties (i) and (ii) mean that $\{u_0, \ldots, u_{k-1}\}$ is in a sense a weak supremum of $\{x_0, \ldots, x_n\}$, so the structure $(\mathcal{T}_k; \leq)$ resembles an upper semilattice. On the other hand, the structure is opposite to upper semilattices in the sense that if a set $\{x_0, \ldots, x_n\}$ does not have a greatest element then it has no supremum. Note also that $\{x_0, \ldots, x_n\}$ is a weak infimum of $\{u_0, \ldots, u_{k-1}\}$, thus any of these sets is definable through the other.

4 The Minimality of $\mathcal{F}_k$

There are several natural examples of *dc*-semilattices, e.g. the semilattice of numberings with completing operations [Se82,Se83] and the semilattice of Bairings with Wadge reducibility and suitable closure operations [Se04]. So it might be of interest to understand better the structure of *dc*-semilattices. Results of this section show that the *dc*-semilattice discussed in the last section is in a sense the simplest nontrivial *dc*-semilattice.

Theorem 4. *Let* $(S; \cup, p_0, \ldots, p_{k-1})$ *be a dc-semilattice and* a *its element such that* $a < p_i(a)$ *for all* $i < k$. *Then the subalgebra* (a) *of* S *generated by* a *is isomorphic to the quotient structure of* $(\mathcal{F}_k; \sqcup, p_0, \ldots, p_{k-1})$.

Proof. With any finite k-forest F we associate an element $f(F) \in (a)$ by induction on the number of elements $|F|$ in F as follows:

$f(\emptyset) = a$;
if $F = T_0 \sqcup \cdots \sqcup T_n$ is a proper forest with trees $T_0, \ldots, T_n$ then $f(F) = f(T_0) \cup \cdots \cup f(T_n)$;
if $F = p_i(G)$ is a tree then $f(F) = p_i(f(G))$.

Let us check that for all finite k-forests F and G we have $F \leq G$ iff $f(F) \leq f(G)$. In the case $F = \emptyset$ we have to show that $a \leq t$, where t is the value in S of some term of signature $\tau = \{\cup, p_0, \ldots, p_{k-1}\}$ constructed from symbol a; this follows easily from axioms of *dc*-semilattices by induction on the term. Now let $F \neq \emptyset$ and $G = \emptyset$; we have to shaw that $f(F) \not\leq f(G)$. By definition of f, $f(G) = a$ and $f(F)$ is the value in S of a τ-term t from the symbol a, and t contains at least one of the functional symbols p_i, $i < k$. By induction on t (and by axioms of *dc*-semilattices), $p_i(a) \leq f(F)$. Since $a < p_i(a)$, $f(F) \not\leq f(G)$.

It remains to consider the case when both F and G are non-empty. In this case the proof is by induction on $|F| + |G|$. The induction basis (i.e. the case when F and G are singletons) is trivial. For $|F| + |G| > 2$, consider the following four subcases.

Subcase 1. F is a proper forest with trees $F_0, \ldots, F_m$.

Then $F \leq G$ iff $F_i \leq G$ for all $i \leq m$ iff (by induction) $f(F_i) \leq f(G)$ for all $i \leq m$ iff $f(F) = f(F_0) \cup \cdots \cup f(F_m) \leq f(G)$.

Subcase 2. F is a tree and G is a proper forest with trees $G_0, \ldots, G_n$.

Then $F \leq G$ iff $F \leq G_j$ for some $j \leq n$ iff (by induction) $f(F) \leq f(G_j)$ for some $j \leq n$ iff $f(F) \leq f(G_0) \cup \cdots \cup f(G_n) = f(G)$.

Subcase 3. $F = p_i(F')$ and $G = p_j(G')$ are k-trees, and $i \neq j$.

Then $F \leq G$ iff $p_i(F') \leq p_j(G')$ iff $F \leq G'$ iff $f(F) \leq f(G')$ iff $f(F) = p_i(f(F')) \leq p_j(f(G')) = f(G)$.

Subcase 4. $F = p_i(F')$ and $G = p_j(G')$ are k-trees, and $i = j$.

Then $F \leq G$ iff $F' \leq G$ iff $f(F') \leq f(G) = p_i(f(G'))$ iff $f(F) = p_i(f(F')) \leq p_i(f(G')) = f(G)$.

From the equivalence just proven and from definition of f it follows that f induces isomorphism of the quotient structure of $(\mathcal{F}_k; \sqcup, p_0, \ldots, p_{k-1})$ onto (a). This concludes the proof of the theorem.

The following corollary of the last theorem shows that the theory of non-linearly ordered *dc*-semilattices of rank k has a minimal model under inclusion.

Corollary 3. *Let $(S; \cup, p_0, \ldots, p_{k-1})$ be a dc-semilattice such that $(S; \leq)$ is not a linear order. Then the quotient structure of $(\mathcal{F}_k; \sqcup, p_0, \ldots, p_{k-1})$ is isomorphic to a substructure of S.*

Proof. Let $x, y \in S$ be incomparable and let $a = x \cup y$. Then a satisfies the conditions of the previous theorem, hence the quotient structure of $(\mathcal{F}_k; \sqcup, p_0, \ldots, p_{k-1})$ is isomorphic to the substructure (a) of $(S; \cup, p_0, \ldots, p_{k-1})$. This completes the proof.

Remark. The reader might wonder on how the linearly ordered *dc*-semilattices look like. It is not difficult to understand such structures completely but we will not do this here.

5 $\mathcal{F}_k$ As a Free Object

In this section we characterize the algebra $(\mathcal{F}_k; \sqcup, p_0, \ldots, p_{k-1})$ as the free (or initial) 1-generated model of the theory with axioms 1) – 3) of Definition 1. Note that initial algebras (even with arbitrary sets of generators) exist and are unique up to isomorphism because the axioms in 1) – 3) are quasiidentities (see [M61,Se96]).

So let $(I_k; \cup, p_0 \ldots, p_{k-1})$ denote the initial model of 1) – 3) with one generator. Recall (see e.g. [M61,Se96]) that I_k may be constructed in two steps. First, construct the syntactic algebra A of τ-terms from a constant symbol c (which represents the generator). Second, factorize A modulo provable equivalence of terms in the theory with axioms 1) – 3). The algebra I_k is free in the sense that for any model S of 1) – 3) and any $a \in S$ there is a unique homomorphism $g : I_k \to S$ with $f(c) = a$ (actually $f([c]) = a$, where $[c]$ is the equivalence class containing c, but we again prefer simpler but not exact notation).

Theorem 5. *The quotient structure of* $(\mathcal{F}_k; \sqcup, p_0, \ldots, p_{k-1})$ *is isomorphic to the initial algebra* I_k.

Proof. By the property of I_k, there is a homomorphism $g : I_k \to \mathcal{F}_k$ with $g(c) = \emptyset$. We will show that g is indeed an isomorphism. To this end, define the function f from finite k-forests to I_k in exactly the same way as the function f in the proof of Theorem 4, only with a replaced by c. We claim that the function f respects the equivalence relation on forests. It suffices to check that $F \leq G$ implies $f(F) \leq f(G)$, and this is done in exactly the same way as in the proof of Theorem 4.

Now it is easy to see that f induces a homomorphism from the quotient structure of $(\mathcal{F}_k; \sqcup, p_0, \ldots, p_{k-1})$ to I_k which is the inverse of g. This completes the proof of the theorem.

The last theorem gives some information about the algebra of k-forests. On the other hand, it provides nontrivial information on the initial algebra I_k: it turns out to be a distributive lattice which satisfies the axiom 4) of *dc*-semilattices.

6 Countable *k*-Forests

In this section we observe that many results established above and in [Se04] may be extended to (at most) countable k-posets $(P; \leq, c)$ without infinite chains. The absence of infinite chains is of course equivalent to well-foundednes of both $(P; \leq)$ and $(P; \geq)$.

Let $\widetilde{\mathcal{P}}_k$, $\widetilde{\mathcal{F}}_k$, $\widetilde{\mathcal{T}}_k$ and $\widetilde{\mathcal{T}}_k^i$ denote the classes of all countable k-posets, countable k-forests, countable trees and countable i-rooted k-trees without infinite chains, respectively. The quasiorder $\leq$, the operation $\sqcup$ and other notions from Section 2 extend to this more general situation in the obvious way. By a *σ-semilattice* we mean un upper semilattice in which every countable set of elements has a supremum. Then it is not hard to obtain analogs of results in Section 2, in particular we have

Proposition 3. *(i) For any* $k \geq 2$, $(\widetilde{\mathcal{F}}_k; \leq)$ *is a wqo, and the rank* $r(\widetilde{\mathcal{F}}_k)$ *is the first non-countable ordinal* ω_1.

(ii) For any $k \geq 2$, $(\mathcal{F}_k; \leq)$ *is an initial segment of* $(\widetilde{\mathcal{F}}_k; \leq)$.

(iii) $w(\widetilde{\mathcal{F}}_2) = 2$ *and* $w(\widetilde{\mathcal{F}}_k) = \omega$ *for* $k > 2$.

(iv) For any $k \geq 2$, *the quotient structure of* $(\widetilde{\mathcal{F}}_k; \leq)$ *is a distributive lattice which is a σ-semilattice.*

(v) The set $\widetilde{\mathcal{T}}_k$ *coincides with the set of non-empty σ-join-irreducible elements of the lattice* $(\widetilde{\mathcal{F}}_k; \leq)$.

The results of Sections 3 and 4 also generalize to countable k-forests without infinite chains. As an example, we give the following straightforward generalization of Theorems 1 and 4. By a *dcσ*-semilattice we mean a *dc*-semilattice

$(S;\cup,p_0,\dots,p_{k-1})$ such that $(S;\cup)$ is a σ-semilattice and the axiom 4) of *dc*-semilattices holds for supremums of countable subsets of S (i.e., $p_i(x)$ is σ-join-irreducible). The operations p_i extend to the countable k-forests in the obvious way.

Theorem 6. *(i) The quotient structure of* $(\widetilde{\mathcal{F}}_k;\sqcup,p_0\dots,p_{k-1})$ *is a dcσ-semilattice.*

(ii) Let $(S;\cup,p_0,\dots,p_{k-1})$ *be a dcσ-semilattice and* a *its element such that* $a<p_i(a)$ *for all* $i<k$. *Then the dcσ-subsemilattice* (a) *of* S *generated by* a *is isomorphic to* $(\widetilde{\mathcal{F}}_k;\sqcup,p_0,\dots,p_{k-1})$.

We conclude this section with observing that main results of [Se04] about Boolean hierarchy of k-partitions have natural generalizations to the countable case, with essentially the same proofs.

Let (P,c) be a countable k-poset without infinite chains and $\mathcal{L}\subseteq P(M)$ be a σ-base (i.e. a base which is a σ-semilattice). A function $S:P\to\mathcal{L}$ is *admissible* if $\cup_x S_x=M$ and $S_x\cap S_y=\cup\{S_z|z\le x,y\}$ for all $x,y\in P$. For any such S, define a map $\tilde{S}:P\to P(M)$ by $\tilde{S}_x=S_x\setminus\cup\{S_y|y<x\}$. It is easy to see that if S is admissible then $\{\tilde{S}_x\}_{x\in P}$ is a partitions of M. Let $\mathcal{L}(P,c)=\{c\circ\tilde{S}|S\in H(P,\mathcal{L})\}$ where $H(P,\mathcal{L})$ is the set of admissible functions $S:P\to\mathcal{L}$ and $\tilde{S}$ is identified with the function from M to P sending $a\in M$ to the unique $p\in P$ with $a\in\tilde{S}_p$. Note that $\mathcal{L}(P,c)\subseteq k^M$, i.e. $\mathcal{L}(P,c)$ is a class of k-partitions of M.

The refined boolean hierarchy of k-partitions over $\mathcal{L}$ is the collection of classes $BH_k^*(\mathcal{L})=\{\mathcal{L}(P,c)|(P,c)\in\widetilde{\mathcal{P}}_k\}$. We define also the collection of classes of k-partitions $FBH_k(\mathcal{L})=\{\mathcal{L}(P,c)|(P,c)\in\widetilde{\mathcal{F}}_k\}$.

Now it is easy to check that analogs Lemmas 1.1 and 1.2 from [Se04] are true for the countable case which is crucial for proving the following relationship between the introduced collections. We call a σ-base $\mathcal{L}$ *σ-reducible* if for every sequence $A_0,A_1,\dots$ in $\mathcal{L}$ there is a pairwise disjoint sequence $B_0,B_1,\dots$ in $\mathcal{L}$ such that $B_i\subseteq A_i$ for all $i<\omega$ and $\cup_i B_i=\cup_i A_i$.

Theorem 7. *Over arbitrary σ-reducible σ-base $\mathcal{L}$, $BH_k^*(\mathcal{L})=FBH_k(\mathcal{L})$, and hence the poset $(BH_k^*(\mathcal{L});\subseteq)$ is a wqo.*

It is also possible to generalize the corresponding proof in [Se04] and obtain the following interesting concrete example of the countable Boolean hierarchy.

Theorem 8. *Over the σ-reducible base $\mathcal{L}$ of open sets in the Baire space ω^ω, the collection $(BH_k^*(\mathcal{L});\subseteq)$ is isomorphic to the quotient structure of $(\widetilde{\mathcal{F}}_k;\le)$.*

The last result characterizes the extension of the Hausdorff difference hierarchy from the case of sets to the case of k-partitions.

7 Conclusion

The structures considered above arise in different situations. E.g., from a result in [H96] it follows that the structure of discontinuity degrees of functions f :

$\omega^\omega \to k$ (i.e., the structure of Wadge degrees of k-partitions of the Baire space) of finite rank is embeddable in the quotient structure of $(\mathcal{F}_k; \leq)$. It is not hard to improve this result and show that it is indeed isomorphic to the quotient structure of $(\mathcal{F}_k \setminus \{\emptyset\}; \leq)$. It is also possible to show that the structure of Wadge degrees of k-partitions of the Baire space of countable rank is isomorphic to the quotient structure of $(\widetilde{\mathcal{F}}_k \setminus \{\emptyset\}; \leq)$. There are also similar close relationships of the structure $(\widetilde{\mathcal{T}}_k; \leq)$ to the Wadge degrees of k-partitions of ω-algebraic domains considered in [Se05]. But this is another story.

References

[DP94] B.A. Davey and H.A. Pristley. *Introduction to Lattices and Order*. Cambridge, 1994.

[H96] P. Hertling. *Unstetigkeitsgrade von Funktionen in der effectiven Analysis*. PhD thesis, FernUniversität Hagen, Informatik-Berichte 208–11, 1996.

[Ko00] S. Kosub. On NP-partitions over posets with an application of reducing the set of solutions of NP problems. *Lecture Notes of Computer Science*, 1893 (2000), 467–476, Berlin, Springer.

[Ko05] S. Kosub. On NP-partitions over posets with an application of reducing the set of solutions of NP problems. *Theory of Computing Systems*, 38(1) (2005), 83–113.

[KW00] S. Kosub and K. Wagner. The boolean hierarchy of NP-partitions. In: *Proc. 17th Symp. on Theor. Aspects of Comp. Sci., Lecture Notes of Computer Science*, 1770 (2000), 157–168, Berlin, Springer.

[Kr60] J.B. Kruskal. Well-quasi-ordering, the tree theorem, and Varzsonyi's conjecture. *Trans. Amer. Math. Soc.*, 95 (1960), 210–225.

[Kr72] J.B. Kruskal. The theory of well-quasi-ordering: a frequently discovered concept. *J. Combinatorics Th.(A)*, 13 (1972), 297–305.

[Ku06] D. Kuske. Theories of orders on the set of words. *Theoretical Informatics and Applications*, 40 (2006), 53–74.

[M61] A.I. Malcev, Constructive algebras, *Uspechi mat. nauk*, 16, No 3 (1961) 3—60 (in Russian, English translation in: The Metamathematics of Algebraic Systems (North Holand, Amsterdam) 1971, p. 148–214).

[Se79] V.L. Selivanov. On the structure of degrees of index sets. *Algebra and Logic*, 18, N 4 (1979), 463–480 (Russian, there is an English translation).

[Se82] V.L. Selivanov. On the structure of degrees of generalized index sets. *Algebra and Logika*, 21, N 4 (1982), 472–491 (in Russian, there is an English translation).

[Se83] V.L. Selivanov. Hierarchies of hyperarithmetical sets and functions. *Algebra and Logic*, 22, N 6 (1983), 666–692 (Russian, there is an English translation).

[Se96] V. L. Selivanov. On recursively enumerable structures. *Annals of pure and applied logic*, 78 (1996), 243–258.

[Se04] V. L. Selivanov. Boolean hierarchy of partitions over reducible bases. *Algebra and Logic*, 43, N 1 (2004), 77–109 (see also http://www.informatik.uni-wuerzburg.de, *Technical Report 276*, Institut für Informatik, Univers-ität Würzburg, 2001.

[Se05] V.L. Selivanov. Variations on the Wadge reducibility. *Siberian Advances in Math.*, 15, N 3 (2005), 44–80.

Third-Order Computation and Bounded Arithmetic

Alan Skelley*

Mathematical Institute, Academy of Sciences of the Czech Republic
Žitná 25, CZ - 115 67 Praha 1, Czech Republic
skelley@math.cas.cz

Abstract. We describe a natural generalization of ordinary computation to a third-order setting and give a function calculus with nice properties and recursion-theoretic characterizations of several large complexity classes. We then present a number of third-order theories of bounded arithmetic whose definable functions are the classes of the EXP-time hierarchy in the third-order setting.
Keywords: bounded arithmetic, recursion theory, computability, computational complexity

1 Introduction

Bounded arithmetic is an important and useful way to approach problems in computational and propositional proof complexity: strong tools from logic and model theory can be applied, and many of the connections are intriguingly not tight, suggesting that it could be possible to skirt around the major barriers of complexity theory. The second-order viewpoint of Zambella and Cook associates second-order theories of bounded arithmetic with various complexity classes by studying the definable functions of strings, rather than numbers. This approach simplifies presentation of the theories and their propositional translations, and furthermore is applicable to complexity classes that previously had no corresponding theories.

In previous work [12], we adapted the second-order viewpoint to PSPACE with the third-order theory W_1^1. In what follows, we generalize this result in several directions: First, by expanding the notion of computation to the third-order setting, essentially allowing a natural way to compute with very large objects, admitting a function calculus with nice properties and obtaining useful recursion-theoretic characterizations of large complexity classes above PSPACE. This computational setting bridges a gap by simultaneously allowing more natural reasoning about the kind of computation captured by theories of bounded arithmetic, while at the same time remaining a natural extension of ordinary computation and complexity. The second direction of generalization is to a full hierarchy of theories for the EXP-time hierarchy in this general setting. We also

* Partially supported by a Canadian NSERC postdoctoral fellowship

show how to apply the recursion-theoretic characterization of PSPACE to obtain a "minimal" theory for that class.

The remainder is organized as follows: In section 2 we describe the third-order setting: first the framework for bounded arithmetic, then for computation, and discuss complexity and recursion theory. Section 3 presents theories of bounded arithmetic and results about definability. We conclude with some open problems.

2 The Third-Order Setting

2.1 Bounded Arithmetic

The three sorts of third-order bounded arithmetic are intended to represent natural numbers, finite sets of natural numbers, and finite sets of such sets. For free and bound variables of these sorts we respectively use $a, b, c, ...$ and $x, y, z, ...$; $A, B, C, ...$ and $X, Y, Z, ...$; and $\mathcal{A}, \mathcal{B}, \mathcal{C}, ...$ and $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, ...$ The language $\mathcal{L}_A^3 := \{0, 1, +, \cdot, |\cdot|_2, \in_2, \in_3, \leq, =_1, =_2\}$ of nonlogical symbols is the same as the set $\mathcal{L}_A^2$ for V^1 but with the addition of the third-order membership predicate $A \in_3 \mathcal{B}$; note the absence of smash ('#') and third-order equality or length. We write $\mathcal{A}(B)$ for $B \in_3 \mathcal{A}$ and similarly for $\in_2$. The second sort closely represent finite binary strings as in e.g. [5] and likewise with the third sort (with strings rather than numbers as bit-indices), so we refer to them respectively as "strings" and "superstrings". We use a tilde: $\tilde{x}$ to denote unspecified sort.

There is a hierarchy of classes $g\Sigma_i^{\mathcal{B}}$ and $g\Pi_i^{\mathcal{B}}$ of formulas in this language analogous to the hierarchies Σ_i^B and Π_i^B of second-order formulas: the subscript counts alternations of third-order quantifiers (bounded string and number quantifiers ignored) and the 'g' denotes general (not strict) quantifier syntax. Note that there is no way to bound third-order quantifiers, but the number and string parameters determine the number of initial bits of superstrings that are relevant to the truth-value of a formula.

2.2 Computation

Our intent now is to capture the nature of string-based computation defined by third-order theories of bounded arithmetic. For this reason, our primary focus is on classes of polynomially-bounded functions (from strings to strings) or similar, as this makes operations such as composition of functions more natural and matches ordinary complexity theory. We are consequently interested in our classes of functions somehow maintaining an exponential-size distinction between the three sorts, as do (standard) theories of bounded arithmetic. Furthermore, our intent when defining third-order complexity classes is that the third-order (superstring) arguments not count towards the resource limits of the machine.

Functions in our setting will be strongly typed (each function has a fixed signature specifying sorts of arguments and output). The domains of the three sorts are: $\mathbb{D}_1 := \mathbb{N}$; $\mathbb{D}_2 := \{S \subset \mathbb{N} : |S| < \infty\}$; and $\mathbb{D}_3 = \{\mathcal{S} \subset \mathbb{D}_2 : |\mathcal{S}| < \infty\}$. Again we shall refer to these as numbers, strings, and superstrings; these sorts

are the same as the intended interpretations of the three sorts of variables in third-order bounded arithmetic and we use similar notation. Function symbols in our calculi will similarly be named $f, g, ...; F, G, ...$; and $\mathcal{F}, \mathcal{G}, ...$ to indicate the sort of **the range of the function**. Let $\mathbb{E} = \mathbb{E}_1 \cup \mathbb{E}_2 \cup \mathbb{E}_3$ be the set of all functions of fixed signature, categorized according to the sort of the output. The 0-1 valued functions (predicates) are referred to as $\mathbb{E}_0 \subset \mathbb{E}_1$.

Such functions are computed by Turing machines or other computational models by receiving number inputs in unary, strings as usual, and superstrings by random access. Outputs of strings or numbers are the same way, while superstrings are either output on a write-only tape, or "by query", as a predicate with a distinguished string input as the characteristic function of the output bits of the superstring-valued function. Precise definitions are in [13]. We are interested primarily in **polynomially bounded** functions. In the context of third-order computation, we mean that the polynomial bound applies to the value of a number output or the length of a string output, and is computed using only the number inputs and the lengths of the string inputs. If there are only superstring inputs, then the bound is a constant, and every superstring-valued function is polynomially bounded.

2.3 Complexity

An ordinary function or language class becomes a complexity class of third-order functions as follows: The notation (various superscripts on the complexity classes) is: For FC a function class, FC^+ is the third-order class with superstring output on write-only tape; for C a class of languages, $C^\diamond$ is the class of third-order predicates, while C° are the functions computed "by query" by predicates in this class. Here we describe some specific cases of complexity classes we are interested in:

First, $\mathrm{FPSPACE}^+$ is the third-order analogue of PSPACE functions. It consists of those polynomially bounded functions computable by a machine in polynomial space (as a function of the string and (unary) number inputs only), where superstring outputs are written onto a write-only output tape, allowing exponential-length superstring outputs. The machine's queries to its superstring inputs must also be polynomially bounded (as a function of its inputs). FEXP^+ is similarly the polynomially bounded exponential-time functions with polynomially bounded access to superstring inputs. In contrast to $\mathrm{FPSPACE}^+$, the polynomial bound is an actual restriction as an exponential time machine could otherwise write exponentially large strings (either as output, or as a query to superstring inputs).

Now for the case of polynomial time, the class FP^+ defined analogously to $\mathrm{FPSPACE}^+$ and FEXP^+ has the property that superstring outputs have polynomial length, due to the time bound of the machines; however, the class P° of polynomially-bounded functions computed by "by query" by polynomial-time machines does not have this restriction. For this reason, $\mathrm{FP}^+ \cup \mathrm{P}^\circ$ is in some contexts a more suitable third-order analogue of P. This is also the case for functions from levels $\Box_i^p$ of the polynomial-time hierarchy, which are computed

by polynomial-time machines with access to an oracle from Σ_{i-1}^p: The third-order class $(\square_i^p)^+$ is restricted to polynomially many bits in its superstring outputs and so $(\square_i^p)^+ \cup (\square_i^p)^\circ$ is a more appropriate definition.

As a final set of examples, the predicate classes $\mathrm{P}^\diamond$, $\mathrm{NP}^\diamond$, $(\Sigma_i^p)^\diamond$, $\mathrm{NEXP}^\diamond$ and $(\Sigma_i^{exp})^\diamond$ are 0-1 valued functions, and are the characteristic functions of machines from the corresponding ordinary complexity classes, modified with polynomially bounded access to superstring inputs.

Some comments are in order concerning these classes. First, and most importantly, the third-order complexity classes discussed thus far, restricted to functions from strings to strings (or string predicates) are the usual complexity classes. There are nevertheless some interesting observations to be made: For example $\mathrm{P}^\diamond \neq \mathrm{NP}^\diamond$, as a predicate in the latter class can determine if a given superstring contains a 1 (up to a bound given by a string argument), while this predicate is clearly not in $\mathrm{P}^\diamond$. The usual argument for Savitch's theorem goes through, at least for (unrelativized) $\mathrm{NPSPACE}^\diamond$: configurations are still described by polynomial-sized strings, including queries to superstring inputs. We conclude that $\mathrm{PSPACE}^\diamond = \mathrm{NPSPACE}^\diamond$.

Now, in order to expand our discussion to the exponential-time hierarchy, we must first address relativizing classes of functions by adding oracles in the form of access to a third-order function. Formally, a **third-order oracle** Turing machine has a number of specified write-only query tapes, each one designated with a sort. The machine may write values on these tapes which are polynomially bounded, in the sense that the numbers (in unary), and lengths of strings written are all bounded by fixed polynomials in the machine's (non-superstring) inputs. When the machine enters the special query state, these tapes are erased, and a value is returned to the machine by way of a special read-only reply tape (with random access in the case of a superstring-valued oracle).

The usual exponential-time hierarchy has definition $\Sigma_i^{exp} = \mathrm{NEXP}^{\Sigma_{i-1}^p}$ [10]. This is equal to Σ_i-TIME(exp), which are the languages computed by exponential time alternating Turing machines with i alternations (starting with existential). Paralleling this definition, we can define the corresponding classes of 0-1 valued functions from $\mathbb{E}_0$. It is important to observe that the queries made of the Σ_{i-1}^p oracle by the NEXP machine in the standard definition are in general of **exponential** size. Our third-order oracle machines can also issue exponentially-long queries to their oracles, but these must be in the form of superstrings, as the string inputs to oracles are restricted to be polynomially bounded per our definition. Consequently the complexity class of the third-order oracle we use will be different.

We therefore define $(\Sigma_1^{exp})^\diamond = \mathrm{NEXP}^\diamond$ and $(\Sigma_i^{exp})^\diamond = (\mathrm{NEXP}^\diamond)^{(\Sigma_{i-1}^{exp})^\diamond}$. In other words, each higher level of the hierarchy is obtained by augmenting nondeterministic exponential time with a third-order oracle for the previous level. Since the queries to this oracle must be polynomially bounded (although this still allows exponential-length superstring inputs to the oracle), it can be seen that this relativization corresponds to unbounded access to an ordinary oracle from the appropriate level of the **quasi-polynomial-time** hierarchy (considered as a

predicate on the superstring inputs): For example, if an NEXP machine writes string and superstring inputs of lengths $p(n)$ and $2^{p(n)}$ respectively to a third-order NEXP oracle, then the query can be answered in nondeterministic time $2^{(p(n))^k}$ for some k, which is exponential in $p(n)$. In terms of the length of the superstring input, $2^{p(n)}$, the quantity $2^{(p(n))^k}$ is quasi-polynomial.

In the hands of an NEXP machine, however, an unbounded (ordinary) oracle from some level of the quasi-polynomial-time hierarchy is no more powerful than one from the same level of the polynomial-time hierarchy, as the machine could simply make longer queries (i.e. $2^{(p(n))^k}$) of this latter oracle. Thus as predicates purely on strings, the levels of our hierarchy correspond precisely with the levels of the ordinary exponential-time hierarchy. Therefore:

Theorem 1. *The predicates represented in the standard model by $\Sigma_0^{\mathcal{B}}$-formulas are precisely $PH^{\diamond}$; for $i \geq 1$ those represented by $g\Sigma_i^{\mathcal{B}}$- and $g\Pi_i^{\mathcal{B}}$-formulas (and also the strict versions of these classes) are precisely $(\Sigma_i^{exp})^{\diamond}$ and $(\Pi_i^{exp})^{\diamond}$, respectively.*

The function classes $(\square_i^{exp})^+ := (\mathrm{FEXP}^{(\Sigma_{i-1}^{exp})^{\diamond}})^+$ are the polynomially bounded functions computed by exponential-time Turing machines relativized with a third-order oracle for a predicate from $(\Sigma_i^{exp})^{\diamond}$, and similarly as functions purely of strings correspond to the usual $\square_i^{exp}$. It should be noted that $(\Sigma_i^{exp})^{\diamond} = (\Pi_i^{exp})^{\diamond}$ seems to imply that the third-order exponential-time hierarchy collapses to the ith level, while this is not known for the ordinary case; The difference is that the assumption $\Sigma_i = \Pi_i$ in the third-order context is stronger, in that it covers also predicates on superstrings.

2.4 Recursion Theory of Functions

First some standard functions: The number functions $\{x + y, x \cdot y\}$, constants 0,1, etc. are as usual. The bit, string successor and concatenation functions $\{\mathrm{bit}(x, Y), s_0(X), s_1(X), X \frown Y\}$ are also standard, but they are operations on binary strings, while our string-like domain $\mathbb{D}_2$ consists of finite sets of natural numbers. We therefore define these functions to operate on the strings represented by the input finite sets, and to output the set representing the desired string. $\{|X|, X \in \mathcal{Y}, 1^x\}$ respectively give the least upper bound of the set (which is one more than the length of the string being represented by the set), the (0-1-valued) characteristic function of $\mathcal{Y}$, and a standard string of x bits (represented by a set of least upper bound $x + 1$). All of the functions described thus far are polynomially bounded.

We now define several operations on these functions. As our focus is on string functions as opposed to the standard recursion-theoretic viewpoint of number functions, we shall comment in each case on how these operations compare to standard operations on number functions.

First, the operation of **composition** defines a function $\tilde{f}(\vec{\tilde{x}}) = t$ by specifying a term t consisting of variables among $\vec{\tilde{x}}$ and other functions, constructed in such

a way that arities and argument types are respected. Observe that this operation allows permutation and renaming of variables.

Define $\tilde{f}$ (of any sort) by **limited recursion** from $\tilde{g}$, $\tilde{h}$ (also of any sort) and l by $\tilde{f}(0,...) = \tilde{g}(...), \tilde{f}(x+1,...) = \tilde{h}(x, \tilde{f}(x,...),...)$ and either $\tilde{f}(x,...) \leq l(x,...)$ or $|\tilde{f}(x,...)| \leq l(x,...)$, as appropriate. This operation corresponds roughly to limited recursion on notation for number functions, as it iterates a function ($\tilde{h}$) a polynomial number of times subject to a bound on growth. **Recursion** is the same operation without the bound on growth.

Define $\tilde{f}$ by **limited doubling recursion** from $\tilde{g}$ and l by $\tilde{f}(0,\tilde{y},...) = \tilde{g}(\tilde{y},...), \tilde{f}(x+1,\tilde{y},...) = \tilde{f}(x,\tilde{f}(x,\tilde{y},...),...)$ and either $\tilde{f}(x,\tilde{y},...) \leq l(x,...)$ or $|\tilde{f}(x,\tilde{y},...)| \leq l(x,...)$, as appropriate. This operation corresponds roughly to limited recursion for number functions, as it iterates a function ($\tilde{g}$) an exponential number of times (by doubling the number of nestings a polynomial number of times) subject to a bound on growth. **Doubling recursion** is the same operation without the bound on growth.

Define $\tilde{f}$ (of any sort) by **limited long recursion** from $\tilde{g}$, $\tilde{h}$ (also of any sort) and l by $\tilde{f}(1^0,...) = \tilde{g}(...), \tilde{f}(X+1,...) = \tilde{h}(x, \tilde{f}(X,...),...)$ and either $\tilde{f}(X,...) \leq l(X,...)$ or $|\tilde{f}(X,...)| \leq l(X,...)$, as appropriate. This operation is similar to the previous one in that it iterates a function an exponential number of times; however, it differs in that the exponentially many iterations are performed directly by using a string as an exponential-length counter. This operation presupposes a suitable string successor function $X+1$.

Define $\mathcal{F}$ by **limited 3-comprehension** from $g, h \in \mathbb{E}_1$ by $\mathcal{F}(..)(X) \leftrightarrow (|X| \leq g(..) \wedge h(X,..) = 0)$.

It should be noted here that the recursion operations, as well as simple composition of functions, appear to be significantly more powerful when applied to superstring-valued functions. This is because in the composition of two such functions, the space may not be available to write down the intermediate value. A space-bounded computation model would then have to query the "inner" function many times (to retrieve bits of its output as needed) in order to compute the outer function. The composition of two polynomially bounded number- or string-valued functions can be computed using the sum of the time requirements (computing first one then the other function), while the required space does not increase. For superstring-valued functions, on the other hand, the time required for the composition as described seems in general to be the product of the time required for each component, while the space required is the sum. If space is not bounded then the intermediate results can be written in full, and thus time and space requirements are as for the composition of number- or string-valued functions.

At this point we can characterize several important complexity classes:

Theorem 2. *1. $FP^+ \cup P^\circ$ is the closure of the initial functions $I = \{0, 1, x+y, x \cdot y, 1^x, |X|, s_0(X), s_1(X), bit(x,Y), X \frown Y, X \in \mathcal{Y}\}$ under composition, limited 3-comprehension and limited recursion with the latter restricted to $\mathbb{E}_1 \cup \mathbb{E}_2$.*

2. *FPSPACE$^+$ is the closure of I under composition, limited 3-comprehension and limited recursion.*
3. *FPSPACE$^+$ is the closure of I under composition, limited 3-comprehension and limited doubling recursion restricted to $\mathbb{E}_1 \cup \mathbb{E}_2$.*
4. *FEXP$^+$ is the closure of I under composition, limited 3-comprehension and limited doubling recursion.*

Proof (sketch). The first point is essentially as in Cobham [4].

FPSPACE$^+$ is contained in the closure of FP$^+$∪P$^\circ$ by limited recursion on $\mathbb{E}_3$, composition and limited 3-comprehension: First, a superstring-valued FP$^+$∪P$^\circ$ function can compute from the input of a PSPACE Turing machine the transition function of the machine as a table listing the next configuration for each given configuration. Another function in FP$^+$∪P$^\circ$ can compose such a function with itself by reading two (polynomial-sized) entries from this table. Therefore after applying limited recursion on these two functions we obtain a third that outputs the 2^x-step transition function and from this it is trivial to extract the value of the original PSPACE function. Conversely, FPSPACE$^+$ is closed under limited recursion (as each such operation increases the space requirements of a function by a polynomial factor) and the other operations.

For point 3, The step function of a PSPACE Turing machine (a polynomial-time string function) can be iterated exponentially many times using limited doubling recursion restricted to $\mathbb{E}_1 \cup \mathbb{E}_2$. Conversely FPSPACE$^+$ is closed under this restriction of limited doubling recursion as the recursion can be unwound with only a polynomial amount of additional space. This characterization is analogous to the one used in Dowd [8]: initial functions closed under limited recursion. Limited recursion in the context of number functions is of exponential length, as is limited doubling recursion in our setting. This in turn is reminiscent of $\mathcal{E}^2$, the second level of the Grzegorczyk hierarchy [9], which is defined similarly except with an initial function of linear growth rate as opposed to $x\#'y$; this was shown by Ritchie [11] to equal the linear space functions.

Finally, with limited doubling recursion on $\mathbb{E}_3$, the step function of an exp-time Turing machine can be iterated exponentially many times. See [3] for a previous recursion-theoretic characterization of the exponential-time number functions. □

3 Third-Order Bounded Arithmetic Theories

Our main theories are W_1^i and TW_1^i, intended to correspond to levels of the exponential-time hierarchy; they are parameterized by the type of induction. These theories are suggested by the RSUV isomorphism and are closely connected to U_2^i and V_2^i, respectively, although we do not claim an actual isomorphism (but one may hold with the unbounded domain versions of these theories).

For $i \geq 0$, W_1^i is a theory over $\mathcal{L}_A^3$. The axioms of W_1^i are B1-B14, L1, L2 and SE of [Cook/Kolokolova], (strict) $\forall^2\Sigma_i^{\mathcal{B}}$-IND and the comprehension schemes $\Sigma_0^{\mathcal{B}}$-2COMP: $(\exists Y \leq t(\overline{x}, \overline{X}))(\forall z \leq a)[\phi(\overline{x}, \overline{X}, \overline{\mathcal{X}}, z) \leftrightarrow Y(z)]$ and $\Sigma_0^{\mathcal{B}}$-3COMP:

$(\exists \mathcal{Y})(\forall Z \leq a)[\phi(\overline{x}, \overline{X}, \overline{\mathcal{X}}, Z) \leftrightarrow \mathcal{Y}(Z)]$, where in each case $\phi \in \Sigma_0^{\mathcal{B}}$ subject to the restriction that neither Y nor $\mathcal{Y}$, as appropriate, occurs free in ϕ.

W_1^1 defined above is slightly different than the version in CSL04 [12]; it includes a string equality symbol and extensionality axiom, but this is a conservative extension. The unusual class of formulas for which we admit induction (a bounded string quantifier followed by a strict $\Sigma_i^{\mathcal{B}}$-formula) is in order for a replacement scheme to be provable; as a result of this scheme, W_1^i ultimately admits full $g\Sigma_i^{\mathcal{B}}$-IND; we omit the details.

Define $\widehat{W_1^i}$ to be the analogous theory with the induction scheme restricted to (strict) $\Sigma_i^{\mathcal{B}}$-formulas. Note that $\widehat{W_1^0} = W_1^0$.

TW_1^i is defined identically as above, but with the following scheme named $\Sigma_i^{\mathcal{B}}$-SIND (string or set induction) in place of $\forall^2\Sigma_i^{\mathcal{B}}$-IND:

$$[\forall X, Y, Z((|Z| = 0 \supset \phi(Z)) \wedge (\phi(X) \wedge S(X, Y) \supset \phi(Y)))] \supset \forall Z \phi(Z)$$

for $\phi \in$(strict)$\Sigma_i^{\mathcal{B}}$, where $S(X, Y)$ is a $\Sigma_0^{\mathcal{B}}$-formula expressing that Y is the lexicographically next finite set after X. Again, TW_1^i admits (string) induction on the more general class of formulas due to a replacement scheme.

TTW_1^i is yet another theory in this vein, with a yet stronger induction scheme named $\Sigma_i^{\mathcal{B}}$-SSIND ("superstring" induction). Note that since (by design) there is no way to bound a third-order object, the scheme refers to a term t, and restricts its attention to the first 2^t bits of the objects. It is intended that this t be some crucial bound from ϕ. The scheme is:

$$[\forall \mathcal{X}, \mathcal{Y}, \mathcal{Z}((\forall X \leq t \neg \mathcal{Z}(X)) \supset \phi(\mathcal{Z})) \wedge (\phi(\mathcal{X}) \wedge S_3(\mathcal{X}, \mathcal{Y}, t) \supset \phi(\mathcal{Y}))] \supset \forall \mathcal{Z} \phi(\mathcal{Z})$$

for $\phi \in$(strict)$\Sigma_i^{\mathcal{B}}$, where $S_3(\mathcal{X}, \mathcal{Y}, z)$ is a $\Sigma_0^{\mathcal{B}}$-formula expressing that when considering only the lowest 2^z bits of the superstrings, $\mathcal{Y}$ is lexicographically next after $\mathcal{X}$.

The scheme $\Sigma_0^{\mathcal{B}}$-superstring-recursion is $\exists \mathcal{X} \phi^{\mathrm{rec}}(S, \mathcal{X})$, where $\phi(Y, \mathcal{X}) \in \Sigma_0^{\mathcal{B}}$, and $\phi^{\mathrm{rec}}(x, \mathcal{X}) \equiv \forall Y \leq |S|(L_2(Y, S) \supset (\mathcal{X}(Y) \leftrightarrow \phi(Y, \mathcal{X}^{<Y})))$. $L_2(X, Y)$ expresses that lexicographically, $X < Y$, while $\mathcal{X}^{<Y}$ is a chop function (i.e., $\mathcal{X}^{<Y}(Z)$ abbreviates the subformula $L_2(Z, Y) \wedge \mathcal{X}(Z)$). ϕ (and therefore also ϕ^{rec}) may have other free variables than the displayed ones, but ϕ must have distinguished string and superstring free variables Y and $\mathcal{X}$. ϕ^{rec} then has $\mathcal{X}$ free as well as a new variable S. This scheme is analogous to that from [2] and follows the presentation from [7].

The scheme $\Sigma_0^{\mathcal{B}}$-superstring-halfrecursion is $\exists \mathcal{X} \phi^{\mathrm{hrc}}(S, \mathcal{X})$, where $\phi(Y, \mathcal{X}) \in \Sigma_0^{\mathcal{B}}$, and $\phi^{\mathrm{hrc}}(S, \mathcal{X}) \equiv \forall Y \leq |S|(L_2(Y, S) \supset (\mathcal{X}(Y) \leftrightarrow \phi(Y, \mathcal{X}^{<Y/2})))$, where $\mathcal{X}^{<Y/2}$ is a chop function returning the first $\frac{Y}{2}$ (as a number) bits of $\mathcal{X}$. ϕ and ϕ^{rec} have the same free-variable conventions and requirements as in the superstring recursion scheme. Then HW_1^0 is the theory W_1^0 with the addition of the $\Sigma_0^{\mathcal{B}}$-superstring-halfrecursion scheme.

3.1 Definability in the Theories

The definability of functions in the third-order setting is a generalization of the usual definition, but the case of superstring-valued functions additionally

includes a mechanism for explicitly reasoning about only an initial segment of the output. This is necessary as superstrings in the theories are formally unbounded, while in the function calculus they are finite.

The following omnibus theorem summarizes results concerning definable functions in the theories; proofs (and the formal definition of definability) are in [13] and are fairly technical, yet generally straightforward. We comment below on especially interesting or unusual points.

Theorem 3. *1. For $i \geq 1$, the $\Sigma_i^{\mathcal{B}}$-definable functions of W_1^i are precisely (FPSPACE$^{(\Sigma_{i-1}^{exp})^\diamond}$)$^+$.*
2. For $i \geq 1$, the $\Sigma_i^{\mathcal{B}}$-definable functions of TW_1^i are precisely (FEXP$^{(\Sigma_{i-1}^{exp})^\diamond}$)$^+$.
3. The $\Sigma_1^{\mathcal{B}}$-definable functions of TTW_1^0 are precisely FEXP$^+$.
4. The $\Sigma_1^{\mathcal{B}}$-definable functions of both HW_1^0 and $\widehat{W_1^1}$ are precisely FPSPACE$^+$.
5. $W_1^0 = TW_1^0$ and the definable functions of this theory are FPH$^+ \cup$FPH$^\circ$, the polytime hierarchy functions in the third-order setting.

Remarks. 1. Uniqueness of the function value is important, otherwise the definable **multi**-functions are (FEXP$^{(\Sigma_{i-1}^{exp})^\diamond}$[wit,poly])$^+$; analogously to [1] for U_2^1.
2. Straightforward.
3. Functions are defined using the superstring-recursion scheme.
4. Straightforward.
5. Equality of theories is by "shortening of cuts"; the theories are conservative extensions of the second-order polytime hierarchy theory V by a standard argument.

□

3.2 An Application of the Function Calculus

W_1^1 does not seem to be a "minimal" theory for PSPACE as witnessing the induction seems to be too hard. (Put another way, the $\Sigma_2^{\mathcal{B}}$-definable functions of W_1^1 might not be contained in FPSPACE$^+$, analogouslt to the situation for, say, S_2^1). An application of the recursion-theoretic characterization previously presented is that one can specify a language $\mathcal{L}_{PS}$ of FPSPACE$^+$ function symbols with open defining equations. There are cases for initial functions and definitions by limited recursion, as well as minimization functions allowing elimination of quantifiers. The resulting theory, $\overline{HW_1^0}$, is universal; it extends HW_1^0 as it contains functions witnessing the halfrecursion scheme; and the extension is conservative. This last point is proved by showing inductively that each function of $\mathcal{L}_{PS}$ is definable in HW_1^0 by a single application of the halfrecursion operation followed by a $\Sigma_0^{\mathcal{B}}$ projection (a subclass of $\Sigma_1^{\mathcal{B}}$-definable). The upshot of all of this is that HW_1^0 is therefore in some sense a minimal theory for FPSPACE$^+$.

4 Further Research

Some particular problems: First, does $\widehat{W_1^1}$ prove the general induction of W_1^1? One approach is the method of [6], namely by KPT witnessing, with HW_1^0 as the

starting point. Second, what about propositional translations of these theories? Partial progress is made in [13]. Third, are any of the theories HW_1^0, W_1^1 or TW_1^1 finitely axiomatizable? Finally, would the conservativity of W_1^1 over HW_1^0 have any complexity-theoretic consequences?

5 Acknowledgment

Thanks to Toniann Pitassi, Charles Rackoff, Alasdair Urquhart, Sam Buss and Stephen Cook for improvements to the presentation and helpful comments; and to the referees, whose detailed suggestions I have but imperfectly heeded.

References

[1] Samuel Buss, Jan Krajíček, and Gaisi Takeuti. On provably total functions in bounded arithmetic theories R_3^i, U_2^i and V_2^i. In Peter Clote and Jan Krajíček, editors, *Arithmetic, proof theory and computational complexity*, pages 116–61. Oxford University Press, Oxford, 1993.

[2] Samuel R. Buss. Axiomatizations and conservation results for fragments of bounded arithmetic. In *CMWLC: Logic and Computation: Proceedings of a Workshop held at Carnegie Mellon University*, pages 57–84. Contemporary Mathematics Volume 106, American Mathematical Society, 1990.

[3] Peter Clote. A safe recursion scheme for exponential time. In Sergei I. Adian and Anil Nerode, editors, *LFCS97*, volume 1234 of *Lecture Notes in Computer Science*, pages 44–52. Springer, 1997.

[4] Alan Cobham. The intrinsic computational difficulty of functions. In Yehoshua Bar-Hillel, editor, *Proceedings of the International Congress for Logic, Methodology and Philosophy of Science*, pages 24–30. North-Holland, 1964.

[5] S. A. Cook. CSC 2429S: Proof Complexity and Bounded Arithmetic. Course notes, URL: "http://www.cs.toronto.edu/∼sacook/csc2429h", Winter 2002.

[6] Stephen Cook and Neil Thapen. The strength of replacement in weak arithmetic. In Harald Ganzinger, editor, *LICS04*, pages 256–264. IEEE Computer Society, July 2004.

[7] Stephen A. Cook. Theories for complexity classes and their propositional translations. In Jan Krajíček, editor, *Complexity of Computations and Proofs*, volume 13 of *Quaderni di Matematica*, pages 175–227. Seconda Università di Napoli, 2004.

[8] Martin Dowd. *Propositional Representation of Arithmetic Proofs*. PhD thesis, University of Toronto, 1979.

[9] A. Grzegorczyk. Some classes of recursive functions. *Rozprawy Matematyczne*, 4:1–46, 1953.

[10] Juris Hartmanis. The collapsing hierarchies. *Bulletin of the EATCS*, 33, September 1987.

[11] R. W. Ritchie. Classes of predictably computable functions. *Transactions of the American Mathematical Society*, 106:139–173, 1963.

[12] Alan Skelley. A third-order bounded arithmetic theory for PSPACE. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *CSL04*, volume 3210 of *Lecture Notes in Computer Science*, pages 340–354. Springer, 2004.

[13] Alan Skelley. *Theories and Proof Systems for PSPACE and the EXP-Time Hierarchy*. PhD thesis, University of Toronto, 2005. Available from ECCC in the 'theses' section.

On Inner Constructivizability of Admissible Sets*

Alexey Stukachev

Sobolev Institute of Mathematics, Novosibirsk, Russia
aistu@math.nsc.ru

Abstract. We consider a problem of inner constructivizability of admissible sets by means of elements of a bounded rank. For hereditary finite superstructures we find the precise estimates of the rank of inner constructivizability: it is equal to ω for superstructures over finite structures and less than or equal to 2 otherwise. We also introduce examples of hereditary finite superstructures with ranks 0, 1, 2. It is shown that hereditary finite superstructure over the field of reals has rank 1.

Notations and terminology used below are standard and corresponds to [3, 1]. We denote the domains of a structure $\mathfrak{M}$ and KPU-model $\mathcal{A}$ by M and A respectively. Further on, for simplicity reasons and without loss of generality, we will consider only structures and KPU-models with relational signatures.

Let $\mathfrak{M}$ be a structure of computable relational signature $\langle P_0^{n_0}, \ldots, P_k^{n_k}, \ldots\rangle$, and let $\mathcal{A}$ be a KPU-model, i.e. a structure of signature containing symbols $U^1, \in^2$, which is a model of the system of axioms KPU. Following [3], $\mathfrak{M}$ is called *Σ-definable (constructivizable) in $\mathcal{A}$* if there exists a computable sequence of Σ-formulas

$$\Phi(x_0, y), \Psi(x_0, x_1, y), \Psi^*(x_0, x_1, y), \Phi_0(x_0, \ldots, x_{n_0-1}, y),$$

$$\Phi_0^*(x_0, \ldots, x_{n_0-1}, y), \ldots, \Phi_k(x_0, \ldots, x_{n_k-1}, y), \Phi_k^*(x_0, \ldots, x_{n_k-1}, y), \ldots$$

such that for some parameter $a \in A$, and letting

$$M_0 \leftrightharpoons \Phi^{\mathcal{A}}(x_0, a), \quad \eta \leftrightharpoons \Psi^{\mathcal{A}}(x_0, x_1, a) \cap M_0^2$$

one has that $M_0 \neq \varnothing$ and η is a congruence relation on the structure

$$\mathfrak{M}_0 \leftrightharpoons \langle M_0, P_0^{\mathfrak{M}_0}, \ldots, P_k^{\mathfrak{M}_0}, \ldots\rangle,$$

where $P_k^{\mathfrak{M}_0} \leftrightharpoons \Phi_k^{\mathcal{A}}(x_0, \ldots, x_{n_k-1}) \cap M_0^{n_k}$, $k \in \omega$,

$$\Psi^{*\mathcal{A}}(x_0, x_1, a) \cap M_0^2 = M_0^2 \setminus \Psi^{\mathcal{A}}(x_0, x_1, a),$$

* This work was supported by the INTAS YSF (grant 04-83-3310), the Program "Universities of Russia" (grant UR.04.01.488), the Russian Foundation for Basic Research (grant 05-01-00481a) and the Grant of the President of RF for Young Scientists (grant MK.1239.2005.1).

$$\Phi_k^{*\mathcal{A}}(x_0, \ldots, x_{n_k-1}, a) \cap M_0^{n_k} = M_0^{n_k} \setminus \Phi_k^{\mathcal{A}}(x_0, \ldots, x_{n_k-1})$$

for all $k \in \omega$ and the structure $\mathfrak{M}$ is isomorphic to the quotient structure $\mathfrak{M}_0/\eta$. In this case we say that the above sequence of formulas together with the parameter a are Σ-*defining* $\mathfrak{M}$ *in* $\mathcal{A}$.

In the present setting, however, it would be more convenient to use an equivalent approach based on the notion of $\mathcal{A}$-constructivizability [3]. An onto mapping (numbering) $\nu : B \to M$ is called an $\mathcal{A}$-*constructivization* of a structure $\mathfrak{M}$ if $B \subseteq A$ is a Σ-definable subset of $\mathcal{A}$, and the numbering equivalence relation

$$\eta_\nu = \{\langle b_0, b_1\rangle | b_0, b_1 \in B,\ \mathfrak{M} \models (\nu(b_0) = \nu(b_1))\}$$

as well as the sets

$$\{\langle k, \langle b_0, \ldots, b_{n_k-1}\rangle\rangle | k \in \omega, b_0, \ldots, b_{n_k-1} \in B, \mathfrak{M} \models P_k(\nu(b_0), \ldots, \nu(b_{n_k-1}))\}$$

are Δ-definable subsets of $\mathcal{A}$. We will say that a structure is $\mathcal{A}$-*constructivizable* if it has an $\mathcal{A}$-constructivization. It is known (see [3]) that a structure $\mathfrak{M}$ is Σ-definable in a KPU-model $\mathcal{A}$ if and only if $\mathfrak{M}$ is $\mathcal{A}$-constructivizable.

Let $\mathcal{A}$ be a KPU-model of signature $\sigma_{\mathcal{A}}$ and let Θ be a Σ-formula of the same signature. For arbitrary Σ-formula Φ of the signature $\sigma_{\mathcal{A}}$, the *relativization* Φ^Θ of formula Φ by formula Θ is defined inductively:

– if Φ is an atomic formula then $\Phi^\Theta \leftrightharpoons \Phi$;
– if $\Phi = \neg\Psi$ then $\Phi^\Theta \leftrightharpoons \neg(\Psi^\Theta)$;
– if $\Phi = (\Psi_1 * \Psi_2)$, $* \in \{\wedge, \vee, \to\}$, then $\Phi^\Theta \leftrightharpoons (\Psi_1^\Theta * \Psi_2^\Theta)$;
– if $\Phi = (Qx \in y)\Psi$, $Q \in \{\forall, \exists\}$, then $\Phi^\Theta \leftrightharpoons (Qx \in y)\Psi^\Theta$;
– if $\Phi = \exists x\Psi$ then $\Phi^\Theta \leftrightharpoons \exists x(\Theta(x) \wedge \Psi^\Theta)$.

It is clear that Φ^Θ is a Σ-formula of signature $\sigma_{\mathcal{A}}$.

Definition 1. *Let $\mathcal{A}$ be a* KPU*-model of computable relational signature $\sigma_{\mathcal{A}} = \langle U^1, \in^2, P_0^{n_0}, \ldots\rangle$, and let $B \subseteq A$ be a transitive Σ-subset defined in $\mathcal{A}$ by some Σ-formula Θ of the signature $\sigma_{\mathcal{A}}$, which contains parameters only from B. $\mathcal{A}$ is said to be* constructivizable inside B *if there is a computable sequence $\Phi(\bar{x}_0, \bar{y})$, $\Phi_=(\bar{x}_0, \bar{x}_1, \bar{y})$, $\Psi_=(\bar{x}_0, \bar{x}_1, \bar{y})$, $\Phi_\in(\bar{x}_0, \bar{x}_1, \bar{y})$, $\Psi_\in(\bar{x}_0, \bar{x}_1, \bar{y})$, $\Phi_U(\bar{x}_0, \bar{y})$, $\Psi_U(\bar{x}_0, \bar{y})$, $\Phi_{P_0}(\bar{x}_0, \ldots, \bar{x}_{n_0-1}, \bar{y})$, $\Psi_{P_0}(\bar{x}_0, \ldots, \bar{x}_{n_0-1}, \bar{y}), \ldots$ of Σ-formulas (all tuples $\bar{x}_0$, $\bar{x}_1$ ... are supposed to be of the same length k – a dimension of the constructivization, tuple $\bar{y}$ has length l), and a tuple of parameters $\bar{b} \in B^l$ such that $\{\bar{a} \in A | \mathcal{A} \models \Phi^\Theta(\bar{a}, \bar{b})\} \subseteq B^k$ and the sequence of the relativized formulas $\langle \Phi^\Theta, (\Phi_=)^\Theta, (\Psi_=)^\Theta, (\Phi_\in)^\Theta, (\Psi_\in)^\Theta, (\Phi_U)^\Theta, (\Psi_U)^\Theta, \Phi_{P_0}^\Theta, \Psi_{P_0}^\Theta, \ldots\rangle$ with parameters $\bar{b}$ are Σ-defining the* KPU*-model $\mathcal{A}$ in $\mathcal{A}$.*

The above notion could also be reformulated in terms of constructivizations, so we will usually speak about $\mathcal{A}$-*constructivizations of* $\mathfrak{M}$ *inside* B. Note also that because of the requirement on parameters to be elements from B we could not, in general, replace in the above definition a tuple $\bar{b}$ by a single $b \in B$.

In the same way, under the same conditions for B, we call a subset $C \subseteq A$ to be Σ-*definable in* $\mathcal{A}$ *inside* B, if C is defined in $\mathcal{A}$ by means of Φ^Θ for some Σ-formulas Φ and Θ with parameters from B.

Suppose now that $\mathcal{A}$ is an admissible set, i.e. a KPU-model in which the set of ordinals is well-founded (see [3]). If, for any $a \in \mathcal{A}$, $\mathrm{rnk}(a)$ denotes the rank of a, we can define a notion of rank for arbitrary subset $B \subseteq A$ in the following way: $\mathrm{rnk}(B) = \sup\{\mathrm{rnk}(b) | b \in B\}$.

Definition 2. *The rank of inner constructivizability of an admissible set $\mathcal{A}$ is the ordinal*

$$\mathrm{cr}(\mathcal{A}) = \inf\{\mathrm{rnk}(B) | \mathcal{A} \textit{ is constructivizable inside } B\}.$$

The next theorem gives the precise estimates of the rank of inner constructivizability for hereditary finite superstructures.

Theorem 1. *Suppose $\mathfrak{M}$ is a structure of computable signature. Then*
1) if $\mathfrak{M}$ is finite then $\mathrm{cr}(\mathbb{HF}(\mathfrak{M})) = \omega$,
2) if $\mathfrak{M}$ is infinite then $\mathrm{cr}(\mathbb{HF}(\mathfrak{M})) \leqslant 2$.

We now begin the proof. Assume as usual that, for any $n \in \omega$, $HF_n(M)$ is the set of all elements from $\mathbb{HF}(\mathfrak{M})$ with rank less or equal to n. It is easy to see that in case when $\mathfrak{M}$ is finite, $HF_n(M)$ is finite for all $n \in \omega$, and hence it is clear that, for any $n \in \omega$, $\mathbb{HF}(\mathfrak{M})$ is not constructivizable inside $HF_n(\mathfrak{M})$, thus the first statement is true. The second statement comes from the following

Theorem 2. *If $\mathfrak{M}$ is infinite then the hereditary finite superstructure $\mathbb{HF}(\mathfrak{M})$ is constructivizable inside $HF_2(M)$.*

Proof. First, we construct an $\mathbb{HF}(\mathfrak{M})$-constructivization ν of the standard model of arithmetic $\mathcal{N} = \langle \omega, \leq, +, \cdot, s, 0\rangle$ inside $HF_2(M)$. For this we will use a cardinal presentation of natural numbers on the set M: with any $n \in \omega$ we connect the collection of all subsets of M containing exactly n elements, i.e.

$$\nu^{-1}(n) \leftrightharpoons \{a \subseteq M \mid \mathrm{card}(a) = n\}.$$

The numeration ν defined in this way is called a *cardinal numeration*. Relative to this numeration, any two subsets of M represent the same natural number if and only if there exists a bijection from one subset onto another. We will represent functions whose domains and ranges are finite subsets of M by means of elements of $\mathbb{HF}(\mathfrak{M})$ with rank 2. Namely, any function $f = \{\langle u_0, v_0\rangle, \ldots, \langle u_n, v_n\rangle\}$, where $u_0, \ldots, u_n, v_0, \ldots, v_n \in M$, is represented by any element (of rank 2) of the form

$$\{w_0, \ldots, w_n, \{u_0, w_0\}, \ldots, \{u_n, w_n\}, \{u_0, v_0, w_0\}, \ldots, \{u_n, v_n, w_n\}\},$$

where $w_0, \ldots, w_n \in M \setminus \{u_0, \ldots, u_n, v_0, \ldots, v_n\}$ are pairwise different (such elements do exist since M is infinite). Let C_f be the set of all such presentations of f, and let

$$C = \cup\{C_f \mid f \text{ is a finite function with } \mathrm{dom}(f) \subseteq M \text{ and } \mathrm{rng}(f) \subseteq M\}.$$

It is clear that $C \subseteq HF_2(M)$ and, moreover, C is Δ_0-definable in $\mathbb{HF}(\mathfrak{M})$. It is also easy to write down Δ_0-formulas defining, for any element $c \in C$ which

represents some finite function f_c, the sets $\mathrm{dom}(f_c)$ and $\mathrm{rng}(f_c)$ — the domain and the range of f_c respectively, and a Δ_0-formula which is true if and only if f_c is a bijection. So it follows that the numeration equivalence relation for the cardinal numeration ν is Σ-definable inside $HF_2(M)$: for any finite $a, b \subseteq M$,

$$\nu(a) = \nu(b) \iff \exists c \in C\ ((f_c \text{ is a bijection}) \wedge (\mathrm{dom}(f_c) = a) \wedge (\mathrm{rng}(f_c) = b)).$$

In the same way, for the natural order relation $\leq$ we have

$$\nu(a) \leq \nu(b) \iff \exists a' \in HF_1(M)\ ((\nu(a') = \nu(a)) \wedge (a' \subseteq b)),$$

$$\nu(a) < \nu(b) \iff \exists a', b' \in HF_1(M)\ ((\nu(a') = \nu(a)) \wedge (b = a' \cup b') \wedge (a' \cap b' = \varnothing) \wedge (b' \neq \varnothing)),$$

hence, since $\nu(a) \neq \nu(b)$ iff $((\nu(a) < \nu(b)) \vee (\nu(b) < \nu(a)))$, we get that both the numbering equivalence relation and the order relation are Δ-definable inside $HF_2(M)$.

For the operations of addition and multiplication we have that

$$\nu(a) + \nu(b) = \nu(c) \iff \exists a', b' \in HF_1(M)\ ((\nu(a') = \nu(a)) \wedge (\nu(b) = \nu(b')) \wedge (c = a' \cup b') \wedge (a' \cap b' = \varnothing)$$

$$\nu(a) \cdot \nu(b) = \nu(c) \iff \exists c' \in HF_2(M)\ ((\cup c' = c) \wedge (\text{“}c' = \{a'_1, \ldots, a'_{\nu(b)}\}\text{”}) \wedge (\text{“}a'_i \cap a'_j = \varnothing \text{ then } i \neq j\text{”}) \wedge (\text{“}\nu(a'_i) = \nu(a) \text{ for all } i\text{”})),$$

where “$c' = \{a'_1, \ldots, a'_{\nu(b)}\}$” denotes the formula

$$\exists c'' \in HF_1(M)\ ((\nu(c'') = \nu(b)) \wedge \forall a' \in c' \exists! x \in a' (x \in c'')).$$

Thus, relative to the cardinal numbering ν, the operations of additions and multiplication of natural numbers are Δ-definable inside $HF_2(M)$.

Recall that, for arbitrary structure $\mathfrak{M}$, a *coding scheme* [5] $\mathcal{C}$ consists of a set $N^{\mathcal{C}} \subseteq M$ and a linear order $<^{\mathcal{C}}$ on $N^{\mathcal{C}}$ such that

$$\langle N^{\mathcal{C}}, <^{\mathcal{C}} \rangle \simeq \langle \omega, < \rangle,$$

and an injective mapping $\pi^{\mathcal{C}}$ from the set of all finite sequences of elements of M into M. For a given coding scheme $\mathcal{C}$ we will denote by $\dot{0}, \dot{1}, \dot{2}, \ldots$ the corresponding elements of $N^{\mathcal{C}}$, relative to $<^{\mathcal{C}}$. Together with $\mathcal{C}$, we will also consider the predicate $Seq^{\mathcal{C}}(x)$ on $\mathfrak{M}$, which is true in case then $x = \pi^{\mathcal{C}}(\emptyset)$ or $x = \pi(\langle m_0, \ldots, m_n \rangle)$ for some $m_0, \ldots, m_n \in M$, and functions $lh^{\mathcal{C}}(x)$, $pr^{\mathcal{C}}(x, \dot{m})$, which gives correspondingly the length and the m-th element of the tuple with code x, and gives $\dot{0}$ in case of mismatch of the arguments. A structure $\mathfrak{M}$ is called *acceptable* [5] if it has a coding scheme $\mathcal{C}$ such that functions and relations $N^{\mathcal{C}}$, $<^{\mathcal{C}}$, $Seq^{\mathcal{C}}$, $lh^{\mathcal{C}}$, $pr^{\mathcal{C}}$ are definable in $\mathfrak{M}$.

We introduce the (multivalued) coding scheme $\mathcal{C}_*$ for coding finite sequences of elements from M by elements from $HF_2(M)$, such that $N^{\mathcal{C}_*} = \nu^{-1}(\omega)$, and

$Seq^{\mathcal{C}_*}$, $lh^{\mathcal{C}_*}$ and $pr^{\mathcal{C}_*}$ are Δ-definable in $\mathbb{HF}(\mathfrak{M})$ inside $HF_2(M)$. The set of codes of a tuple $\langle m_0, \ldots, m_k\rangle \in M^{k+1}$ in the coding scheme $\mathcal{C}_*$ is equal, by definition, to the set of all elements of the form

$$\{\{m_0, u_0\}, \ldots, \{m_k, u_0, \ldots, u_k\}, u_0, \ldots, u_k\},$$

there $u_0, \ldots, u_k$ are pairwise different elements from M such that $\{u_0, \ldots, u_k\} \cap \{m_0, \ldots, m_k\} = \varnothing$. It is easy to see that the relation $Seq^{\mathcal{C}_*}$ and the functions $lh^{\mathcal{C}_*}$ and $pr^{\mathcal{C}_*}$ are Δ-definable inside $HF_2(M)$.

Having a cardinal $\mathbb{HF}(\mathfrak{M})$-constructivization of the standard model of arithmetic $\mathcal{N}$, the coding scheme $\mathcal{C}_*$, and some fixed constructivization γ (in sense of the classical theory of constructive models) of the admissible set $\mathbb{HF}(\mathcal{N})$, we construct the $\mathbb{HF}(\mathfrak{M})$-constructivization ν_* of $\mathbb{HF}(\mathfrak{M})$ inside $HF_2(M)$ in the following way. Suppose $a \in \mathbb{HF}(\mathfrak{M})$; we let $(\nu_*)^{-1}(a)$ to be equal to the set of all elements of the form

$$\{a_\varkappa, \{m_0, u_0\}, \ldots, \{m_k, u_0, \ldots, u_k\}, u_0, \ldots, u_k\},$$

where $\varkappa \in HF(\omega)$ and $m_0, \ldots, m_k \in M$ are such that $a = \varkappa(m_0, \ldots, m_k)$ in the notations of [3], the set $a_\varkappa \subseteq M$ satisfies the condition $\nu(a_\varkappa) = \gamma^{-1}(\varkappa)$, and elements $u_0, \ldots, u_k$ from M are pairwise different and $\{u_0, \ldots, u_k\} \cap \{m_0, \ldots, m_k\} = \{u_0, \ldots, u_k\} \cap a_\varkappa = \{m_0, \ldots, m_k\} \cap a_\varkappa = \varnothing$.

The numeration ν_* defined in such way is, in fact, a constructivization of $\mathbb{HF}(\mathfrak{M})$ inside $HF_2(M)$. Indeed, the equality relation and the membership relation are defined by mutual recursion in the following way:

$$\varkappa_1(\bar{m}_1) \in \varkappa_2(\bar{m}_2) \iff \exists \varkappa' \in \varkappa_2(\varkappa_1(\bar{m}_1) = \varkappa'(\bar{m}_2)),$$

$$\varkappa_1(\bar{m}_1) \subseteq \varkappa_2(\bar{m}_2) \iff \forall \varkappa' \in \varkappa_1 \exists \varkappa'' \in \varkappa_2(\varkappa'(\bar{m}_1) = \varkappa''(\bar{m}_2)),$$

$$\varkappa_1(\bar{m}_1) = \varkappa_2(\bar{m}_2) \iff (\varkappa_1(\bar{m}_1) \subseteq \varkappa_2(\bar{m}_2)) \wedge (\varkappa_2(\bar{m}_2) \subseteq \varkappa_1(\bar{m}_1)).$$

Since the recursive part of this definition corresponds to the preimage of the set of natural numbers $\nu^{-1}(\omega)$, there exist Σ-formulas which define the numeration equivalence relation and the preimage of the membership relation for ν_* inside $HF_2(M)$.

Examples of structures $\mathfrak{M}$, for which $\mathrm{cr}(\mathbb{HF}(\mathfrak{M})) = 2$, are infinite models of empty signature, dense linear orders, and, more interesting, the structure $\langle \omega, s\rangle$ of natural numbers with successor function. Indeed, if we denote by $\mathrm{Th}_{WM}(\mathfrak{M})$ the theory of a structure $\mathfrak{M}$ in the language of weak monadic second order logic, then the following lemma is true.

Lemma 1. *If* $\mathfrak{M}$ *is infinite and* $\mathrm{Th}_{WM}(\mathfrak{M})$ *is decidable then* $\mathrm{cr}(\mathbb{HF}(\mathfrak{M})) = 2$.

Proof. Suppose, for a contradiction, that $\mathrm{cr}(\mathbb{HF}(\mathfrak{M})) < 2$. Then, in particular, the standard model of arithmetic $\mathcal{N}$ is $\mathbb{HF}(\mathfrak{M})$-constructivizable inside $HF_1(M)$, hence $\mathcal{N}$ is interpretable in $\mathfrak{M}$ by means of weak monadic second order logic. So $\mathrm{Th}(\mathcal{N}) \leq_m \mathrm{Th}_{WM}(\mathfrak{M})$, and so decidability of $\mathrm{Th}_{WM}(\mathfrak{M})$ implies decidability of the elementary theory of the standard model of arithmetic, a contradiction.

From the Büchi result [2] about decidability of $\mathrm{Th}_{WM}(\langle\omega, s\rangle)$ and the previous lemma we get that $\mathrm{cr}(\mathbb{HF}(\langle\omega, s\rangle)) = 2$.

An example of structure $\mathfrak{M}$ for which $\mathrm{cr}(\mathbb{HF}(\mathfrak{M})) = 0$ is, obviously, the standard model of arithmetic $\mathcal{N}$. An example of structure for which the hereditary finite superstructure has rank of inner constructivizability 1, is the field $\mathbb{R}$ of real numbers. First, we establish one general result.

Lemma 2. *If $\mathbb{P}$ is a field of characteristic 0 then the standard model of arithmetic is constructivizable in $\mathbb{HF}(\mathbb{P})$ inside $HF_1(\mathbb{P})$.*

Proof. We build an $\mathbb{HF}(\mathbb{P})$-constructivization μ of the standard model of arithmetic $\langle\omega, \leq, +, \cdot, s, 0\rangle$ inside $HF_1(\mathbb{P})$. Since $\mathbb{P}$ is a field of characteristic 0, the set $\mathbb{N} = \{0, 1, 1+1, \ldots\}$ is a subset of $\mathbb{P}$. As the requested constructivization we take a mapping $\mu : \mathbb{N} \to \omega$ defined as follows: $\mu^{-1}(n) = \underbrace{1 + \ldots + 1}_{n}$ for all $n \in \omega$.

The set of "natural numbers" $\mathbb{N} \subseteq \mathbb{P}$ is Σ-definable in $\mathbb{HF}(\mathbb{P})$ inside $HF_1(\mathbb{P})$: for $t \in \mathbb{P}$ we have

$$t \in \mathbb{N} \iff \mathbb{HF}(\mathbb{P}) \models \exists a\, ((a \subseteq \mathbb{P}) \wedge (0 \in a) \wedge \forall x \in a \\ (x \neq 0 \to \exists y \in a(x = y + 1)) \wedge (t = \max(a)),$$

where $t = \max(a)$ denotes the formula $(t \in a) \wedge \neg(t + 1 \in a)$. The numeration equivalence relation for μ coinsides with the equality relation on $\mathbb{N}$, the order relation is Δ-definable in $\mathbb{HF}(\mathbb{P})$ inside $HF_1(\mathbb{P})$: for $n, m \in \mathbb{N}$

$$\mu(n) \leq \mu(m) \iff \mathbb{HF}(\mathbb{P}) \models \exists a \exists b\, ((a = \{0, 1, \ldots, n\}) \\ \wedge (b = \{0, 1, \ldots, m\}) \wedge (a \subseteq b)),$$

$$\mu(n) \not\leq \mu(m) \iff \mu(m) < \mu(n) \iff (\mu(m) \leq \mu(n)) \wedge (n \neq m)).$$

The operations of addition and multiplication on $\mathbb{N}$ are induced by the corresponding operations of the field $\mathbb{P}$, and so they are Δ-definable in $\mathbb{HF}(\mathbb{P})$ inside $HF_1(\mathbb{P})$.

Corollary 1. *If $\mathbb{P}$ is a field of characteristic 0, then the weak monadic second order theory $\mathrm{Th}_{WM}(\mathbb{P})$ is undecidable. In particular, weak monadic second order theories $\mathrm{Th}_{WM}(\mathbb{R})$, $\mathrm{Th}_{WM}(\mathbb{Q}_p)$ and $\mathrm{Th}_{WM}(\mathbb{C})$ are undecidable.*

Theorem 3. $\mathrm{cr}(\mathbb{HF}(\mathbb{R})) = 1$.

Proof. By Lemma 2, the standard model of arithmetic is constructivizable in $\mathbb{HF}(\mathbb{R})$ inside $HF_1(\mathbb{R})$. For the existence of a constructivization of $\mathbb{HF}(\mathbb{R})$ inside $HF_1(\mathbb{R})$, a necessary and sufficient condition is the existence of a Σ-definable inside $HF_1(\mathbb{R})$ coding scheme for finite sequences of reals.

We introduce the coding scheme for finite sequences of reals by the pairs of finite sets of reals. A tuple $\langle a_0, \ldots, a_{n-1}\rangle \in \mathbb{R}^n$ is represented by the set of pairs $\langle\{a_0, \ldots, a_{n-1}\}, \{q_0, \ldots, q_{n-1}\}\rangle$, where elements $q_0, \ldots, q_{n-1} \in \mathbb{R}$ are defined in the following way: we find the least distance $d = \min\{|a_i - a_j| \mid i, j < n, a_i \neq a_j\}$ between distinct elements of the tuple and let $q_i = a_i + \frac{d}{2^{i+2}}$ for all $i < n$

(under this assumption $q_0, \ldots, q_{n-1}$ are pairwise different even in case then some of $a_0, \ldots, a_{n-1}$ are equal. The set of pairs coding finite sequences of reals is Σ-definable inside $HF_1(\mathbb{R})$ since there exists the corresponding constructivization of natural numbers. The projecting function is Δ-definable inside $HF_1(\mathbb{R})$: $a_i = pr(\langle\{a_0, \ldots, a_{n-1}\}, \{q_0, \ldots, q_{n-1}\}\rangle, \mu^{-1}(i))$ if and only if there exists $q_i \in \{q_0, \ldots, q_{n-1}\}$ such that $|a_i - q_i| = \frac{d}{2^{i+2}}$. In the same way it is easy to show that the function lh in the described coding scheme is also Δ-definable inside $HF_1(\mathbb{R})$.

We define a constructivization μ_* of the admissible set $\mathbb{HF}(\mathbb{R})$ inside $HF_1(\mathbb{R})$ in the following way. Suppose $a \in \mathbb{HF}(\mathbb{R})$; we let $(\mu_*)^{-1}(a)$ to be equal to the set of all triples of the form

$$\langle \mu^{-1}(\gamma(\varkappa)), \{a_0, \ldots, a_n\}, \{q_0, \ldots, q_n\}\rangle,$$

where elements $\varkappa \in HF(\omega), a_0, \ldots, a_n \in \mathbb{R}$ are such that $a = \varkappa(a_0, \ldots, a_n)$, $\gamma : \omega \to HF(\omega)$ is a constructivization of the admissible set $\mathbb{HF}(\mathcal{N})$, and the pair $\langle\{a_0, \ldots, a_n\}, \{q_0, \ldots, q_n\}\rangle$ is coding the tuple $\langle a_0, \ldots, a_n\rangle$ in the coding scheme described above.

The mapping μ_* thus defined is a constructivization (of dimension 3) of the admissible set $\mathbb{HF}(\mathbb{R})$ inside $HF_1(\mathbb{R})$.

From Theorem 1, in particular, could be derived some constructive analogs of some results (namely, of Theorems 18, 19, 20) from [4] about the definability in multisorted languages, where the type of a variable describes the rank of its possible values.

References

1. J. Barwise: Admissible Sets and Strucrures. Springer-Velag. Berlin. (1975)
2. J.R. Buchi: Weak second order arithmetic and finite automata. Z. Math. Logik Grundl. Math.**6** (1960) 66-92
3. Yu.L. Ershov: Definability and Computability. Plenum. New York. (1996)
4. R. Montague: Recursion theory as a branch of model theory. Proceedings of the Third International Congress for Logic, Methodology and Philosophy of Science. Amsterdam. (1967) 63-86
5. Y.N. Moschovakis: Elementary induction on abstract structures. Amsterdam. (1974)

A Sharp Phase Transition Threshold for Elementary Descent Recursive Functions

Arnoud den Boer and Andreas Weiermann

Fakulteit Bètawetenschappen
Departement Wiskunde
P.O. Box 80010
3508 TA Utrecht
The Netherlands
avdenboer@gmail.com,weierman@math.uu.nl

Abstract. Harvey Friedman introduced natural independence results for the Peano axioms via certain schemes of combinatorial well-foundedness. We consider here parameterized versions of this scheme and classify exactly the threshold for the transition from provability to unprovability in PA. For this purpose we fix a natural bijection between the ordinals below ε_0 and the positive integers and obtain an induced natural well ordering $\prec$ on the positive integers. We classify the asymptotic of the associated global count functions. Using these asymptotics we classify precisely the phase transition for the parameterized hierarchy of elementary descent recursive functions and hence for the combinatorial well-foundedness scheme. Let $\mathrm{CWF}(g)$ be the assertion

$$(\forall K)(\exists M)(\forall m_0,\dots,m_M)[\forall i \le M(m_i \le K+g(i)) \to \exists i < M(m_i \preceq m_{i+1})].$$

Let $f_\alpha(i) := i^{\mathrm{F}_\alpha^{-1}(i)}$ where F_α^{-1} denotes the functional inverse of the α-th function from the fast growing hierarchy. Then

$$\mathrm{PA} \vdash \mathrm{CWF}(f_\alpha) \iff \alpha < \varepsilon_0.$$

Keywords: proof theory, phase transition, elementary descent recursive functions, multiplicative number theory

1 Introduction

The Peano Axioms were originally designed hoping that every true statement in the language for natural numbers is a consequence of these axioms. It has therefore been a great surprise when Gödel showed in 1931 that there are true statements about the natural numbers which do not follow from the Peano Axioms (PA). The example Gödel came with, was somewhat artificial and thus not completely satisfying. (It looked like the sentence 'this sentence is true but unprovable').

Since then logicians have therefore been searching for mathematically relevant examples for independent statements. A breakthrough has been obtained

in 1977 by Paris and Harrington [7] who showed that a slight modification of the finite Ramsey theorem is unprovable in PA.

Around 1980 H. Friedman established further striking natural examples for independent statements. He showed that the miniaturization of Kruskal's theorem is not provable in predicative analysis. Moreover he introduced principles of combinatorial well-orderedness and combinatorial well-quasi-orderedness as paradigms for independent assertions [9].

In 1995 he studied jointly with Sheard [5] combinatorial well-orderedness principles with respect to abstract elementary recursive ordinal notation systems. In this article we fix a concrete example for an elementary recursive ordinal notation system for ε_0 which goes back to Schütte 1977.

For this specific natural well-ordering we are able to classify exactly the phase transition from provability to unprovability for the underlying principle of combinatorial well-orderedness. This is part of a general research program on phase transitions in logic and combinatorics initiated by the second author (See, for example, [11–13]). This type of phase transition is different from phase transitions known for random graphs or the satisfiability problem where a probabilistic setting is assumed. In some sense it is closer to the situation in physics where phase transition is a type of behaviour wherein small changes of a parameter of a system cause dramatic shifts in some globally observed behaviour of the system, such shifts being usually marked by a sharp 'threshold point'. An everyday life example of such thresholds are ice melting and water boiling temperatures.

The results of this paper reflect specific properties of the natural well-odering of ε_0, in particular numbertheoretic aspects of the coding. The approach is related to Arai's investigation on the slowly well-orderedness of ε_0 [1] but instead of a norm based approach we work directly with natural number codes for ordinals. We therefore had to employ methods from multiplicative number theory (Dirichlet series, Rankin's method) instead of additive methods to obtain the asymptotic of the count functions. Nevertheless in the unprovability part we make essential use of Arai's result. In particular we do not obtain the threshold for unprovability by an iteration of a renormalization operator as in the additive setting.

Moreover, we adapt parts of Arai's treatment to the current situation. It is still quite mysterious why this is possible and it seems that this problem is closely related to Burris central problem 12.21 [4] on finding general principles to explain why local additive results lift to global multiplicative results. In our situation we have a lift from an additive independence result to a multiplicative one.

1.1 Notation and definitions

With $\mathbb{N}$ we denote the natural numbers, starting at 0. Let $(p_i)_{i\geq 1}$ enumerate the prime numbers in increasing order. Let $\mathbb{P}$ be the set of all primes. By primitive recursion let us define the following linear ordering on $\mathbb{N}$:

$$m \prec n :\Leftrightarrow \Big(m \neq n \& \big(n = 0 \vee m = 1 \vee [\frac{m}{gcd(m,n)} = p_{m_1} \cdot \ldots \cdot p_{m_k}$$

$$\& \frac{n}{gcd(m,n)} = p_{n_1} \cdot \ldots \cdot p_{n_l} \& \forall i \leq k \exists j \leq l(m_i \prec n_j])\Big).$$

Then $\langle N \setminus \{0\}, \prec \rangle \simeq \langle \varepsilon_0, < \rangle$. The corresponding isomorphism $\varphi : N \setminus \{0\} \to \varepsilon_0$ is defined as follows. Assume that $\alpha = \omega^\beta + \gamma$ is in Cantor normal form. Then $\varphi(\alpha) := p_{\varphi(\beta)} \cdot \varphi(\gamma)$. Then additive indecomposable ordinal numbers are mapped onto multiplicative indecomposable natural numbers and surjectivity of the map follows by representing numbers by their associated prime factor decomposition.

This obersation gives rise to consider multiplicative number systems in the sense of Burris [4]. A multiplicative number system $\langle A, P, \cdot, 1, M \rangle$ is a countable free commutative monoid $\langle A, \cdot, 1 \rangle$ with P the set of indecomposable elements ('primes'), and M a multiplicative norm on A, that is a function $M : A \to \mathbb{N}$ such that $M(a) = 1 \Leftrightarrow a = 1$, such that $M(a \cdot b) = M(a) \cdot M(b)$ for all $a, b \in A$ and such that for every $n \geq 2$ the set $\{a \in A : M(a) = n\}$ is finite. Let $q_1 := p_2$ and $q_{k+1} := p_{q_k}$ for $k \geq 1$. Thus q_k has order type ω_k an iterated tower of omega's of hight k. If we put $q_0(d) := d$ and $q_{k+1}(d) := p_{q_k(d)}$ for $k \geq 1$ then $q_k(d)$ has order type $\omega_k(d)$, an iterated tower of omega's of hight k with a finite number d on the top. So $q_{k+1}(d)$ is the d-th member of the fundamental sequence for q_{k+2}.

For $K \geq 1$, let

$$Q_K := \{m \in \mathbb{N} : m \prec q_K\}$$

and define a norm M on Q_K simply by putting $M(n) := n$.
Then $\langle Q_K, \mathbb{P} \cap Q_K, \cdot \restriction (Q_K \times Q_K), 1, M \restriction Q_K \rangle$ is a multiplicative system. Let

$$C_{Q_K}(n) := \#\{a \in Q_K : M(a) \leq n\}$$

be its global count function.

For technical reasons we introduce a notation for iterated exponentiation. We write $a_1 := a, a_{k+1} := a^{a_k}, a_0(d) := d, a_{k+1}(d) := a^{a_k(d)}$ for $a, d \in \mathbb{R}, k \in \mathbb{N}$. In [3] the following lower- and upperbounds are proven via multiplicative number theory. The basic idea was to apply Rankin's method to appropriate Dirichlet generating series.

Lemma 1. (a) *Let* $\mathrm{e} = 2.71828\ldots$ *denote the Euler number. Let* $K \geq 3$ *and*

$$T(K) := \max\{e_4^{\mathrm{e}_3}, \mathrm{e}_K\}.$$

Then

$$C_{Q_K}(n) \geq \exp\left(2^{2-K} \frac{\ln(n)}{\ln_{K-1}(n)}\right)$$

for all $n \geq T(K)$.

(b) *For all* $K \geq 3$ *there is a constant* V *such that for all* n

$$C_{Q_K}(n) \leq \exp\left(V \frac{\ln(n)}{\ln_{K-1}(n)}\right.$$

Furthermore we conjecture

$$\ln(C_{Q_K}(n)) \sim \frac{\pi^2}{6\ln(2)} \left(\frac{\ln(n)}{\ln_{K-1}(n)} \right) \tag{1}$$

for $K \geq 3$.

It is also natural to consider ordinal segments as additive number systems in the sense of Burris [4]. An additive number system $\langle A, P, \cdot, 1, N \rangle$ is a countable free commutative monoid $\langle A, \cdot, 1 \rangle$ with P the set of indecomposable elements, and with N an additive norm on A, that is a function $N : A \to \mathbb{N}$ such that $N(a) = 0 \Leftrightarrow a = 1$, such that $N(a \cdot b) = N(a) + N(b)$ for all $a, b \in A$ and such that such that for every $n \geq 1$ the set $\{a \in A : N(a) = n\}$ is finite. In our situation it is natural to consider the following norm. Let $N(1) := 0$ and

$$N(\Pi_{i \in I} p_i^{m_i}) := \sum_{i \in I} m_i \cdot (N(i) + 1).$$

Then obviously N is an additive norm.
Let $c_{Q_K}(n) := \#\{a \in Q_K : N(a) = n\}$ be the local count function.
Bounds for the local count function have already been obtained in the literature on additive number theory, and even the asymptotic behaviour. For example a famous theorem of Hardy and Ramanujan says

$$c_{Q_2}(n) \sim \frac{\exp\left(\pi\sqrt{\frac{2}{3}n}\right)}{4\sqrt{3n}}$$

Related results for the set Q_K for $K \geq 3$ have been obtained by Yamashita in [14]. One has, for example,

$$\ln\left(c_{Q_2}(n)\right) \sim \frac{\pi^2}{6} \left(\frac{n}{\ln_{K-2}(n)} \right)$$

for $K \geq 3$. Not the analogy with conjecture (1).

1.2 Summary of the result

For any limit ordinal $\lambda < \varepsilon_0$, let $(\lambda[n])_{n \in \mathbb{N}}$ be the canonical fundamental sequence for λ. Thus if $\lambda = \omega^{\alpha_1} + \ldots + \omega^{\alpha_k} > \alpha_1 \geq \ldots \geq \alpha_k = \beta + 1$, then $\lambda[n] = \omega^{\alpha_1} + \ldots + \omega^{\alpha_{k-1}} + \omega^{\beta} \cdot n$ and if $\lambda = \omega^{\alpha_1} + \ldots + \omega^{\alpha_k} > \alpha_1 \geq \ldots \geq \alpha_k \in Lim$, then $\lambda[n] = \omega^{\alpha_1} + \ldots + \omega^{\alpha_{k-1}} + \omega^{\alpha_k[n]}$.
For all ordinals $\alpha \leq \varepsilon_0$ we define function $\mathrm{F}_\alpha : \mathbb{N} \to \mathbb{N}$ as follows
$\mathrm{F}_0(x) := 2^x$, $\mathrm{F}_{\alpha+1}(x) := \mathrm{F}_\alpha^{x+1}(x)$, where the upper index denotes the number of iterations, and $\mathrm{F}_\lambda(x) := \mathrm{F}_{\lambda[x]}(x)$ if λ is a limit.

A basic result is that PA proves the totality of F_α iff $\alpha < \varepsilon_0$. Thus PA does not prove the assertion $(\forall x)(\exists y)[\mathrm{F}_{\varepsilon_0}(x) = y]$. See, e.g., [5] for a proof. Of course the inverse function $\mathrm{F}_{\varepsilon_0}^{-1}$ is provably recursive in PA. [Let us briefly indicate the argument since we need if for stating the next theorems. It is well known that

the graph of $\mathrm{F}_{\varepsilon_0}$ elementary recursive (see, e.g., the appendix of [2] for a proof of this folklore result). Then $F^{-1}_{\varepsilon_0}(x)$ is the smallest $z \leq x$ such that $\mathrm{F}_{\varepsilon_0}(z) > x$.] In this article we establish the following result.

Theorem 1. (a) *If* $\alpha = \varepsilon_0$ *then*

$$\mathrm{PA} \nvdash (\forall K)(\exists M)\Big((\forall m_0, \cdots, m_{M-1}) \\ (\forall i < M)[m_i \leq K + i^{|i|_{\mathrm{F}^{-1}_\alpha(i)}}] \Rightarrow (\exists i < M-1)[m_i \preceq m_{i+1}]\Big).$$

(b) *If* $\alpha < \varepsilon_0$ *then*

$$\mathrm{PA} \vdash (\forall K)(\exists M)\Big((\forall m_0, \cdots, m_{M-1}) \\ (\forall i < M)[m_i \leq K + i^{|i|_{\mathrm{F}^{-1}_\alpha(i)}}] \Rightarrow (\exists i < M-1)[m_i \preceq m_{i+1}]\Big).$$

So if we define the function:

$$\mathrm{D}(K, h) := \max\{M : (\exists m_0 \succ \cdots \succ m_{M-1})(\forall i < M)[m_i \leq K + i^{|i|_{h(i)}}]\}$$

our theorem is equivalent to

$$\mathrm{PA} \vdash (\forall K)(\exists M) M = \mathrm{D}(K, \mathrm{F}^{-1}_\alpha) \iff \alpha < \varepsilon_0.$$

This theorem is the multiplicative analogue of the following result of Arai [1] and Weiermann [11]:

Theorem 2. (a) *If* $\alpha = \varepsilon_0$ *then*

$$\mathrm{PA} \nvdash (\forall K)(\exists M)\Big((\forall m_0, \cdots, m_{M-1}) \\ (\forall i < M)[N(m_i) \leq K + |i| \cdot |i|_{\mathrm{F}^{-1}_\alpha(i)}] \Rightarrow (\exists i < M-1)[m_i \preceq m_{i+1}]\Big)$$

(b) *If* $\alpha < \varepsilon_0$ *then*

$$\mathrm{PA} \vdash (\forall K)(\exists M)\Big((\forall m_0, \cdots, m_{M-1}) \\ (\forall i < M)[N(m_i) \leq K + |i| \cdot |i|_{\mathrm{F}^{-1}_\alpha(i)}] \Rightarrow (\exists i < M-1)[m_i \preceq m_{i+1}]\Big)$$

Or, with

$$\mathrm{L}(K, h) := \max\{M : (\exists m_0 \succ \cdots \succ m_{M-1})(\forall i < M)[N(m_i) \leq K + |i| \cdot |i|_{h(i)}]\}$$

our theorem is equivalent to

$$\mathrm{PA} \vdash (\forall K)(\exists M)[M = \mathrm{L}(K, \mathrm{F}^{-1}_\alpha)] \iff \alpha < \varepsilon_0.$$

So by replacing the additive norm N with the multiplicative norm M, and replacing the function $K + |i| \cdot |i|_{\mathrm{F}^{-1}_\alpha(i)}$ with $K + i^{|i|_{\mathrm{F}^{-1}_\alpha}}$, we get again an independence result. The first is obtained by using bounds on the local countfunction

c_{Q_K}, the latter by using bounds on the global count function C_{Q_K}. This suggests the existence of a relation between local additive and global multiplicative. In fact, this parallelism is stated as an open problem (12.21) in the book of Burris [4].

In [1] it is shown that, with $l(i) = |i|^2$, $\mathrm{F}_{\varepsilon_0}$ is bounded by $K \mapsto \mathrm{L}(2K+16, l)$. Therefore the latter function is not provably total in PA. In Section 3 of this paper we show that $\mathrm{F}_{\varepsilon_0}$ is also bounded by a function which involves D. This yields the unprovability assertion.
For the provability result, Section 2, we show that for $\alpha < \varepsilon_0$, D is bounded from above by a function which is primitive recursive in F_α. This implies that D is provable recursive in PA.

Of course, we also need to show that the assertion about which the independence result is retrieved, is true indeed. This is a simple consequence of König's Lemma (every finitely-branched infinite tree has a path), and the fact that an descending chain of ordinals cannot be infinite. Remember that $\langle N \setminus \{0\}, \prec\rangle \simeq \langle \mathcal{E} \leq \varepsilon_0, <\rangle$

Lemma 2. *Let* $||\ \ ||$ *be any norm and let* $f : \mathbb{N} \to \mathbb{N}$ *be any function . Then the assertion*

$$(\forall K)(\exists M)\forall \alpha_0, \ldots, \alpha_M(\forall i \leq M : ||\alpha_i|| \leq K + f(i) \Rightarrow \exists i : \alpha_i \preceq \alpha_{i+1})$$

is true in the standard model.

Proof. Let

$$b := \{< \alpha_0, \ldots, \alpha_M >: (\forall i \leq M)[||\alpha_i|| \leq K + f(i)] \& \alpha_0 \succ \ldots \succ \alpha_M\}.$$

Suppose the Lemma is false. Then $(\exists K)(\forall M)[\exists < \alpha_0, \ldots, \alpha_M > \in b]$.
Then b is an infinite tree. b is also finitely branched, for suppose $< \alpha_0, \ldots, \alpha_M > \in b$. If $< \alpha_0, \ldots, \alpha_{M+1} > \in b$ then $||\alpha_{M+1}|| \leq K + f(m+1)$, so there are only finite possible succesors.
By König's Lemma, there is a path $f : \mathbb{N} \to \mathbb{N}$, $(\forall i)[< f(0), f(1), \ldots, f(i) > \in b]$. But then $f(0), f(1), \ldots$ are isomorph with an infinite descending chain of ordinals, which is impossible.

2 The provability assertion

With $|x|$ we denote the binary length of x. Thus $|0| := 0$ and $|x| := \lceil \log_2(x+1) \rceil$ for $x > 0$. We call a function $h : \mathbb{N} \to \mathbb{N}$ *unbounded* if h is weakly increasing and $\lim_{x\to\infty} h(x) = \infty$. If h is unbounded, we let $h^{-1}(x) := \min\{n \in \mathbb{N} : x < h(n)\}$. We call an unbounded function h *log-like* if $(\forall x > 0)[h(x-1) < h(x) \Rightarrow (\exists y)[x = 2^y]]$. We call an unbounded function h *exp-like* if $(\forall x)[h(x) \in \{0\} \cup \{2^y : y \in \mathbb{N}\}]$. For $n \geq 1$ let $\ln_1(n) := \max\{(1, \ln(n)\}$ and $\ln_{k+1}(n) := \max\{1, \ln_1(\ln_k(n))\}$.

Lemma 3. (a) *If* $x \geq 4$ *then* $x^2 \leq 2^x$.
(b) *If* $x \geq 3$ *then* $3_n(x) \leq 2_n(2x)$.

(c) $|2_K(y)|_K \geq y$.
(d) $2_{K-1}(|N+1|_K) \leq N+1$ *for all* $K \geq 4, N+1 \geq 2_{K-1}(K-2)$.
(e) *If* $K \geq 4$ *and* $1 \leq m_0 \leq K+1$ *then* $m_0 \preceq q_{K-1}$.
(f) *If* h *is loglike then*h^{-1} *is explike.*
(g) *If* F *is explike then* F^{-1} *is loglike.*

Proof. Assertions (a) and (b) are contained in Proposition 14 from [1].
Assertion (c) is proved by induction on K. If K=1 then

$$|2_1(y)| = \lceil \log_2(2^y+1) \rceil \geq \lceil \log_2(2^y) \rceil = y.$$

Further $|2_{K+1}(y)|_{K+1} = |\quad |2_1(2_K(y))| \quad |_K \geq |2_K(y)|_K \geq y$.
For proving assertion (d) we first show that it is true for $N+1 = 2_{K-1}(K-2)$ by using induction on $K \geq 4$.
$K = 4 : 2_{4-1}(|2_{4-1}(4-2)|_4) = 2_4 = N+1$.
$K > 4 : 2_K(|N+1|_{K+1}) = 2_1(2_{K-1}(||N+1||_K) \leq 2_1(|N+1|)$.
Take $N+1 = 2_{K-1}(K-2)$. Then

$$2_K(|2_{K-1}(K-2)|_{K+1}) \leq 2_1(|2_{K-1}(K-2)|) = 2_1(2_{K-2}(K-2)) = 2_{K-1}(K-2).$$

Since $2_{K-1}(|N+2|_K) - 2_{K-1}(|N+1|_K) \leq 1$, the assertion follows for all $N+1 \geq 2_{K_1}(K-2)$
Proof of assertion (e). For $K = 4$ it is checked by hand.
For $K \geq 5$ we prove the assertion by induction on m_0.
Suppose that $m_0 \succ q_{K-1}$.
If $q_{K-1}|m_0$ then, since $m_0 \neq q_{K-1}$, $K+1 \geq m_0 > q_{K-1} > K+1$. Contradiction.
Thus q_{K-1} is not a divisor of m_0. Since q_{K-1} is a prime number this implies that $\gcd(m_0, q_{K-1}) = 1$. Then by definition of $\prec$ there is a prime $p_j|m_0$ s.t. $q_{K-1} \prec p_j$, i.e. $p_{q_{K-2}} \prec p_j$, i.e. $q_{K-2} \prec j$. (Here we use the fact that $\forall a, b \neq 0, 1 \quad a \prec b$ iff $p_a \prec p_b$, which follows easily from the definition of $\prec$.)
But $j < m_0 \leq K+1$, and thus $j \leq K$. Since $K-1 \geq 4$ we obtain $j \preceq q_{K-2}$ by induction hypothesis. Contradiction.
For a proof of assertion (f) assume that h is loglike. $h^{-1}(x) = \min\{n \in N : h(n) > x\}$. Suppose $h^{-1}(x) = m \neq 0$. Then $h(m) > x$ and $h(m-1) \leq x$, hence $h(m) > h(m-1)$. Hence there exists a y such that $m = 2^y$.
Thus $(\forall x)h^{-1}(x) \in \{0\} \bigcup \{2^y : y \in N\}$, hence, h^{-1} is exp-like.
(g): Let F be exp-like. $F^{-1}(x) = \min\{n \in N : F(n) > x\}$. Let $x > 0$ be arbitrary.
Suppose $m = F^{-1}(x) > F^{-1}(x-1) = m'$. If $F(m') > x$ then $m = F^{-1}(x) \leq m' = F^{-1}(x-1)$ but we assumed $m > m'$.
Also impossible is $F(m') < x$ because then $x - 1 < F(m') < x$ which is contradictory.
Hence, $F(m') = x$, hence, $x = 2^y$ for some y.

Theorem 3. *Let* h *be the log-like function* $h = \mathrm{F}_\alpha^{-1}(i)$ *for some* $\alpha < \varepsilon_0$. *Let* $K \geq 4$ *and let* V *be as in assertion* (b) *of Lemma 1 where we assume that* V *is a positive integer. Then* $\mathrm{D}(K, h) \leq \max\{2\mathrm{F}_\alpha(K), 2_K(K-2), 2_{K+1}(5V)\}$

Proof. Fix $K \geq 4$ and let $N_1 := \max\{2 \cdot \mathrm{F}_\alpha(K), 2_K(K-2), 2_{K+1}(5V)\}$.

W.l.o.g. choose an arbitrary sequence $m_0, \ldots, m_{n-1}$ s.t. $m_0 \succ \cdots \succ m_{n-1}$ and $m_i \leq K + i^{(|i|_{h(i)})}$ for all $i = 0, \cdots, n-1$. We need to show $n \leq N_1$.
We proof this by contradiction. Assume $n > N_1$.

The function h is log-like, so $\mathrm{F}_\alpha = h^{-1}$ is exp-like. From this fact together with $K \geq 4$, it follows that there exists an $N \geq 4$ such that $N_1 = 2^{N+1}$.
We have $K \geq 4$ and $m_0 \leq K + 0^{|0|_{h(0)}} \leq K + 1$, so by Lemma 3.5 $m_0 \preceq q_{K-1}$.
By transitivity of $\preceq$, $m_i \prec q_{K-1}$ for all $i = 0, \cdots, n-1$.
Since $n > N_1 = 2^{N+1}$ we thus have $m_{2^N}, \cdots, m_{2^{N+1}-1} \in Q_{K-1}$.

The function h is log-like so $h(i) = h(2^N)$ for all $i \in [2^N, 2^{N+1} - 1]$.
Let $k := K + (2^{N+1} - 1)^{|2^{N+1}-1|_{h(2^N)}}$. Then we have $m_i \leq K + (i)^{|i|_{h(i)}} \leq k$ for all $i \in [2^N, 2^{N+1} - 1]$. Hence $C_{Q_{K-1}}(k) \geq card([2^N, 2^{N+1} - 1]) = 2^N$.
By assertion (b) of Lemma 1 we also have $C_{Q_{K-1}}(k) \leq \exp(V \frac{\ln_1(k)}{\ln_{K-2}(k)})$.
To reach a contradiction we'll show that $\exp(V \frac{\ln_1(k)}{\ln_{K-2}(k)}) < 2^N$, which is equivalent to

$$V \frac{\ln_1(k)}{N} < \ln_{K-2}(k) \ln(2). \tag{2}$$

Step one: $\frac{\ln_1(k)}{N} \leq \ln(2) \cdot (1 + 2|N+1|_K)$
Proof of step one:
By definition $N_1 \geq 2\mathrm{F}_\alpha(K)$, therefore $2^N \geq \mathrm{F}_\alpha(K) = h^{-1}(K)$.
By definition we have $h^{-1}(K) = \min\{n : K < h(n)\}$, so $K < h(h^{-1}(K))$.
We have $2^N \geq h^{-1}(K)$ thence $h(2^N) \geq h(h^{-1}(K)) > K$ (since h is weakly increasing) and hence $K \leq h(2^N) - 1$.
Thus

$$|N+1|_{h(2^N)-1} \leq |N+1|_K. \tag{3}$$

An easy induction on K shows $2_K(K-2) \geq 2^{K+1}$, hence

$$2^{N+1} = N_1 \geq 2_K(K-2) \geq 2^{K+1},$$

and thus $N \geq K$.
For k we have using (3)

$$\begin{aligned} k &= K + (2^{N+1} - 1)^{|2^{N+1}-1|_{h(2^N)}} \\ &\leq K + (2^{N+1})^{|N+1|_{h(2^N)-1}} \\ &= K + 2^{(N+1)|N+1|_{h(2^N)-1}} < N + 2^{(N+1)|N+1|_K} \\ &\leq 2^{N+1} \cdot 2^{(N+1)|N+1|_K} \end{aligned}$$

(since $N + y \leq y \cdot 2^{N+1}$ for all $y \geq 1$.)
Hence

$$\ln_1(k) \leq \ln(2^{N+1} \cdot 2^{(N+1)|N+1|_K}) = (N+1)(|N+1|_K + 1) \ln(2)$$

And thus

$$\frac{\ln_1(k)}{N} \leq \frac{(N+1)}{N}(|N+1|_K+1)\ln(2) \leq \left(\frac{5}{4}|N+1|_K+\frac{5}{4}\right)\ln(2)$$

(since $N \geq 4$)

$$\leq (\frac{5}{4}|N+1|_K+1+\frac{1}{4}\cdot|N+1|_K)\ln(2)$$

(since $|N+1|_K \geq 1$)

$$\leq (2|N+1|_K+1)\ln(2).$$

Step two: $V\cdot(1+2|N+1|_K) \leq \ln_{K-2}(k)$
(This together with step one proves (2) and hence the contradiction.)
Proof of step two:
Let $x := V(1+2|N+1|_K)$. We'll show $e_{K-2}(x) \leq k$.
From $N_1 \geq 2_{K+1}(5V)$ we conclude $N+1 \geq 2_K(5V)$ and obtain

$$|N+1|_K \geq |2_K(5V)|_K \geq 5V.$$

The last inequality holds by assertion (c) of Lemma 3. The assertion follows from $5V \geq 5 > 4$ which holds because V is a positive integer. This gives

$$\begin{aligned} 2x &= 2V(1+2|N+1|_K) < 4V+4V|N+1|_K \\ &< |N+1|_K V + 4V|N+1|_K = 5V|N+1|_K \\ &\leq (|N+1|_K)^2. \end{aligned}$$

Hence we get

$$e_{K-2}(x) \leq 3_{K-2}(x) \leq 2_{K-2}(2x)$$

by assertion (b) of Lemma 3. Further

$$2_{K-2}(2x) \leq 2_{K-2}((|N+1|_K)^2) \leq 2_{K-1}(|N+1|_K)$$

by applying assertion (a) of Lemma 3. $|N+1|_K \geq 4$ yields $(|N+1|_K)^2 \leq 2^{|N+1|_K}$ hence $2_{K-2}((|N+1|_K)^2) \leq 2_{K-2}(2^{|N+1|_K}) = 2_{K-1}(|N+1|_K)$.

Using assertion (d) of Lemma 3 we obtain

$$e_{K-2}(x) \leq 2_{K-1}(|N+1|_K) \leq N+1 < k$$

and we have reached a contradiction.
(The last inequality $N+1<k$ is true because

$$\begin{aligned} k &= K+(N_1-1)^{|N_1-1|_{h(2^N)}} \\ &\geq (N_1-1)^{|N_1-1|_{h(2^N)}} \\ &\geq (N_1-1)^1 = 2^{N+1}-1 > N+1). \end{aligned}$$

Corollary 1. *If* $\alpha < \varepsilon_0$ *then* $\mathrm{PA} \vdash (\forall K)(\exists M)M = \mathrm{D}(K,h)$

Proof. $\mathrm{D}(K,h)$ is bounded by a function which is primitive recursive in F_α, hence provably total in PA.

3 The unprovability assertion

Define the functions g_1, g, r as follows:
$r(n) := 2n + 16$,
$g_1(n) := \max\{2_{n+2}(n+1), 2_1(2_1(21)-1), 2^{T(n+3)}\}$,
where T is the function from assertion 2 of Lemma 1.
$g(n) := 6^{2_1(3n+20)-1} \cdot 2^{2_1(3n+20)} \cdot 2^{g_1(n)}$,
$l(i) := |i|^2$

Lemma 4. (a) $2^{(|i|-1)} \leq i \leq 2^{|i|} - 1$.
(b) *If* $i > g_1(n)$ *then* $l(|i|) \geq n + 3 + r(n)$.
(c) $i > g_1(n)$ *then* $(|i| - 1) \geq 8l(|i|)^2$.
(d) $q_{m+r(n)} \cdot 2^{g_1(n)} \leq g(n)$.
(e) $N(q_n) = n + 1$.
(f) *If* $m \succ q_n$ *then* $N(m) > n + 1$.

Proof. Proof of assertion (a). $|i| = \alpha$ yields $2^\alpha \leq i \leq 2^{\alpha+1} - 1$ hence $2^{|i|-1} \leq i \leq 2^{|i|} - 1$.
Proof of assertion (b). $i > g_1(n) \geq 2_{n+2}(n+1)$ yields

$$l(|i|) \geq ||2_{n+2}(n+1)||^2 = |2_{n+1}(n+1)+1|^2 \geq |2_{n+1}(n+1)|^2 = (2_n(n+1)+1)^2$$

For $n = 1$ we obtain $(2_1(1+1)+1)^2 = 25 > 22 = 3n + 19$.
For $n > 1$ observe that $(2_n(n+1)+1)^2$ grows faster in n then $3n + 19$.
Proof of assertion (c). If $\gamma \geq 21$ then $2^\gamma > 8(\gamma+1)^4$.
From $\beta \geq 8 \cdot 22^4$ we conclude $(\frac{1}{8}\beta)^{\frac{1}{4}} - 1 \geq 21$ hence

$$2^{(\frac{1}{8}\beta)^{\frac{1}{4}}-1} > 8((\frac{1}{8}\beta)^{\frac{1}{4}} - 1 + 1)^4 = \beta.$$

Applying assertion 1 to $|i|$ gives $2^{(||i||-1)} \leq |i| \leq 2^{||i||} - 1$, and applying assertion 1 again to these bounds gives $2^{2^{(||i||-1)}-1} \leq i \leq 2^{2^{||i||}-1} - 1$.
From $i \geq 2_1(2_1(21) - 1)$ we conclude $||i|| \geq 22$ hence $8l(|i|)^2 = 8||i||^4 \geq 8 \cdot 22^4$.
Hence $2^{(\frac{1}{8}8||i||^4)^{\frac{1}{4}}-1} > 8||i||^4$, therefore $2^{||i||-1} > 8l(|i|)^2$ and thus

$$|i| - 1 \geq 2^{||i||-1} - 1 \geq 8||i||^4.$$

Proof of assertion (d). Put $m := n + 3$ and $z(n) := 6n \ln_1(n)$. Note that z is increasing in n and that $p_n \leq z(n)$. This last property follows from $p_n \leq n(\ln_1(n) + \ln_1 \ln_1(n) - \frac{1}{2})$ for $n \geq 20$, which is proven in [8].
By repeated application we get $q_k \leq z^{(k)}(2)$,
and thus $q_{m+r(n)} \cdot 2^{g_1(n)} \leq z^{(m+r(n))}(2) \cdot 2^{g_1(n)} \leq z^{(3n+20)}(2) \cdot 2^{g_1(n)}$.
We claim

$$z^{(k)}(n) \leq 6^{2_1(k)-1} \cdot n^{2_1(k)}.$$

This follows by induction: $z^{(1)}(n) \leq 6n^2$ and

$$\begin{aligned} z^{(k+1)}(n) &= z(z^{(k)}(n)) \\ &\leq z(6^{2_1(k)-1} \cdot n^{2_1(k)}) \\ &\leq 6(6^{2_1(k)-1} \cdot n^{2_1(k)})^2 \\ &= 6^{2_1(k+1)-1} \cdot n^{2_1(k+1)}. \end{aligned}$$

So we arrive at

$$q_{m+r(n)} \cdot 2^{g_1(n)} \leq z^{(3n+20)}(2) \cdot 2^{g_1(n)} \leq 6^{2_1(3n+20)-1} \cdot 2^{2_1(3n+20)} \cdot 2^{g_1(n)} = g(n).$$

Assertion (e) is obvious. Assertion (f) is proved by main induction on n and subsidiary induction on $N(m)$. Assume that $m \succ q_n$. Then $m = p_s \cdot t$. If $p_s = q_n$ then necessarily $t \neq 1$ and the assertion follows. Assume that $q_s \succ q_n$. If $n = 1$ then $N(q_s) > 2$ and we are done. If $n > 1$ then $s \succ q_{n-1}$ and the induction hypothesis yields $N(s) > n$ and the assertion follows immediately.

Theorem 4. *Let h be the log-like function $h(i) = \mathrm{F}^{-1}_{\varepsilon_o}(i)$, with inverse $h^{-1} = \mathrm{F}_{\varepsilon_0}$. Then $\mathrm{D}(g(n), h) \geq \mathrm{F}_{\varepsilon_0}(n)$ for all n.*

Proof
Let $m := n + 3$. Recall $l(i) := |i|^2$ and

$$\mathrm{L}(r(n)), l) = \max\{M : (\exists m_0 \succ \ldots \succ m_{M-1})(\forall i)[N(m_i) \leq r(n) + |i| \cdot |i|_{l(i)}]\}.$$

Choose a sequence

$$l_0 \succ \ldots \succ l_{M_0}$$

with $N(l_i) \leq r(n) + |i| \cdot |i|_{l(i)}$ for all i, and such that M_0 is maximal (i.e. $M_0 = \mathrm{L}(r(n), l) - 1$).
Since $|i|^2 \geq 1$ we obtain $|i|_{|i|^2} \leq |i|$ hence $|i| \cdot |i|_{l(i)} = |i| \cdot |i|_{|i|^2} \leq |i| \cdot |i| = l(i)$. And thus $N(l_i) \leq r(n) + l(i)$ for all i.

From this sequence, we'll construct a sequence

$$m_0 \succ \ldots \succ m_{h^{-1}(n)}$$

with $m_i \leq g(n) + i^{|i|_{h(|i|)}}$ for all i. This will prove the assertion.

First we observe that $l_0 = q_{r(n)}$.
For suppose otherwise. Then either $l_0 \prec q_{r(n)}$ or $l_0 \succ q_{r(n)}$. In the first case we have $0 \succ q_{r(n)} \succ l_0 \succ \ldots \succ l_{M_0}$. Since $N(q_n) = n + 1$ holds for all n we obtain $N(q_{r(n)}) = r(n) + 1 = r(n) + |0| \cdot |0|_{l(0)}$ hence

$$N(l_i) \leq r(n) + |i| \cdot |i|_{l(i)} \leq r(n) + |i + 1| \cdot |i + 1|_{l(i+1)}.$$

Hence M_0 is not maximal. Contradiction.

In the case that $l_0 \succ q_{r(n)}$, we either have that $\exists \alpha > 1 \quad l_0 = q_{r(n)} \cdot \alpha$ and then

$$N(l_0) = N(q_{r(n)}) + N(\alpha) > N(q_{r(n)}) = r(n) + 1 = r(n) + |0| \cdot |0|_{l(0)}.$$

Contradiction.
Or we have that $\exists\beta > r(n) \& \exists\gamma \quad l_0 = q_\beta \cdot \gamma$ and then assertion (f) of Lemma 4 yields

$$N(l_0) > N(q_\beta) = \beta + 1 > r(n) + 1 = r(n) + |0| \cdot |0|_{l(0)}.$$

Contradiction.

For $0 \leq i \leq g_1(n)$ we put

$$m_i := q_{m+r(n)} \cdot 2^{g_1(n)-i}.$$

Then obviously $0 \succ m_0 \succ \ldots \succ m_{g_1(n)}$. Further $m_i \leq g(n)$ follows by assertion (d) of by Lemma 4.
For $g_1(n) < i \leq h^{-1}(n)$ we define

$$k(i) := 2^{(|i|-1)(|i|_{h(i)}-1)}$$

and

$$Q^m(\leq k(i)) := \{l \prec q_m : l \leq k(i)\}$$

Further let $enum_{Q^m(\leq k(i))}$ be the enumeration function of $Q^m(\leq k(i))$ with respect to $\prec$.
(So $enum_{Q^m(\leq k(i))}(2^{|i|}-i)$ is the $(2^{|i|}-i)$-th element of the set $\{l \prec q_m : l \leq k(i)\}$ ordered by $\prec$. Below we show that such an element indeed exists).
We put

$$m_i := q_m(l_{|i|}) \cdot enum_{Q^m(\leq k(i))}(2^{|i|} - i).$$

Observe that $l_{|i|}$ is welldefined since $|i| \leq i \leq h^{-1}(n) \leq \mathrm{L}(r(n), l)$ where the last inequality is proven in [1].
For all $i > g_1(n)$ we have

$$m_{g_1(n)} = q_{m+r(n)} = q_m(q_{r(n)}) = q_m(l_0) \succ q_m(l_{|i|})$$

since $l_0 \succ l_{|i|}$. And thus $m_i \prec m_{g_1(n)}$ for all $i > g_1(n)$.

If $|i| = |i+1|$ then $k(i) = k(i+1)$ and $q_m(l_{|i|}) = q_m(l_{|i+1|})$ and

$$enum_{Q^m(\leq k(i))}(2^{|i|} - i) \succ enum_{Q^m(\leq k(i+1))}(2^{|i+1|} - (i+1)).$$

Thus

$$m_i \succ m_{i+1}.$$

If $|i| < |i+1|$ then $l_{|i|} \succ l_{|i+1|}$ thence $q_m(l_{|i|}) \succ q_m(l_{|i+1|})$.
Also

$$enum_{Q^m(\leq k(i))}(2^{|i|} - i) \prec q_m(l_{|i|})$$

and

$$enum_{Q^m(\leq k(i+1))}(2^{|i+1|} - (i+1)) \prec q_m(l_{|i|}).$$

Therefore

$$m_i \succ m_{i+1}.$$

So we have shown $m_{g_1(n)} \succ m_{g_1(n)+1} \succ \ldots \succ m_{h^{-1}(n)}$.
Now we have to show that $m_i \leq g(n) + i^{|i|_{h(i)}}$ holds for those i.

In [3] it is proven by induction on $N(m)$ that $m \leq 2^{2 \cdot N(m)^2}$ for all $m \geq 1$. Using this we obtain

$$\begin{aligned} m_i &\leq q_m(l_{|i|}) \cdot k(i) \leq 2^{2(N(q_m(l_{|i|})))^2} \cdot k(i) \\ &= 2^{2(m+N(l_{|i|}))^2} \cdot k(i) \leq 2^{2(m+r(n)+l(|i|))^2} \cdot k(i). \end{aligned}$$

Using assertion (a) of Lemma 4 and assertions (b) and (c) of 4, we get

$$\begin{aligned} m_i &\leq 2^{2(2l(|i|))^2} \cdot k(i) \\ &= 2^{8l(|i|)^2+(|i|-1)(|i|_{h(i)}-1)} \\ &\leq 2^{(|i|-1)+(|i|-1)(|i|_{h(i)}-1)} \\ &= (2^{(|i|-1)})^{|i|_{h(i)}} \\ &\leq i^{|i|_{h(i)}} \leq i^{|i|_{h(i)}} + g(n) \end{aligned}$$

We still had to show that the $(2^{|i|}-i)$-th element of $Q^m(\leq k(i))$ exists. Adressing this we'll show that

$$\#Q^m(\leq k(i)) \geq 2^{|i|} - 1.$$

Note that $h^{-1}(n) \geq i$ yields $h(i) \leq n$. From $i > g_1(n) \geq 2^{T(m)}$ we conclude $k(i) = 2_1((|i|-1)(|i|_{h(i)}-1)) \geq 2_1((|i|-1)(|i|_n-1)) \geq 2_1(|i|-1) \geq T(m)$. So by assertion (a) of Lemma 1,

$$C_{Q_m}(k(i)) \geq \exp(2^{2-m}\frac{\ln(k(i))}{\ln_{m-1}(k(i))}).$$

The inequality $i > g_1(n) \geq 2_{n+2}(n+1)$ implies $2^{2-m}(|i|_n - 1) \geq \ln_{m-1}(2_1((|i|-1)^2))$, since the lefthandside grows faster in i than the righthandside, and for $i = 2_{n+2}(n+1)$ the lefthandside is greater than the righthandside.
Hence

$$2^{2-m}(|i|_n - 1) \geq \ln_{m-1}(2_1((|i|-1)^2)),$$

thus

$$\begin{aligned} 2^{2-m}(|i|_{h(i)} - 1) &\geq 2^{2-m}(|i|_n - 1) \\ &\geq \ln_{m-1}(2_1((|i|-1)(|i|_{h(i)}-1)) \\ &= \ln_{m-1}(k(i)) \end{aligned}$$

thence

$$\begin{aligned} 2^{2-m}\ln(k(i)) &= 2^{2-m}(|i|_{h(i)} - 1)(|i|-1)\ln(2) \\ &\geq (|i|-1)\ln(2)\ln_{m-1}(k(i)) \\ &= \ln(2^{|i|-1})\ln_{m-1}(k(i)) \end{aligned}$$

so $2^{2-m}\frac{\ln(k(i))}{\ln_{m-1}(k(i))} \geq \ln(2^{|i|-1})$ and

$$C_{Q_m}(k(i)) \geq \exp(2^{2-m}\frac{\ln(k(i))}{\ln_{m-1}(k(i))}) \geq 2^{|i|-1}.$$

Corollary 2. $\mathrm{PA} \nvdash (\forall K)(\exists M) M = \mathrm{D}(K, \mathrm{F}_{\varepsilon_0}^{-1})$

Proof. $\mathrm{F}_{\varepsilon_0}$ is not provably total in PA, hence $\mathrm{D}(g(n), \mathrm{F}_{\varepsilon_0}^{-1})$ is not provably total in PA, and the assertion follows.

References

1. Toshiyasu Arai: On the slowly well-orderedness of ε_0. Mathematical Logic Quarterly 48 (2002) 125-139.
2. Lev Beklemishev: Proof-theoretic analysis by iterated reflection. Archive for Mathematical Logic 42 (6), 515-552 (2003)
3. Arnoud den Boer: An independence result of PA using multiplicative number theory. Preprint, University of Utrecht 2006. Available via WWW at: `http://www.math.uu.nl/people/weierman/`
4. S.N. Burris: Number Theoretic Density and Logical Limit Laws. Mathematical Surveys and Monographs 86. American Mathematical Society 2001.
5. Harvey Friedman and Michael Sheard: Elementary descent recursion and proof theory. Annals of Pure and Applied Logic 71 (1995), 1-45.
6. J.R. Norris, Markov Chains, Cambridge University Press, Cambridge, 1997
7. J. Paris and L. Harrington: A mathematical incompleteness in Peano arithmetic, Handbook of Mathematical Logic (J. Barwise, ed.), North-Holland, Amsterdam, 1977, 1133-1142.
8. Rosser, J. Barkley, Schoenfeld, Lowell. Approximative formulas for some functions of prime numbers. Illinois J. Math. 6, 64-94, 1962.
9. Rick L. Smith: The consistency strength of some finite forms of the Higman and Kruskal theorems. In *Harvey Friedman's Research on the Foundations of Mathematics*, L. A. Harrington et al. (editors), (1985), pp. 119–136.
10. K. Schütte: Proof Theory. Springer, Berlin 1977.
11. Andreas Weiermann. An application of graphical enumeration to PA. To appear in the Journal of Symbolic Logic.
12. A. Weiermann, *Analytic Combinatorics, proof-theoretic ordinals and phase transitions for independence results*, Ann. Pure Appl. Logic, 136 (2005), 189–218.
13. A. Weiermann: Phase transition thresholds for some natural subclasses of the computable functions. This proceedings.
14. M. Yamashita: Asymptotic estimation of the number of trees, Trans. IECE Japan, 62-A (1979), 128-135 (in Japanese).

Some Reflections on the Principle of Image Collection

Albert Ziegler[1]

Ludwig-Maximilian's University, Munich 81673, Germany

Abstract. This article considers two alternative and formally weaker forms of Fullness, one of the axioms of constructive Zermelo–Fraenkel set theory. The relation to other axioms is analysed and some results that previously invoked Fullness are shown to be provable with the weaker forms too, sometimes even more easily.

1 Introduction

To study sets in and create a formal system for constructive mathematics, Aczel [2, 3] introduced the constructive Zermelo-Fraenkel set theory CZF.

In contrast to classical ZF, intuitionistic logic is used and some axioms are replaced by more constructive versions. Firstly, Separation is only allowed for bounded formulas[1] adhering to the goal of predicativeness. Bounded Seperation is equivalent to asserting the existence of binary intersections of arbitrary sets. Secondly, Replacement is strengthened to Strong Collection, an axiom constructing not only images of sets under functions (like Replacement) but under total relations. It states for every formula θ and each set a that if the formula defines a total class-relation on a, i.e. if $\forall x \in a \exists y \theta(x, y)$, then there is a set b whose elements stand in bitotal relation to those of a via that relation. This means that:

$$\forall x \in a \exists y \in b\ \theta(x, y) \wedge \forall y \in b \exists x \in a\ \theta(x, y).$$

Thirdly, Foundation is reduced to the (classically equivalent) principle of Set Induction which states the truth of a formula for all sets if the truth for each set follows from the truth for all of its elements. In short, this allows proving a statement for all sets by induction over the $\in$-relation. Finally, CZF omits the impredicative Powerset Axiom in favour of the axiom of Subset Collection. This complex looking axiom states that for each formula θ and for all A, B there is a C such that:
$\forall x \in A \exists y \in B \theta(x, y) \to \exists z \in C(\forall x \in a \exists y \in z \theta(x, y) \wedge \forall y \in z \exists x \in a\ \theta(x, y))$
In presence of the other axioms of CZF, including Strong Collection, this is equivalent to the following axiom of Fullness, which plays an important role in this article. For A and B sets, $\mathrm{mv}(A, B)$ denotes the class of all total relations from A to B, often thought of as $m$ulti$v$alued functions. The class $\mathrm{mv}(A, B)$

[1] Recall that a formula is called *bounded* in case it contains only restricted quantifiers $\forall a \in b$ or $\exists a \in b$

does not have to be a set, but Fullness demands that it has got a subset such that every total relation contains a total relation in the subset. Such a set will then be called *full* in $\mathrm{mv}(A, B)$.

The remaining axioms of ZF, Extensionality, Pairing and Union, remain unchanged. Thus the full list of Axioms is:
Extensionality, Pairing, Union, Seperation for bounded formulas, Strong Collection, Subset Collection, Set Induction.

Choice principles are not included in CZF. In particular, full AC should not be added, as it implies the law of excluded middle for bounded formulas [6]. However, the addition of weaker versions of AC is often considered, such as countable or dependant choice or often Aczel's Presentation Axiom.

The fragment of CZF without Subset Collection and Set Induction and where only Replacement instead of Strong Collection is included is called CZF_0. We would like to work in the theory CZF without Subset Collection, but in fact mostly use only CZF_0 and will indicate the one occasion when we use more.

2 The Principle of Image Collection

In [5], Crosilla, Ishihara and Schuster proposed the principle of Refinement, being a formally weaker principle than Fullness. It demands for every set A and B a set D collecting subsets of A, so that each $r \in \mathrm{mv}(A, B)$ contains an $f \in \mathrm{mv}(A, B)$ from A to B such that the fibre of every element of B is an element of D. So Refinement(A, B) means:

$$\exists D \forall r \in \mathrm{mv}(A, B) \exists f \subseteq r : f \in \mathrm{mv}(A, B) \wedge \forall b \in B f^{-1}(b) \in D$$

where $f^{-1}(b)$ stands for $\{a \in A | (a, b) \in f\}$, just as $f(a)$ in the following means $\{b \in B | (a, b) \in f\}$, the set of all values of f at the point a. The name of the axiom stems from the quasi-topological intuition of mv-functions from A to B as coverings of A indexed by B. The set D then collects subsets of A such that each covering has a refinement to a covering using only sets in D. I will now consider the dual principle that I would like to call Image Collection. It states that the sets of images under these f's for each element in A can be collected in a set E of subsets of B. So ImageColl(A, B) means:

$$\exists E \forall r \in \mathrm{mv}(A, B) \exists f \subseteq r : f \in \mathrm{mv}(A, B) \wedge \forall a \in A f(a) \in E$$

Like Refinement this is a consequence of Fullness:

Proposition 1. *Fullness(A,B)* $\rightarrow$ *Image Collection(A,B)*

Proof. Such a set E can be obtained from the full set C and the set A by defining

$$E := \{f(a) | f \in C \wedge a \in A\}$$

This is a set by Replacement, and every $r \in \mathrm{mv}(A,B)$ contains an f such that $f \in C$ so that all $f(a)$ for $a \in A$ are in E.

Analogous to the view of Refinement as stating that every covering of A indexed by B can be refined to a covering such that the sets covering A all belong to a collection D, it states the following: There is a collection E such that not the sets covering, but the sets containing for an $a \in A$ the indices of covering sets that cover a can be found in a collection E for a suitable refinement independent of a. Loosely speaking, E contains the possibilities for the pile of the covering sets above each $a \in A$. Although the dual statement about the fibres has found more applications as yet, this view is not completely artificial: for example in formal topology, it means that if X is a set, then the α_x-sets of refinements (in the sense of a refinement of a topology) of set-induced (in the sense that the relation $\|-$ is given by a set) ct-space-structures on X with fixed index set S can all be collected in a set E (for definitions, confer [1]). Here the mv-functions from X to S are the "$\|-$" relations.

Although AC is not accepted as constructive[2] and its addition to CZF destroys the constructive content of the theory, a lot about Image Collection can be understood considering the Axiom of Choice (AC). AC states that every relation contains a function, for which the image collection would be the trivial collection of all singleton sets of elements of B. Thus we get:

Proposition 2. *AC(A, B)* $\rightarrow$ *ImageColl(A, B)*

This severely handicaps the potential of Image Collection to prove results, as they are limited to those that already follow from AC. In this vein, one is lead to the following description of the impact of Image Collection(A, B) on the other principles of CZF except Subset Collection: The more AC(A, B) is violated, the more mv-relations exist which do not reduce to functions but only to more complex relations, the more complex subsets of B have to be contained in E, the less the Powerset axiom is violated. So Image Collection states, very laxly speaking, that the violation of the Powerset axiom and the Axiom of Choice together isn't too big.

The fact that it follows from AC is both a blessing and a curse for the applicability of Image Collection. It is a curse because there cannot be too many consequences, for each consequence has to follow from AC. For example, although it is something of a substitute for fullness, it does not guarantee any uncountable set and thus should perhaps at least be combined with other (weak) principles that see to this to get interesting results. Yet finding consequences of AC that only need Image Collection is of course an important accomplishment. And in some way it is also a blessing, for it can now be seen very quickly what consequences Image Collection doesn't have: For example it does not imply that the detachable subsets of a given set can be integrated in one set, because if it did, even AC would do this. Yet AC implies excluded middle for bounded formulas and thus also that every subset is detachable, so it would imply the existance of the powerset of the given set, which it clearly does not do (even in ZF without the Powerset Axiom). In particular Image Collection does not imply its dual,

[2] At least when viewed as an axiom about extensional sets. It is however even derivable in Martin-Loef type theory.

Refinement, which in turn does imply that the detachable subsets of any set form a set [5]. And of course it is seen that Image Collection is a real weakening of Fullness, i.e. it doesn't imply Fullness.

Yet Image Collection is not wholly without consequences; in particular, it is not trivial. The exponentiation axiom, whose single instances are denoted by Exp(A,B), claims that for every pair (A,B) of sets, the functions with domain A and range included in B form a set. Exponentiation alone does not prove Fullness, as Lubarsky has shown [7]. But under the principles of CZF without Subset Collection or in the theory CZF_0 we get:

Theorem 1. *If E is an image collection for A and B and Exp(A,E) holds, then Fullness(A,B) follows.*

In particular we have: ImageColl + Exp → Fullness

Proof. This is dual to the analogous statement about Refinement, and even the proof goes along similar lines (confer [5]): It is to show that the class C of mv-functions f with $f(a) \in E$ for all elements $a \in A$ is a set, so it suffices to show that the class C' of all relations f with $f(a) \in E$ for all elements $a \in A$ is a set. However, for each $f : A \to E$ let $r_f = \bigcup\{\{a\} \times f(a) | a \in A\}$. Let the set C'', contain the r_f for all $f : A \to E$. Now this C'' is equal to the class C' and a set by Replacement (which is entailed by Strong Collection). So C, which is full in $\mathrm{mv}(A, B)$, is a set.

It is interesting to note that although Fullness is not a consequence of Exponentiation, it doesn't seem to take much to go from Exponentiation to Fullness. For example Image Collection, Refinement and the Presentation Axiom all do the trick.

If A is countable, Image Collection(A, B) is a consequence of countable choice. Perhaps most applications of Image Collection, should these be found, will view the principle as a weakening of AC rather than as a reduction of Fullness (and thereby of the powerset axiom).

3 The Principle of Big Image Collection

An interesting variation seems to be the following:

Definition 1. *We define BigImageColl(A, B) to hold iff*

$$\exists F \forall r \in \mathrm{mv}(A, B) \exists f \subseteq r : f \in \mathrm{mv}(A, B) \wedge f[A] \in F$$

where $f[A] = \bigcup_{a \in A} f(a) = \{b \in B | \exists a \in A : (a, b) \in f\}$. The assertion of BigImageColl(A, B) for all sets A and B will be called Big Image Collection.

The content of this statement is not reduced to nil by the presence of AC, as here F collects the whole images of the relations, and even functions can have interesting images. The intuituion that Big Image Collection might be a consequence of Image Collection (just 'fuse together' the images of the different

arguments under the same relation) proves false, as the results below will show. Of course it is still a consequence of fullness, for if C is full in $\mathrm{mv}(A, B)$, then we can obtain a big image collection by defining $F = \{f[A] | f \in C\}$.

Big Image Collection is not only a direct a consequence of Fullness, it can also be seen as a special single instance of the Subset Collection scheme. Take $\theta(x, y, u)$ to be the simple formula $(x, y) \in u$ to get Big Image Collection.

Even if only one single instance, Big Image Collection is not without consequences:

Proposition 3. $\mathrm{BigImageCollection}(B, B \cup \{B\}) \to \{S \subseteq B | S \,\mathrm{detachable}\}$ *is a set.*

Proof. To see this, for each detachable S consider the function which is the identity on S and constantly B on the rest; its image contains S and perhaps $\{B\}$. So after purging {B} from every set of the Big Image Collection, the result is a set containing the set of detachable subsets of B. The part "$\{B\}$" could also have been replaced by any other set disjoint from B and that it really is disjoint requires Set Induction. Of course, if B is bijective to $B \cup \{B\}$, which is often the case, for example if $B = \omega$, the antecedent could also have been Big Image Collection(B, B), which looks nicer.

The consequence that the detachable subsets form a set is, by the way, equivalent to the instance of Exponentiation Exp(B,2) or to Exp(B, Y) for all discrete Y as was shown in [4]. So even simple instances of Big Image Collection imply the existance of quite a lot of uncountable sets.

Some results that previously invoked Fullness may be proved instead by only using Big Image Collection, sometimes even in an easier way. An example is the Main Lemma in [1], Appendix 2, which states that the strongly adequate subsets form a set. This result can be used to show, for example, that the formal points of a locally compact regular formal topology form a set.

Theorem 2. *Let S be a set with two binary set relations $\prec$ and $\approx$ on S, then:* $\mathrm{BigImageCollection}(W, S) \to \{B \subseteq S | B \,\mathrm{strongly\ adequate\ for}\, W\}$ *is a set*

Proof. Let W be the set giving the $\prec$ relation. Recall that a subset α of S is called strongly adequate iff

$$b, c \in \alpha \to b \approx c$$

$$a \in \alpha \to \exists b \in \alpha b \prec a$$

$$b \prec a \to \exists c \in \alpha (b \approx c \to c = a)$$

The proof given in [1] doesn't need to be altered a lot so that it only requires Big Image Collection and not the whole of Fullness, but it becomes easier in that it doesn't need the extra Lemma 54 from [1]:

It will be shown that the class of strongly adequate subsets, A, is a subset of E, the big image collection of W and S. So take any $\alpha \in A$. Then $R_\alpha =$

$\{((a,b),c) \in W \times S | c \in \alpha \wedge (b \approx c \rightarrow c = a)\} \in \mathrm{mv}(W,S)$. This is total for every adequate α, thus by virtue of Big Image Collection there is a $R \in \mathrm{mv}(W,S)$ with $R \subseteq R_\alpha$ for which $R[W] \in E$. But thanks to Lemma 56 in [1], which states that if R is a total subrelation of R_α, then $R[W] = \alpha$, we get $\alpha = R[W] \in E$, which finishes the proof.

However, the principle of Big Image Collection as a whole is even equivalent to that of Fullness:

Proposition 4. $\mathrm{BigImageColl}(A, A \times B) \rightarrow \mathrm{Fullness}(A,B)$

Proof. Every total relation r from A to B corresponds to a relation r' from A to $A \times B$ assigning to every $a \in A$ the pairs (a,b) such that $(a,b) \in r$. By the antecedent some total subrelation f of this has an image contained in the big image collection of A and $A \times B$ - but this image itself is a total subrelation of $r \subseteq A \times B$. Thus the big image collection of A and $A \times B$ is full in $\mathrm{mv}(A,B)$.

But although as a principle Big Image Collection is equivalent to fullness, its single instances, like the one invoked for the previous result, need not be equivalent to any instance of Fullness. So on the level of single instances Big Image Collection still seems to be a proper weakening of Fullness, and when proving results it is interesting to note when only the former is needed.

The principle of Big Image Collection should not be confused with the concept of fullness in [8]. The latter does not allow to go for a subrelation and so implies the Powerset axiom, as was shown in [3]. Indeed, most variations of Fullness that omit the transition to the subrelation are equivalent to the Powerset axiom, even the modified Image Collection' stating that for every mv-function the images of all elements can be collected in a set.

4 Conclusions

Fullness means that all total relations contain total relations that are gathered in a single set C. This can be weakened by collecting only pointwise preimages (fibres) or dually pointwise images. It can also be formally weakened by collecting complete images (right projections) or dually complete preimages (left projections). The corresponding axioms are Refinement, which has many consequences as shown in [5], dually Image Collection, which is also implied by AC and yields Fullness in the presence of Exponentiation, Big Image Collection, which is even equivalent to Fullness, and its dual that has not been graced by attention here, as it is of course trivial: The domain of any total relation from A to B is obviously the whole of A.

References

1. Aczel, P.: Aspects of general topology in constructive set theory. Ann. Pure Appli. Logic bf 137 (2006) 3–29

2. Aczel, P.: The type theoretic interpretation of constructive set theory. Logic Colloquium '77, North–Holland, Amsterdam (1978) 55–66
3. Aczel, P., Rathjen, M.: Notes on Constructive Set Theory. Institut Mittag–Leffler Preprint **40**(2000/01)
4. Aczel, P., Crosilla, L., Ishihara, H., Palmgren, E., Schuster, P.: Binary refinement implies discrete exponentiation. Typoscript
5. Crosilla, L., Ishihara, H., Schuster, P.: On constructing completions. J. Symbolic Logic **70, iss. 3** (2005) 969–978
6. Diaconescu, R.: Axiom of choice and complementation. Proc. Amer. Math. Soc **51** (1975) 176–178
7. Lubarsky, R.: Independence results around constructive ZF Ann. Pure Appl. Logic **132** (2005), 209–225
8. Troelstra, A.S., van Dalen, D.: Constructivism in Mathematics. North Holland, Amsterdam (1988)

Information Content and Computability in the n-C.E. Hierarchy

Bahareh Afshari

Department of Pure Mathematics, School of Mathematics
University of Leeds, Leeds LS2 9JT, UK

In 1944, Post [9] set out to relate computational structure to its underlying information content. Since then, many computability-theoretic classes have been captured, in the spirit of Post, via their relationships to the lattice of computably enumerable (c.e.) sets. In particular, we have Post's [9] characterisation of the non-computable c.e. Turing degrees as those of the simple, or hypersimple even, sets; Martin's Theorem [6] showing the high c.e. Turing degrees to be those containing maximal sets; and Shoenfield's [10] characterisation of the non-low_2 c.e. degrees as those of the atomless c.e. sets (that is, of co-infinite c.e. sets without maximal supersets).

In this talk, and in Afshari, Barmpalias and Cooper [1], we initiate the extension of Post's programme to computability-theoretic classes of the n-c.e. sets.

For basic terminology and notation, see Cooper [4], Soare [11], or Odifreddi [7].

References

1. B. Afshari, G. Barmpalias and S. B. Cooper, Characterising the highness of hyper-hyperimmune d.c.e. sets, in preparation.
2. G. Barmpalias, Hypersimplicity and Semicomputability in the Weak Truth Table Degrees, *Archive for Math. Logic* Vol.**44**, Number 8 (2005) 1045–1065.
3. G. Barmpalias and A. Lewis, The Hypersimple-free c.e. wtt degrees are dense in the c.e. wtt degrees, to appear in *Notre Dame Journal of Formal Logic*.
4. S. B. Cooper, *Computability Theory*, Chapman & Hall/ CRC Press, Boca Raton, FL, New York, London, 2004.
5. A. H. Lachlan, On the lattice of recursively enumerable sets, *Trans. Am. Math. Soc.* **130** (1968), 1–37.
6. D. A. Martin, Classes of recursively enumerable sets and degrees of unsolvability, *Z. Math. Logik Grundlag. Math.* **12** (1966), 295–310.
7. P. Odifreddi, *Classical recursion theory Vols. I,II* Amsterdam Oxford: North-Holland, 1989, 1999.
8. J. C. Owings, Recursion, metarecursion and inclusion, *Journal of Symbolic Logic* **32** (1967), 173–178.
9. E. L. Post, Recursively enumerable sets of positive integers and their decision problems, *Bull. Am. Math. Soc.* **50** (1944), 284–316.
10. J. R. Shoenfield, Degrees of classes of r.e. sets, *J. Symbolic Logic* **41** (1976), 695–696.
11. R. I. Soare, *Recursively enumerable sets and degrees*, Springer-Verlag, Berlin, London, 1987.

Computing with Newtonian Machines

Edwin Beggs and John V Tucker

University of Wales Swansea

The talk will consider to what extent machines operating under Newton's laws can compute more than a Turing machine. Various examples will be given, and their advantages and drawbacks discussed. The examples will be studied using a methodology to analyse and classify the physical sub-theories allowing the computations and hyper-computations. The work is part of our programme to discover just what pieces of classical mechanics are necessary to ensure that machines can only compute what a Turing machine can compute.

References

1. E J Beggs and J V Tucker, *Computations via experiments with kinematic systems*, `http://www-compsci.swan.ac.uk/~csjvt/JVTPublications/Bagatelle%28PreprintMarch04%29.pdf`, Technical Report 5-2004, Department of Computer Science, University of Wales Swansea, March 2004.
2. E J Beggs and J V Tucker, *Embedding infinitely parallel computation in Newtonian kinematic systems*, `http://www-compsci.swan.ac.uk/~csjvt/JVTPublications/infinite%20parallelism.pdf`, Applied Mathematics and Computation, accepted.
3. E J Beggs and J V Tucker, *Can Newtonian systems, bounded in space, time, mass and energy compute all functions?*, Theoretical Computer Science, accepted.

Infinite Time Turing Computation

Barnaby Dawson

Mathematics Department
Bristol University

Infinite time Turing computation is a generalisation of Turing computation to computers with infinite tapes and an infinite amount of time to work on those tapes. The implementation of such a computer will be described and the implications of the assumption that all sequences can be viewed as the output of such a computer shall be elucidated. In particular links with the axiom of choice and the coninuum hypothesis will be shown.

On a Problem of J. Paris

C. Dimitracopoulos[1⋆] and A. Sirokofskich[2]

[1] Department of History and Philosophy of Science, University of Athens, GR-157 71 Zografou, Greece, cdimitr@phs.uoa.gr
[2] Department of Mathematics, University of Athens, GR-157 84 Zografou, Greece, asirokof@math.uoa.gr

A question asked by J. Paris (see Problem 34 in the list in [1]) is whether or not Δ_n induction implies Σ_n collection. We sketch alternative proofs of results of T. Slaman ([3]) and N. Thapen ([4]) concerning this problem, especially for the case $n = 1$. Our proofs depend on results of C. Dimitracopoulos and J. Paris ([2]) concerning relationships between Σ_n collection and (versions of) Σ_n pigeonhole principle.

References

1. P. Clote and J. Krajíček: *Open problems*, Oxford Logic Guides, 23, Arithmetic, proof theory, and computational complexity (Prague, 1991), 1–19, Oxford Univ. Press, New York, 1993.
2. C. Dimitracopoulos and J. Paris: *The pigeonhole principle and fragments of arithmetic*, Z. Math. Logik Grundlag. Math. 32 (1986), 73–80.
3. T. A. Slaman: *Σ_n-bounding and Δ_n-induction*, Proc. Amer. Math. Soc. 132 (2004), 2449–2456.
4. N. Thapen: *A note on Δ_1 induction and Σ_1 collection*, Fund. Math. 186 (2005), no. 1, 79–84.

⋆ Research co-funded by the European Social Fund and National Resources - (EPEAEK II) PYTHAGORAS II

Risk Management in Grid Computing

Odej Kao[1] and Karim Djemame[2]

[1] PC^2 - University of Paderborn, 33102 Paderborn, Germany
[2] School of Computing, University of Leeds, Leeds LS2 9JT, UK

The Grid [1] offers scientists and engineering communities scalable, secure, high-performance mechanisms for discovering and negotiating access to computational, storage and network resources in a seamless virtual organisation. It promises to make it possible for scientific collaborations to share resources on an unprecedented scale, and for geographically distributed groups to work together in ways that were previously impossible.

Grid technologies have reached a high level of development, but Grid adopters underline core shortcomings related to security, trustworthiness, and dependability of the Grid for commercial applications and services. End-users require application execution with the desired priority and quality, and Service Level Agreements (SLAs) negotiation to define all aspects of the business relationship with Grid providers. Nevertheless, providers are still cautious on adoption as agreeing on SLAs including penalty fees is a business risk: for example a system failure can lead to SLA violation. Providers need risk assessment methods as decision support for accepting/rejecting SLAs, price/penalty negotiation, activating fault-tolerance actions, and for capacity and service planning. Grid end-users need the estimation and aggregated confidence information for provider selection and fault-tolerance/penalty negotiations.

The *Risk Assessment and Management for Grids* research project (AssessGrid) [2], recently funded by the European Commission, brings together various academic and industrial partners to address Grid risk awareness and consideration in SLA negotiation, self-organising fault-tolerant actions, and capacity planning. It will develop and integrate methods for risk management in all Grid layers. The corner stones are risk management scenarios reflecting the perspective of Grid end-users, resource brokers, and resource providers. The results will support all Grid actors by increasing the transparency, reliability, and trustworthiness as well as providing an objective foundation for planning and management of Grid activities. Thus, this research will supply Next Generation Grids with additional innovative and required components to close the gap between SLAs as concept and accepted tool for commercial Grid uptake.

References

1. Berman, F., Cox, G.C., Hey, A.J.G.: Grid Computing - Making the Global Infrastructure a Reality. Wiley, 2003
2. AssessGrid: Risk Assessment and Management for Grids. http://www.assessgrid.org/, 2006

Some Mathematical Properties of Propositional Input Resolution Refutations with Non-Tautological Resolvents

Annelies Gerber*

University Paris 6, Department of Computer Science, 75013 Paris, France
agerber03@yahoo.co.uk, Annelies.Gerber@univ-rouen.fr

Abstract. The question of whether or not a given set S of initial propositional clauses possesses a resolution proof is far from trivial. We aim to geometrically reproduce all necessary information encoded in propositional resolution proofs. Here, we give a mathematical characterisation of propositional input resolution refutations with non-tautological resolvents based on notions from Euclidean geometry. For a particular class of such proofs we develop a method to check whether or not a given set S of initial clauses admits such proofs. We comment on possible generalisations of this approach.

* This work is supported by the Holcim Foundation, Switzerland

NdE - Normalization During Extraction

Mircea-Dan Hernest[1]

Laboratoire d'Informatique (LIX)
École Polytechnique
F-91128 Palaiseau - FRANCE
danher@lix.polytechnique.fr

We present a methodology for improving the implementation of the **NbE** (Normalization by Evaluation) normalization algorithm [1, 2] in *call-by-value* functional programming languages like SCHEME [3]. Such optimizations are meant to heavily (at least from an empirical, observational viewpoint) decrease the run-time complexity of the NbE-normalization of long sequences of nested term applications $(\mathbf{t_n}..(\mathbf{t_2}(\mathbf{t_1}\mathbf{t_0}))..)$. A situation of this kind occurs for example in the case of the extraction of a *modulus of uniform continuity* for a closed term $\mathbf{t}$ of Goedel's $\mathbf{T}$ of type $(\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$. The aforementioned extraction is by means of Kohlenbach's Monotone functional "Dialectica" Interpretation [4] and proceeds from a proof of the *hereditarily extensional equality* of $\mathbf{t}$ to itself. This example was implemented in the MINLOG proof-system [5], hence a machine DEMOnstration will be available for the **NdE** optimization of the call-by-value NbE-normalization.

References

1. Berger, U., Eberl, M., Schwichtenberg, H.: Normalization by evaluation. In Möller, B., Tucker, J., eds.: Prospects for Hardware Foundations. Volume 1546 of LNCS. Springer Verlag (1998) 117–137
2. Berger, U., Eberl, M., Schwichtenberg, H.: Term rewriting for normalization by evaluation. Information and Computation **183**(1) (2003) 19–42 International Workshop on Implicit Computational Complexity (ICC'99).
3. Cadence Research Systems: Chez Scheme. http://www.scheme.com (2006)
4. Kohlenbach, U.: Analysing proofs in Analysis. In Hodges, W., Hyland, M., Steinhorn, C., Truss, J., eds.: Logic: from Foundations to Applications, Keele, 1993. European Logic Colloquium, Oxford University Press (1996) 225–260
5. Schwichtenberg, H., Others: Proof- and program-extraction system MinLog. Free code and documentation at http://www.minlog-system.de (2006)

A Similarity Criterion for Proofs*

Stefan Hetzl

Institut für Computersprachen (E-185),
TU-Vienna, Favoritenstraße 9,
1040 Vienna, Austria
hetzl@logic.at

Abstract. This talk is about the "characteristic clause sets" of sequent calculus proofs (for first-order classical logic) and about their expressiveness as a similarity criterion for proofs.
The characteristic clause sets have first been introduced in order to study cut-elimination: They are used as the main tool of the cut-elimination method Ceres (Baaz,Leitsch 2000).
In this talk we present recent results and work in progress 1) on the class of proofs having the same characteristic clause sets and 2) on proofs having strongly related (in the sense of subsumption) clause sets.
We show that the characteristic clause sets are invariant under a number of syntactic transformations of sequent calculus proofs that are generally perceived not to change the character of the mathematical argument underlying the proof (e.g. NNF-Transformation, rule permutations,...).
Furthermore we will give a comparison to other techniques used for analyzing the questions of equality and similarity of proofs.

* supported by the Austrian Research Fund (project no. P17995)

On Real Primitive Recursive Functions and Differential Algebraicity

Akitoshi Kawamura

Tokyo Institute of Technology

In 1996 Moore introduced a class of real-valued "recursive" functions by analogy with Kleene's formulation of the classical recursion theory. While his concise characterization of the class offers unique insight into continuous-time computation and has inspired numerous subsequent works, technically it seems to suffer some gaps. In this informal talk I focus on his "primitive recursive" functions and try to specify the problem. In particular, I discuss possible attempts to remove the ambiguity in the behavior of the primitive recursion operator on partial functions. Different modifications keep different parts of the original claims, but it turns out that in any case the purported relation to differential algebraicity, and hence to Shannon's GPAC model, needs fix.

Non-unitary Quantum Walks: Exploring the Space Between Classical and Quantum Computing.

Viv Kendon

School of Physics and Astronomy, University of Leeds, Leeds LS2 9JT, UK
V.Kendon@leeds.ac.uk

Abstract. Quantum versions of random walks have markedly different properties from classical random walks, such as faster mixing or spreading. These properties have been exploited to create several quantum algorithms. It has also been observed that making the quantum walk slightly less than perfectly quantum can actually improve the useful behaviour, such as even faster mixing to the uniform distribution. In this talk I will explain how to generalise a quantum walk into a non-unitary dynamics that can interpolate between classical and quantum behaviours by tuning a single parameter. In this way, the transition between quantum and classical behaviour can be located and characterised.

For a review of non-unitary behaviour in quantum walks, see [1]. As an example of typical behaviour, figure 1 shows the mixing time of the coined quantum walk on cycles of sizes 48 to 52, as a function of the applied decoherence rate. The mixing time decreases to a sharp minimum, then rises again to the classical value. The optimal behaviour from a computational point of view is thus intermediate between quantum and classical.

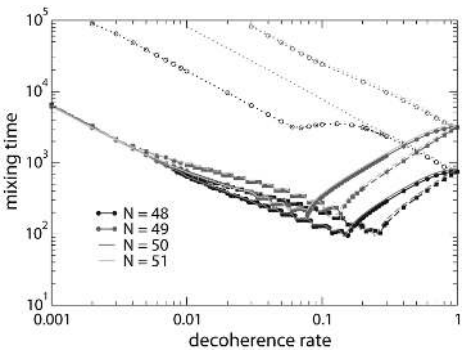


Fig. 1. Numerical data for the mixing time on cycles of size $N = 48$ to $N = 51$, for decoherence applied to coin only (dotted), position only (dashed) and both (solid).

1. Kendon, V.: Decoherence in quantum walks – a review. In preparation. (2006)

Logical Characterization of the Counting Hierarchy

Juha Kontinen

Department of Mathematics and Statistics
University of Helsinki, Finland
juha.kontinen@helsinki.fi

The counting hierarchy CH is the analogue of the polynomial hierarchy PH, the building block being probabilistic polynomial time PP instead of NP. We show that the extension of first-order logic by second-order majority quantifiers of all arities describes exactly the problems in the counting hierarchy.

This result is based on definability results which show that using the k-ary Majority quantifier and first-order logic, the relativized k-ary Majority quantifier (for $k > 1$) and the k-ary second-order existential quantifier are uniformly expressible. This characterization holds on all structures, i.e., without ordering or any numeric predicates. However, assuming ordering and predicates for $+$ and $\times$, we get exact correspondence between the levels C_kP of CH and sentences of quantifier-rank k, where the rank depends on the majority quantifiers only.

We also discuss the possibility of replacing the majority quantifiers by general proportional quantifiers saying "more than an r-fraction of relations", where $0 < r < 1$, for this result to remain valid.

Gödelian Foundations of Non-Computability and Heterogeneity In Economic Forecasting and Strategic Innovation

Sheri M. Markose

Economics Department and Centre for Computational Finance and Economic Agents (CCFEA), University of Essex, Wivenhoe Park Colchester C04 3SQ, UK.
E-mail:scher@essex.ac.uk

Abstract. The lack of effective procedures to determine winning strategies cleverly identified by Brian Arthur (1994) in an informal statement of this problem in a stock market game called the Minority or El Farol game results in the adoption of a multiplicity of meta- models for forecasting and strategizing by agents. The presence of contrarian payoff structures or hostile agents in a game theoretic framework are shown to result in the fundamental non-computable fixed point that corresponds to Gödel's undecidable proposition. Any best response function of the game which is constrained to be a total computable function then represents the productive function of the Emil Post (1944) set theoretic proof of the Gödel Incompleteness result. The productive function implements strategic innovation and objects of novelty or 'surprise' result in undecidable structure changing dynamics in the system. Oppositional or contrarian structures, self-reflexive calculations and the necessity to innovate to out-smart hostile agents are ubiquitous in economic systems. However, as first noted in Binmore (1987) and Spear (1987), extant game theory and economic theory cannot model the strategic and logical necessity of Gödelian indeterminism in economic systems. This paper reports on some formal results developed in Markose (2002, 2004, 2005) on the implications of the Gödelian incompleteness result for economics.

Arthur, W.B., (1994). "Inductive Behaviour and Bounded Rationality", *American Economic Review*, 84:406-411.
Binmore, K. (1987), "Modelling Rational Players: Part 1", *Journal of Economics and Philosophy*, 3:179-214.
Markose, S.M, 2005 , "Computability and Evolutionary Complexity : Markets as Complex Adaptive Systems (CAS)", *Economic Journal*, 115:F159-F192.
Markose, S.M, 2004, "Novelty in Complex Adaptive Systems (CAS): A Computational Theory of Actor Innovation", *Physica A: Statistical Mechanics and Its Applications*, 344:41- 49. *Details in Essex, Econ.Dept. DP No. 575.*
Post, E.(1944). "Recursively Enumerable Sets of Positive Integers and Their Decision Problems", *Bulletin of American Mathematical Society*, 50:284-316.
Spear, S.(1989), "Learning Rational Expectations Under Computability Constraints", *Econometrica* , 57:889-910.

Sequent Calculi and the Identity of Proofs

Richard McKinley

University of Bern

Abstract. Until recently, no satisfactory denotational semantics for classical logic existed, and hence no satisfactory answer to the question "when are two proofs equal?" Recently, several independent approaches to this question have arisen: each gives a class of models which captures the equalities on proofs required to perform cut-reduction in a particular formalism.
This presentation will show that such approaches are heavily dependent on the particular variant of the sequent calculus one chooses. Taking the additive and multiplicative presentations of the classical sequent calculus as examples, we show that what is an evident equality of proofs in one presentation is decidedly unclear in the other. We examine how these two separate notions of proof equality might be synthesized by examining Hughes' Hybrid logic (which has rules which are simultaneously additive and multiplicative) and Bruennler's SKS (a deep inference calculus for classical logic).

The Key to the Universe, Part 2

Robert K Meyer

Automated Reasoning Group
Computer Science Lab
Research School of Information Sciences & Engineering
Australian National University

The key to the universe is the magnificent accord between Curry-style combinators and matching relational postulates for the semantics of a wide range of logics. This key works most impressively in the filter models of lambda calculus in intersection type theory developed with Barendregt by Coppo, Dezani and the Torino school. I examine in the present talk the degree to which the further logical modelling of Boolean complementation can be added to this picture.

Questions Concerning the Usefulness of Small Universal Systems.

Liesbeth De Mol

Centre for Logic and Philosophy of Science
University of Gent, Blandijnberg 2, 9000 Gent, Belgium

In the fifties and sixties of the 20th century, there seemed to have been a small hype for finding the smallest possible Universal Turing machine. The last few years interest in this subject was triggered again, partly due to the highly criticized book by Stephen Wolfram, which, in its turn, gave rise to another kind of hype. Given this and related work the question concerning the significance of finding shortest universal systems should at least be taken into consideration again. It will be shown that it is far from trivial to investigate the "behaviour" of universal systems within a reasonable time and space, especially if one wants to consider other systems than CA. It will be argued that finding (short) universal systems might be interesting iff. a certain condition is fulfilled: when it is possible to generate non-trivial "behaviour" for practically feasible initial conditions. This discussion will be related to some preliminary research of the author on the possibility of universal binary tag systems.

Basic Model Theory for Bounded Theories

Morteza Moniri

Department of Mathematics, Shahid Beheshti University, Iran

Preservation theorems for some special classes of first-order formulas (e.g. existential formulas) are well-known in classical model theory. Notions like quantifier elimination or model completeness are also well studied in model theory. We define similar notions and prove similar theorems for bounded formulas of bounded theories (i.e. theories axiomatizable by bounded formulas). We naturally define when a bounded theory has bounded quantifier elimination, is bounded model complete, or has bounded model companion. We also prove some basic theorems on these notions. These provide natural extensions of questions like P=NP or NP=coNP in the context of bounded theories. We also study applications of these general results on some well-known theories of classical bounded arithmetic like PV and CPV, and some intuitionistic versions of them (via Kripke models). These results are based on a paper to appear in AML, and also a work in progress.

The Method Of Approximation In Real Computation

Kerry Ojakian

Instituto Superior Técnico (Lisbon)

Campagnolo, Costa, and Moore [2002] found connections between discrete complexity classes and analog classes defined on the reals. Building on these ideas Bournez and Hainry [2005] found function algebra descriptions of classes of functions defined via computable analysis. We develop a general collection of tools which allow us to nicely compare different classes of functions via a notion of "approximation" We apply these general tools to obtain some previous results and prove new ones, along the lines of the aforementioned authors.

References

[2002] M. Campagnolo, C. Moore, J. Costa: An analog characterization of the Grzegorczyk hierarchy. Journal of Complexity **18** (2002) 977–100

[2005] O. Bournez, E. Hainry, Elementarily Computable Functions Over the Real Numbers and **R**-Sub-Recursive Functions. Theoretical Computer Science **348** (2005) 130–147

Long Games with a Short Memory: The Church Synthesis Problem over Countable Ordinals

Alexander Rabinovich and Amit Shomrat

Sackler Faculty of Exact Sciences
Tel-Aviv Univ., Israel 69978.
{rabinoa, shomrata}@post.tau.ac.il

Two fundamental results of automata theory are the decidability of the Monadic second-order Logic of Order (MLO) over ω and the computability of the Church synthesis problem over ω. In their classical paper [BL], Büchi and Landweber reduce the Church synthesis problem to ω-length two-person games of perfect information, where the winning condition is definable by an MLO-formula (the so called "regular" games). Their main theorem states:

Theorem (Büchi-Landweber).

1. *In every such game, one of the players has a winning strategy.*
2. *The winning player has a winning strategy that is definable by a finite automaton with output.*
3. *There exists an algorithm that, given such a game (that is, a formula defining the winning condition), determines which of the two players has a winning strategy and constructs a finite-automaton defining such a strategy.*

Büchi generalized the concept of an automaton to allow automata to "work" on a countable ordinal and used this to show that the MLO-theory of any countable ordinal is decidable (see [BS]). Büchi and Landweber state that their result could also be generalized to all countable ordinals [BL]. However, we show:

Theorem. *The Büchi-Landweber theorem holds for a countable ordinal α iff $\alpha < \omega^\omega$. In particular, there is a regular game of length ω^ω in which none of the players has a finite-state winning strategy.*

Our proof uses both game theoretical techniques and the "composition method" developed by Feferman-Vaught, Shelah and others (see e.g. [Sh]).

We will also discuss the decidability of the Church synthesis problem over a given countable ordinal and an exact characterization of games (i.e., formulas) in which one of the players has a finite-state winning strategy.

References

[BL] J. R. Büchi, L. H. Landweber, Solving Sequential Conditions by Finite-State Strategies, Transactions of the AMS, Vol. 138 (Apr. 1969), pp. 295-311.

[BS] J. R. Büchi, D. Siefkes, The Monadic Second-order Theory of all Countable Ordinals, Springer Lecture Notes 328 (1973), pp. 1-126.

[Sh] S. Shelah, The Monadic Theory of Order, Annals of Mathematics, Ser. 2, Vol. 102 (1975), pp. 379-419.

Extensions of the Semi-Lattice of the 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

For every recursive ordinal α a reducibility on all sequences of sets of natural numbers of length α is defined. The induced equivalence classes are called α-enumeration degrees. The 1-enumeration degrees coincide with the enumeration degrees and for all $\alpha < \beta$ there exists a natural embedding of the α-enumeration degrees into the β-enumeration degrees which is proper.

In the talk we discuss several properties of the α-enumeration degrees. Analogues of the Selman's Theorem, The Minimal Pair Theorem, The Exact Pair Theorem and the existence of Quasi-minimal degrees are demonstrated.

A jump operation on α-enumeration degrees is defined and the respective Jump inversion theorem is proved. We show also that the Σ^0_2, ω-enumeration degrees are dense.

* Research partially supported by the Sofia University Science Fund

Genericity and Nonbounding

Mariya Ivanova Soskova*

University of Leeds

The structure of the semi lattice of the enumeration degrees has been investigated from many aspects. One aspect is the bounding and nonbounding properties of the enumeration degrees.

Definition 1. *Let a and b be two enumeration degrees. We say that a and b form a minimal pair in the semi-lattice of the enumeration degrees if:*

1. $a > 0$ *and* $b > 0$.
2. *For every enumeration degree* c $(c \leq a \wedge c \leq b \rightarrow c = 0)$.

In [2] Cooper, Sorbi, Lee and Yang proved that every Δ_2^0 degree bounds a minimal pair, but there exists a Σ_2^0 degree that does not bound a minimal pair.

Definition 2. *A set A is n-generic if for every Σ_n^0 set X of strings*

$$\exists \tau \subset A(\tau \in X \vee \forall \rho \supseteq \tau(\rho \notin X))$$

Copestake on the other hand proved in [1] that the degree of every 2-generic set bounds a minimal pair and conjectured that in contrast to the Turing case where every 1-generic degree bounds a minimal pair as proved in [5] there is a 1-generic set whose enumeration degree does not bound a minimal pair. Using the infinite injury priority method we construct a Σ_2^0 1-generic set, whose enumeration degree does not bound a minimal pair, generalizing the former result and proving Copestake's longstanding conjecture.

Theorem 1. *There exists a 1-generic enumeration degree a, that does not bound a minimal pair in the semi-lattice of the enumeration degrees.*

References

1. K. Copestake, *1-genericity in the enumeration degrees*, J. Symbolic Logic **53** (1988), 878–887.
2. S. B. Cooper, Andrea Sorbi, Angsheng Li and Yue Yang, *Bounding and nonbounding minimal pairs in the enumeration degrees*, to appear in the J. Symbolic Logic.
3. R. I. Soare, *Recursively enumerable sets and degrees*, Springer-Verlag, Heidelberg, 1987.
4. S. B. Cooper, *Computability Theory*, Chapman & Hall/CRC Mathematics, Boca Raton, FL, 2004.
5. P. G. Odifreddi, *Classical Recursion Theory, Volume II*, North-Holland/Elsevier, Amsterdam, Lausanne, New York, Oxford, Shannon, Singapore, Tokyo 1999.

* Research partially supported by the Marie Curie Early Training Site MATHLOGAPS (MEST-CT-2004-504029)

Author Index

Abdul Rauf, Rose Hafsah 2
Afshari, Bahareh 289
Andréka, Hajnal 12
Arslanov, Marat 15

Beggs, Edwin 290
Blanck, Jens 24
Bruni, Riccardo 37

Costa, Jose Felix 47
Cotogno, Paolo 58

Dawson, Barnaby 291
De Mol, Liesbeth 303
den Boer, Arnoud 268
Dimitracopoulos, Costas 292
Djemame, Karim 293

Elbl, Birgit 68

Fouche, Willem L. 78

Gaßner, Christine 85, 95
Gavryushkin, Alexander 105
Gerber, Annelies 294
Gerhardy, Philipp 109
Gordeev, Lev 119

Harman, Neal A. 129
Hernest, Mircea-Dan 295
Hetzl, Stefan 296
Hirata, Kouichi 140
Hirowatari, Eiju 140
Hodges, Andrew 1

Kalimullin, Iskander 150
Kao, Odej 293
Kawamura, Akitoshi 297
Kendon, Viv 298
Koepke, Peter 160
Komendantskaya, Ekaterina 170
Kontinen, Juha 299

Li, Chung-Chih 182
Longley, John 193
Lopez-Valdes, Maria 206

Markose, Sheri 300
McKinley, Richard 301
Meyer, Robert K. 302
Miyahara, Tetsuhiro 140
Moniri, Morteza 304
Mycka, Jerzy 47

Nagy, Benedek 216

Ojakian, Kerry 305

Potgieter, Petrus Hendrik 226
Puzarenko, Vadim 236

Rabinovich, Alexander 306

Seda, Anthony 170
Selivanov, Victor 241
Shomrat, Amit 306
Siders, Ryan 160
Sirokofskich, Alla 292
Skelley, Alan 251
Soskov, Ivan N. 307
Soskova, Mariya Ivanova 308
Stukachev, Alexey 261

Tucker, John V. 290

Vályi, Sándor 216

Weiermann, Andreas 268

Ziegler, Albert 282