Table of Contents

Preface

CiE 2005: New Computational Paradigms

`http://www.illc.uva.nl/CiE/`

The cooperation *Computability in Europe* (**CiE**) is an informal European network covering computability in theoretical computer science and mathematical logic, ranging from application of novel approaches to computation to set-theoretic analyses of infinitary computing models. The cooperation consists of fourteen main nodes and includes over 400 researchers; it is coordinated from Leeds (UK). More information about **CiE** can be found in Barry Cooper's introductory paper to the LNCS proceedings volume and at

`http://www.amsta.leeds.ac.uk/pure/staff/cooper/cie.html`.

CiE 2005 is a conference on the special topic "New Computational Paradigms" held in Amsterdam in June 2005. It was initiated by and will serve as a focus point for the informal cooperation **CiE**. The topic of "New Computational Paradigms" covers connections between computation and physical systems (*e.g.*, quantum computation, neural nets, molecular computation) but also higher mathematical models of computation (*e.g.*, infinitary computation or real computation).

We understand **CiE 2005** as an interdisciplinary venue for researchers from computer science and mathematics to exchange ideas, approaches and techniques in their respective work, thereby generating a wider community for work on new computational paradigms that allows uniform approaches to diverse areas, the transformation of theoretical ideas into applicable projects, and general cross-fertilization transcending disciplinary borders. The goal to reach out to both computer scientists and mathematicians resulted in many surprises to both communities: after many years and decades of experience in academia, we were astoundingly surprised about the differences in general practice between two research communities that seem to be so close in content. For mathematicians, the proceedings distributed at a conference are just an abstract collection – having an abstract printed there doesn't count as a publication; for computer scientists, acceptance of papers at conferences to be printed in the proceedings is a key indicator of research success.

These two very different approaches to the submission, acceptance and prepublication of abstracts in the two communities makes it very hard to organize a conference that appeals to both mathematicians and computer scientists. This phenomenon is known to many conference organizers in logic, but there are few known solutions. Some mathematics conferences have been inviting computer scientists as plenary speakers or special session speaker, but have been unsuccessful in attracting larger numbers of submissions from the CS community, other conferences have adopted the computer science organization and depend on the mathematicians playing according to computer science rules.

We believe that in an area like computability theory which genuinely belongs to both communities, these asymmetric solutions don't work. We need conferences that accommodate the style of work of both mathematicians and computer scientists and consequently, we have been catering for both of them: we have a formal proceedings volume in the LNCS series and an informal abstract booklet in the ILLC Publication series that covers extended abstracts, but also informal abstracts in the style of a mathematics conference.

We believe that we successfully managed to walk on the ridge between mathematics and computer science.

In the style of mathematics conferences, there will be a *postproceedings* volume that will be reviewed according to journal standards. In addition to this, there will be special issues of the journals *Theoretical Computer Science*, *Mathematical Structures in Computer Science* and *Theory of Computing Systems* covering special aspects of the conference. All speakers will be eligible for being invited to submit a paper to one of those postconference publications.

The Programme Committee for **CiE 2005** that ensured the scientific quality of our conference consisted of K. Ambos-Spies (Heidelberg), A. Atserias (Barcelona), J. van Benthem (Amsterdam), B. Cooper (Leeds, *Chair*), P. van Emde Boas (Amsterdam), S. Goncharov (Novosibirsk), B. Löwe (Amsterdam, *Chair*), D. Normann (Oslo), H. Schwichtenberg (München), A. Sorbi (Siena), I. Soskov (Sofia), L. Torenvliet (Amsterdam), J. Tucker (Swansea), and J. Wiedermann (Prague). The conference is sponsored by the *Koninklijke Nederlandse Akademie van Wetenschappen* (KNAW), *Nederlandse Organisatie voor Wetenschappelijk Onderzoek* (NWO), the *Institute for Logic, Language and Computation* (ILLC), the *European Association for Theoretical Computer Science* (EATCS) and the *Association for Symbolic Logic* (ASL).

Last, but definitely not least, we would like to take the opportunity to thank the numerous helpers for their support with the conference. Marjan Veldhuisen was responsible for catering, room reservations, the social event, registration and accommodation. Her support was absolutely necessary for our conference and it could not have taken place without her. Stefan Bold was the contact person for technical support and the book exhibit, Samson de Jager was our layouter not only for this booklet but also for the LNCS volume. Marco Vervoort created the content management system that we used for the webpage and the scientific evaluation procedure in the Programme Committee. We received a lot of support from the ILLC infrastructure (the director Frank Veltman and the interim manager Wil van Zijl-Barbe) and the building management of the *Euclides gebouw.* For practical matters during the conference, our student volunteers from the MSc in Logic and the PhD programme of the GPiL (Graduate Programme in Logic) were instrumental for the success of the conference: Edgar Andrade, Ioanna Dimitriou, Caroline Foster, Höskuldur Hlynsson, Vincent Kieftenbeld, Henrik Nordmark, Brian Semmes, Yanjing Wang, and Joost Winter. We would like to thank all of them with our deepest gratitude, and let the diverse list of nationalities in the list of organizers and helpers (China, Colombia, Germany, Great Britain, Greece, Iceland, Mexico, the Netherlands, Sweden, and United States of America) stand as a symbol for the European and global integration that we want to achieve with the **CiE** project.

For the most current information about the conference, we refer the reader to our webpage

`http://www.illc.uva.nl/CiE/`

Amsterdam & Leeds, June 2005 B. C. B. L. L. T.

Query learning of Horn formulas revisited[⋆]

Jose L. Balcázar

LARCA Research Group, Departament LSI
Universitat Politècnica de Catalunya
Barcelona, Spain
balqui@lsi.upc.edu
http://www.lsi.upc.edu/~balqui

Abstract. We discuss a number of (mostly open) questions related to the learnability of Horn clauses from queries. We consider that the currently existing solution is satisfactory only to a certain extent, and that progress on the related questions stated here could contribute positively to advancing the state of the art in several related fields. Our contributions here are a new analysis of the existing algorithm, a new algorithm based on novel ideas, and the complete proof of a related structural result.

1 Introduction

Horn formulas are a crucial knowledge representation mechanism. Even within the propositional realm, they come very close to an optimal balance between efficient algorithmics and flexibility as representation mechanisms, both in the sense of the quantity of naturally arising concepts that allow such a representation, and in the sense of the easiness of explaining which concepts are these. There are some extensions that still maintain efficient algorithmics, but their corresponding semantics are not so crisp; beyond these, one very easily falls into a powerful propositional representation system whose major algorithmic questions are at least NP-hard. Conversely, simpler mechanisms are much too limited for many applications. The essential intuitions of Horn propositional logic are the driving force for a number of successful restrictions of first-order logic.

We consider here the problem solved in [2]: the learnability of Horn formulas via queries. It is proved there that these formulas (conjunctions of Horn clauses) are learnable from membership and equivalence queries in polynomial time. We propose here to study the following extensions or refinements of this known fact, which we subsequently discuss:

Question 1. Is it possible to learn Horn formulas via queries in less demanding query learning models?

[⋆] This work started with the visit of the author to University of Illinois at Urbana-Champaign, with a grant of the NATO Scientific Program, and has been supported in part by the Spanish Government project TIC2004–07925–C03–02 (TRANGRAM), the EU research network PASCAL, and matching funds from CIRIT 2001SGR-00252.

Question 2. Is it possible to learn multivalued dependency formulas via queries? Same question for such less demanding models.

Question 3. Is the HORN1 algorithm from [2] optimal with respect to the total number of queries or is it possible to improve on it? If there is an improved algorithm, with respect to the total number of queries, is there one such algorithm that still works in polynomial time?

Question 4. Does there exist a different algorithm for learning Horn formulas in the standard model and in polynomial time?

Question 5. Is there an alternative way of proving the correctness of the algorithms in [2]?

It is easy to see that a natural order to tackle these questions is in reverse order; on the other hand, we found more natural to state them in the given order. In fact, here we provide solutions to questions 5 and 4, except for the fact that we work here only with definite Horn clauses to simplify the analysis; and contribute hints of progress towards the solution of question 3 and, to some extent, 2 (it must be pointed out that the main result regarding question 2 was announced already in [6]; the complete proof is given here for the first time).

2 Preliminaries and Notation

We are in a propositional logic setting, with n boolean variables out of which we construct models, that is, binary words of length n. Models are denoted as x_i and similar syntactical objets. Thus the set of all models is the n-dimensional boolean cube, endowed with its usual bitwise partial order, denoted $x \leq y$, or $x < y$ for the proper order. Literals, terms, and propositional formulas are defined in the usual way, and are satisfied by a model x if they evaluate to true on it; we use the standard notation $x \models F$. Sometimes we say that a model violates a formula to mean that it evaluates to false on the model. We denote $\top$ the model consisting of all trues. Operations $\wedge$ and $\vee$ on models apply bitwise, and can be extended by associativity and commutativity to theories, that is, sets of models; we also adhere to the standard convention that the union of an empty theory is the all-false model $\bot$, whereas the intersection of an empty theory is the top model $\top$. We frequently overload the numeral notation by denoting true as 1 and false as 0, as well as denoting $1(x)$ the set of variables with value true in x, and $0(x)$ those with value false. When convenient, these sets of variables will be seen as positive terms; more generally we use greek letters to denote positive terms. The Hamming weight of a model is the number of variables it assigns to true; the Hamming weight of a set of models is the sum of the Hamming weights of its elements.

Horn clauses are disjunctions containing exactly one positive literal (definite Horn clauses) or none (nondefinite Horn clauses). Horn formulas are conjunctions of Horn clauses. We propose to approach our study of Horn learnability here avoiding as much as possible these syntactic peculiarities, and to resort instead to a semantic approach based on the following structural property, which is known since the earliest works on Horn logic and, *mutatis mutandis*, holds also beyond propositional domains:

Theorem 1. *1. A propositional theory is a Horn theory, that is, can be axiomatized by a conjunction of Horn clauses, if and only if it is closed under intersection.*
2. The smallest Horn theory that contains a given theory is its closure under intersection.

For this propositional case, the proof is not difficult and can be found in a number of references (for instance in [16]). For the sake of comparison with our later contributions, we state the following easy but crucial step in the proof:

Lemma 1. *Consider a Horn clause, and two models x and y that satisfy it. Then $x \wedge y$ also satisfies it.*

From this lemma it immediatelly follows that each Horn clause satisfied by a theory T is also satisfied by the closure of T under intersection.

In our learning model, a learning algorithm poses queries to a teacher, and gets answers corresponding to a particular target; the target itself is just a set of models. Then, learning Horn formulas means that there is a learning algorithm that identifies a Horn axiomatization of the target, provided one exists. The interaction takes place through the most common query learning model (sometimes called minimally adequate teachers) where two sorts of queries can be posed: membership queries where the learner asks about whether a model of its choice belongs to the theory, and equivalence queries where the learner guesses a Horn axiomatization of the target and gets a counterexample, or is told that the guess is correct and the process finishes. Note that the inductive bias (Horn formulas) coincides here with the language to express equivalence queries: this setting is sometimes called proper learning. We will denote H^* the target, and we will assume it to be axiomatized by a conjunction of definite Horn clauses. We will point out explicitly the point where this definiteness assumption is used.

3 On Question 5: An Alternative Validation of a Known Algorithm

This section provides one answer to question 5. Indeed, we describe a new invariant that explains in a different way the inner workings of the algorithms of Angluin, Frazier, and Pitt [2], and we also take the opportunity to review the algorithm itself.

Our version is a bit of a hybrid between both algorithms HORN and HORN1 in [2]; it is the version for which our argumentations seem to be most clear, and all the essential ideas and intuitions of [2] are the same.

3.1 The Algorithm

Like in usual learning algorithms in the presence of equivalence queries, we assume that the algorithm only stops upon receiving a positive answer to an equivalence query, so that termination implies correctness. To prove termination, we bound the number of queries in terms of the number of variables and the number of clauses in an arbitrary Horn axiomatization of the target H^*; the computing time between queries is easily seen to be polynomial.

At each point of the algorithm, it maintains a set P of all the positive examples seen so far. The algorithm maintains also a sequence $x_1, \ldots, x_t$ of negative examples; the indices i, j, k run from 1 to t unless otherwise specified. For the sake of the argument, we also assume that it maintains some other additional models: for each $i < j$, we have models $x_{i,j}$, $y_{i,j}$, and $z_{i,j}$, whose properties will be described below; these are not part of the original algorithm. We will abbreviate the fact that a model z is a positive example, that is, belongs to the target, or $z \models H^*$, by the simple expression $z\oplus$; likewise, the fact that y is a negative example will be abbreviated $y\ominus$.

The query is formed with all clauses that can be constructed from the x_i and do not contradict previous positive examples: $H = \bigwedge\{1(x_i) \to v\}$ where v runs over all variables in $0(x_i)$ such that $\forall z \in P(x_i \leq z \to z \models v)$. Observe that, if this condition is not fulfilled for some $z \in P$, then $z \not\models 1(x_i) \to v$, which cannot be a logical consequence of H^* since z is a positive example.

A positive counterexample is treated just by adding it to P; this is tantamount to erasing from H the clauses contradicting it: being a positive counterexample, it surely violated some clause in H, which no longer will be there. (In the terminology of [2], this step is called *reducing* the hypothesis.)

A negative counterexample y satisfies the query H. We use y to either refine some x_i into a smaller counterexample, or to add x_{t+1} to the list. Specifically, let

$$i := \min(\{j \mid (x_j \wedge y) \ominus \text{ and } x_j \wedge y < x_j\} \cup \{t+1\})$$

and then refine x_i into $x'_i = x_i \wedge y$, in case $i \leq t$, of else make $x_{t+1} = y$, subsequently increasing t. The value of i is found through membership queries on all the $x_j \wedge y$. The handling of $x_{i,j}$, $y_{i,j}$, and $z_{i,j}$ will be explained as need arises; they are needed only by our correctness proof.

3.2 The Proof

We will prove that each x_i always violates different clauses in any arbitrary Horn axiomatization of H^*; this implies that $t \leq m$, the number of clauses in the axiomatization chosen.

In the argumentation, we will act as if the algorithm indeed maintains $x_{i,j}$, $y_{i,j}$, and $z_{i,j}$, for $i < j$, which have the following intuitive meaning. Clearly each x_j is the intersection of a number of negative examples. For $i < j$, we make sure that each example that refined x_j could not refine instead x_i, as the algorithm does; but we will distinguish, among those examples that lead to x_j, which ones skipped x_i because $(x_i \wedge y)\oplus$ from those that skipped x_i because $x_i \wedge y = x_i$, that is, $x_i \leq y$; it suffices to keep the intersection of the two distinct sorts. That is, $x_j = x_{i,j} \wedge y_{i,j}$, for each $i < j$. The reason is that, from the perspective of x_i, it is true that all y that refined x_j did correctly so, but for two different causes.

As for the z's, they will eventually allow us to prove that indeed each two different x's are chasing different clauses of the target. The reader familiar with [1] will detect a striking similarity with our approach (see also [8] for further discussion).

It becomes handy to introduce a bit more of notation with respect to the consequences of an antecedent with respect to the target and the hypothesis, and to use it to state separately the main property of a negative counterexample. The notion of $H(x)$ is a key concept in [14].

Definition 1. *By $H^*(x)$ we denote the set (sometimes, the conjunction, or even the model having that set of true variables) of all the variables that are consequences of $1(x)$ under the theory H^*; that is, $H^*(x) = \{v \mid H^* \models 1(x) \to v\}$. For the query H, the set $H(x)$ is defined similarly.*

Lemma 2. *Let x_i be one of the negative examples used to construct the query H, and let $y\ominus$ be a counterexample above x_i, that is, $x_i \leq y$; then, $x_i \leq H^*(x_i) \leq H(x_i) \leq y$.*

Proof. Seen as sets, $H^*(x_i) \subseteq H(x_i)$, since a variable that is indeed logical consequence of x_i cannot be disproved by any positive example having it at zero and still being above x_i. Of course $1(x_i) \subseteq H^*(x_i)$. Finally, since y is a negative counterexample, it must satisfy H, which includes a clause $1(x_i) \to v$ for each of the variables in $H(x_i)$. To satisfy them all, being $x_i \leq y$, requires $H(x_i) \subseteq 1(y)$.■

3.3 Invariants

We will prove that the algorithm maintains the following invariants, besides the facts that the x_i's are negative and the examples in P are positive: for all $i < j$,

$(I0)$ $x_i \wedge x_{i,j} \leq z_{i,j} \leq x_{i,j}$
$(I1)$ $z_{i,j}\oplus$
$(I2)$ $y_{i,j} \models H^*(x_i)$
$(I3)$ $x_j = x_{i,j} \wedge y_{i,j}$

Coming back to the intuitive explanation we just saw before, for $i < j$, x_j is made up of the intersection of counterexamples that skipped x_i (whatever its value was at that time), and these may have skipped x_i for two reasons: either they were above x_i, and then still are since x_i only can have changed by refinement, or their intersection with x_i was positive. Those that skipped x_i for the first reason are intersected together into $y_{i,j}$; whereas the rest are intersected together into $x_{i,j}$, and of course x_j is the intersection of them all. Moreover, those that got added to $y_{i,j}$ were positive for the current hypothesis, so that not only they were above x_i but also above all the consequences $H^*(x_i)$, and this remains true if x_i is refined. On the other hand, those that gave a positive intersection with x_i will always leave a positive example in the area indicated in $(I0)$. We make precise and argue formally now all these indications.

Lemma 3. *Consider the four invariants for fixed $i < j$, and suppose they hold before refining x_i into x'_i; then they still hold after the refinement.*

Proof. Refining x_i in $(I0)$ and changing nothing else reduces or leaves untouched the leftmost term, and does not affect the other two, so that $(I0)$ is maintained. Since $x'_i \leq x_i$, we have $H^*(x'_i) \subseteq H^*(x_i)$, so that refining x_i and leaving $y_{i,j}$ untouched maintains $(I2)$; x_i does not even appear in the other two invariants. Note that, formally, we have not used at all the inequality $i < j$. ∎

Theorem 2. *The invariants initially hold, and processing each counterexample maintains them.*

Proof. Initially $t = 0$; in fact the invariants are trivially true for $t \leq 1$. To study the processing of a counterexample, pick two values $i < j$ such that one of the corresponding models is refined. In case x_i is refined, the previous lemma gives us the invariance. Thus, we analyze now what happens when we refine x_j instead, with respect to any i but again under $i < j$. We will need to adjust the remaining variables so that the invariants are recovered, as follows. Let $x'_j = x_j \wedge y$ be the result of the refinement, and assume first that $(x_i \wedge y)\oplus$; then we recover $(I3)$ by setting $x'_{i,j} = x_{i,j} \wedge y$, and need a positive example that falls in the appropriate range to complete $(I0)$:

$$x_i \wedge x'_j = x_i \wedge x_j \wedge y =$$
$$= (x_i \wedge y) \wedge (x_i \wedge x_j) \leq (x_i \wedge y) \wedge z_{i,j} \leq x_j \wedge y = x'_j$$

Having just assumed $(x_i \wedge y)\oplus$, and knowing that $z_{i,j}\oplus$, the closure of H^* under bitwise intersection provides $z'_{i,j} = ((x_i \wedge y) \wedge z_{i,j})\oplus$, thus reestablishing $(I0)$ and $(I1)$. Of course $(I2)$ does not change.

Assume now that $x_i \leq y$. Then we choose to recover $(I3)$ by setting a new $y'_{i,j} = y_{i,j} \wedge y$, and leave everything else untouched, so that we have only to argue that $(I2)$ is again true. Knowing $y_{i,j} \models H^*(x_i)$, it remains to prove that $y \models H^*(x_i)$. This will follow now from the fact that y is a negative counterexample, so that it satisfies the query H; consider any variable $v \in 0(x_i)$ such that $H^* \models 1(x_i) \to v$: then, for positive examples such as those in P, $x_i \leq z \to z \models v$, so that $H \models 1(x_i) \to v$ as well. For y that satisfies the query H with $x_i \leq y$, we obtain $y \models v$. Of course, for the variables in $1(x_i)$ the fact that $x_i \leq y$ suffices.

To complete the invariance analysis, consider now what happens when no x_i was refined and y was added as x_{t+1}. We have the choose fresh values for $x_{i,t+1}$, $y_{i,t+1}$, and $z_{i,t+1}$, and for each i the way of doing it will depend on the reason why x_i was not refined.

If $x_i \leq y = x_{t+1}$, we pick $x_{i,t+1} = z_{i,t+1} = 1^n$; the assumption that H^* consists of definite Horn clauses is used here to make sure that $1^n\oplus$. This establishes (I0) and (I1) for this pair. To establish $(I3)$ we are forced to pick $y_{i,t+1} = x_{t+1} = y$, and indeed this is convenient since y was a negative counterexample, $x_i \leq y$, and then, by lemma 2, we have $y \models H^*(x)$. ∎

3.4 Termination

It remains to prove that the invariant guarantees that t is no more than the number of clauses in an arbitrary Horn axiomatization of H^* (for instance, a smallest one). Since each x_i is negative, each falsifies one of these clauses. We prove that no two of them can falsify the same clause. (Actually we prove something stronger as in [2]; this observation leads to the new algorithm in the next section.)

Theorem 3. *If two different negative models x_i and x_j both violate a clause $\alpha \to v$, then $H^* \not\models \alpha \to v$.*

Proof. Consider a clause with $H^* \models \alpha \to v$, such that $x_i \not\models \alpha \to v$, where α is a monotone term, and likewise $x_j \not\models \alpha \to v$ for $i \neq j$. Then $x_i \models \alpha$ and $x_i \not\models v$, and likewise for x_j: $x_j \models \alpha$ and $x_j \not\models v$. From $(I3)$ we know $x_j = x_{i,j} \wedge y_{i,j}$. However, $x_i \models \alpha$ and $H^* \models \alpha \to v$ imply that $v \in H^*(x_i)$, so that by $(I2)$ we obtain that $y_{i,j} \models v$, so that the only way to have $x_j \not\models v$ is that $x_{i,j} \not\models v$ either. By the second inequality in $(I0)$, $z_{i,j} \not\models v$ as well.

On the other hand, $x_j \models \alpha$ with $x_j \leq x_{i,j}$, and with $x_i \models \alpha$, imply that $x_i \wedge x_{i,j} \models \alpha$; by the first equality in $(I0)$, $z_{i,j} \models \alpha$ as well; therefore $z_{i,j} \not\models \alpha \to v$, contradicting the fact that it is a positive example of H^* with $H^* \models \alpha \to v$. ∎

Therefore, if H^* has an axiomatization with m clauses, then the number of models x_i will not exceed m; the rest of the argument to evaluate the number of queries is as in the original reference: each negative counterexample must either create a new x_i, which may happen at most m times, or refine an existing one, which may happen at most n times per model, for a total bound of no more than $m(n + 1)$ negative counterexamples; each positive counterexample being added to P reduces the possibilities for the right hand side of one clause of the hypothesis, and that literal will never be recovered back as a right hand side for the same x_i due to the presence in P of the same positive example, so that at most mn positive counterexamples can be seen. Jointly, this allows for no more than $O(mn)$ equivalence queries. Membership queries are made only to process negative counterexamples, and at most m such queries are made for each negative counterexample, for a total of at most $m^2(n + 1)$ membership queries. The whole algorithm is easily seen to be programmable in polynomial time.

4 On Question 4: An Alternative Algorithm

We use the intuition provided by the previous section to propose a new algorithm that refines several refinable examples with every negative counterexample. Unfortunately, we are unable to evaluate the supposedly faster parallel progress made in this way, and hence we cannot prove a better running time than in [2]. However, the ideas might be useful to progress along this line (see question 3).

The idea of this variant is as follows. The termination argument only requires from us to argue that each x_i pursues a different clause. If we consider separately a given pair i, j, it is actually irrelevant whether $i < j$ or $i > j$ for the termination argument. Therefore, for each i and j, we define the parameterized statement

$$(I_{i,j}) \equiv \exists\, x_{i,j}, y_{i,j}, z_{i,j}$$
$$(x_i \wedge x_{i,j} \leq z_{i,j} \leq x_{i,j} \wedge z_{i,j} \oplus \wedge y_{i,j} \models H^*(x_i) \wedge x_j = x_{i,j} \wedge y_{i,j})$$

and we aim at maintaining the following invariant:

$$(I^*) \qquad \forall\, i, j\, (i \neq j) \Rightarrow (I_{i,j} \vee I_{j,i})$$

which is sufficient to guarantee the bound m on the number of examples x_i and, thus, the termination argument. In order to choose which (refinable) models to refine, we classify (pairs of) those refinable ones into three sorts of pairs, according to whether $I_{i,j}$, $I_{j,i}$, or both hold.

- if $(I_{i,j} \wedge I_{j,i})$: then we can refine either, and $(I_{i,j} \vee I_{j,i})$ will be maintained;
- if $(I_{i,j} \vee I_{j,i})$, but not both: then we must choose which one to refine, in such a way that we do not lose the invariant.

The first of these statements follows from lemma 3 since refining x_i will maintain $I_{i,j}$ as seen there. The difficulty now is to drive the refinements described in the second statement.

Now, to guide our choices, we maintain a directed graph G without self-loops, which we will prove acyclic later on, and whose vertices are the negative examples x_i; and we set an edge from x_i to x_j whenever we have lost the guarantee that $I_{j,i}$ holds. Therefore, in such case, we are only able to maintain $I_{i,j}$, so that we cannot refine both x_i and x_j: if both can be refined, we will refine only x_i, which maintains $I_{i,j}$, just resorting again to 3.

Thus, given a negative counterexample y for the hypothesis H, we consider the restriction G' of G to the set R of refinable models: $R = \{i|(x_i \wedge y)\ominus < x_i\}$; assuming it for now to be nonempty, we will choose a subset $L \subseteq R$ of vertices which we will refine. This set L consists some of the vertices that we call *covered*, plus one additional vertex i_0. A vertex labeled with model x_i is covered if there is another vertex labeled with model x_k if $(x_i \wedge y) \leq x_k \leq y$. Here is why we consider these covered vertices.

Lemma 4. *Assume the following: x_i and x_j are both refinable, $I_{i,j}$ holds, and x_i is covered, say by x_k. Then, we can refine both x_i and x_j, and still maintain $I_{i,j}$.*

Proof. Simply take $y'_{i,j} = y_{i,j} \wedge y$ and leave $x_{i,j}$ unchanged, so that $x_{i,j} \wedge y'_{i,j} = x_{i,j} \wedge y_{i,j} \wedge y = x_i \wedge y$, the result of the refinement that we will denote $x'_i = x_i \wedge y \leq x_k \leq y$, the last two inequalities coming from the covering. Here, lemma 2 gives $y \models H^*(x_k)$ and thus $y \models H^*(x'_i)$ because the latter is smaller. On the other hand, from the invariant we know that $y_{i,j} \models H^*(x_i)$ and thus $y_{i,j} \models H^*(x'_i)$ again because the latter is smaller. Thus the conjunction $y_{i,j} \wedge y = y'_{i,j} \models H^*(x'_i)$. With respect to the invariants relating $z_{i,j}$, which is unchanged, they are maintained upon reducing x_i as in lemma 3. ∎

Now, this argument would help us to choose refinable covered vertices but this does not guarantee that there is any. Instead of arguing that indeed there are some, we prove that we can refine one more additional example, that will be found indeed by a process analogous to the one in the HORN algorithms of [2]. Within G', the restriction of G to the refinable vertices R, we employ any appropriate algorithm to find $L \subseteq R$ such that the following conditions hold.

First, as indicated, L consists of some (not necessarily all) covered vertices C plus one more vertex i_0: $L = C \cup \{i_0\}$; second, there is no path in G leading from a vertex in L into a vertex in $R - C = (R - L) \cup \{i_0\}$ (we do not consider paths of length zero here). In particular, there is no edge from x_k, $k \in L$, to x_ℓ, $\ell \in R - L$, that is, for these pairs $I_{k,\ell}$ holds, including $I_{i_0,\ell}$. Likewise, there is no edge from C into i_0, that is, I_{k,i_0} holds for $k \in C$.

Such a nonempty L always exists, as soon as we argue that G is acyclic: it would suffice to take $C = \emptyset$ and pick for i_0 a sink of G'. However, possibly we can take larger sets L in specific cases. Now: all the edges from $x_\ell \in R - C$ to $x_k \in L$ are added to G' (except the self-loop on i_0), which maintains acyclicity since we have ensured that there is no path to close a circuit with such an edge; and, subsequently, the models at the vertices of L are refined.

It remains to argue that $I_{i,j}$ is maintained for all pairs except those corresponding to edges added to G. Only the cases where either of them is in L require attention, since the other models are not refined. For all the pairs where one (say k) or both of the two vertices is in C, thus covered, we just apply lemma 4, knowing that $I_{k,\ell}$ holds for $k \in L$ and $\ell \in R - L$, and I_{k,i_0} as well. The invariants in the opposite direction may be lost (or might have been lost before), thus we add to G the corresponding edges. Finally, the only remaining case is $I_{i_0,\ell}$ for $\ell \in R - L$, that is, where x_{i_0} is refined and x_ℓ is not, and we have already argued that $I_{i_0,\ell}$ holds before refining, so it is maintained by lemma 3.

Taken together, we have argued that:

Lemma 5. *Assume that counterexample y leads to a nonempty set R of refinable models among x_i. If G is maintained as indicated and L is chosen as indicated, then G is always acyclic and the invariant (I^*) is maintained.*

We still must explain how the sequence of negative examples grows, which of course happens when $R = \emptyset$. Then the following holds: for all i, either $(x_i \wedge y)\oplus$ (it can be added to P) or $x_i \wedge y$ is not smaller than x_i, that is, $x_i \leq y$. We add y as new vertex x_{t+1}, and add to G edges from x_{t+1} to all x_i for which $x_i \leq y$. For the other x_i, no edge to or from x_{t+1} is added. Clearly, this operation does not introduce cycles in G. It remains to argue that $I_{i,j}$ is true for all newly created pairs except those where we have added an edge to G. That is:

Lemma 6. *Assume that counterexample y does not allow any refinement, that is, $R = \emptyset$. Then $x_{t+1} = y$ can be completed with values for the other variables in such a way that $I_{i,t+1}$ is true for all i, and $I_{t+1,i}$ is true for all i such that $(x_i \wedge y)\oplus$.*

Proof. Consider first some $x_i \leq y = x_{t+1}$, for which we only have to establish $I_{i,t+1}$. We set $x_{i,t+1} = z_{i,t+1} = 1^n\oplus$, being positive thanks to our assumption that only definite Horn clauses are used. We also set $y_{i,t+1} = y$, so that $x_{i,t+1} \wedge y_{i,t+1} = 1^n \wedge y = y = x_{t+1}$. From $x_i \leq y$ and lemma 2 once more, we obtain $y_{i,t+1} = y \models H^*(x_i)$, and the remaining inequalities are immediate.

Now we consider the case where $(x_i \wedge y)\oplus$, and must argue both $I_{i,t+1}$ and $I_{t+1,i}$. Of course we take $z_{i,t+1} = z_{t+1,i} = (x_i \wedge y)\oplus$, and $y_{i,t+1} = y_{t+1,i} = 1^n$, which makes sure trivially that $y_{i,t+1} \models H^*(x_i)$ and $y_{t+1,i} \models H^*(x_{t+1})$. This leaves as only choices $x_{i,t+1} = y$ and $x_{t+1,i} = x_i$, which satisfactorily relate to $z_{i,t+1}$ and $z_{t+1,i}$ according to the inequalities of the invariant. ∎

Thus, we have completed the proof that the invariants are maintained according to an algorithm that refines, on the basis of each negative counterexample y, the models x_i corresponding to a set L constructed as indicated, or adds y as a new element x_{t+1}. From the invariant, the same proof as in theorem 3 ensures that the algorithm must stop within, at most, the same number of queries as the previously known algorithms.

5 On Question 3: Towards a Query-Optimal Algorithm

This section is much shorter, and its only aim is to survey the current knowledge regarding the limitations that we must expect on our algorithms.

In fact, from [10], we know that any algorithm learning Horn formulas from a polynomial number of membership queries and additional equivalence queries must make at least $\Omega(\frac{mn}{\log n + \log m})$ equivalence queries; moreover, a generalization of the halving algorithm given in the same reference achieves this bound, not only for Horn formulas but for any conjunctions of arbitrary clauses, with polynomially many memberships but at the price of the need of unbounded computational power.

Considering all the queries jointly, the lower bound of Arias, Khardon, and Servedio [4] is $O(m(m+n))$. That is, any algorithm learning Horn formulas from membership and equivalence queries must make at least that many queries. Note that the AFP algorithm pays an extra factor m (or n in case $m < n$). There is some room for improvement, and as of today we bet on the nonoptimality of AFP rather than on the coarseness of the lower bound. We believe that the lower bound known is actually the true upper bound (up to a constant factor); however, we also believe that the algorithms attaining this bound on the number of queries might be forced to do so at the price of not being polynomial time.

More particularly, we feel possible that an algorithm exists that is able to refine optimally each counterexample so that every antecedent maintained is actually the antecedent of a clause in a saturated axiomatization of the target; and in such a way that each equivalence query provides one more clause through a linear number of membership queries. Note that this idea goes back to a scheme discarded in AFP: using membership queries to walk the counterexample down. The problem of choosing what variables to refine for each membership query, though, could be harder than polynomial time, and we tend to expect that there is a transformation so that the use of an oracle for the hypergraph transversal problem (to help us choose sets of variables that get set simultaneously to zero) would provide a query-optimal, possibly nonpolynomial time, algorithm. Whether, assuming this bold conjecture true, a polynomial-time particular case of hypergraph transversal ([12] or [13]) would allow for this process, or for the learnability of wide interesting subclasses of Horn logic, is much too early to answer.

6 On Question 2: Multivalued Dependency Formulas

We discuss here a syntactic variant of Horn clauses whose study might allow us to gain additional intuitions. It is a class of propositional formulas introduced in [17] on the basis of their parallel to multivalued dependencies in databases. Normal forms for databases (notably the Boyce-Codd normal form and its widely used relaxation 3NF, or third normal form) try to avoid certain functional dependencies, where the value of some attribute is fully determined by the values of other attributes. A standard approach is to normalize the relation, that is, decompose it into several relations; and the possibility of correctly decomposing a relation is not characterized exactly by functional dependencies but by the somewhat more complicated multivalued dependencies.

Now, functional dependencies admit a syntactic parallel with Horn clauses that goes, actually, much beyond a syntactic parallel, and indeed their deduction calculi are isomorphic. Applying a similar transformation to multivalued dependencies, one can obtain Sagiv's multivalued dependency formulas, that are conjunctions of multivalued dependency clauses. These are defined precisely as follows: they must have the form $\alpha \rightarrow \beta \vee \gamma$, for disjoint terms α, β, and γ that satisfy the additional condition that their union is the set of all the variables. These implications are naturally called multivalued dependency clauses. See [18] for details on all these issues.

Hermo and Lavin [15] have studied these formulas from the point of view of query learning and approximate fingerprints, proving that they are not learnable with only membership queries or with only equivalence queries. They suggested that multivalued dependency formulas could be learnable from membership and equivalence queries in a similar way to the Horn clause learning algorithm, but, as far as we know, the issue is open as of today. On the basis of the previous sections, we consider that a potential advance could stem from a clarification of the semantic properties that these formulas may enjoy, along the line of the property of closure under intersection characterizing Horn clauses. We provide here one such characterization, hoping for either its usefulness towards a learning algorithm or, possibly, for its intrinsic interest. The statement was announced in [6] but there was just a hint of a proof, which appears here complete for the first time. Our main technical ingredient is as follows.

Definition 2. *Consider a set of binary tuples T. We say that $x \in T$ is a focus of T if, for every $y \in T$, $x \vee y$ is not the top model $\top$.*

Note that, for $y = x$, this implies that the focus x itself is not $\top$. Note also that there may be multiple foci of a given T. In the particular case of a set of two models, either both or neither are foci. Additionally, a theory containing $\top$ has no foci at all.

Definition 3. *Consider a propositional theory T. A focused intersection of T is a model that can be obtained as the intersection of all the members of a subtheory $T' \subseteq T$ that has at least one focus (of T').*

We are ready for the main result of this section: a semantic characterization of the theories defined by Horn or multivalued dependency clauses.

Theorem 4. *A propositional theory can be axiomatized by a conjunction of Horn or multivalued dependency clauses if and only if it is closed under (the operation of taking a) focused intersection.*

We prove this statement through a series of lemmas. The first one is the analogue of lemma 1.

Lemma 7. *Consider a multivalued dependency clause, and a propositional theory T that satisfies it. Assume that T has a focus. Then the intersection of all the members of T satisfies the clause.*

Proof. Let the clause be $\alpha \to \beta \vee \gamma$. If there is a model $y \in T$ with $y \not\models \alpha$ then the intersection also has the same property, and therefore satisfies the clause. Thus, we assume that all elements of T, including some fixed focus x of T, satisfy α, and by (the soundness of) modus ponens they satisfy either β or γ; we classify them into T_β and T_γ accordingly, and remove the focus x from whichever side it fell into. Note that the only model satisfying α and both β and γ is $\top$, which cannot belong to T since otherwise T would have no foci. Thus T_β and T_γ are disjoint.

It is easy to see that the intersection y_β of all the models in T_β (even if this set is empty) satisfies β. A bit less obviously, for all $z \in T_\gamma$, the zeros of z are in β, because the three terms α, β and γ jointly cover all the variables and z satisfies both α and γ; so that the intersection y_γ of T_γ also has all the zeros in γ. Observe that the intersection of all of T, by associativity and commutativity, is exactly $x' = x \wedge y_\beta \wedge y_\gamma$.

Recall that x satisfies α and, by the implication, also either β or γ. Assume first it satisfies γ: since $y_\beta \models \beta$, and the three terms include all the variables, $x \vee y_\beta = \top$, and $x \vee y = \top$ as well for every $y \in T_\beta$ because $y_\beta \leq y$: then x would not be a focus. The only way is that $T_\beta = \emptyset$ so that no such y exists. Then $y_\beta = \top$, and x' satisfies γ since it is the intersection of three models that do.

The other case, dually, is when $x \models \beta$. As we just saw, y_γ has all the zeros in β, so $x \vee y_\gamma = \top$, and we argue exactly as in the previous case. ∎

In general, we will apply this lemma to subtheories, as in definition 4. Let us point out here that the condition that β and γ be disjoint has not been used in the proof; we come back to this observation later on. Now we can concentrate on the forward direction of our main characterization.

Lemma 8. *Assume that T is axiomatized by a conjunction of Horn or multivalued dependency clauses. Then T is closed under focused intersection.*

Proof. Let $T' \subseteq T$ have a focus x. Consider each of the clauses that participate in the axiomatization: whether they are Horn, or multivalued dependency clauses, by the previous lemmas we know that they are satisfied by the intersection x' of all the models in T'; therefore, x' satisfies all the axioms of T, whence it must belong to T. ∎

Let us prove the converse now. Assuming that T is closed under focused intersection, we must show how to axiomatize it by a conjunction of Horn or multivalued dependency clauses: we simply consider all the clauses of these sorts that are true for all of T. We must prove that their conjunction axiomatizes T, that is, a model x satisfies them all if and only if $x \in T$. The "if" part is obvious since these clauses are all true for all of T. Thus it remains to see that a model x that is not in T violates some such clause that is true of T. Our notation will rely strongly on the following definition: for a theory T and a model x, the subset $T_x \subseteq T$ is

$$T_x = \{y \in T \mid x \leq y\}$$

The following is argued essentially as in the characterization of Horn theories, but we sketch the proof for the sake of completeness. It covers the case where the intersection of the models in T above x remains above x.

Lemma 9. *Let $x \notin T$. Consider the intersection z of all the models in T_x. If $x \neq z$, then x violates a Horn clause that is true of T.*

Proof. Clearly $x \leq z$, so that if they differ then $x < z$. We consider a clause of the form $1(x) \to v$ where $v \in V$ is a variable that is true in z and false in x; clearly $x \not\models 1(x) \to v$. However, $T \models 1(x) \to v$: indeed, for each $y \in T$, either $y \notin T_x$, and then it falsifies $1(x)$, or $y \in T_x$, in which case $z \leq y$ forces v to be true in y so that $y \models 1(x) \to v$ as well. ∎

The final case corresponds to x being indeed the intersection of all of T_x, which no longer means that it must be in T since the theory may not be closed under arbitrary intersections: we are actually assuming that $x \notin T$. Let us discard beforehand a special case, namely, $x = \top$, or $T_x = \emptyset$. This is easily handled by the nondefinite Horn clause $\bigwedge_{v \in V} v \to \square$ (where V is the set of all the propositional variables): it excludes $\top$ and no other model, so that it is indeed satisfied by T (since $x = \top \notin T$) and falsified by x.

Thus from now on we assume that $x < \top$. We prove that this case is covered by the multivalued dependency clauses.

Lemma 10. *Let T be a theory closed under focused intersection, and let $x \notin T$, with $x < \top$. Assume that the intersection of all the models in T_x is precisely x. Then x violates a multivalued dependency clause that is true of T.*

Proof. Consider a subset $T' \subseteq T_x \subseteq T$ such that x is still the intersection of all the models in T', but T' has, under this condition, minimal Hamming weight.

Note that, in particular, this implies that each pair y, z in T' reaches $y \vee z = \top$; otherwise, $y \wedge z$ would belong to T by closure under focused intersection, thus to T_x as well, and replacing both in T' by this intersection would reduce the Hamming weight.

As a consequence, fixed any $y \in T'$, all the other elements of T', and their intersection as well, have value true for all those variables that y sets to false. Note also that x is not all true and thus T' is nonempty.

Pick any arbitrary $y \in T'$; since $x \notin T$ but $y \in T_x$, they differ, and $x < y$. Let z_y be the intersection of $T' - \{y\}$, so that $x = y \wedge z_y$. As just argued in the previous paragraph, $y \vee z_y = \top$. Also, $T' - \{y\} \neq \emptyset$ since otherwise $x = y$ (but note that z_y may not be in T).

Let X be the variables satisfied by x, and likewise Y and Z for y and z_y respectively. Consider the clause $\phi = (1(x) \to 1(y) \vee 1(z))$, which is then a multivalued dependency clause (technically, the ones of x should be removed from both disjuncts of the right hand side but this is in fact irrelevant). The minimality of T' (and the fact that $x < y$, so y is not all zeros) implies that $x < z_y$ since otherwise we could cross y off from T' and reduce Hamming weight.

Therefore, $x < y$ and $x < z_y$, which jointly imply that x falsifies ϕ. We prove now that in fact T satisfies it, so that it belongs to the axiomatization we constructed in the first place, and this completes the proof that each model not in T falsifies at least one of the axioms, which is our current claim.

Assume, therefore, that some model $w \in T$ falsifies this clause; that is, it satisfies its left hand side but falsifies both disjuncts of the right hand side. Satisfying the left hand sice means $x \leq w$, so that $w \in T_x$. Falsifying $1(y)$ implies that $w \wedge y < y$. If we can prove that $w \wedge y \in T$ then we are done, since both are above x, thus both are in T_x, and $w \wedge y \in T_x$ as well: it could have been used instead of y in T', contradicting again the minimality of T'.

Here is where closure under focused intersection plays its role: we simply prove that $w \vee y < \top$, and since both w and y are in T, their focused intersection must be as well. Thus it only remains to prove that w and y have a common zero, and for this we use the single remaining property of w, that of not satisfying the second disjunct of the right hand side of ϕ. Namely, $w \not\models 1(z_y)$ means that $1(z_y) \cap 0(w) \neq \emptyset$. Now, recalling $x \leq w$, it follows $0(w) \subseteq 0(x)$, hence $0(w) = 0(w) \cap 0(x)$; and from $x = y \wedge z_y$ so that $0(x) = 0(y) \cup 0(z_y)$. Thus,

$$1(z_y) \cap 0(w) = 1(z_y) \cap 0(w) \cap 0(x) = 1(z_y) \cap 0(w) \cap (0(y) \cup 0(z_y))$$

Applying distributivity,

$$1(z_y) \cap 0(w) = (1(z_y) \cap 0(w) \cap 0(y)) \cup (1(z_y) \cap 0(w) \cap 0(z_y))$$

where the second argument of the union is obviously empty; therefore, given that $1(z_y) \cap 0(w) \neq \emptyset$ the set $1(z_y) \cap 0(w) \cap 0(y)$ is equally nonempty and the larger set $0(w) \cap 0(y)$ is nonempty too, as was to be shown. ■

Taken together, the lemmas prove the main theorem in this section. We can use the same techniques to characterize the case of using only multivalued dependency clauses, without the company of Horn clauses; the property is somewhat less elegant. We first note that a model with less than two variables set to false satisfies every multivalued dependency clause, since it has to falsify both disjuncts of the right hand side in order to falsify the clause.

Theorem 5. *A propositional theory T can be axiomatized by a conjunction of multivalued dependency clauses if and only if it is closed under focused intersection, contains $\top$ and, for each model $x \notin T$, $x = \bigwedge T_x$.*

We only sketch the proof since it uses the same techniques: the “if” part follows from lemma 10, by considering all the multivalued dependency clauses satisfied by T, and picking $x \notin T$ (so $x \neq \top$); then the properties about x and the closure of T are exactly what we need to apply lemma 10 and prove that x falsifies some of these clauses; thus the conjunction of these classes axiomatizes T. For the “only if” part, the observation just before this theorem says that T contains $\top$, and lemma 8 proves that it is closed under focused intersection; pick $x \notin T$, and consider T_x. To prove $x = \bigwedge T_x$ it suffices to see that, for each $v \in 0(x)$, there is $y \in T_x$ for which $v \in 0(y)$: indeed, such y is the model that sets only v to zero. Then, since in this case $x \leq y$ and $y \in T$, we have that $y \in T_x$.

7 On Question 1: the Incomplete Membership Teacher

This was the original technical motivation to start research along the line described here. In fact, there were two additional, less technical reasons. First, this author’s experience in teaching the algorithms from [2], where, in nearly all opportunities, some person in the audience would raise a hand and ask about the possibility of refining several examples at once, a very natural alternative that is easy to think of; and having no answer beyond “we do not know whether that idea can be turned into a better algorithm”, given once and again, felt (and still feels) annoying. Second, a number of interesting results obtained recently in the author’s research group ([11], [7], [5]) connect with closure operators, and in fact the simplest such case, still general enough to provide intuitions on all of them, is the operator mapping x into $H^*(x)$ (with our notation), which is isomorphic to those used in Formal Concept Analysis and has close connections to Database Theory.

As for the Learning Theory motivation proper, it is not difficult to argue that it would be preferable to be able to let the teacher skip some membership queries, that would be answered “I don’t know”. This model of incomplete memberships has been studied in the past, and there are several studies of algorithms for learning monotone DNF in this model ([3], [9], [19]). We found extremely interesting the fact that one important advance [9] came from considering a method of learning monotone DNF on the basis of computing intersections (refinements) of positive examples, which is essentially the same thing as refining negative examples in the Horn case, close to antimonotone. Thence the natural question of whether it is possible to extend the learning algorithm for Horn clauses up to this weaker model. Important efforts together with valuable co-workers notwithstanding, this problem remains, to our knowledge, open today, as is, of course, the same problem for multivalued dependency formulas.

References

1. D. Angluin. Learning Regular Sets from Queries and Counterexamples. *Information and Computation* 75 (1987), 87–106.
2. D. Angluin, M. Frazier, L. Pitt. Learning Conjunctions of Horn Clauses. *Machine Learning* 9 (1992), 147–164.
3. D. Angluin, D. K. Slonim. Randomly Fallible Teachers: Learning Monotone DNF with an Incomplete Membership Oracle. *Machine Learning* 14,1 (1994), 7–26.
4. M. Arias, R. Khardon, R. Servedio. Polynomial Certificates for Propositional Classes. Int. Conf. Computational Learning Theory 2003.
5. J. Baixeries, J.L. Balcázar. Characterization and Armstrong relations for Degenerate Multivalued Dependencies using Formal Concept Analysis. To appear in International Conference on Formal Concept Analysis 2005.
6. J.L. Balcázar, J. Baixeries. Characterization of Multivalued Dependencies and Related Expressions. Int. Conf. Discovery Science 2004.
7. J.L. Balcázar, G. Casas-Garriga. On Horn Axiomatizations for Sequential Data. To appear in International Conference on Database Theory 2005.
8. J.L. Balcázar, J. Diaz, R. Gavaldà, O. Watanabe. Algorithms for Learning Finite Automata from Queries: A Unified View. In: Advances in Algorithms, Languages, and Complexity, D.-Z. Du and K.-I. Ko (eds.), Kluwer Academic Publishers, 1997.
9. N. H. Bshouty, N. Eiron. Learning Monotone DNF from a Teacher that Almost does not Answer Membership Queries. *Journal of Machine Learning Research* 3 (2002), 49–57.
10. N. H. Bshouty, S. A. Goldman, T. R. Hancock, S. Matar. Asking Questions to Minimize Errors. *Journal of Computer and System Sciences* 52 (1996), 268–286.
11. G. Casas-Garriga. Summarizing Sequential Data with Closed Partial Orders. To appear in SIAM Data Mining Conference 2005.
12. C. Domingo, N. Mishra, L. Pitt. Efficient Read-Restricted Monotone CNF/DNF Dualization by Learning with Membership Queries. *Machine Learning* 37, 1 (1999), 89–110.
13. T. Eiter, G. Gottlob. Identifying the Minimal Transversals of a Hypergraph and Related Problems. *SIAM Journal on Computing* 24, 6 (1995), 1278–1304.
14. M. Frazier, L. Pitt. Learning From Entailment: An Application to Propositional Horn Sentences. Int. Conference Machine Learning 1993, 120–127.
15. M. Hermo, V. Lavín. Negative Results on Learning Dependencies with Queries. Seventh International Symposium on Artificial Intelligence and Mathematics (2002).
16. R. Khardon, D. Roth. Reasoning with Models. *Artificial Intelligence* 87 (1996), 187–213.
17. Yehoshua Sagiv. An Algorithm for Inferring Multivalued Dependencies with an Application to Propositional Logic. *Journal of the ACM* 27, 2 (1980), 250–262.
18. Y. Sagiv, D. Delobel, D. Scott Parker, R. Fagin. An Equivalence Between Relational Database Dependencies and a Fragment of Propositional Logic. *Journal of the ACM* 28,3 (1981), 435–453. Corrigendum: *Journal of the ACM* 34,4 (1987), 1016–1018.

19. H.-U. Simon. How Many Missing Answers Can Be Tolerated by Query Learners? *Theory of Computing Systems* 37,1 (2004), 77–94

The internal logic of Bell's states[⋆]

Giulia Battilotti and Paola Zizzi

Dipartimento di Matematica Pura ed Applicata
Università di Padova
via G. Belzoni n.7
I–35131 Padova, Italy

Abstract. We introduce logical judgements for the internal logic of a quantum computer with two qubits, in the two particular cases of non-entanglement (separable states) and maximal entanglement (Bell's states). To this aim, we consider an internal (reversible) measurement which preserves the probabilities by mirroring the states. We then obtain logical rules obeying the reflection principle which illustrate the different computational behaviour of separable and Bell's states.

1 Introduction

The main aim of our work is to look for the internal logic of quantum computation [9], illustrating the point of view of a hypotetical "internal observer" who lives inside the black box. Such an observer, introduced in [15], can perform "internal" (reversible) measurements in the quantum system. The idea is that internal measurements give rise to logical assertions [1], [2], which are then treated following the reflection principle as in basic logic [12]. By the reflection principle, logical connectives are the result of importing some pre-existing metalinguistic links between assertions into the formal language. We then obtain adequate connectives, corresponding to the *physical* links which are present inside the black box.

In [2] whe have considered a toy-model quantum computer with one qubit and we have obtained an interpretation of the *superposition* of the two basis states in terms of the additive conjunction "&" (and, dually, with the additive disjunction "$\oplus$"). The resulting logic is paraconsistent [7], [6], [10], and symmetric, like basic logic. We recall that in a paraconsistent logic, both the non-contradiction and the excluded middle principle do not hold. Here, we introduce a model of two qubits. This makes it possible to deal with two different physical links occurring between two qubits of the register: *maximal entanglement* (the two qubits are a Bell pair) and *non entanglement* (the two qubits state is separable).

2 Measurements and Mirrors

To obtain the judgements for the two qubits model, we extend the definition of the internal measurement to the case of two qubits. We remind that, in a

[⋆] Work supported by the research project "Logical Tools for Quantum Information Theory", Department of Pure and Applied Mathematics, University of Padova.

Hilbert space C^2, the internal measurement of one qubit is given by a unitary 2×2 complex matrix [15]. In such a model, the judgements are obtained by means of a particular internal measurement, called "mirror measurement" [2], given by the matrices:

$$M = e^{i\phi} \begin{pmatrix} \alpha & 0 \\ 0 & \alpha^* \end{pmatrix} \tag{1}$$

where $\alpha\alpha^* = |\alpha|^2 = 1$. We have:

$$M(a|0\rangle + b|1\rangle) = e^{i\phi}(\alpha a|0\rangle + \alpha^* b|1\rangle) \tag{2}$$

So, our mirrors are "quasi-identities"; actually, they modify the longitude of the qubit in the Bloch sphere, that is, the probability amplitudes:

$$a \to a' = e^{i\phi}\alpha a$$

$$b \to b' = e^{i\phi}\alpha^* b$$

and preserve the "internal truth" given by the probabilities, since $|a'|^2 = |a|^2$ and $|b'|^2 = |b|^2$. For this reason, we have chosen mirror-matrices to witness the internal truth and the consequent logical judgements, as we shall see in the next section.

We now extend the mirror-matrices to C^4. If $M_1 = e^{i\phi_1} \begin{pmatrix} \alpha & 0 \\ 0 & \alpha^* \end{pmatrix}$ and $M_2 = e^{i\phi_2} \begin{pmatrix} \beta & 0 \\ 0 & \beta^* \end{pmatrix}$, the tensor product $M = M_1 \otimes M_2$ given by:

$$M = e^{i(\phi_1+\phi_2)} \begin{pmatrix} \alpha\beta & 0 & 0 & 0 \\ 0 & \alpha\beta^* & 0 & 0 \\ 0 & 0 & \alpha^*\beta & 0 \\ 0 & 0 & 0 & \alpha^*\beta^* \end{pmatrix} = e^{i\phi} \begin{pmatrix} \gamma & 0 & 0 & 0 \\ 0 & \delta & 0 & 0 \\ 0 & 0 & \delta^* & 0 \\ 0 & 0 & 0 & \gamma^* \end{pmatrix} \tag{3}$$

is also a mirror matrix. In fact, the most general state of C^4 in the computational basis is:

$$|\psi\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle \tag{4}$$

and one has:

$$M|\psi\rangle = e^{i\phi}(\gamma a|00\rangle + \delta b|01\rangle + \delta^* c|10\rangle + \gamma^* d|11\rangle) \tag{5}$$

and again we have: $a \to a' = e^{i\phi}\gamma a$... and so on, then probabilities are preserved: $|a'|^2 = |a|^2$... and so on.

Note that, if $|\psi\rangle$ is one of the Bell states $|\psi_\pm\rangle = 1/\sqrt{2}(|00\rangle \pm |11\rangle)$, its mirroring is: $M|\psi_\pm\rangle = 1/\sqrt{2}e^{i\phi}(\gamma|00\rangle \pm \gamma^*|11\rangle)$. Similarly, for the Bell's state $|\phi_\pm\rangle = 1/\sqrt{2}(|01\rangle \pm |10\rangle)$, we have $M|\phi_\pm\rangle = 1/\sqrt{2}e^{i\phi}(\delta|01\rangle \pm \delta^*|10\rangle)$. Then Bell states behave as a single particle in the mirroring, since the result has the same form as (2). This fact will be shown in Sect.4 in logical terms.

3 From Mirrors to Judgements in the Black Box

We recall the line of thought followed for the case of the one qubit model (see [2]). Inside the Black Box, a hypothetical internal observer **P** is equipped with mirror-matrices and then can perform reversible measurements. Outside the Black Box, instead, an external observer **G** can perform standard quantum measurements, represented by projectors. **G** has a standard quantum logic [3].

It is well known that performing a standard quantum measurement in the given basis, e.g. $|0\rangle, |1\rangle$, on a qubit $|q\rangle = a|0\rangle + b|1\rangle$, means to apply one of the two projectors P_0 or P_1, breaking the superposition and obtaining one of the two basis states. So the observer **G** can "read" the value of the qubit as $|0\rangle$, asserting: "$|0\rangle$ is true" or $|1\rangle$, asserting: "$|1\rangle$ is true". So, let us suppose that a standard quantum measurement is applied and a result $A \in \{|0, |1\rangle\rangle\}$ is obtained. Then **G** asserts "A is true", written as:

$$\vdash A$$

Conversely, denoting by $A^\perp$ the opposite result, **G** asserts "$A^\perp$ is true", written as:

$$\vdash A^\perp$$

By the no cloning theorem [14], after the measurement, **G** can assert *only one* of the two. The same does not happen to the internal observer **P**, who applies a mirror to $|q\rangle$. In fact, any mirror is the sum of the two projectors:

$$M = e^{i\phi} \begin{pmatrix} \alpha & 0 \\ 0 & \alpha^* \end{pmatrix} = e^{i\phi}\alpha P_0 + e^{i\phi}\alpha^* P_1 \tag{6}$$

so that $M|q\rangle = e^{i\phi}\alpha P_0|q\rangle + e^{i\phi}\alpha^* P_1|q\rangle$. Hence **P** obtains a *superposition* of the two results obtainable by **G**. We write then both the above judgements together:

$$\vdash A \qquad\qquad \vdash A^\perp$$

What is a couple of possibilities for **G** is instead a unique fact for **P**! By the reflection principle, a connective corresponds to a link between judgements. As in [12], we make the connective "&" correspond to the above couple, putting

$$\vdash A\&A^\perp \equiv \qquad \vdash A \qquad \vdash A^\perp \tag{7}$$

Then

$$\vdash A\&A^\perp \tag{8}$$

"$A\&A^\perp$ is true" is the judgement put by **P** inside the Black Box, concerning the value of the qubit $|q\rangle$.

Now, let us consider a two-qubit model, that is a Black Box equipped with a register $|\psi\rangle$ of two qubits $|q_1\rangle, |q_2\rangle$. Fixed a basis of C^4, e.g. the computational basis $|00\rangle, |01\rangle, |10\rangle, |11\rangle$, the external observer, who performs a standard quantum measurement in that basis, applies one of the four projectors $P_{00}, P_{01}, P_{10}, P_{11}$.

Let us suppose that she finds an answer $A \in \{|0\rangle, |1\rangle\}$ for $|q_1\rangle$ and an answer $B \in \{|0\rangle, |1\rangle\}$ for $|q_2\rangle$. Then she has a register of two classical bits, and her assertion is:

$$\vdash A, B \tag{9}$$

where the comma stands for the register link between the two classical bits. We interpret the link by the multiplicative connective on the right of the sequent, which is called "par", written as "$\oslash$". "Par" has the same physical meaning of the tensor product "times", written $\otimes$, which, however, is used in linear logic and basic logic to interpret the comma on the left of the sequent. Then, as in [12] we put the equation:

$$\vdash A \oslash B \quad \equiv \quad \vdash A, B \tag{10}$$

What are all the possible judgements? If the measurements of the two qubits are independent, four combinations are possible:

$$\vdash A, B \quad \vdash A, B^{\perp} \quad \vdash A^{\perp}, B \quad \vdash A^{\perp}, B^{\perp}$$

and so four judgements are obtainable:

$$\vdash A \oslash B \qquad \vdash A \oslash B^{\perp} \qquad \vdash A^{\perp} \oslash B \qquad \vdash A^{\perp} \oslash B^{\perp} \tag{11}$$

This is the case of a pair of unentangled qubits. On the contrary, let us consider a Bell pair. In such a case the two measurements are related, thus not all combinations are possible: if $\vdash A, B$ is a result, then $\vdash A^{\perp}, B^{\perp}$ is the only other. Note that **G** is in general unaware of the link existing between $|q_1\rangle$ and $|q_2\rangle$ inside the Black Box, so that the same register link is used outside. Then, in the case of entanglement, two judgements are possible outside:

$$\vdash A \oslash B \qquad \vdash A^{\perp} \oslash B^{\perp} \tag{12}$$

As in the case of one qubit, the external observer can put only one of the possible judgements. Again, the judgement of the internal observer is given by a mirror measurement, and mirrors of C^4 are obtainable as a linear combination of the four projectors:

$$M = e^{i\phi}(\gamma P_{00} + \delta P_{01} + \delta^* P_{10} + \gamma^* P_{11}) \tag{13}$$

So **P**, who applies M to the register $|\psi\rangle$, obtains:

$$M|\psi\rangle = e^{i\phi}(\gamma P_{00}|\psi\rangle + \delta P_{01}|\psi\rangle + \delta^* P_{10}|\psi\rangle + \gamma^* P_{11}|\psi\rangle) \tag{14}$$

that is the superposition of the possible values obtainable outside. If $|\psi\rangle$ is not a maximally entangled state, every projector gives a result and the judgement of the internal observer is obtained as a superposition of the four judgements in (11), that is:

$$\vdash (A \oslash B)\&(A \oslash B^{\perp})\&(A^{\perp} \oslash B)\&(A^{\perp} \oslash B^{\perp}) \tag{15}$$

If $|\psi\rangle$ is a Bell state, for example $|\psi_{\pm}\rangle$, one has $M|\psi_{\pm}\rangle = e^{i\phi}(\gamma P_{00}|\psi_{\pm}\rangle + \gamma^{*}P_{11}|\psi_{\pm}\rangle)$. The same holds for the other Bell states. The result of the measurement of a maximally entangled state is the superposition of the two judgements (12), that is:

$$\vdash (A \oslash B)\&(A^{\perp} \oslash B^{\perp}) \tag{16}$$

Now, let us consider the internal measurement without any reference to the external one. We know that $\mathbf{P}$ achieves a superposition $A\&A^{\perp}$ measuring $|q_1\rangle$ and another $B\&B^{\perp}$ measuring $|q_2\rangle$. The two results are linked by two different register links present in the black box, that is non entanglement and maximal entanglement. For non entanglement we write $\asymp$, while, for maximal entanglement $\bowtie$. The mirror measurement gives the judgements:

$$\vdash (A\&A^{\perp}) \asymp (B\&B^{\perp}) \tag{17}$$

and

$$\vdash (A\&A^{\perp}) \bowtie (B\&B^{\perp}) \tag{18}$$

respectively.

We put the two reflection principles, writing $\oslash_0$ (no correlation) and $\oslash_1$ (maximum correlation) for the two corresponding binary connectives:

$$\vdash (A\&A^{\perp}) \oslash_0 (B\&B^{\perp}) \equiv \ \vdash (A\&A^{\perp}) \asymp (B\&B^{\perp}) \tag{19}$$

and

$$\vdash (A\&A^{\perp}) \oslash_1 (B\&B^{\perp}) \equiv \ \vdash (A\&A^{\perp}) \bowtie (B\&B^{\perp}) \tag{20}$$

So we have two kinds of internal judgements concerning our register of two qubits:

$$\vdash (A\&A^{\perp}) \oslash_0 (B\&B^{\perp}) \tag{21}$$

and

$$\vdash (A\&A^{\perp}) \oslash_1 (B\&B^{\perp}) \tag{22}$$

As for the states that are neither separable nor maximally entangled, we argue that connectives $\oslash_x$, $x \in (0,1)$ (where x is a degree of correlation), should be introduced, perhaps leading to a kind of fuzzy logic.

4 Towards a calculus of judgements

Of course, the internal judgements (21) and (22) must be equivalent to the superposition of the external ones (15) and (16), that is we have to prove the following equivalence:

$$\vdash (A\&A^{\perp}) \oslash_0 (B\&B^{\perp}) \Longleftrightarrow \vdash (A \oslash B)\&(A \oslash B^{\perp})\&(A^{\perp} \oslash B)\&(A^{\perp} \oslash B^{\perp}) \tag{23}$$

in the non entangled case, and the equivalence:

$$\vdash (A\&A^{\perp}) \oslash_1 (B\&B^{\perp}) \Longleftrightarrow \vdash (A \oslash B)\&(A^{\perp} \oslash B^{\perp}) \tag{24}$$

in the maximally entangled case.

This can be achieved disassembling the judgements and then assembling them the other way around, as we show in the following pair of derivations:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash (A \oslash B)\&(A^{\perp} \oslash B^{\perp})}{\dfrac{\vdash A \oslash B}{\vdash A, B}\oslash \quad \dfrac{\vdash A^{\perp} \oslash B^{\perp}}{\vdash A^{\perp}, B^{\perp}}\oslash}\&}{\vdash (A\&A^{\perp}) \bowtie (B\&B^{\perp})}\&congr}{\vdash (A\&A^{\perp}) \oslash_1 (B\&B^{\perp})}\oslash_1}{} \tag{25}$$

for the maximally entangled case, and

$$\dfrac{\dfrac{\dfrac{\vdash (A \oslash B)\&(A \oslash B^{\perp})\&(A^{\perp} \oslash B)\&(A^{\perp} \oslash B^{\perp})}{\dfrac{\vdash A \oslash B}{\vdash A, B}\oslash \quad \dfrac{\vdash A \oslash B^{\perp}}{\vdash A, B^{\perp}}\oslash \quad \dfrac{\vdash A^{\perp} \oslash B}{\vdash A^{\perp}, B}\oslash \quad \dfrac{\vdash A^{\perp} \oslash B^{\perp}}{\vdash A^{\perp}, B^{\perp}}\oslash}\&}{\vdash (A\&A^{\perp}) \asymp (B\&B^{\perp})}\&cont}{\vdash (A\&A^{\perp}) \oslash_0 (B\&B^{\perp})}\oslash_0 \tag{26}$$

for the non entangled case.

We now explain how one must read the above derivations, which are performed by the internal observer. The key point, once again, is the superposition link. We remind that, in basic logic, which is a quantum linear logic, the additive connective & is obtained putting the following definitional equation:

$$\Gamma \vdash A\&B \quad \equiv \quad \Gamma \vdash A \quad \Gamma \vdash B \tag{27}$$

which does not admit any context besides the active formulae A and B (visibility of basic logic). On the contrary, the same equation in any non-quantum logic can be written with a context C:

$$\Gamma \vdash A\&B, C \quad \equiv \quad \Gamma \vdash A, C \quad \Gamma \vdash B, C \tag{28}$$

But, in this case, one would derive distributivity of the multiplicative connective $\oslash$ with respect to the additive connective & (e.g. see [12]). Inside the black box, we have (at least!) two kinds of distinct links for registers and hence we can deal with two different versions of the equation for &:

$$\Gamma \vdash (A\&A^{\perp}) \bowtie (B\&B^{\perp}) \quad \equiv \quad \Gamma \vdash A, B \quad \Gamma \vdash A^{\perp}, B^{\perp} \tag{29}$$

for maximal entanglement, and:

$$\Gamma \vdash (A\&A^{\perp}) \asymp (B\&B^{\perp}) \quad \equiv \quad \Gamma \vdash A, B \quad \Gamma \vdash A^{\perp}, B \quad \Gamma \vdash A, B^{\perp} \quad \Gamma \vdash A^{\perp}, B^{\perp} \tag{30}$$

for non entanglement. Note that this last equation could be derived from the classical one!

Then, of course, a classical logician would derive only the rules for the unentangled case, that correspond to a classical use of the context. On the contrary, in the equation (29), we have an odd use of the context, corresponding to entanglement. Notice moreover that, due to visibility, the basic logic equation (27) represents a lower bound for both the equations valid in the black box, that is (29) and (30). This means again that the external observer, unaware of the actual kind of correlations present in the black box, but aware of her unawareness, must, as Scepticism suggests, suspend judgement. Hence, for her judgements, no context at all.

Let's us go back and explain the above derivations (25) and (26). As we have seen in basic logic, the definitional equations give rise to formation and implicit reflection rules, each one corresponding to a direction of the equivalence. Such rules can be found in the above couple of derivations, which can be read top-down and bottom-up, providing the required equivalences between the judgements. The premise Γ, used in the definitional equations, is not present in our judgements yet, because its justification inside the Black Box is under study. In particular, in (25) we have the new rule "&*congr*", which follows from the definitional equation (29), where "*congr*" is for "congruence", as equation (29) resembles a congruence rule. It shows in logical terms that entanglement is a particular form of superposition. In (26) the rule "&*cont*" (for "context"), coming from the definitional equation (30), that is equivalent to the classical equation (28), is a form of a classical &-rule of sequent calculus.

In our opinion, the derivations, despite their simplicity, are already quite informative for a logical calculus which aims to grasp the efficiency of quantum computation. In fact, they show how the "quantum parallelism", in the entangled case, can be obtained by only one half of the derivation branches! This is achieved thanks to the rule "&*congr*". As for the non entangled case, the rule "&*cont*" relizes a classical parallelism (superposition without entanglement).

Note that derivation (25) has the same form of the following one, that shows how the judgements $\vdash A$ and $\vdash A^{\perp}$ can be assembled and disassembled in the one qubit case (cf. [2]):

$$\cfrac{\cfrac{\vdash A \& A^{\perp}}{\vdash A \quad \vdash A^{\perp}}}{\vdash A \& A^{\perp}} \tag{31}$$

In this sense we think that a Bell's state seen from inside the quantum computer can be assimilated to a single particle. Notice that here "inside" means that the quantum computer is embedded in a non-commutative geometry background [15]. In other words a 2-qubits register is a fuzzy sphere with four elementary cells each one encoding a two-qubits string $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ (see fig. (1)). In the maximally entangled case, the fuzzy sphere has two cells, each one with doubled surface area, and encoding the two-bits string $|00\rangle$ and $|11\rangle$ (see fig. (2)). The latter situation resembles the case of one qubit, where the fuzzy sphere has two elementary cells, each one encoding the one bit string $|0\rangle$ and $|1\rangle$ (see fig. (3)).

The fact that a Bell's state (as seen from inside a quantum computer) "pretends" to be a single particle, while we think it is not, is at the origin of all paradoxes related to entanglement. For example, "non-locality" is just a problem of the external observer, who lives in a local space-time. Instead, the Bell's state, as seen from inside a quantum computer, lives in a non-local space, which is the fuzzy sphere. Moreover, as far as causality is concerned, let us consider the cut rule:

$$\frac{A \vdash B \quad B \vdash C}{A \vdash C}\,cut$$

which is a causal relation. But the cut rule as such is not admissible inside the Black Box (as we have seen in [2]), since it corresponds to a projective measurement performed in the external world. Thus we argue that the usual meaning of causality is absent in the case of a quantum computer on a fuzzy sphere (internal logic). This is the reason why an external observer, who lives in a causal world, sees as a paradox the non-causal behaviour of a Bell's state.

Notice that the model of a quantum computer in a non-commutative geometry can be identified with a model of Computational Loop Quantum Gravity (CLQG) [16]. For a review on Loop Quantum Gravity (LQG) see for example [11]. This is equivalent to consider a quantum computer at the Planck scale. Causality at the Planck scale is a very controversial issue, and some authors, like Sorkin and collaborators [4] are inclined to believe in a sort of micro-causality that they discuss in terms of causal sets. However, the fact is that the light cone at the Plank scale might be "smeared" by the very strong quantum fluctuations of the metric field (the "quantum foam" [13]) and this would indicate that (micro) causality is lost at that scale at least in its usual setting. At the light of our logical result, i.e., that the cut rule as such is not admissible inside the Black Box, we are now lead to argue that causality is absent at the fundamental scale.

To conclude, it is just the intrinsic non-locality of non-commutative geometry which leads to a modification of micro-causality (see for example [5] in support of this idea). Then, it is non-commutative geometry itself which "trivializes" at once both the "paradoxes" of non-locality and non-causality of entanglement. Recall that non-commutative geometry was exploited in [15] to describe the inner quantum world, and here some logical consequences are analyzed, producing an internal logic. We think that our internal approach can lead to a logical explanation of the different computational behaviour (speed up!) of quantum systems with respect to classical ones. In fact we describe the quantum computation itself, without mixing it with the effects of the external measurements, which would break quantum parallelism.

References

1. Battilotti, G.: Basic logic and quantum computing: logical judgements by an insider observer, Proc. FQI04, 16-19 april 2004, Camerino, Italy, International Journal of Quantum Information **3**,1 (2005) 105-109. arXiv: quant-ph/0407057.
2. Battilotti, G. Zizzi, P.: Logical Interpretation of a Reversible Measurement in Quantum Computing, arXiv: quant-ph/0408068.
3. Birkhoff, G., von Neumann, J.: The Logic of Quantum Mechanics, Annals of Mathematics, **37** (1936) 823-843.

4. Bombelli, L., Lee, J., Meyer, D. and Sorkin, R.: Space-time as a causal set, Phys. Rev. Lett. **59** (1987) 521-524.
5. Chong-Sun Chu, Ko Furuta and Takeo Inami: Locality, Causality and Noncommutative Geometry. arXiv: hep-th/0502012.
6. Dalla Chiara, M. L., Giuntini, R.: Paraconsistent Quantum Logic, Foundations of Physics **19**(1989) 891-904.
7. Dalla Chiara, M. L., Giuntini, R., Leporini, R.: Quantum Computational Logics. A Survey, arXiv: quant-ph/0305029.
8. Girard, J. Y.: Linear Logic, Theoretical Computer Science **50**(1987) 1-102.
9. Nielsen, N. A., Chuang, I. L.: *Quantum Computation and Quantum Information*, Cambridge University Press (2000).
10. Priest, G.: Paraconsistent logic, in Handbook of Philosophical logic, second edition, D. Gabbay and F. Guenthner Eds., Kluwer Academic Publishers 2002.
11. Rovelli, C.: *Quantum Gravity*, Cambridge, U.K. Univ. Press (2004).
12. Sambin, G., Battilotti, G. and Faggian, C.: Basic Logic: Reflection, Symmetry, Visibility, The Journal of Symbolic Logic, **65** (2000) 979-1013.
13. Wheeler, J. A.: *Geometrodynamics*, Academic Press, New York (1962).
14. Wootters, W. K., Zureck, W. H.: A single quantum cannot be cloned, Nature **299** (1982) 802-803.
15. Zizzi, P. A.: Qubits and Quantum Spaces, Proc. FQI04, 16-19 april 2004, Camerino, Italy. International Journal of Quantum Information, **3**,1 (2005) 287-291. arXiv: quant-ph/0406154.
16. Zizzi, P. A.: A minimal model for quantum gravity, Physics Letters A, to appear. arXiv: gr-qc/0409069.

Fig. 1

Two unentangled qubits.

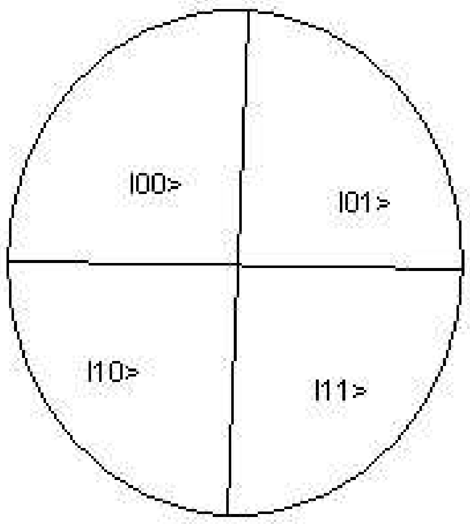


Fig. 2

A Bell's state

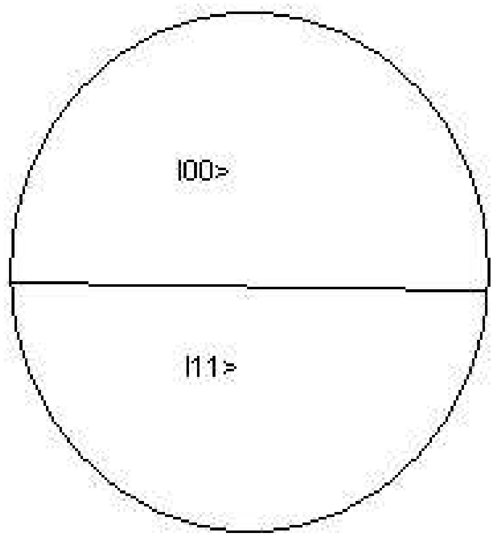
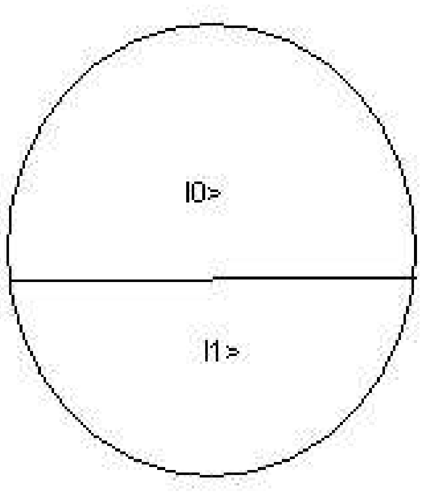


Fig. 3

One qubit



Logical Characterizations of $\mathrm{P}_{\mathcal{K}}$ and $\mathrm{NP}_{\mathcal{K}}$ over an Arbitrary Structure $\mathcal{K}$

Olivier Bournez[1], Felipe Cucker[⋆2], Paulin Jacobé de Naurois[1], and Jean-Yves Marion[1]

[1] LORIA, 615 rue du Jardin Botanique, BP 101
54602 Villers-lès-Nancy Cedex
Nancy, France
{Olivier.Bournez,Paulin.De-Naurois,Jean-Yves.Marion}@loria.fr
[2] Department of Mathematics, City University of Hong Kong
83 Tat Chee Avenue
Kowloon, Hong Kong
macucker@math.cityu.edu.hk

Abstract. We focus on the BSS model of computation over arbitrary structures. We propose a modification of the logical characterizations of $\mathrm{P}_{\mathcal{K}}$ and $\mathrm{NP}_{\mathcal{K}}$ given by Grädel and Gurevich in order to make them applicable to any computational structure $\mathcal{K}$.

1 Introduction

In the last decades complexity theory developed in many directions to offer a broad perspective of the complexity of computations. Two directions relevant to our work are the extension of complexity theory to domains other than finite alphabets and the characterizations of complexity classes in machine-independent terms.

A seminal paper for the first direction above is the one by Blum, Schub and Smale [1], where a theory of computation and complexity that allowed an ordered ring or field as alphabet for the space of admissible inputs was developed. The authors emphasized the case when the ring is the field of real numbers, $\mathbb{R}$, bringing the theory of computation into the domain of analysis, geometry and topology. Later on extended to the more abstract level of computations over arbitrary structures in [2, 3], this BSS model, among other things, makes use of the extensively developed subject of the theory of discrete computations, initiated by the work of Turing [4].

In the classical setting of computation over finite alphabets, a great amount of work has been done, along the second direction to provide machine independent characterization of the major complexity classes. Such characterizations lend further credence to the importance of the complexity classes considered, relate them to issues relevant to programming and verification, and, in general,

⋆ Partially supported by City University of Hong Kong SRG grant 7001290.

help understand the notion of computability in a whole perspective. Several approaches for designing such characterizations have been chosen, among which one can find descriptive complexity (global methods of finite model theory).

Descriptive complexity began with the work of Fagin [5], who proved, in the classical setting, that the class NP can be characterized as the class of sets describable within existential second-order logic. Subsequently, Vardi and Immerman [6–8] used this approach to characterize P. Several other characterizations exist, for classes like LOGSACE [9] or PSPACE [10–18]. An overview of the subject can be found in [19, 20]. These characterizations, however, applied originally only to the classical setting of computation over finite alphabets since only finite models were considered.

In [21], the notion of $\mathbb{R}$-structure has been introduced, and characterization of deterministic polynomial time $\mathrm{P}_{\mathbb{R}}$ and non-deterministic polynomial time $\mathrm{NP}_{\mathbb{R}}$ in terms of logics over these $\mathbb{R}$-structures were provided. These results have been later on extended in [22], in order to capture other complexity classes, and in [23] over structures other than $\mathbb{R}$.

In this paper, we extend the notions of $\mathbb{R}$-structures to $\mathcal{K}$-structures over an arbitrary computational structure $\mathcal{K}$, extend the notion of first order logic and second order logic over $\mathbb{R}$-structures of [21] to first and second-order logic over $\mathcal{K}$-structures. We also define a proper notion of fixed-point rule, different from [21], allowing to capture $\mathrm{P}_{\mathcal{K}}$, and propose a characterization of $\mathrm{NP}_{\mathcal{K}}$. Our characterizations, while only slightly different from the ones of [23], apply to any computational structure. In particular, when $\mathcal{K}$ is $\mathbb{Z}_2$, they coincide with the classical ones.

Indeed, in the characterizations of [21, 22], some logic is hidden in basic computations over the real numbers. In particular, these characterizations assume that the set of real numbers is equipped with a sign function and a multiplication: therefore they cannot be trivially extended, say, to the additive setting or to the field of complex numbers. Similar restrictions apply to the more general characterizations of [23], where it is required that the underlying computational structure contains an expansion of $(\mathbb{N}, 0, 1+, -, <, \max, \min, \Sigma, \Pi)$. Here again, this does not apply to the field of complex number, or to finite structures. Our characterizations overcome these drawbacks.

2 Computing over Arbitrary Structures

This section is devoted to a brief exposition of the BSS model of computation, based on [24] and [3].

2.1 Arbitrary Structures

The following notion of *structure* describes the domain of elements over which a machine computes, as well as the operations and relations that can be performed over these elements.

Definition 1. (Structure): *A* structure

$$\mathcal{K} = (\mathbb{K}, \{op_i\}_{i\in I}, rel_1 \ldots, rel_l, \mathbf{0}, \mathbf{1})$$

is given by some underlying set $\mathbb{K}$, *some operators* $\{op_i\}_{i\in I}$, *and a finite number of relations* $rel_1, \ldots, rel_l$.

Constants correspond to operators of arity 0. While the index set I may be infinite, the number of operators with arity greater than 1 is assumed to be finite, that is, only symbols for constants may be infinitely many. We will not distinguish between operator and relation symbols and their corresponding interpretations as functions and relations of the same arity, respectively over the underlying set $\mathbb{K}$. We assume that the equality relation $=$ is a relation of the structure, and that there are at least two constant symbols, with different interpretations (denoted by $\mathbf{0}$ and $\mathbf{1}$ in our work) in the structure.

The main examples of structures are $\mathbb{Z}_2 = (\{0,1\}, =, \mathbf{0}, \mathbf{1})$, $\mathbb{R} = (\mathbb{R}, +, -, *, /, <, \{c \in \mathbb{R}\})$ and $\mathbb{R}_{ovs} = (\mathbb{R}, +, -, <, \{c \in \mathbb{R}\})$. When considered over $\mathbb{Z}_2$, the notions of BSS computability and complexity coincide with the one of the classical Turing model.

2.2 BSS Machines

Denote by $\mathcal{K}$ a structure as above. We now define machines over $\mathcal{K}$ following the lines of [24]. A machine over $\mathcal{K}$ is roughly a Turing machine with a bi-infinite tape, where the tape cells holds elements of $\mathbb{K}$. A machine can perform operations and relations of $\mathcal{K}$ over its tape elements at unit cost. More formally,

Definition 2. (Machine): *A* machine *over* $\mathcal{K}$ *consists of an input space* $\mathcal{I} = \mathbb{K}^* = \bigcup_{n\in\mathbb{N}} \mathbb{K}^n$, *an output space* $\mathcal{O} = \mathbb{K}^*$, *and a register space*[3] $\mathcal{S} = \mathbb{K}_* = \{(x_i, \ldots, x_j), i \in \mathbb{Z}, j \in \mathbb{Z}, x_{i\leq k\leq j} \in \mathbb{K}\}$, *together with a connected directed graph whose nodes labelled* $0, \ldots, N$ *correspond to the set of different* instructions *of the machine. These nodes are of one of the six following types: input, output, computation, copy, branching and shift nodes.*

1. *Input nodes*. There is only one input node and it is labelled with 0. Associated with this node there is a next node $\beta(0)$, and the input map $g_I : \mathcal{I} \to \mathcal{S}$.
2. *Output nodes*. There is only one output node which is labelled with 1. It has no next nodes, once it is reached the computation halts, and the output map $g_O : \mathcal{S} \to \mathcal{O}$ places the result of the computation in the output space.
3. *Computation nodes*. Associated with a node m of this type there are a next node $\beta(m)$ and a map $g_m : \mathcal{S} \to \mathcal{S}$. The function g_m replaces the component indexed by 0 of $\mathcal{S}$ by the value $op(w_0, \ldots, w_{n-1})$ where $w_0, w_2, \ldots, w_{n-1}$ are components 0 to $n-1$ of $\mathcal{S}$ and op is some operation of the structure $\mathcal{K}$ of arity n. The other components of $\mathcal{S}$ are left unchanged. When the arity n is zero, m is a constant node.

[3] In the original paper by Blum, Shub and Smale, this is called the *state* space. We rename it *register* space to avoid confusions with the notion of 'state' in a Turing machine.

4. *Copy nodes.* Associated with a node m of this type there are a next node $\beta(m)$ and a map $g_m : \mathcal{S} \to \mathcal{S}$. The function g_m performs one of the following actions:
   – Replace the component indexed by 0 by a copy of the component indexed by 1. This is denoted as a *copy left*.
   – Replace the component indexed by 0 by a copy of the component indexed by -1. This is denoted as a *copy right*.
   – Exchange the component indexed by 0 and the component indexed by 1 . This is denoted as a *switch*.
5. *Branch nodes.* There are two nodes associated with a node m of this type: $\beta^+(m)$ and $\beta^-(m)$. The next node is $\beta^+(m)$ if $rel(w_0, \ldots, w_{n-1})$ is true and $\beta^-(m)$ otherwise, where $w_0, w_2, \ldots, w_{n-1}$ are components 0 to $n-1$ of $\mathcal{S}$ and rel is some relation of the structure $\mathcal{K}$ of arity n.
6. *Shift nodes.* Associated with a node m of this type there is a next node $\beta(m)$ and a map $\sigma : \mathcal{S} \to \mathcal{S}$. The σ is either a left or a right shift.

As in the classical Turing model of computation, complexity classes over an arbitrary structure $\mathcal{K}$ can be defined such as $\mathrm{P}_{\mathcal{K}}$ and $\mathrm{NP}_{\mathcal{K}}$, as well as the notions of reduction and completeness. Over the three major structures, natural complete problems exist for $\mathrm{NP}_{\mathcal{K}}$, and the question $\mathrm{P}_{\mathcal{K}} = \mathrm{NP}_{\mathcal{K}}$ remains open.

3 First-Order Logic on $\mathcal{K}$-structures

In the rest of the paper, denote by $\mathcal{K}$ a structure as above. We will consider machines over $\mathcal{K}$, as well as the complexity classes $\mathrm{P}_{\mathcal{K}}$ and $\mathrm{NP}_{\mathcal{K}}$.

3.1 Definitions

We assume that the reader is familiar with first-order mathematical logic. In order to capture computation over arbitrary structure, we extend the notion of $\mathbb{R}$-Structures from [21, 24] to the notion of $\mathcal{K}$-Structures. Our $\mathcal{K}$-Structures are particular instances of the meta-finite structures of Grädel and Gurevich [23].

Definition 3. **($\mathcal{K}$-Structures)***: Let L_s, L_f be finite vocabularies, where L_s may contain relation and function symbols with their arities, and L_f contains function symbols only, with their arities. An $\mathcal{K}$-structure of signature $\sigma = (L_s, L_f)$ is a pair $\mathcal{D} = (\mathcal{U}, \mathcal{F})$ consisting of*

(i) *a finite logical structure $\mathcal{U}$ of vocabulary L_s, called the* skeleton *of $\mathcal{D}$, whose (finite) universe A is also called the* universe *of $\mathcal{D}$, and*
(ii) *a finite set $\mathcal{F}$ of functions $f_i^{\mathcal{D}} : A^{k_i} \to \mathbb{K}$ interpreting the function symbols f_i in L_f, with the corresponding arities k_i.*

$\mathcal{K}$-Structures as above are a generalization of the concept of logical structure, over a fixed computational structure $\mathcal{K}$, in order to denote inputs to algorithms performed by BSS machines over $\mathcal{K}$. The skeleton of a $\mathcal{K}$-structure is used for describing the finite, discrete part of a structure, while the set $\mathcal{F}$ is used for describing its computational part.

Example 1. Consider for instance the real case $\mathbb{R}$, and the $\mathrm{NP}_{\mathbb{R}}$-complete problem 4FEAS: decide whether a given polynomial of degree 4 has a real zero. Inputs to this decision problem are polynomials of degree 4, which can be encoded by $\mathbb{R}$-structures of signature $(L_s, L_f) = (\emptyset, \{f_0, f_1, f_2, f_3, f_4\})$ as follows: The universe A contains one variable for each variable of the polynomial. The interpretation of the function $f_4 : A^4 \to \mathbb{R}$ of L_f gives the (real) coefficient for each monomial of degree 4, and similarly for the other coefficients with $f_3, \dots f_0$.

We present now the notion of first-order logic over $\mathcal{K}$-structures. A key feature of this logic is that the quantified variable range only over the skeleton of the structures, and not over $\mathcal{K}$. Assume that V is a fixed, countable set of variables.

Definition 4. (First-Order Logic for $\mathcal{K}$-Structures)*: The language of* first-order logic *for $\mathcal{K}$-structures,* $\mathrm{FO}_{\mathcal{K}}$*, contains for each signature $\sigma = (L_s, L_f)$ a set of terms and formulas. We first define terms, of which there are two kinds. When interpreted in a $\mathcal{K}$-structure $\mathcal{D} = (\mathcal{U}, \mathcal{F})$ with universe A, each term t takes values either in A, in which case we call it an* index term*, or in $\mathbb{K}$, in which case we call it a* number term*. Terms are defined inductively as follows.*

- **(i)** *The set of index terms is the closure of the set V of variables under the application of functions symbols of L_s.*
- **(ii)** *If $h_1, \dots, h_k$ are index terms, and X is a k-ary function symbol of L_f, then $X(h_1, \dots, h_k)$ is a number term.*
- **(iii)** *If $t_1, \dots, t_{k_i}$ are number terms, and op_i is an operation of the computational structure $\mathcal{K}$ of arity k_i, $op_i(t_1, \dots, t_{k_i})$ is also a number term. In particular, operations of arity 0 of $\mathcal{K}$ yield constant number terms.*

Atomic formulas *are defined as follows.*

- **(i)** *equalities $h_1 = h_2$ of index terms are atomic formulas.*
- **(ii)** *If $t_1, \dots, t_{k_i}$ are number terms, and rel_i is a relation of the computational structure $\mathcal{K}$ of arity k_i, $rel_i(t_1, \dots, t_{k_i})$ is an atomic formula. In particular we may consider equalities $t_1 = t_2$ of number terms.*
- **(iii)** *If $h_1, \dots, h_k$ are index terms, and P is a k-ary relation symbol in L_s, $P(h_1, \dots, h_k)$ is also an atomic formula.*

The set of formulas *of* $\mathrm{FO}_{\mathcal{K}}$ *is the smallest set containing all atomic formulas and which is closed under Boolean connectives, $\vee, \wedge, \neg$, and quantification $(\exists v)\psi$ and $(\forall v)\psi$.*

Definition 5. (Interpretation of First-Order Logic for $\mathcal{K}$-Structures)*: We adopt the usual interpretation. The main point is that the quantifiers range over the universe of the $\mathcal{K}$-Structure: a formula $(\exists x)\psi$ where ψ contains variables $x, y_1, \dots, y_n$, $n \geq 0$ is* `true` *at $b \in A^n$ if there exists $a \in A$ such that $\psi^{\mathcal{D}}$ is* `true`*, where x is interpreted as a and y_i is interpreted as b_i, for $i = 1, \dots, n$. Similarly for a formula $(\forall x)\psi$ being interpreted as $\neg(\exists x)\neg\psi$. For details, see the Appendix.*

Definition 6. (Order)*: Let $\sigma = (L_s, L_f)$ be a signature. A $\mathcal{K}$-structure $\mathcal{D} = (\mathcal{U}, \mathcal{F})$ of signature σ with universe A is* ordered *if there is a binary relation symbol $\leq$ in L_s whose interpretation $\leq^{\mathcal{D}} \in \mathcal{U}$ is a total order on A.*

Remark 1. Our order relation replaces the ranking function of [21, 22, 24, 23]. In these papers, the ranking function is a function in L_f, defining a bijection between A and $\{0, \ldots, |A| - 1\}$. This uses the property that $\mathbb{R}$ is ordered and contains at least the natural numbers. It is not applicable in our setting of arbitrary structure, where no such requirement can be made on $\mathcal{K}$.

Remark 2. It is checkable in first-order logic over $\mathcal{K}$-structures that a structure is ordered. Based on this order relation, one can also define in first-order logic a lexicographic order over A^k for any $k \in \mathbb{N}$. Therefore, we will freely use the symbol $\leq_k$ for a total order over A^k. In what follows, we will only consider ordered $\mathcal{K}$-structures. Note also that the minimal element of A^k and the maximal one with respect to this lexicographic order are definable in first-order logic. We will denote them 0 and $n^k - 1$ respectively.

Definition 7. (Fixed Point Rule)*: Fix an signature $\sigma = (L_s, L_f)$, D a relation symbol of arity $r \geq 0$ and Z a function symbol of arity r, both not contained in this signature. Let $H(D, t_1, \ldots, t_r), I_1(D, t_1, \ldots, t_r), \ldots, I_{k-1}(D, t_1, \ldots, t_r)$, $k \geq 0$ be first-order formulas over the signature $(L_s \cup \{D\}, L_f \cup \{Z\})$ with free variables $t_1, \ldots, t_r$. Let $F_1(Z, t_1, \ldots, t_r), \ldots, F_k(Z, t_1, \ldots, t_r)$ be number terms over the signature $(L_s \cup \{D\}, L_f \cup \{Z\})$ with free variables $t_1, \ldots, t_r$. We allow D to appear several times in H and the I_i's, but we do not require that its arguments are $(t_1, \ldots, t_r)$. The only restriction is that the number of free variables in H and the I_i's coincide with the arity of D. A similar remark holds for the F_i's and Z. For any $\mathcal{K}$-structure $\mathcal{D}$ of signature σ and any interpretation $\zeta : A^r \to \mathcal{K}$ of Z and $\Delta \subseteq A^r$ of D, respectively, the number terms $F_1(Z, t_1, \ldots, t_r), \ldots, F_k(Z, t_1, \ldots, t_r)$ define functions*

$$
\begin{aligned}
F^{\mathcal{D}}_{1,\zeta}, \ldots, F^{\mathcal{D}}_{k,\zeta} \ &: \ A^r \to \mathcal{K} \\
u_1, \ldots, u_r &\to \begin{cases} [F_1(Z \leftarrow \zeta, t_1, \ldots, t_r \leftarrow u_1, \ldots u_r)]^{\mathcal{D}} \\ \vdots \\ [F_k(Z \leftarrow \zeta, t_1, \ldots, t_r \leftarrow u_1, \ldots u_r)]^{\mathcal{D}}, \end{cases}
\end{aligned}
$$

where $[F_i(Z \leftarrow \zeta, t_1, \ldots, t_r \leftarrow u_1, \ldots u_r)]^{\mathcal{D}}$ is obtained from $F_k(Z, t_1, \ldots, t_r)$ by replacing any occurrence of Z by ζ, any occurrence of t_j by u_j, and interpreting the whole number term in $\mathcal{D}$. Also, the formulas $H(D, t_1, \ldots, t_r)$ and $I_i(D, t_1, \ldots, t_r)$, $1 \leq i \leq k-1$ define relations

$$
\begin{aligned}
H^{\mathcal{D}}_{\Delta}, I^{\mathcal{D}}_{i,\Delta} &\subseteq A^r \\
u_1, \ldots, u_r &\to \begin{cases} [H(D \leftarrow \Delta, t_1, \ldots, t_r \leftarrow u_1, \ldots u_r)]^{\mathcal{D}} \\ [I_1(D \leftarrow \Delta, t_1, \ldots, t_r \leftarrow u_1, \ldots u_r)]^{\mathcal{D}} \\ \vdots \\ [I_{k-1}(D \leftarrow \Delta, t_1, \ldots, t_r \leftarrow u_1, \ldots u_r)]^{\mathcal{D}}. \end{cases}
\end{aligned}
$$

Let us now consider the sequence of pairs $\{\Delta^i, \zeta^i\}_{i\geq 0}$ with $\Delta^i \subseteq A^r$ and $\zeta^i : A^r \to \mathcal{K}$ inductively defined by

$$\begin{aligned}
\Delta^0(x) &= \texttt{false} \quad \textit{for all } x \in A^r \\
\zeta^0(x) &= \mathbf{0} \quad \textit{for all } x \in A^r \\
\Delta^{i+1}(x) &= \begin{cases} H^{\mathcal{D}}_{\Delta^i}(x) & \textit{if } \Delta^i(x) = \texttt{false} \\ \texttt{true} & \textit{otherwise} \end{cases} \\
\zeta^{i+1}(x) &= \begin{cases} F^{\mathcal{D}}_{1,\zeta^i}(x) & \textit{if } \neg\Delta^i(x) \wedge I^{\mathcal{D}}_{1,\Delta^i}(x), \textit{ else} \\ \vdots & \\ F^{\mathcal{D}}_{k-1,\zeta^i}(x) & \textit{if } \neg\Delta^i(x) \wedge I^{\mathcal{D}}_{k-1,\Delta^i}(x), \textit{ else} \\ F^{\mathcal{D}}_{k,\zeta^i}(x) & \textit{if } \neg\Delta^i(x), \textit{ else} \\ \zeta^i(x). & \end{cases}
\end{aligned}$$

Since $\Delta^{i+1}(x)$ only differs from $\Delta^i(x)$ in case the latter is `false`*, one has that $\Delta^j = \Delta^{j+1}$ for some $j < |A|^r$. In this case, we also have that $\zeta^j = \zeta^{j+1}$. We denote these fixed points by D^∞ and Z^∞ respectively and call them the* fixed points of $H(D, t_1, \dots, t_r)$, the $F_i(Z, t_1, \dots, t_r)$, and the $I_i(D, t_1, \dots, t_r)$ on $\mathcal{D}$. *The* fixed point rule *is now stated as follows. If $F_i(Z, t_1, \dots, t_r)$, $1 \leq i \leq k$ are number terms as previously, and $H(D, t_1, \dots, t_r)$, $I_i(D, t_1, \dots, t_r)$, $1 \leq i \leq k$ are first-order formulas as previously, then*

$$\mathbf{fp}[D(t_1, \dots, t_r) \leftarrow H(D, t_1, \dots, t_r)](u_1, \dots, u_r)$$

is a first-order formula of signature (L_s, L_f), and

$$\mathbf{fp}[Z(t_1, \dots, t_r) \leftarrow F_i(Z, t_1, \dots, t_r), I_i(D, t_1, \dots, t_r), H(D, t_1, \dots, t_r)](u_1, \dots, u_r)$$

is a number term of signature (L_s, L_f). Their interpretations on a given $\mathcal{K}$-structure $\mathcal{D}$ are D^∞ and Z^∞, respectively.

Remark 3. In [21, 22, 24], the fixed point rule allows to define only number terms. Thus, the authors need to introduce another rule, the maximization rule, which allows them to compute characteristic functions of relations of L_s as number terms, and perform logical operations such as AND, NOT, OR, by the appropriate use of multiplication, addition and a sign function. This approach, however, is not appropriate to our setting of an arbitrary structure, where no specific requirement is made on the functions and operations of the computational structure. We introduce therefore the possibility to define relations as fixed point, such as D^∞ in the definition above, which enables us to perform legally all kinds of logical operations in first-order logic. While our definition of the fixed point rule is more complicated, it makes the need for a maximization rule obsolete and enables us to prove our results over arbitrary structures, which can be specialized to be the real numbers with addition and order $\mathbb{R}_{ovs}$, or the complex numbers. Moreover, it allows us to capture also classes decided by machines having no constant node, by an appropriate specialization of the computational structure.

The fixed point rule allows us to defined an extension of the first-order logic, as follows.

Definition 8. (Fixed Point First-Order Logic for $\mathcal{K}$-Structures)*: Fixed point first-order logic for $\mathcal{K}$-structures, denoted by* $\mathrm{FP}_{\mathcal{K}}$*, is obtained by augmenting first-order logic* $\mathrm{FO}_{\mathcal{K}}$ *with the fixed point rule.*

3.2 Characterizing $\mathbf{P}_{\mathcal{K}}$

It is clear that, for any signature $\sigma = (L_s, L_f)$, one can encode in polynomial time finite ordered $\mathcal{K}$-structures of signature σ as elements of $\mathbb{K}^*$ of polynomial size. For $\mathcal{D}$ an ordered $\mathcal{K}$-structure of signature σ, we will denote by $e(\mathcal{D})$ its encoding. Details can be found in the Appendix.

Definition 9. (Decision Problem of Ordered $\mathcal{K}$-structures)*: Let $\sigma = (L_s, L_f)$ be a signature containing a binary relation symbol $\leq$. We denote by Struct(σ) the set of ordered $\mathcal{K}$-structures of signature σ. A decision problem of ordered $\mathcal{K}$-structures of signature σ is a subset of Struct(σ). Modulo a polynomial encoding of ordered $\mathcal{K}$-structures, any decision problem of ordered $\mathcal{K}$-structures can be seen as a decision problem over $\mathcal{K}$ in the BSS model, and vice versa.*

This yields the following characterization of $\mathrm{P}_{\mathcal{K}}$.

Theorem 1. *Let S be a decision problem of ordered $\mathcal{K}$-structures. Then the following statements are equivalent.*

- **(i)** $S \in \mathrm{P}_{\mathcal{K}}$.
- **(ii)** *There exists a sentence ψ in fixed point first-order logic such that $S = \{\mathcal{D} | \mathcal{D} \models \psi\}$.*

Example 2. Recall the signature $(L_s, L_f) = (\emptyset, \{f_0, f_1, f_2, f_3, f_4\})$ given in Example 1 for encoding inputs to the 4FEAS problem. Consider $X : A \to \mathcal{K}$ whose interpretation in a $\mathcal{K}$-structure $\mathcal{D}$ of signature $(L_s, L_f \cup X)$ instantiates the variables of the polynomial. The problem 4EVAL of checking that a given polynomial of degree 4 evaluates to 0 at a given point is in $\mathrm{P}_{\mathcal{K}}$. It can be formulated as follows: There exists a sentence ϕ in fixed point first-order logic over the signature $(L_s, L_f \cup X)$ such that

$$\mathrm{4EVAL} = \{\mathcal{D} | \mathcal{D} \models \phi\}.$$

4 Second-Order Logic on $\mathcal{K}$-structures

4.1 Definitions

Recall that $\mathcal{K}$-structures can be naturally seen as points in $\mathcal{K}^*$. A decision problem of ranked $\mathcal{K}$-structures is then a decision problem, where the (positive) inputs are encoded as $\mathcal{K}$-structures, and satisfy the natural property of being ranked.

Definition 10. (Existential Second-Order Logic For $\mathcal{K}$-Structures)*:* *We say that ψ is an* existential second-order sentence *(of signature $\sigma = (L_s, L_f)$) if $\psi = \exists Y_1, \ldots, \exists Y_r \phi$, where ϕ is a first-order sentence in* $\mathrm{FO}_\mathcal{K}$ *of signature $(L_s, L_f \cup \{Y_1, \ldots, Y_r\})$. The function symbols $Y_1, \ldots, Y_r$ are called* function variables*. Existential second-order sentences are interpreted in $\mathcal{K}$-structures $\mathcal{D}$ of signature σ in the following way: a sentence $\mathcal{D} \models (\psi = \exists Y_1, \ldots, \exists Y_r \phi)$ if there exist functions $X_1, \ldots, X_r : A^{r_i} \rightarrow \mathbb{K}$ with r_i the arity of Y_i, such that the interpretation of ψ taking $Y_i^\mathcal{D}$ to be X_i yields* `true`*. The set of second-order sentences together with this interpretation constitutes* existential second-order logic *and is denoted by* $\exists\mathrm{SO}_\mathcal{K}$*.*

Existential second-order logic has at least the expressive power of fixed point first-order logic. Indeed:

Proposition 1. *For every sentence ψ of signature σ in* $\mathrm{FP}_\mathcal{K}$ *there exists a sentence $\widetilde{\psi}$ of signature σ in* $\exists\mathrm{SO}_\mathcal{K}$ *such that, for every $\mathcal{K}$-structure $\mathcal{D}$,*

$$\mathcal{D} \models \psi \text{ if and only if } \mathcal{D} \models \widetilde{\psi}.$$

4.2 Characterizing $\mathrm{NP}_\mathcal{K}$

Theorem 2. *Let S be a decision problem of ordered $\mathcal{K}$-structures. Then the following statements are equivalent.*

(i) $S \in \mathrm{NP}_\mathcal{K}$.

(ii) *There exists an existential second-order sentence ψ such that $S = \{\mathcal{D} | \mathcal{D} \models \psi\}$.*

Example 3. Recall the signature $(L_s, L_f) = (\emptyset, \{f_0, f_1, f_2, f_3, f_4\})$ given in Example 1 for encoding inputs to the 4FEAS problem. Consider $X : A \rightarrow \mathcal{K}$ whose interpretation in a $\mathcal{K}$-structure $\mathcal{D}$ instantiates the variables of the polynomial. Consider a formula ϕ in fixed point first-order logic over the signature $(L_s, L_f \cup X)$, whose interpretation for a $\mathcal{K}$-structure $\mathcal{D}$ checks whether the evaluation of the polynomial at the point $X^\mathcal{D} \in \mathcal{K}^{|A|}$ is 0. A instance of the result above is

$$4\mathrm{FEAS} = \{\mathcal{D} | \mathcal{D} \models \exists X \phi\}.$$

Note that by Proposition 1 ϕ can be expressed in existential second-order logic.

References

1. Blum, L., Shub, M., Smale, S.: 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
2. Goode, J.B.: Accessible telephone directories. Journal for Symbolic Logic**59** (1994) 92–105
3. Poizat, B.: Les Petits Cailloux. Aléas (1995)

4. Turing, A.M.: On computable numbers, with an application to the *entscheidungsproblem.* In: Proc. London Math. Society. Volume 42 of 2. (1936) 230–265
5. Fagin, R.: Generalized first order spectra and polynomial time recognizable sets. In Karp, R., ed.: Complexity of Computation. SIAM-AMS (1974) 43–73
6. Vardi, M.: Complexity of relational query languages. In: ACM Symposium on Theory of Computing. (1982) 137–146
7. Immerman, N.: languages which capture complexity classes. In: ACM Symposium on Theory of Computing. (1983) 347–354
8. Immerman, N.: Relational queries computable in polynomial time. Information and Control **68** (1986) 86–104
9. Gurevich, Y.: Algebras of feasible functions. In: Twenty Fourth Symposium on Foundations of Computer Science, IEEE Computer Society Press (1983) 210–214
10. Moschovakis, Y.: Abstract recursion as the foundation for a theory of algorithms. In: Computation and Proof Theory (Proc. of the ASL Meeting in Aachen. Lecture notes in Mathematics, Springer Verlag, Berlin (1983)
11. Gurevich, Y., Shelah, S.: Fixed-point extensions of first-order logic. Annals of Pure and Applied Logic **32** (1986) 265–280
12. Immerman, N.: Languages that capture complexity classes. SIAM Journal of Computing **16** (1987) 760–778
13. Bonner, A.J.: Hypothetical datalog: complexity and expressibility. In: Proceedings of the International Conference on Database Theory. Volume 326 of LNCS., Berlin (1988) 144–160
14. Abiteboul, S., Vianu, V.: Fixed point extensions of first-order logic and datalog-like languages. In: Proceedings of the Fourth Annual Symposium on Logic in Computer Science, Washington, D.C., IEEE Computer Society Press (1989) 71–79
15. Leivant, D.: Inductive definitions over finite structures. Information and Computation **89** (1990) 95–108
16. Abiteboul, S., Simon, S., Vianu, V.: Non-deterministic languages to express deterministic transformations. In: Proc. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. (1990)
17. Abiteboul, S., Vianu, V.: Datalog extensions for database queries and updates. Journal of Computer and System Sciences **43** (1991) 62–124
18. Immerman, N.: Dspace$[n^k]$ = var$[k+1]$. In: Proc. of the 6th IEEE Symp. on Structure in Complexity Theory. (1991) 334–340
19. Ebbinghaus, H.D., Flum, J.: Finite Model Theory. Perspectives in Mathematical Logic. Springer-Verlag, Berlin (1995)
20. Immerman, N.: Descriptive complexity. Springer Verlag (1999)
21. Grädel, E., Meer, K.: Descriptive complexity over the real numbers. In: Proceedings of the 27th Annual ACM Symp. on the Theory of Computing. (1995) 315–324
22. Cucker, F., Meer, K.: Logics which capture complexity classes over the reals. J. of Symb. Logic (1999) 363–390
23. Grädel, E., Gurevich, Y.: Metafinite model theory. Information and Computation **140** (1998) 26–81
24. Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and Real Computation. Springer-Verlag (1998)

Appendix

Interpretation of First-Order Logic for $\mathcal{K}$-Structures

Let $\sigma = (L_s, L_f)$ be a signature, and $\mathcal{D}$ be a $\mathcal{K}$-structure of signature σ. For any index term $h(x_1, \dots, x_n)$ containing $n \geq 0$ variables, and any point $a \in A^n$, the interpretation of the function symbols of h in $\mathcal{D}$ extend to a natural interpretation $h^{\mathcal{D}}(a) \in A$ of h in $\mathcal{D}$ at the point a, where x_i is interpreted as a_i for $i = 1, \dots, n$. The same holds with number terms $t(x_1, \dots, x_n)$, the interpretation $t^{\mathcal{D}}(a)$ at a lying in $\mathbb{K}$. The interpretation of the relation symbols of L_s in $\mathcal{D}$ and the relations of $\mathcal{K}$ enable us to associate *truth values* with atomic formulas evaluated at point in A^n. Thus, if $h_1, \dots, h_k$ are index terms and $t_1, \dots, t_l$ are number terms containing the variables $x_1, \dots, x_n$, and $a \in A^n$, we say that $r(h_1, \dots, h_k)$ is `true` in $\mathcal{D}$ if $(h_1^{\mathcal{D}}, \dots, h_k^{\mathcal{D}}) \in r^{\mathcal{D}}$ and `false` otherwise, and, for a l-ary relation rel of $\mathcal{K}$, $rel(t_1, \dots, t_l)$ is `true` if $(t_1^{\mathcal{D}}, \dots, t_k^{\mathcal{D}}) \in rel$ in $\mathcal{K}$, and `false` otherwise. The interpretation of the logical connectives $\vee$, $\wedge$ and $\neg$ is the usual. A formula $(\exists x)\psi$ where ψ contains variables $x, y_1, \dots, y_n$, $n \geq 0$ is `true` at $b \in A^n$ if there exists $a \in A$ such that $\psi^{\mathcal{D}}$ is `true`, where x is interpreted as a and y_i is interpreted as b_i, for $i = 1, \dots, n$. Similarly for a formula $(\forall x)\psi$ being interpreted as $\neg(\exists x)\neg\psi$. Proceeding recursively, we can associate with any formula ϕ with free variables $x_1, \dots, x_n$ over σ and any $a \in A^n$ a truth value. If this value is `true` we say that $\mathcal{D}$ satisfies $\phi(a)$, and we denote this by $\mathcal{D} \models \phi(a)$. A formula with no variable is called a *sentence*. If T is a set of sentences over σ, $\mathcal{D}$ satisfies T if and only if $\mathcal{D}$ satisfies all the sentences of T. This is denoted as $\mathcal{D} \models T$. The $\mathcal{K}$-structures $\mathcal{D}$ such that $\mathcal{D} \models T$ are called the *models* of T. The sentences of T are called *axioms* and T is a *theory*.

Encoding of Ordered $\mathcal{K}$-structures

Let $\sigma = (L_s, L_f)$ be a signature containing a binary relation symbol $\leq$, and let $\mathcal{D}$ be an ordered $\mathcal{K}$-structure of signature σ. Next, replace all relations in the skeleton by the appropriate characteristic functions $\chi : A^k \to \{\mathbf{0}, \mathbf{1}\} \subseteq \mathbb{K}$. Thus, we get a structure with skeleton a set A and functions $X_1, \dots, X_t$ of the form $X_i : A^{k_i} \to \mathbb{K}$. Each of these functions X_i can be represented by a tuple $\xi = (x_0, \dots, x_{m_i - 1}) \in \mathbb{K}^{m_i}$ with $m_i = |A|^{k_i}$ and $x_j = X_i(\overline{a}(j))$ where $\overline{a}(j)$ is the jth tuple in A^{k_i} with respect to the lexicographic order on A^{k_i} induced by $\leq$. The concatenation

$$e(\mathcal{D}) = \xi_1.\xi_2.\dots.\xi_t$$

of these tuples gives the encoding $e(\mathcal{D}) \in \mathbb{K}^{m_1 + \dots + m_t}$.

Clearly, for a fixed finite signature and for any finite $\mathcal{K}$-structure $\mathcal{D}$ of size n, the size of $e(\mathcal{D})$ is bounded by some polynomial n^l where l depends only on the signature. Thus, appending zeros to $e(\mathcal{D})$ if necessary, we can also view $e(\mathcal{D}) = (x_0, \dots, x_{l-1})$ as a single function $X_{\mathcal{D}} : A^l \to \mathbb{K}$. This means that one can encode an ordered $\mathcal{K}$-structure by a single function from the ordered set $\{0, \dots, n^l - 1\}$ into $\mathbb{K}$. Moreover, this encoding can be performed in polynomial time.

Proof of Theorem 1

Let us first prove the (**i**) $\Rightarrow$ (**ii**) direction. Assume that the signature is $\sigma = (L_s, L_f)$, and that it contains a binary relation symbol $\leq$. Let $S \in \mathrm{P}_{\mathcal{K}}$, and let M be a polynomial time BSS machine over $\mathcal{K}$ deciding S. Assume that, for any ordered $\mathcal{K}$-structure $\mathcal{D}$ of size n, encoded in $\mathbb{K}^*$ by $e(\mathcal{D})$, the computation time of M on $e(\mathcal{D})$ is bounded by n^m, for some $m \in \mathbb{N}$. Assume also that the size of $e(\mathcal{D})$ is n^m. This is without loss of generality since it suffices to add some padding $\mathbf{0}$'s to the encoding $e(\mathcal{D})$ of $\mathcal{D}$. Assume also without loss of generality that the state space of M is bounded by n^m. A point in this space has coordinates $(x_0, x_1, \ldots, x_{n^m-1})$. In the following, we will use the formalism of a Turing machine, with a scanning head moving in $\mathbb{K}^m$ instead of a state space $\mathbb{K}_*$.

Consider the lexicographic order $\leq_m$ on A^m induced by the order $\leq$ on A. By Remark 2, $\leq_m$ is defined in first-order logic, and induces a bijection between A^m and the set $\{0, \ldots, n^m - 1\}$. Therefore, for any $t \in A^m$ we will define:

$$t - 1 = \max_{s \in A^m} \{s <_m t\}$$
$$t + 1 = \min_{s \in A^m} \{t <_m s\}.$$

It is clear that these elements are definable in first-order logic over ordered $\mathcal{K}$-structures, thus we will freely use these notations hereafter. Note however that, when t is minimal in A^m, we have $t - 1 = t$. A similar remark holds for t maximal with $t + 1 = t$. Following this notation, we will identify A^m with $\{0, \ldots, n^m - 1\}$.

Next, we assume that the number of nodes of M is bounded by 2^k, $k \in \mathbb{N}$. Note that k is fixed, independent of $\mathcal{D}$ or n. Thus, the nodes of M will be denoted as elements in A^k, with the proviso that $|A| \geq 2$. We also assume that they correspond to the first elements of A^m with respect to the lexicographic order $\leq_m$. Since their number is constant, they are all definable in first-order logic. We will therefore denote them with constants $v_{type} \in A^m$, where *type* denotes the type of the node as in Definition 2. Recall also the constant $0 \in A^m$, corresponding the the minimal element of A^m as in Remark 2.

Consider now the following relation symbol, not contained in L_s, of arity $4m$:

$$\texttt{Node}(v, t, pos, c) = \texttt{true} \text{ iff}$$
$$\text{the current node of } M \text{ at step } t \text{ is } v \text{ and}$$
$$\begin{cases} pos = 0 & \text{and the head of } M \text{ is on cell } c \\ pos = 1 & \text{and the head of } M \text{ is not on cell } c. \end{cases}$$

Consider also the following function symbol, not contained in L_f, of arity $2m$:

$$\texttt{Cell}(t, c) = \text{content of cell } c \text{ at step } t.$$

If `Cell` can be defined in $\mathrm{FP}_{\mathcal{K}}$ by a number term Z, then the implication (**i**) $\Rightarrow$ (**ii**) holds. The $\mathcal{K}$-structure $\mathcal{D}$ is accepted by M if and only if after n^m steps, the content of the first cell is $\mathbf{1}$. That is, $\texttt{Cell}(n^m - 1, 0) = \mathbf{1}$, were $n^m - 1$ is the maximal element in A^m, as defined in Remark 2. Node and Cell can be defined inductively as follows:

$$\begin{aligned}
\texttt{Node}(input, 0, pos, c) &\leftarrow [(pos = 0) \vee (c = 0)] \wedge [(pos = 1) \vee (c \neq 0)] \\
\texttt{Node}(v, 0, pos, c) &\leftarrow \texttt{false} \text{ if } v \neq input \\
\texttt{Node}(v, t+1, pos, c) &\leftarrow \texttt{Node}(v_{op_i}, t, pos, c) \text{ for } v_{op_i} \text{ s.t. } v = \beta(v_{op_i}) \\
&\quad \vee\ \texttt{Node}(v_{shift_l}, t, pos, c+1) \text{ for } v_{sh_l} \text{ s.t. } v = \beta(v_{shift_l}) \\
&\quad \vee\ \texttt{Node}(v_{shift_r}, t, pos, c-1) \text{ for } v_{sh_r} \text{ s.t. } v = \beta(v_{shift_r}) \\
&\quad \vee\ \texttt{Node}(v_{copy_l}, t, pos, c) \text{ for } v_{copy_l} \text{ s.t. } v = \beta(v_{copy_l}) \\
&\quad \vee\ \texttt{Node}(v_{copy_r}, t, pos, c) \text{ for } v_{copy_r} \text{ s.t. } v = \beta(v_{copy_r}) \\
&\quad \vee\ \texttt{Node}(v_{switch}, t, pos, c) \text{ for } v_{switch} \text{ s.t. } v = \beta(v_{switch}) \\
&\quad \vee\ \texttt{Node}(v_{output}, t, pos, c) \text{ for } v_{output} \text{ s.t. } v = \beta(v_{output}) \\
&\quad \vee\ \texttt{Node}(v_{rel_i}, t, 0, c) \wedge rel_i(\texttt{Cell}(t, c), \ldots, \texttt{Cell}(t, c + k_i)) \\
&\qquad \wedge pos = 0 \quad \text{for } v_{rel_i} \text{ s.t. } v = \beta^+(v_{rel_i}) \\
&\quad \vee\ \texttt{Node}(v_{rel_i}, t, 0, c) \wedge \neg rel_i(\texttt{Cell}(t, c), \ldots, \texttt{Cell}(t, c + k_i)) \\
&\qquad \wedge pos = 0 \quad \text{for } v_{rel_i} \text{ s.t. } v = \beta^-(v_{rel_i})
\end{aligned}$$

and:

$$\texttt{Cell}(0, c) \leftarrow X^{\mathcal{D}}(c)$$

$$\texttt{Cell}(t+1, c) \leftarrow \begin{cases} op_i(\texttt{Cell}(t, c), \ldots, \texttt{Cell}(t, c + k_i)) & \text{if } \texttt{Node}(v_{op_i}, t, 0, c) \\ \texttt{Cell}(t, c+1) & \text{if } \texttt{Node}(v_{copy_l}, t, 0, c) \\ \texttt{Cell}(t, c-1) & \text{if } \texttt{Node}(v_{copy_r}, t, 0, c) \\ \texttt{Cell}(t, c+1) & \text{if } \texttt{Node}(v_{switch}, t, 0, c) \\ \texttt{Cell}(t, c-1) & \text{if } \texttt{Node}(v_{switch}, t, 0, c-1) \\ \texttt{Cell}(t, c) & \text{otherwise} \end{cases}$$

This proves that Cell can be defined in $\mathrm{FP}_{\mathcal{K}}$ as the fixed point of the inductive definition above, from which (**i**) $\Rightarrow$ (**ii**) follows.

To prove (**ii**) $\Rightarrow$ (**i**), we only need to prove that formulas and number terms defined with the fixed point rule can be evaluated in polynomial time. Since the number of updates is polynomially bounded, one only needs to apply the inductive definition a polynomial number of times, which ends the proof. □

Proof of Proposition 1

It suffices to prove that the fixed point rule of Definition 7 can be expressed within existential second-order logic over $\mathcal{K}$-structures. Assume F_i, I_i, H, $1 \leq i \leq k$, Z and D are as in Definition 7. Every occurrence of $\mathbf{fp}[Z(t_1, \ldots, t_r) \leftarrow F_i(Z, t_1, \ldots, t_r), I_i(D, t_1, \ldots, t_r), H(D, t_1, \ldots, t_r)](u_1, \ldots, u_r)$ and of $\mathbf{fp}[D(t_1, \ldots, t_r) \leftarrow H(D, t_1, \ldots, t_r)](u_1, \ldots, u_r)$ in ψ will be replaced by $Z(u_1, \ldots, u_r)$ and $D(u_1, \ldots, u_r)$, respectively. Denote by ϕ the resulting formula.

For any r-ary function $F : A^r \to \mathbb{K}$, denote by $\widetilde{F}$ the $r-ary$ formula with variables $(u_1, \ldots, u_r)$ defined by $\widetilde{F}(u_1, \ldots, u_r) = (F(u_1, \ldots, u_r) = \mathbf{1})$.

Then, $\widetilde{\psi}$ is

$$\begin{aligned}
&\exists Z \exists D \forall (u_1, \ldots, u_r)\\
&\widetilde{D}(u_1, \ldots, u_r) \Leftrightarrow H(\widetilde{D}, u_1, \ldots, u_r)\\
\wedge\ & F_1(Z, (u_1, \ldots, u_r)) = Z(u_1, \ldots, u_r) \Leftrightarrow I_1(\widetilde{D}, u_1, \ldots, u_r)\\
&\vee F_2(Z, (u_1, \ldots, u_r)) = Z(u_1, \ldots, u_r) \Leftrightarrow \begin{cases} \quad \neg(I_1(\widetilde{D}, u_1, \ldots, u_r))\\ \wedge \quad I_2(\widetilde{D}, u_1, \ldots, u_r)\end{cases}\\
&\vdots\\
&\vee F_{k-1}(Z, (u_1, \ldots, u_r)) = Z(u_1, \ldots, u_r) \Leftrightarrow \begin{cases} \quad \neg(I_1(\widetilde{D}, u_1, \ldots, u_r))\\ \vdots \\ \wedge \neg(I_{k-2}(\widetilde{D}, u_1, \ldots, u_r))\\ \wedge \quad I_{k-1}(\widetilde{D}, u_1, \ldots, u_r)\end{cases}\\
&\vee F_k(Z, (u_1, \ldots, u_r)) = Z(u_1, \ldots, u_r) \Leftrightarrow \begin{cases} \quad \neg(I_1(\widetilde{D}, u_1, \ldots, u_r))\\ \vdots \\ \wedge \neg(I_{k-2}(\widetilde{D}, u_1, \ldots, u_r))\\ \wedge \neg(I_{k-1}(\widetilde{D}, u_1, \ldots, u_r))\end{cases}\\
\wedge\ & \phi.
\end{aligned}$$

□

Proof of Theorem 2

Let $S \in \mathrm{NP}_{\mathcal{K}}$ be a problem of ordered $\mathcal{K}$-structures of signature $\sigma = (L_s, L_f)$. By definition of $\mathrm{NP}_{\mathcal{K}}$, there exists $r \in \mathbb{N}$, a function symbol Y of arity r not in L_f, and a decision problem H of ordered $\mathcal{K}$-structures of signature $(L_s, L_f \cup Y)$, such that H belongs to $\mathrm{P}_{\mathcal{K}}$ and

$$S = \{\mathcal{D} \in Struct(\sigma) | \exists Y \ (\mathcal{D}, Y) \in H\}.$$

By Theorem 1, there exists a fixed point first-order formula ϕ that describes H, and thus,

$$\mathcal{D} \in S \text{ if and only if } \mathcal{D} \models \exists Y \phi.$$

By Proposition 1, ϕ can be replaced by an equivalent second-order formula $\exists Z \varphi$ with φ a first-order formula. Then,

$$\mathcal{D} \in S \text{ if and only if } \mathcal{D} \models \exists Y \exists Z \varphi,$$

which shows (**i**) $\Rightarrow$ (**ii**). For the other direction, it suffices to guess an interpretation for the second-order quantified functions, and to check in $\mathrm{P}_{\mathcal{K}}$ that the first-order formula induced is satisfied. □

Fuzzy Interval-valued Processes Algebra[⋆]

Yixiang Chen and Jie Zhou

School of Mathematics, Physics and Informatics
Shanghai Normal University
Shanghai 200234, People's Republic of China

Abstract. In this paper, the authors propose a new model called as fuzzy interval-valued CCS (FICCS for short) to solve the problem of nondeterministic choices arises in concurrency and communication systems. This model is designed in such a way that an interval value is assigned to the action of a prefix and an interval value distribution to components of a parallel composition and a summation based on the meta logic of fuzzy interval logic. Following the study of standard CCS, the authors give the syntax of FICCS and its two versions of operational semantics due to existence of two kinds of order-relations between interval truth values in fuzzy interval logic. We also introduce four fuzzy interval-valued CCS (FI_4CCS, for short). Then, we consider the notion of quantization of processes from FICCS into FI_4CCS. A congruence theorem has been given indicating that this quantization transfers the operational semantics from FICCS into FI_4CCS.

1 Introduction

As what Hoare stated in the preface of Milner's book named with " Communication and Concurrency" [MIL89]. Milner's CCS (Calculus of Communicating Systems)[MIL89] is the model of representing not only interactive concurrent systems, such as communications protocols, but also much of what is familiar in traditional computation e.g. data structures and storage regimes. A similar model, CSP (Communicating Sequential Processes) was independently received by Hoare[HOA85]. Both models have inspired a school of researchers throughout the world, who have contributed to its refinement, development and application.

Among them, probabilistic CCS is one of important developments. One of its feature is that each command of a process summation expression is guarded by a probability and the sum of these probabilities is 1, i.e., it obeys the stochastic condition [GSST]. For example, summation process $p_1P_1+p_2P_2$, where P_1 and P_2 are sub-processes, however, p_1 and p_2 are non-negative real numbers satisfying the stochastic condition $p_1+p_2=1$ and being designed to sub-processes P_1 and P_2 as the probability of execution respectively. This model is called as stochastic

⋆ This work was partly supported by National Key Research and Development Program (Grant NO. 2002CB312004), National Foundation of Natural Sciences of China (Grant No: 60273052), and the key Project of Educational Commission of Shanghai (Grant No: 02DZ46).

probabilistic CCS. Recently, Ying introduced a new model of probabilistic processes, called additive probabilistic process algebra (APPA, for short) [YING02]. In APPA, the probability satisfies the sub-stochastic condition, i.e., the sum of probabilities is less than or equal to 1. APPA is called as sub-stochastic probabilistic CCS. In whether stochastic or sub-stochastic probabilistic CCS, the probabilities have the same feature, which is that if subprocess P is executed with the probability p then it is not executed with the probability $1-p$. In other words, for each subprocess in a summation process, the sum of its probabilities of execution and non-execution is just 1.

One can consider about other tow cases. One is that the sum is less than 1, and the other that the sum is larger than 1. The first case depicts that there are some things that are unknown, since the sum of probabilities of execution and non-execution for a process is less than 1. The second one says that there are some things that are contradictive due to the sum of probabilities being larger than 1. Both cases can appear in a distributed systems.

In this paper, we employ the interval value $[a, b]$ in fuzzy interval logic to represent them in such a way that a is the degree (instead of the probability) of subprocess's execution, and $1 - b$ the degree of its non-execution. A little more general, one proposed the notion of interval truth value, which is

$$[a, b] \quad (a, b \in [0, 1]),$$

where a can be interpreted as a degree of affirmative assertion for a given statement, $1 - b$ as a degree of negative assertion for the same statement.

The purpose of this paper is to propose a new model to deal with nondeterministic choice, based on the interval truth value and the fuzzy interval logic, that we refer to as fuzzy interval-valued process algebra (FICCS, for short). FICCS is an interval extension of CCS with interval values instead of probabilities. It borrows some ideas of probabilistic CCS's. Its underlying (meta) logic is the fuzzy interval logic proposed by Mukaidono and Kikuchi [MUKK90]. So, our FICCS model is completely distinct from the probabilistic model.

As well-known, stochastic and vagueness are at least two forms of nondeterminism and imprecision's appearance in the real world. The toss of a fair coin is a classic stochastic example, and "young" and "old" are well-known concepts of vagueness. The situation in the computer world is similar, e.g., stochastic processes, and imprecise inputs. As what we have pointed above, in order to deal with stochastic phenomena, one uses methods from probability theory and statistics. Probabilistic processing is one of resolving stochastic phenomena in the computer world [Jones [JON90], Larsen and Skou [LS94], McIver and Morgan [MM01], Seidel [SEI95], van Glabbeek et al. [GSS95], Ying [YING02]]. For the denotational semantics of probabilistic processing, one proposed probabilistic powerdomain [Alvarez-Manilla et al. [AMJK03], Jones and Plotkin [JP89],Jung and Tix [JT98],Moshier and Jung [MJ02]], which comes from the classic powerdomain. Probabilistic predicate transformers is the third method to deal with stochastic phenomena [He et al. [HSM97], Kozen [KOZ81], McIver and Morgan [MM01], Morgan et al.[MMS96], Ying [YING03]].

These fuzzy analogues, however, have not been well considered, although as earlier as in 1968, Zadeh pioneers the notion of fuzzy algorithm. In 2003, Chen and Zhou in [CZ03] proposed a kind of fuzzy processing system, called interval CCS. The interval CCS is of the additive model, which is followed from Ying's work. The most recently, Chen and Jung explored the notion of fuzzy predicate transformers [CJ04]. Chen and Plotkin proposed the notion of fuzzy healthy condition for fuzzy predicate transformers [CP04]. This paper aims at the proposal on fuzzy interval-valued process algebra, a non-additive model of CCS, based on the interval truth value in fuzzy interval logic. Its syntax and semantics are given.

The paper is organized as follows. Section 2 recalls some basic notions in fuzzy interval logic. Four particular interval truth values, $\mathbf{T} = [1, 1]$ for true, $\mathbf{F} = [0, 0]$ for false, $\mathbf{U} = [0, 1]$ for unknown and $\mathbf{C} = [1, 0]$ for contradiction, are considered. The notation $\mathbf{I_4}$ will denote the set of these four interval truth values. Two kinds of ordering relation between interval truth values, called truth order and vague order respectively, are introduced. In section 3, the authors define the syntax of FICCS based on the $\mathbf{I}$ and one of FI$_4$CCS based on the $\mathbf{I_4}$. With corresponding these two syntaxes, two kinds of operational semantics of FICCS and FI$_4$CCS are introduced. In Section 4, the authors consider about the notion of quantizers of interval truth values, which is indeed a function from $\mathbf{I}$ into $\mathbf{I_4}$ preserving the finite meets. Based on the quantizer, the authors introduce the notion of omega quantization of processes, which is a mapping from the set $\mathcal{P}$ of interval-valued processes of FICCS into the set $\mathcal{P}_4$ of four interval-valued process of FI$_4$CCS. A congruence theorem indicating that omega quantization preserves the operational semantics is given.

2 Fuzzy Interval Logic

Fuzzy interval logic was proposed by Mukaidono and Kikuchi[MUKK90]. In this fuzzy interval logic, propositions can take interval truth values, which can be viewed as special cases of linguistic truth values [ZAD75] or the general cases of numerical truth values.

2.1 Interval Truth Value

An *interval truth value* is expressed as

$$[a, b] \quad (a, b \in [0, 1]),$$

where a is interpreted as a degree of affirmative assertion for a given statement, $1 - b$ as a degree of negative assertion for the same statement.

The set consisting of all interval truth values is expressed as

$$\mathbf{I} = \{[a, b] \mid a,\ b \in [0, 1]\}.$$

Four interval truth values $[0, 0], [1, 1], [1, 0]$ and $[0, 1]$ are very special. For $[0, 0]$, the affirmative degree is 0, but the negative degree is the largest 1. So, it stands for false, which is denoted by **F**. For $[1, 1]$, the affirmative degree is 1, its negative degree, however, is 0. So, it is used to stand for truth, denoted by **T**. In the case of $[1, 0]$, both affirmative degree and negative degree are 1. So, it presents the greatest contradictive. We us it to stand for the contradiction, denoted by **C**. Finally, it is clear that both affirmative and negative degrees of $[0, 1]$ are 0. So, it says nothing. We us it to represent the unknown, denoted by **U**. The set of these four interval truth values is denoted by $\mathbf{I_4}$. That is,

$$\mathbf{I_4} = \{\mathbf{F}, \mathbf{T}, \mathbf{C}, \mathbf{U}\}.$$

Notice that a and b are not asked to satisfy the condition of a less than b. In other word, b is possibly less than a. So, there are three cases of relations between a and b: $a = b$, $a < b$ and $a > b$.

If $a = b$ then $a + (1 - b) = 1$. That means that the sum of the affirmative degree and the negative degree is 1. It is what the probabilistic case considers about. These interval truth values will be called as *centers*. $\mathbf{I}_\zeta$ will denote the set of all centers, and $\zeta, \zeta', \ldots$ will rang over centers.

If $a < b$ then $a + (1 - b) = 1 - (b - a) < 1$. That means that the sum of affirmative degree and negative degree is less than 1. So, there is some event not known. For example, if we take $a = 1/3, b = 2/3$, then both affirmative and negative degrees are $1/3$. Their sum is $2/3$. So, the remain $1/3$ is of the unknown.

If $a > b$ then $a + (1 - b) = 1 + (a - b) > 1$. So, the sum of affirmative and negative degrees is larger than 1. For example, we take $a = 2/3$ and $b = 1/4$, then $a + (1 - b) = 2/3 + 3/4 = 17/12$. Thus, $1 - (a + (1 - b)) = b - a = 5/12$. The $5/12$ is of contradiction, for it can be of the affirmation and also of the negation. So, in this case, contradiction occurs.

Another interesting case is that $a + b = 1$. Thus, $1 - b = a$. So, both degrees of affirmation and negation are a. These interval truth values will be called as *numerics*. The notation $\mathbf{I}_\nu$ will denote the set of all numerics, and $\nu, \nu', \ldots$ will rang over numerics.

We have seen that degrees of true (affirmation) and false (negation) for a statement are given independently, which is just of the feature of fuzzy logic, and completely different from the probability logic [NIL86]. But, the latter can be thought of as a special case of fuzzy interval truth value.

2.2 Truth Order and Vagueness Order

Two kinds of partial order relations $\prec$ and $\subset$ are defined in the set **I** as follows:

Definition 1. [MUK94] Let $x = [x_a, x_b]$ and $y = [y_a, y_b]$ be elements of **I**. Define the truth order relation $\prec$ as

$$y \prec x \text{ if and only if } y_a \ \leq \ x_a \text{ and } y_b \ \leq \ x_b.$$

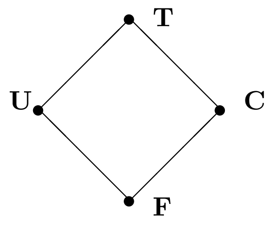
From the definition, one can find out that if x is larger than y in the truth order, then the affirmative degree of x's is larger than one of y's, the negative degree is, however, exactly oppositive. That is, the negative degree of x's is smaller than one y's. This is the reason that we call such relation as the truth order relation. Under the truth order, the smallest element is **F**, and the greatest one is **T**.

Definition 2. [MUK94] Let $x = [x_a, x_b]$ and $y = [y_a, y_b]$ be elements of **I**. Define the vague order relation $\subset$ as

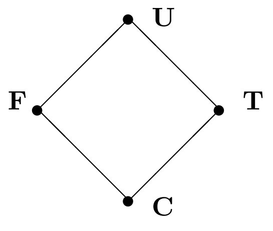
$$y \subset x \text{ if and only if } x_a \le y_a \text{ and } y_b \le x_b.$$

Notice that if the y is a subset of the x then y is the less than x under the vague relation. But, the inverse is not true in general. In fact, the contradiction truth value **C**=$[1, 0]$ is the least element under the vague order relation. Surely it is not a subset of any interval value truth except itself. The unknown truth value **U**=$[0, 1]$ is, however, the greatest one.

Both induced orders on $\mathbf{I_4}$ are shown in the following graphs.



$(\mathbf{I_4}, \prec, \vee, \wedge)$



$(\mathbf{I_4}, \subset, \cup, \cap)$

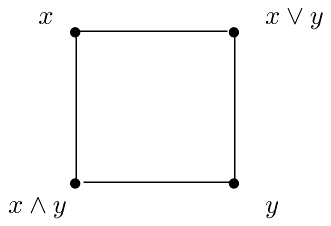
2.3 Lattice Operators

Both truth order and vague order make **I** and $\mathbf{I_4}$ be lattices. From the viewpoint of logic, however, we have logical operations such as truth OR($\vee$), truth AND($\wedge$), vague OR($\cup$), and vague AND($\cap$), which are defined as follows.

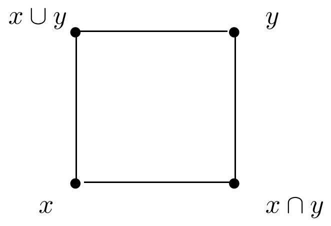
For elements $x = [x_a, x_b]$ and $y = [y_a, y_b]$ of **I**, we define

- $x \vee y = [\max(x_a, y_a), \max(x_b, y_b)]$
- $x \wedge y = [\min(x_a, y_a), \min(x_b, y_b)]$
- $x \cup y = [\min(x_a, y_a), \max(x_b, y_b)]$
- $x \cap y = [\max(x_a, y_a), \min(x_b, y_b)]$.

Both $(\mathbf{I}_4, \vee, \wedge)$ and $(\mathbf{I}_4, \cup, \cap)$ are sublattices of **I**.



$(\mathbf{I}, \prec, \vee, \wedge)$



$(\mathbf{I}, \subset, \cup, \cap)$

We will use $\sqsubset$ and $\sqcup$, $\sqcap$ to represent any one of the two partial order relations and its induced lattice operators as a general notation.

3 Syntax and operational semantics of FICCS

This section will introduce the model of fuzzy interval-valued CCS (FICCS, for short) to deal with nondeterministic choice, based on the interval truth value of **I**. FICCS is an extension of CCS with interval values, similar to, but not probabilistic CCS. The authors acknowledge, however, that probabilistic CCS leads us to the goal.

3.1 Syntax of FICCS

Our FICCS model originally comes from Milner's CCS by adding interval truth value to the combinators of prefix, composition, summation, restriction, relabelling and of constants. So, some basic notions of CCS such as actions etc. are needed. Now we present the syntax of FICCS specifically.

Let Δ be a set, whose elements are called names of actions, $\overline{\Delta}$=$\{\overline{a} : a \in \Delta\}$ the set of co-names of actions, and $\Gamma = \Delta \cup \overline{\Delta}$, whose elements are called as labels. For any $a \in \Delta$, we take $\overline{\overline{a}} = a$. In addition, let τ stand for the silent action. Then, $\mathcal{ACT} = \Gamma \cup \{\tau\}$ will be the set of actions. A relabelling function f is a function from $\mathcal{ACT}$ to $\mathcal{ACT}$ satisfying $f(\overline{l}) = \overline{f(l)}$ and $f(\tau) = \tau$.

We use $\mathcal{X}$ to denote the set of interval-valued process variables, and $\mathcal{K}$ to denote the set of interval-valued process constants; we let $X, Y, \ldots$ rang over $\mathcal{X}$, and $A, B, \ldots$ over $\mathcal{K}$.

The syntax of FICCS is given by

$$E ::= \ X \mid A \mid \alpha_x.E \mid \sum_{i\in I} x_i E_i \mid \boldsymbol{x}(E_1 \mid \ldots \mid E_n) \mid E\backslash L \mid E[f].$$

In the syntax of FICCS, $X \in \mathcal{X}$, $A \in \mathcal{K}$, $\alpha \in \mathcal{ACT}$. $L \subseteq \Gamma$, and $f : \Gamma \to \Gamma$ is a relabelling function. I is a finite indexing set. x, x_i are interval truth values, and $\boldsymbol{x}$ is a vector of interval truth values.

Clearly, if one removed the x, x_i and $\boldsymbol{x}$ from the interval-valued processes, then it would be the syntax of standard CCS.

The set of all interval-valued process expressions E is denoted by Ψ, and the set of all interval-valued processes, i.e., interval-valued process expressions that contain no variables at all, is denoted by $\mathcal{P}$. Milner also called these processes agents in [MIL89].

In this paper, we assume that the interval-valued process constants are only defined by using the equation $A \overset{\text{def}}{=} P$, where P is an interval-valued process. In other words, the interval-valued process constant is declared by an interval-valued process.

The prefix $\alpha_x.E$ performs the action α with interval value x.

In the interval-valued parallel composition $\boldsymbol{x}(E_1| \ldots |E_n)$,

$$\boldsymbol{x} = (x_1, \ldots, x_n, x_{12}, \ldots, x_{1n}, x_{23}, \ldots, x_{2n}, \ldots, x_{n-1,n})$$

x_i stands for the interval-value with which E_i behaves independently leaving other components undisturbed for each $i = 1, \ldots, n$, and x_{ij} expresses the interval-value with which E_i communicates with E_j for any $i, j \leq n$ with $i < j$.

In the interval-valued summation $\sum_{i\in I} x_i E_i$, $x_i \in \mathbf{I}$ is the interval-value with which the summation process chooses E_i among $\{E_j : j \in I\}$ for each $i \in I$. E_i is a sub-process expression of the summation E. For instance,

$$P = [0.1, 0.8]\alpha_{\mathbf{T}}.P_1 + \mathbf{U}P_2 + \mathbf{T}\beta_{\mathbf{U}}.P_3,$$

where $\alpha_{\mathbf{T}}.P_1, P_2$ and $\beta_{\mathbf{U}}.P_3$ are sub-processes of the summation process P, and $[0.1, 0.8], \mathbf{U}$ and $\mathbf{T}$ are interval values of these sub-processes respectively.

In the sequel, all interval-valued summations are supposed to be finite. We define

$$\mathbf{0} \overset{\text{def}}{=} \textstyle\sum_{i\in\emptyset} x_i E_i$$

It will follow from the operational semantics given below that $\mathbf{0}$ is the inactive process having no transitions. For each $K \in \mathcal{K}$, we assume that there is a defining equation $K \overset{\text{def}}{=} P$ of K, where $P \in \mathcal{P}$. The operational semantics (or more general, behaviors) of an interval-valued constant is completely determined by its defining equation.

We have defined the syntax of FICCS, and if we substitute all the appearance of interval values by those elements of $\mathbf{I_4}$ in the definition above, we get a new system, namely FI$_4$CCS, which apparently satisfies FI$_4$CCS $\subseteq$ FICCS. We denote the set of four interval-valued process expressions with Ψ_4 and the set of four interval-valued processes with $\mathcal{P}_4$.

3.2 Operational Semantics of FICCS

The operational semantics of FICCS is given by an interval-valued transition system below.

Definition 3. An $\mathbf{I}$-transition system (over $\mathcal{ACT}$) is an ordered pair (S, T) in which
(1) S is a set of states, and
(2) $T \subseteq S \times \mathcal{ACT} \times \mathbf{I} \times S$ is a set of transitions such that
(3) for any $s, s' \in S$, $\alpha \in \mathcal{ACT}$, and $x_1, x_2 \in \mathbf{I}$, $s \xrightarrow{\alpha_{x_1}} s'$ and $s \xrightarrow{\alpha_{x_2}} s'$ imply $x_1 = x_2$, where we write $s \xrightarrow{\alpha_x} s'$ for $(s, \alpha, x, s') \in T$.

Intuitively, $\mathbf{I}$-transition system $s \xrightarrow{\alpha_x} s'$ means that s performs action α with interval value x and then behaves like s'. Similarly, we can define $\mathbf{I_4}$-transition systems.

With the auxiliary definition above, we may render the operational semantics of FICCS now. The semantics is firstly given by the $\mathbf{I}$-transition system $(\Psi, \longrightarrow)$, where $\longrightarrow$ is defined by the following inference rules, in which the names **Act**,

Sum, **Com**, **Res**, **Rel** and **Con** indicate that the rules are associated respectively with Prefix, Summation, Composition, Restriction, Relabelling and with Constants.

The Operational Semantics of FICCS

Act

$$\frac{}{\alpha_x.E \xrightarrow{\alpha_x} E}$$

Sum

$$\frac{E_j \xrightarrow{\alpha_y} E'}{\sum_{i\in I} x_i E_i \xrightarrow{\alpha_{y\sqcap x_j}} E'}$$

Com$_i$

$$\frac{E_i \xrightarrow{\alpha_y} E'_i}{\boldsymbol{x}(E_1|\dots|E_n) \xrightarrow{\alpha_{y\sqcap x_i}} \boldsymbol{x}(E_1|\dots|E_{i-1}|E'_i|E_{i+1}|\dots|E_n)}$$

Com$_{ij}$

$$\frac{E_i \xrightarrow{\alpha_x} E'_i, \quad E_j \xrightarrow{\overline{\alpha}_y} P'_j}{\boldsymbol{x}(E_1|\dots|E_n) \xrightarrow{\tau_{x\sqcap y\sqcap x_{ij}}} \boldsymbol{x}(E_1 \mid \dots \mid E_{i-1} \mid E'_i \mid E_{i+1} \mid \dots \mid E_{j-1} \mid E'_j \mid E_{j+1} \mid \dots \mid E_n)}$$

where $1 \leq i < j \leq n$.

Res

$$\frac{E \xrightarrow{\alpha_x} E'}{E\backslash L \xrightarrow{\alpha_x} E'\backslash L}(\alpha,\ \overline{\alpha} \notin L)$$

Rel

$$\frac{E \xrightarrow{\alpha_x} E'}{E[f] \xrightarrow{f(\alpha)_x} E'[f]}$$

Con

$$\frac{P \xrightarrow{\alpha_x} P'}{A \xrightarrow{\alpha_x} P'}$$

where $A \overset{\text{def}}{=} P$ is the defining equation of A.

Actually, we may render two versions of operational semantics with respect to the known two kinds of operators, one depends on the truth order relation $\prec$, in which these operator are $\vee$ and $\wedge$, and the other on the vague order relation $\subset$, whose operators are $\cup$ and $\cap$.

That means, if one uses the first kind of operators $\vee$ and $\wedge$ to replace notations $\sqcup$ and $\sqcap$ in the syntax of FICCS respectively, then one gets the first version of FICCS, called $\mathcal{TOS}$. On the other hand, if the second kind of operators $\cup$ and $\cap$ replace these notations $\sqcup$ and $\sqcap$ respectively, then the second version of FICCS is obtained. This version is called $\mathcal{VOS}$. These two versions are distinct semantically.

Example 1. The process

$$E' = [0.2, 0.5]\alpha_{[0.4,0.6]}.E + [0.7, 0.4]\alpha_{[0.2,0.8]}.E + [0.7, 0.7]\beta_{[0.1,0.2]}.\gamma_{\mathbf{T}}.E$$

has the following transition:

$$E' \xrightarrow{\alpha_x} E.$$

Due to the immediate derivation of $\alpha_{[0.4,0.6]}.E \xrightarrow{\alpha_{[0.4,0.6]}} E$ or $\alpha_{[0.2,0.8]}.E \xrightarrow{\alpha_{[0.2,0.8]}} E$ respectively in the system of $\mathcal{TOS}$, we get that $x = [0.2, 0.5] \wedge [0.4, 0.6] = [0.2, 0.5]$, or, $x = [0.7, 0.4] \wedge [0.2, 0.8] = [0.2, 0.4]$. In the system of $\mathcal{VOS}$, however, we have that $x = [0.2, 0.5] \cap [0.4, 0.6] = [0.4, 0.5]$, or, $x = [0.7, 0.4] \cap [0.2, 0.8] = [0.7, 0.4]$.

E' also has the following transitions.

$$E' \xrightarrow{\beta_y} \gamma_{\mathbf{T}}.E \xrightarrow{\gamma_{\mathbf{T}}} E$$

where $y = [0.7, 0.7] \wedge [0.1, 0.2] = [0.1, 0.2]$ in $\mathcal{TOS}$ and $y = [0.7, 0.7] \cap [0.1, 0.2] = [0.7, 0.2]$ in $\mathcal{VOS}$.

Note that, if one restricts the interval truth values in the $\mathbf{I}_4$, then one can get the operational semantics of FI$_4$CCS and the four interval-valued transition system $(\Psi_4, \ \longrightarrow_4)$. Naturally, we have $(\Psi_4, \longrightarrow_4) \subseteq (\Psi, \longrightarrow)$.

4 Quantization

In this section, we consider about the transition from $\mathbf{I}$ into $\mathbf{I_4}$. We then consider about the transition of processes induced by this transition of interval truth values. Both generate together the congruence theorem.

We have noticed that interval truth value sets considered have two versions: $\mathbf{I}$ and its subset $\mathbf{I_4}$. The latter is a sublattice of the previous one. So, one can consider one kind of homomorphisms from $\mathbf{I}$ into $\mathbf{I_4}$. Based on the operational semantics, we pay attention to the meet-homomorphisms, i.e., functions preserving finite meets of $\mathbf{I}$.

4.1 Quantizers of Interval Truth Values

Definition 4. A truth-quantizer from $\mathbf{I}$ into $\mathbf{I_4}$ is a function preserving finite meets with respect to the truth order relation $\prec$. Similarly, a vague-quantizer from $\mathbf{I}$ into $\mathbf{I_4}$ is a function preserving finite meets with respect to the vague order relation $\subset$. A quantizer will mean a truth-quantizer or a vague-quantizer if no confusion occurs. ω, ω' will rang over the set of all quantizers.

Let $\boldsymbol{x} = (x_1, \ldots, x_n)$ be a vector of interval truth values. We define $\omega(\boldsymbol{x}) = (\omega(x_1), \ldots, \omega(x_n))$.

In the next, we will show that each center ζ induces a vague-quantizer and a numeric ν does a truth-quantizer. Thus, we have a family of such quantizers.

Now, let us take any center $\zeta = (a, b) \in \mathbf{I}_\zeta$. Then it divides the set $\mathbf{I}$ into four parts: $\Box_\zeta\mathbf{U}$, $\Box_\zeta\mathbf{C}$, $\sqsubset\!\sqsupset_\zeta\mathbf{F}$ and $\sqsubset\!\sqsupset_\zeta\mathbf{T}$, where

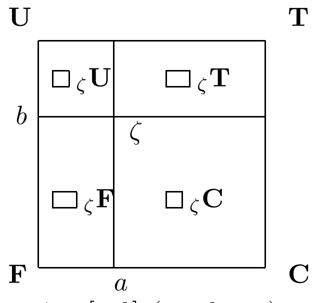
– $\Box_\zeta\mathbf{U} = \{x = (x_a, x_b) \in \mathbf{I} \mid x_a \leq a, b \leq x_b\}$
– $\Box_\zeta\mathbf{C} = \{x = (x_a, x_b) \in \mathbf{I} \mid a \leq x_a, x_b \leq b \text{ but } x \neq (a, b)\}$
– $\sqsubset\!\sqsupset_\zeta\mathbf{F} = \{x = (x_a, x_b) \in \mathbf{I} \mid x_a < a, x_b < b\}$ and
– $\sqsubset\!\sqsupset_\zeta\mathbf{T} = \{x = (x_a, x_b) \in \mathbf{I} \mid a < x_a, b < x_b\}$.

Hence, $\mathbf{I} = \Box_\zeta\mathbf{U} \cup \Box_\zeta\mathbf{C} \cup \Box_\zeta\mathbf{T} \cup \Box_\zeta\mathbf{F}$.

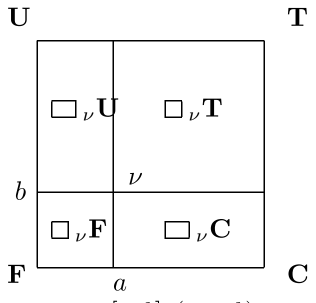
Similarly, for a numeric $\nu \in \mathbf{I}_\nu$, one can define $\Box_\nu\mathbf{T}, \Box_\nu\mathbf{F}, \Box_\nu\mathbf{U}$ and $\Box_\nu\mathbf{C}$. They are sets below, respectively.

- $\Box_\nu\mathbf{T} = \{x = (x_a, x_b) \in \mathbf{I} \mid a \leq x_a, b \leq x_b\}$
- $\Box_\nu\mathbf{F} = \{x = (x_a, x_b) \in \mathbf{I} \mid x_a \leq a, x_b \leq b \text{ but } x \neq (a, b)\}$
- $\Box_\nu\mathbf{U} = \{x = (x_a, x_b) \in \mathbf{I} \mid x_a < a, b < x_b\}$ and
- $\Box_\nu\mathbf{C} = \{x = (x_a, x_b) \in \mathbf{I} \mid a < x_a, x_b < b\}$.

We also have that $\mathbf{I} = \Box_\nu\mathbf{T} \cup \Box_\nu\mathbf{F} \cup \Box_\nu\mathbf{U} \cup \Box_\nu\mathbf{C}$.



$\zeta = [a, b], (a + b = 1)$



$\nu = [a, b], (a = b)$

Based on the division of $\mathbf{I}$ by a center ζ and a numeric ν, we define two kinds of quantizers from $\mathbf{I}$ into $\mathbf{I_4}$. The specific definitions are below.

Definition 5. For a given center ζ and numeric ν, we define functions, denoted still by ζ and ν from $\mathbf{I}$ into $\mathbf{I_4}$ respectively by

$$\zeta(x) = \begin{cases} \mathbf{U}, & \text{if } x \in \Box_\zeta\mathbf{U} \\ \mathbf{C}, & \text{if } x \in \Box_\zeta\mathbf{C} \\ \mathbf{T}, & \text{if } x \in \Box_\zeta\mathbf{T} \\ \mathbf{F}, & \text{if } x \in \Box_\zeta\mathbf{F} \end{cases} \quad \nu(x) = \begin{cases} \mathbf{T}, & \text{if } x \in \Box_\nu\mathbf{T} \\ \mathbf{F}, & \text{if } x \in \Box_\nu\mathbf{F} \\ \mathbf{U}, & \text{if } x \in \Box_\nu\mathbf{U} \\ \mathbf{C}, & \text{if } x \in \Box_\nu\mathbf{C}. \end{cases}$$

Proposition 1. *For a given center ζ, the transition function ζ is a vague-quantizer. Simultaneously, for a given numeric ν, the transition function ν is a truth-quantizer.*

4.2 Omega Quantization of Processes

In this paragraph, we pay attention to setting up an omega quantization, denoted as q_ω, of processes from $\mathcal{P}$ into $\mathcal{P}_4$ induced by a quantizer ω from $\mathbf{I}$ into $\mathbf{I_4}$. A congruence theorem will be given, which shows a reasoning rule as follows.

$$\frac{P \xrightarrow{\alpha_x} P'}{q_\omega(P) \xrightarrow{\alpha_{\omega(x)}} q_\omega(P').}$$

Definition 6. For a given quantizer ω and any interval-valued process $P \in \mathcal{P}$, the ω-quantization of P is inductively defined as follows:

1. If $P = \mathbf{0}$, then $q_\omega(P) = \mathbf{0}$.
2. If P is an interval-valued constant A, then $P = A$ is defined by the equation of $A \overset{\text{def}}{=} Q$, where Q is an interval-valued process, and then $q_\omega(P) = q_\omega(A) \overset{\text{def}}{=} q_\omega(Q)$.
3. If $P = \alpha_x.Q$, then $q_\omega(P) = \alpha_{\omega(x)}.q_\omega(Q)$.
4. If $P = \sum_{i\in I} x_i P_i$, then $q_\omega(P) = \sum_{i\in I} \omega(x_i) q_\omega(P_i)$.
5. If $P = \boldsymbol{x}(P_1|\dots|P_n)$, then $q_\omega(P) = \omega(\boldsymbol{x})(q_\omega(P_1)|\cdots|q_\omega(P_n))$.
6. If $P = Q\backslash L$, then $q_\omega(P) = q_\omega(Q)\backslash L$, and
7. If $P = Q[f]$, then $q_\omega(P) = q_\omega(Q)[f]$.

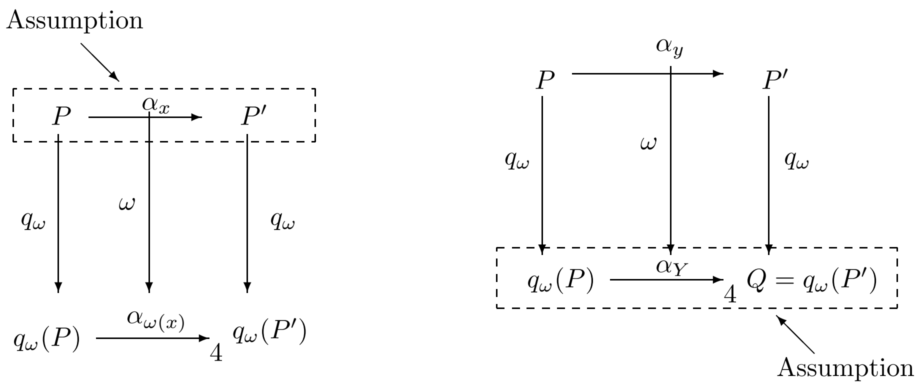
The operation of omega quantization, indeed, induces a map from $\mathcal{P}$, the set of all interval-valued processes, into $\mathcal{P}_4$, the set of all four value processes for a given quantizer ω from $\mathbf{I}$ into $\mathbf{T_4}$. This ω quantization of processes is denoted as q_ω. The following theorem shows that both quantizer ω and its induced omega quantization generate the congruence theorem.

Theorem 1. *For any pair of interval-valued processes P and P' and a given quantizer ω, $P \xrightarrow{\alpha_x} P'$ implies $q_\omega(P) \xrightarrow{\alpha_{\omega(x)}}_4 q_\omega(P')$.*

Proof. This proof is done by using structural induction on the process P and the transition rules from **Act** through **Con**.

Theorem 2. *Suppose that P is an interval-valued process in $\mathcal{P}$, Q is a four interval-valued process in $\mathcal{P}_4$, $Y \in \mathbf{I_4}$, and ω is a quantizer function from $\mathbf{I}$ into $\mathbf{I_4}$. Then, $q_\omega(P) \xrightarrow{\alpha_Y}_4 Q$ implies that there exist $P' \in \Psi$ and $y \in \mathbf{I}$ such that $P \xrightarrow{\alpha_y} P'$, $q_\omega(P') = Q$, and $Y = \omega(y)$.*

Proof. We also can render the proof by induction on the construction of P.



Pictures of Theorem 1 and 2

5 Conclusion and Future Work

This paper has introduced the new model of process calculus called fuzzy interval-valued process algebra (FICCS, shortly) to resolve nondeterminism that arises in concurrent and communication systems based on the fuzzy interval value logic. This model is an extension of standard CCS of Milner's. In FICCS, an interval truth value is assigned to either process or action and the interval truth valued vector to a summation process or a composition process, which are similar to the way Ying tackles with the additive probabilistic process algebra model [YING02]. When interval truth values become numerics, these interval-valued processes are those which probabilistic CCS considers about. But, they are complete distinct due to the distinct logic used. In our model, the logic is fuzzy interval logic. In probabilistic model, however, the logic is the probabilistic logic. As a result, the operators involved are very distinct too. In FICCS, the operators are lattice operators such as the meet and union. In probabilistic model, however, they are numeric operators such as the multiplication and plus.

This paper has established the syntax of FICCS and its two versions of operational semantics due to existence of two kinds of order-relations between interval truth values in fuzzy interval logic. For special four interval truth values **F**, **T**, **C** and **U** representing for false, true, contradiction and unknown respectively, a corresponding four fuzzy interval-valued CCS (FI_4CCS, for short) has been established. Then, this paper mainly pays attention to quantization of processes from FICCS into FI_4CCS. A congruence theorem has been given indicating that this quantization transfers the operational semantics from FICCS into FI_4CCS.

This paper is just at the beginning of study of the fuzzy process algebra. The future work will be to fulfil the theory of fuzzy process algebra and its application in the practice of computer technology.

References

[AMJK03] M. Alvarez-Manilla, A. Jung and K. Keimel. The probabilistic powerdomain for stably compact spaces. 2003.

[BH97] C.Baier & H.Hermanns. Weak bisimulation for fully probabilistic processes. Techn.Bericht IMMD-VII/1-97, Universität Erlangen,1997.

[BK97] C.Baier & M.Kwiatkowska. Domain equations for probabilistic processes. Technical Report CSR-97-7, University of Birmingham,1997.

[CHEN95] Yixiang Chen. Algebraic structures of interval truth value logic. *Journal of Engineering Mathematics,* vol.12.No.1, pp.123-126,1997.

[CJ04] Yixiang Chen and Achim Jung. An introduction to fuzzy predicate transformers. Invited talk on the third International Symposium on Domain Theory, May 10-14, 2004, Xi'an China.

[CP04] Yixiang Chen and Gordon Plotkin. Healthy Fuzzy Predicate Transformers. (Preparing).

[CZ03] Yi-Xiang Chen and Jie zhou. Interval - value CCS. In Yin-Ming Liu et al, editors, *The Proceeding of Conference on Fuzzy Procceesing Theories and Applicatoin*, Tsinghua University Press and Springer Press, pages 101-104, 2003.

[GSST] R.J.van Glabbeek, S.A.Smolka, B.Steffen and C.M.N.Tofts. Reactive, Generative and stratified models of probabilistic processes. In: *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science, Philadelphia,* PA, pp.130-141, 1990.

[GSS95] R.J. van Glabbeek, S.A. Smolka, B.Steffen. Reactive, generative, and stratified models of processes.*Information and Computation*, 121:59-80, 1995.

[HSM97] J.He, K.Seidel, A.K. MCIver. Probabilistic models for the guarded command language. *Science of Computer Programming,* 28(171-192), 1997.

[HOA85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall,1985.

[JON90] C. Jones. Probabilistic non-determinism. PhD thesis, Unicersity of Edinburgh, Edinburgh, 1990. Also published as Techniccal report No. CST-63-90.

[JP89] C. Jones and G.Plotkin. A probabilistic powerdomain of evalutions. In *Proceedings of the 4th Annual Symposium on Logic in Computer Science,* pages 186-195. IEEE Computer Society Press, 1989.

[JUN03] A. Jung. Stable compact spaces and the prbabilistic powerspace construction. ENTCS, 87.2003.

[JT98] A. Jung and R.Tix. The troublesome probabilistic powerdomain. in A. Edalat, A.Jung, K.Keimel, and M.Kwiatkowska, editors, *Proceedings of the third Workshop on Computation and Approximation*, Volme 13 of *Electronic Notes in Theoretical Computer Science.* Elsevier Science Publishers B.V., 1998. 23 pages.

[KOZ81] D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Science,* 22(328-350), 1981.

[LS94] K.G. Larsen, A.Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1-28, 1991.

[MM01] A.K. McIver, C. Morgan. Partial correctness for probabilistic demonic programs. *Theoretical Computer Science ,* 266(513-541), 2001.

[MIL89] R.Milner. *Communication and Concurrency.* International Series in Computer Science. Prentice Hall, 1989.

[MIL99] R.Milner. *Communicating and Mobile Systems: the $\pi-$Calculus.* Cambridge University Press, 1999.

[MIS00] M.Mislove. Nondeterminism and probabilistic choices: obeying the laws. C. Palamidessi (Ed.): CONCUR 2000 - Concurrency Theory 11th International Conference, University Park, PA, USA. Proceedings.LNCS 1877,2000.

[MMS96] C. Morgan, A.McIver, K. Seidel. Probsbilistic predicate transformers.*ACM Trans. Programmiing Languages and Systems,* 18(325-353), 1996.

[MOR97] J.M. Morris. Non-deterministic expressions and predicate transformers. *Information Processing Letters*, 61: 241-246, 1997.

[MJ02] M.A. Moshier and A. Jung. A logic for probabilities in semantics. In Julian Bradfield, editors, *Computer Science Logic*, volume 2471 of *Lecture Notes in Computer Science,* pages 216-231. Springer Verlag, 2002.

[MUK92] Masao Mukaidono. Interval logic and its extension. *IEEE International Conference on Fuzzy Systems,* San Diego, pp.579-586,1992.

[MUK94] M.Mukaidono. Algebraic structures of truth values in fuzzy logic. In *Fuzzy Logic and Fuzzy Control.* Lecture Notes in Artificial Intelligence 833, Springer-Verlag, pp.15-21.

[MUKK90] M.Mukaidono & H.Kikuchi. A proposal on fuzzy interval logic. *Journal of Japan Society for Fuzzy Theory and Systems,* vol.2.No.2, pp.97-110,1990.

[NIL86] N.Nilsson. Probabilistic logic. *Artificial Intelligence*, Vol.28, pp.71-88,1986.

[PlO80] G.D. Plotkin. Dijkstra's predicate transformers and Smyth's powerdomains. In D. Bjørner, editor, *Abstract Software Specifications*, volume 86 of *Lecture Notes in Computer Science*, 1980, pages 527-553.

[PLO81] G.D.Plotkin. A structured approach to operational semantics. Report DAIMI FN-19, Aarhus University, 1981.

[SEI95] K. Seidel. Probabilistic communicating processes. *Theoretical Computer Science*, 152:219-249, 1995.

[SMY83] M.B. Smyth. Power domains and predocate transformers: a topological view. *Lecture Notes in Computer Science 154,* 1983, pp.662-675.

[YING02] Mingsheng Ying. Additive models of probabilistic processes. *Theoretical Computer Science*, 275 (2002) 481-519.

[YING03] M.Ying. Reasoning about probabilistic sequential programs in a probabilistic logic. *Acta Informatica*,39:315-389, 2003.

[ZAD75] L.A.Zadeh. The concept of a linguistic variable and its application to approximate reasoning, parts 1-3. *Information Science*, vol.8, pp.199-249, 301-357, vol.9, pp.43-80.

[ZAD68] L.A. Zadeh. Fuzzy algorithms. *Information and Control,* 12(2):94-102, 1968.

[ZAD72] L.A. Zadeh. On fuzzy algorithms. *Library of Congress Cataloging-in-Publication*: *Fuzzy sets, fuzzy logic, and fuzzy systems:* selected papers by L.A. Zadeh; editors: G. J. Klir, and B. Yuan., pages: 127-147. World Scientific Publishing Co Pte Ltd, 1996.

Computable Analysis in Systems and Control

Pieter Collins

Centrum voor Wiskunde en Informatica
Amsterdam, The Netherlands
`http://homepages.cwi.nl/~collins/`
`Pieter.Collins@cwi.nl`

A formal theory of computation of real-valued and set-valued functions is important for the analysis and verification of nonlinear systems. In this talk, I will discuss computable analysis and topology for (semi)continuous multivalued maps, and show how this can be used to study the computability of reachable, viable and invariant sets.

Polynomial complexity for analog computation

Jose Felix Costa[1] and Jerzy Mycka[2]

[1] Department of Mathematics, I.S.T.,
Universidade Tecnica de Lisboa, Lisboa, Portugal
`http://fgc.math.ist.utl.pt/jfc.htm`
`fgc@math.ist.utl.pt`
[2] Institute of Mathematics,
University of Maria Curie-Sklodowska,
Lublin, Poland

The theory of analog computation, where the internal states of a computer are continuous rather than discrete, has enjoyed a recent resurgence of interest. We have been working towards recursive definitions of computational classes of functions over reals. Let us recall the concept of a real recursive function, and the corresponding class REC($\mathbb{R}$). The class REC($\mathbb{R}$) of real recursive vectors is generated from the real recursive scalars 0, 1, -1, and the real recursive projections by the following operators: composition, differential recursion (i.e a solution of the Cauchy problem whenever it is of the class C^1 on the largest interval containing 0 in which a unique solution exists), infinite limits.

We will describe a function as subexponential if its Laplace transform is defined along the whole positive real axis. Now we can propose the definition of the class AnalogP, which can be interpreted as the class of real recursive functions computed with polynomial restrictions on their values and times of computation. The class of real recursive functions computable in deterministic polynomial time, designated by AnalogP, is defined as the collection of all real recursive functions such that, in any step of the construction, their components are subexponential. There are many quite interesting properties of this class. Here let us point out the most important results. Any recursive function in the domain of natural numbers computable in deterministic polynomial time has a real extension in AnalogP. A function $f\colon \mathbb{N} \to \mathbb{N}$ is said to be an admissible restriction of F if there exists a sequence (α_i) such that $F(\alpha_i) \in \mathbb{N}$ and $f(i) = F(\alpha_i)$ for all natural numbers i. Let $H\colon \mathbb{R} \to \mathbb{R}$ be a real recursive function in AnalogP such that, in each step of the construction, it grows slower than $\lambda x.\ \mathrm{e}^{\log(x)^k}$, for some k, up to a multiplicative constant. If an admissible restriction h of H exists, then h is computable in deterministic polynomial time.

The use of Laplace transform makes AnalogP meaningful, since physical measures in physical theories do have a Laplace transform, having then a controlled growth. In this sense our complexity classes have a physical meaning. This attempt allow us to consider problems of structural complexity as problems in Analysis and, in our opinion, it opens to consideration new perspectives over classical computational complexity, for example in the context of the conjecture P $\neq$ NP.

Constructive set theory with operations

Laura Crosilla[1,2] and Andrea Cantini[1]

[1] University of Florence
Florence, Italy
[2] `http://www.philos.unifi.it/persone/crosilla_eng.htm`
`Laura.Crosilla@unifi.it`

We present an extension of constructive Zermelo–Fraenkel set theory (Aczel '78) which is enriched with urelements as well as operations. The urelements represent elements of a combinatory algebra extended with natural numbers. The notion of operation is quite general and it is shown to be essentially non-extensional. By combining these features, the theory allows for a form of application of sets to sets and for the basic axioms to be expressed in a uniform way.

From the proof theoretic point of view, we single out a theory of constructive sets with operations which has the same strength as Peano arithmetic. Interestingly, this weak system still has the expressive power to represent a general theory of inductive definitions.

On Learning Conjunctions of Horn$^{\supset}$ Clauses[⋆]

Joxe Gaintzarain, Montserrat Hermo, and Marisa Navarro

Departamento de L.S.I., Facultad de Informática,
P.O. Box 649
20080-San Sebastián, Spain.

Abstract. $Horn^{\supset}$ is a logic programming language which extends usual $Horn$ clauses with intuitionistic implication. In this paper we study the learnability of the concepts that are represented by conjunctions of propositional $Horn^{\supset}$ clauses.

1 Introduction

The problem of learning efficiently an unknown boolean formula under some determined protocols has been widely studied. It is well known that, even restricted to propositional formulas, the problem is hard [3, 10] in the usual learning models. Among these, we find the *PAC model* introduced by Valiant [19] and the *prediction model* introduced by Pitt and Warmuth [18]. Both are *passive* models in the sense that the learning algorithm has no control over the labeled examples, however they can be transformed into *active* ones by adding the ability to make membership queries. Another usual model is the *exact query model* introduced by Angluin [1], where the learning algorithm can make equivalence and/or membership queries.

In the line of learning subclasses of boolean formulas, D. Angluin, M. Frazier and L. Pitt [2] presented a positive result: a polynomial-time algorithm that used equivalence and membership queries for exactly learning conjunctions of propositional $Horn$ clauses. In this field we can also find negative results showing that there are not efficient learning algorithms for a given class. Even there exist relative results which, without solving the problem, give evidence to the effect that some class is not learnable. This kind of relative result is based on the concept of prediction-preserving reduction [18, 3].

From the point of view of logic programming, different approaches for extending Horn clauses have been studied. Some of them consider to incorporate into the language a new implication symbol, $\supset$, with the aim of structuring logic programs in some blocks with local clauses [4, 7–9, 13–17]. These extensions can also be seen as a sort of inner modularity in logic programming [5].

We consider here a particular extension named $Horn^{\supset}$. This programming language has been formally studied in [9, 8, 4, 11, 17]. In [4] a natural extension of classical first order logic $\mathcal{FO}$ with the intuitionistic implication ($\supset$), named $\mathcal{FO}^{\supset}$, was presented as the underlying logic of the programming language $Horn^{\supset}$.

[⋆] This work has been partially supported by CICYT-project TIC2001-2476-C03-03

It was also proved that the "good properties" that verify $Horn$ clauses (as a programming language) with respect to its underlying logic $\mathcal{FO}$ are conserved by $Horn^{\supset}$ with respect to $\mathcal{FO}^{\supset}$. In concrete, general models can be restricted to Herbrand-like models, each program has a canonical model and the operational semantics is an effective subcalculus of a complete calculus for $\mathcal{FO}^{\supset}$ (see also [12, 11]).

Our aim is to study the learnability of propositional $Horn^{\supset}$ clauses since this class of formulas, extending propositional $Horn$ clauses, seems to be a good candidate for improving the learnability results obtained by D. Angluin, M. Frazier and L. Pitt [2].

This paper presents our first results on such study and it is organized as follows: In Section 2 the preliminary notions in the area of learning are given. In Section 3 the programming language $Horn^{\supset}$ (restricted to the propositional case) is presented. In Section 4 we show that any model-based translation from $Horn^{\supset}$ clauses to equivalent $Horn$ clauses obtains, in general, an exponential number of clauses. This result suggests us that finding a polynomial-time algorithm for learning $Horn^{\supset}$ language may be difficult. In section 5 we show that boolean formulas are prediction preserving reducible to $Horn^{\supset}$ clauses using membership queries. As a consequence, we obtain the main result of this paper: conjunctions of propositional $Horn^{\supset}$ clauses are not predictable even with membership queries under cryptographic assumptions. We conclude, in Section 6, by summarizing our results.

2 Learning preliminaries

Most of the terminology used in this section is borrowed from [3, 6]. Let S be a fixed domain and let S^* be the set of strings over S. $|w|$ denotes the length of the string w. Strings in S^* will represent both examples and concept names.

A *representation of concepts* (or *representation class*) $\mathcal{C}$ is any subset of $S^* \times S^*$. We interpret an element $\langle u, x \rangle$ of $S^* \times S^*$ as consisting of a *concept name* u and an *example* x. For instance, the representation of concepts $\mathcal{C}_{BF}$ contains pairs $\langle u, x \rangle$ where u is a boolean formula and x is any interpretation satisfying such formula. Define the *concept represented by* u as $K_{\mathcal{C}}(u) = \{x \mid \langle u, x \rangle \in \mathcal{C}\}$. The set of *concepts represented by* $\mathcal{C}$ is $\{K_{\mathcal{C}}(u) \mid u \in S^*\}$.

We use two models of learning, both of them fairly standard: Angluin's model of exact learning with queries [1] and the model of prediction with membership queries as defined by Angluin and Kharitonov [3].

2.1 The exact query learning model

Let $\mathcal{C}$ be a representation class. An *exact learning algorithm with queries* is an algorithm A that takes as input a bound s on the size of the target concept representation. It may make any number of queries or requests, the responses to which are determined by the unknown target concept c. The algorithm A must eventually halt with an output concept name v. The concept $K_{\mathcal{C}}(v)$ is interpreted as a A's guess of the target concept. The most common kinds of queries are *membership* and *equivalence* queries. A *membership query* takes a string x in S^* as input and returns 1 if $x \in c$ and 0 otherwise. An *equivalence query* takes a concept name h as input and returns 1 if $K_{\mathcal{C}}(h) = c$ and a counterexample, in the symmetric difference of c and $K_{\mathcal{C}}(h)$, otherwise.

The algorithm A runs in polynomial time if its running time (counting one step for each query) is bounded by a polynomial in s and the length of the largest counterexample. We say that A *exactly learns* a representation of concepts $\mathcal{C}$, if and only if for all positive integer s, for all concept name u with $|u| \leq s$, when A runs with input s and a target concept $c = K_{\mathcal{C}}(u)$, A outputs a concept name v such that $c = K_{\mathcal{C}}(v)$

A representation of concepts $\mathcal{C}$ is *polynomially learnable* if there is a learning algorithm A that runs in polynomial time and exactly learns $\mathcal{C}$. In this paper, we always suppose that A uses membership and equivalence queries.

2.2 The prediction model with membership queries

A *prediction with membership algorithm*, or *pwm-algorithm* is a possible randomized algorithm A that takes as input a bound s as above, a bound n on the length of examples, and an accuracy bound ϵ. It may make three different kinds of queries or requests, the responses to which are determined by the unknown target concept c and an unknown probability distribution $\mathcal{D}$ on S^n, as follows:

– A *membership query* takes a string x in S^* as input and returns 1 if $x \in c$ and 0 otherwise.

– A *request for a random classified example* takes no input and return a pair $\langle x, b\rangle$, where x is a string chosen independently according to $\mathcal{D}$ and $b = 1$ if $x \in c$ and $b = 0$ otherwise.

– A *request for an element to predict* takes no input and returns a string *sol* chosen independently according to $\mathcal{D}$.

The algorithm A may make any number of membership queries or requests for random classified examples. However, A must eventually make one and only one request for an element to predict and eventually halt with an output 1 or 0 without making any further query or request. The output is interpreted as A's guess of how the target concept classifies the element *sol*. We say that A runs in polynomial time if its running time (counting one step for each query or request) is bounded by a polynomial in s, n, and $\frac{1}{\epsilon}$.

We say that A *predicts* a representation of concepts $\mathcal{C}$ if and only if for all positive integers s and n, for all positive rational ϵ, for all concept name $u \in S^*$ with $|u| \leq s$, when A runs with inputs s, n, and ϵ, and a target concept $c = K_{\mathcal{C}}(u)$ and $\mathcal{D}$, A asks queries or requests and the probability that the output of A is not equal to the correct classification of *sol* by c is at most ϵ.

A representation of concepts $\mathcal{C}$ is *polynomially predictable with membership queries* if and only if there is an algorithm A that runs in polynomial time and predicts $\mathcal{C}$.

Lemma 1. *[3] If a representation of concepts is polynomially learnable, then it is polynomially predictable with membership queries.*

2.3 Reducibility among prediction problems

To compare the difficulty of learning problems in the prediction model we use *pwm-reducibility* as defined in [3]. It is denoted by $\leq_{pwm}$.

Definition 1. *Let $\mathcal{C}$ and $\mathcal{C}'$ be representations of concepts. Let $\top$ and $\bot$ be two different symbols not occurring in S. Then $\mathcal{C}$ is* pwm-reducible *to $\mathcal{C}'$, if and only if there exist three mappings g, f and h with the following properties:*

1. There is a nondecreasing polynomial q such that for all natural numbers s and n and for every $u \in S^*$ with $|u| \leq s$, $g(s, n, u)$ is a string u' of length at most $q(s, n, |u|)$.

2. For all natural numbers s and n, for every $u \in S^*$ with $|u| \leq s$, and for every $x \in S^*$ with $|x| \leq n$, $f(s, n, x)$ is a string x' and $x \in K_{\mathcal{C}}(u)$ if and only if $x' \in K_{\mathcal{C}'}(g(s, n, u))$.

3. For all natural numbers s and n, for every $u \in S^*$ with $|u| \leq s$, and for every $x' \in S^*$, $h(s, n, x')$ is either $\top$, $\bot$ or a string x. If $h(s, n, x') = \top$ then $x' \in K_{\mathcal{C}'}(g(s, n, u))$; if $h(s, n, x') = \bot$ then $x' \notin K_{\mathcal{C}'}(g(s, n, u))$; and otherwise $x' \in K_{\mathcal{C}'}(g(s, n, u))$ if and only if $x \in K_{\mathcal{C}}(u)$. Moreover, h is computable in time bounded by a polynomial in s, n and $|x'|$.

In the property (2), and independently, in the property (3), the expression "$x \in K_{\mathcal{C}}(u)$" can be replaced with "$x \notin K_{\mathcal{C}}(u)$", as discussed in [3]. We denote these options by properties (2)' and (3)' respectively.

The only properties of this reducibility that are needed in this paper were established in [3]:

Lemma 2. *The pwm-reduction is transitive, i.e., let $\mathcal{C}$, $\mathcal{C}'$, and $\mathcal{C}''$ be representations of concepts, if $\mathcal{C} \leq_{pwm} \mathcal{C}' \leq_{pwm} \mathcal{C}''$ then $\mathcal{C} \leq_{pwm} \mathcal{C}''$.*

Lemma 3. *Let $\mathcal{C}$ and $\mathcal{C}'$ be representations of concepts. If $\mathcal{C} \leq_{pwm} \mathcal{C}'$ and $\mathcal{C}'$ is polynomially predictable with membership queries, then $\mathcal{C}$ is also polynomially predictable with membership queries.*

3 The programming language ***Horn***$^{\supset}$

In this section we introduce the programming language *Horn*$^{\supset}$ by showing its syntax and its model semantics. Although the language is in general a first order language (see [9, 4]), in this paper we shall restrict our presentation only to this language in the propositional setting.

3.1 The syntax

The syntax is an extension of the (propositional) *Horn* clause language by adding the intuitionistic implication $\supset$ in goals and clause bodies. Let Σ be a set of propositional variables (or signature). The Σ-clauses, named D, and the Σ-goals, named G, are recursively defined as follows (where v stands for any propositional variable in Σ):

$$G ::= v \mid G_1 \wedge G_2 \mid D \supset G \qquad\qquad D ::= v \mid G \rightarrow v \mid D_1 \wedge D_2$$

A *Horn*$^{\supset}$ Σ-program is a finite set (or conjunction) of Σ-clauses. The main difference with respect to *Horn* clauses is the use of a "local" clause set D in goals of the kind $D \supset G$ (and therefore also in clause bodies).

Example 1. The following set with three clauses is a *Horn*$^{\supset}$ program over signature $\Sigma = \{a, b, c, d\}$

$$\{((b \rightarrow c) \supset c) \rightarrow a,\ b,\ ((a \wedge (b \rightarrow c)) \supset (((b \rightarrow c) \wedge (a \rightarrow d)) \supset a)) \rightarrow d\}$$

The second program clause is simply b. The first and the third program clauses are of the form $G \rightarrow v$. In the first clause, the goal G is $(b \rightarrow c) \supset c$. That is, it contains a local set with one program clause. In the third clause, the goal G is of the form $D_1 \supset (D_2 \supset G_3)$, where $D_1 = a \wedge (b \rightarrow c)$, $D_2 = (b \rightarrow c) \wedge (a \rightarrow d)$, and $G_3 = a$. D_1 and D_2 are local sets with two clauses and they can also be written $D_1 = \{a, (b \rightarrow c)\}$ and $D_2 = \{(b \rightarrow c), (a \rightarrow d)\}$ respectively.

3.2 The model semantics

Model semantics for the programming language *Horn*$^{\supset}$ is based on some Kripke structures which, in the propositional setting, can be defined as follows.

Definition 2. *Given a signature Σ, the model semantics for the propositional Horn$^{\supset}$ is given by the set of Kripke Σ-structures $\underline{\mathsf{Mod}}(\Sigma) = \{K_I \mid I \subseteq \Sigma\}$ where K_I denotes the partially ordered set of worlds $\{J \subseteq \Sigma \mid I \subseteq J\}$.*

Well-formed Σ-formulas are built, from propositional variables in Σ, using classical connectives ($\neg$, $\wedge$, $\vee$ and $\rightarrow$) and the intuitionistic implication ($\supset$). The satisfaction relation $\models_{\Sigma}$ between a Σ-structure K_I and a Σ-formula φ requires the formula φ to be forced in the minimal world I in K_I. The forcing relation is defined between (the associated interpretation to) a world and a formula.

Definition 3. *Let $K_I \in \underline{\mathsf{Mod}}(\Sigma)$ and φ a Σ-formula. We say that*

(a) *$K_I \models_{\Sigma} \varphi$ (K_I satisfies φ) iff $I \Vdash_{\Sigma} \varphi$ (φ is forced in I)*
(b) *The binary* forcing relation *$\Vdash_{\Sigma}$ (or simply $\Vdash$ if there is no confusion about the signature) is inductively defined as follows:*
$I \nVdash False$
$I \Vdash v$ iff $v \in I$ for $v \in \Sigma$
$I \Vdash \neg\varphi$ iff $I \nVdash \varphi$
$I \Vdash \varphi \wedge \psi$ iff $I \Vdash \varphi$ and $I \Vdash \psi$
$I \Vdash \varphi \vee \psi$ iff $I \Vdash \varphi$ or $I \Vdash \psi$
$I \Vdash \varphi \rightarrow \psi$ iff if $I \Vdash \varphi$ then $I \Vdash \psi$
$I \Vdash \varphi \supset \psi$ iff for all $J \subseteq \Sigma$ such that $I \subseteq J$: if $J \Vdash \varphi$ then $J \Vdash \psi$

Example 2. Let φ be the formula (in this case a goal) $((a \wedge c) \rightarrow b) \supset (c \wedge b)$. $I \Vdash \varphi$ for $I = \{a, b, c\}$, $I = \{a, c\}$ and $I = \{b, c\}$. $I \nVdash \varphi$ for $I = \{a, b\}$, $I = \{a\}$, $I = \{b\}$, $I = \{c\}$ and $I = \emptyset$. Note, for instance, that $\{a, b\} \Vdash (a \wedge c) \rightarrow b$ and $\{a, b\} \nVdash (c \wedge b)$.

This forcing relation does not behave monotonically with respect to the world ordering for general formulas. For instance, $a \to b$ is forced in the world $I = \emptyset$ but it is not forced in $J = \{a\}$. We say that a formula is *persistent* whenever the forcing relation behaves monotonically for it.

Definition 4. *A Σ-formula φ is* persistent *when for any Σ-interpretation I, if $I \Vdash \varphi$ then $J \Vdash \varphi$ for any Σ-interpretation J such that $I \subseteq J$.*

From the following proposition we obtain, in particular, that any Σ-goal G is a persistent formula.

Proposition 1. *[4] Any $v \in \Sigma$ is persistent. Any formula $\varphi \supset \psi$ is persistent. If φ and ψ are persistent then $\varphi \vee \psi$ and $\varphi \wedge \psi$ are persistent.*

Definition 5. *Two formulas φ and ψ are* semantically equivalent *if both have the same meaning in each structure in $\underline{\mathsf{Mod}}(\Sigma)$. In other words, if both are forced in the same Σ-interpretations.*

4 Trying to learn $\boldsymbol{Horn}^{\supset}$ programs with $\boldsymbol{Horn}$ programs

A way to learn a $Horn^{\supset}$ program P might consist on using the well-known algorithm of Angluin, Frazier and Pitt [2] for learning a conjunction of $Horn$ clauses equivalent to P. In this section, we show that this simple idea does not work if we are looking for a learning algorithm that runs in polynomial time. We prove that any translation from a $Horn^{\supset}$ Σ-program P to an equivalent $Horn$ Σ-program $\widehat{P}$ yields a number of clauses that is exponential in the size of P.

Definition 6. *For each $Horn^{\supset}$ Σ-clause D, let $Models(D)$ be the set $\{I \subseteq \Sigma \mid I \Vdash D\}$. Let $Min(D)$ be the set $\{I \subseteq \Sigma \mid I \nVdash D$ but $J \Vdash D$, for all $J \subset I\}$. That is, $Min(D)$ contains the "minimal" interpretations not forcing D.*

In the sequel, we intentionally consider I as a set or as a conjunction, as convenient.

Definition 7. *For each $Horn^{\supset}$ Σ-clause D, the $Horn$ Σ-program $\widehat{D}$ is defined as follows:*

For $D = v$, $\qquad \widehat{D} = \{v\}$

$$\text{For } D = G \to v, \quad \widehat{D} = \begin{cases} \emptyset, & \text{if } Models(D) = \mathcal{P}(\Sigma) \\ \bigcup_{I \in Min(D)} \{I \to v\} & \text{in other case} \end{cases}$$

In the following theorem we prove that D and $\widehat{D}$ are semantically equivalent, that is, they have the same models.

Theorem 1. *For each Σ-interpretation I and each $Horn^{\supset}$ Σ-clause D it holds: $I \Vdash \widehat{D} \iff I \Vdash D$*

Proof. For $D = v$, the theorem is trivial. Let D be $G \to v$.

($\Longrightarrow$) Let us suppose that $I \not\Vdash D$. Then $v \notin I$. Since $I \not\Vdash D$, $Min(D)$ is not empty. Therefore there exists some $J \in Min(D)$ such that $J \subseteq I$ and $J \to v$ is a clause of the program $\widehat{D}$. Then $I \Vdash J$ and $v \notin I$ imply $I \not\Vdash \widehat{D}$.

($\Longleftarrow$) Let us suppose that $I \Vdash D$. If $v \in I$ then trivially $I \Vdash \widehat{D}$. If $v \notin I$ then $I \not\Vdash G$. Let $J \to v$ be a clause in the program $\widehat{D}$ for some $J \in Min(D)$ (if $\widehat{D}$ were empty then trivially $I \Vdash \widehat{D}$). If J were a (proper) subset of I, by persistence of goals we obtain $J \not\Vdash G$. But this implies $J \Vdash D$ which contradicts $J \in Min(D)$. That is, each $J \in Min(D)$ is not a subset of I and then trivially $I \Vdash J \to v$. Therefore $I \Vdash \widehat{D}$. ■

Corollary 1. *Each Horn$^{\supset}$ Σ-program P is equivalent to a Horn Σ-program $\widehat{P}$.*

Now we are going to consider a concrete *Horn*$^{\supset}$ clause D whose $\widehat{D}$ needs to have an exponential number of clauses with respect to the symbols in D.

Lemma 4. *Let D be the following Σ-clause*

$$([(a_{11} \to b) \wedge (a_{12} \to b) \wedge \cdots \wedge (a_{1n} \to b)] \supset b \wedge$$

$$\cdots$$

$$[(a_{n1} \to b) \wedge (a_{n2} \to b) \wedge \cdots \wedge (a_{nn} \to b)] \supset b\ ) \to a$$

where $\Sigma = \{a_{ij} \mid i, j \in \{1, \ldots, n\}\} \cup \{b, a\}$. Each Σ-interpretation of the form $\{a_{1k_1}, a_{2k_2}, \ldots, a_{nk_n}\}$, with $k_j \in \{1, \ldots, n\}$ belongs to $Min(D)$.

Proof. Let I be one of such Σ-interpretations. Without loss of generality, let us suppose I to be $\{a_{11}, \ldots, a_{n1}\}$. The given clause D is $(G_1 \wedge \cdots \wedge G_n) \to a$ whereeach G_i is the goal $((a_{i1} \to b) \wedge \cdots \wedge (a_{in} \to b)) \supset b$. First let us prove that for each proper subset $J \subset I$, it holds that $J \Vdash D$. Since there exists some $a_{i1} \notin J$, then $J \Vdash (a_{i1} \to b) \wedge \cdots \wedge (a_{in} \to b)$ and$J \not\Vdash b$. Then $J \not\Vdash G_i$ and therefore $J \Vdash (G_1 \wedge \cdots \wedge G_n) \to a$. Now let us see that $I \not\Vdash D$. Since $a_{11} \in I$, for every Σ-interpretation K such that $I \subseteq K$ and $K \Vdash (a_{11} \to b) \wedge \cdots \wedge (a_{1n} \to b)$ it holds that $K \Vdash b$ and therefore $I \Vdash G_1$. Similarly, we can obtain $I \Vdash G_i$ for each $i \in \{1, \ldots, n\}$ and then, since $a \notin I$, $I \not\Vdash D$. ■

The set of *Horn* clauses $\widehat{D}$ obtained by Definition 7 from the *Horn*$^{\supset}$ program D in Lemma 4 contains at least these n^n clauses:

$$\{I \to a \mid I \text{ is } \{a_{1k_1}, a_{2k_2}, \ldots, a_{nk_n}\},\ \text{with } k_j \in \{1, \ldots, n\}\} \qquad (1)$$

The next result shows that this set of *Horn* clauses is non-redundant.

Lemma 5. *Any set of Horn clauses equivalent to (1) has at least n^n clauses.*

Proof. Denote by $I_r \to a$ the r-th clause in (1), for $1 \le r \le n^n$. I_r is not a model of the r-th clause in (1), but satisfies any other clause in (1). In addition, for each pair I_i, I_j with $1 \le i \neq j \le n^n$, the intersection $I_i \cap I_j$ is a model of (1).

Suppose that there exists a set H of $Horn$ clauses equivalent to (1) whose number of clauses is smaller than n^n. There must be at least two different Σ-interpretations I_i and I_j that falsify the same clause c in H. Since we are dealing with $Horn$ clauses, the interpretation $I_i \cap I_j$ falsifies c and therefore $I_i \cap I_j$ is not a model of H which is a contradiction. ∎

5 Boolean Formulas are pwm-reducible to $Horn^{\supset}$ programs

In this section we present a relative solution about the non-learnability of the class $Horn^{\supset}$. We show that if $Horn^{\supset}$ programs were learnable then boolean formulas would be also learnable. Namely, we show that a set of especial $Horn^{\supset}$ programs can simulate boolean formulas. The simulation is in the model of prediction with membership queries, where the notion of simulation is characterized by the concept of pwm-reduction.

The pwm-reduction from boolean formulas $\mathcal{C}_{BF}$ to $Horn^{\supset}$ is divided in two stages. Let Σ be a fixed signature. Let $GHorn^{\supset}$ (respectively $DHorn^{\supset}$) be the representation class whose elements are $\langle u, x\rangle$, where x is a Σ-interpretation and u is a Σ-goal G (respectively a Σ-clause D). First we prove that $GHorn^{\supset}$ is pwm-reducible to $DHorn^{\supset}$ and then we present a pwm-reduction from $\mathcal{C}_{BF}$ to $GHorn^{\supset}$.

Lemma 6. *$GHorn^{\supset} \leq_{pwm} DHorn^{\supset}$*

Proof. Let Σ be a fixed signature and b be a fixed variable that is not in Σ. Consider the representation class $GHorn^{\supset}$ over signature Σ and $DHorn^{\supset}$ over signature $\Sigma \cup \{b\}$. For each Σ-goal G, $G \to b$ is a $\Sigma \cup \{b\}$-clause. For any Σ-interpretation I it holds that $I \Vdash_{\Sigma} G \Longleftrightarrow I \nVdash_{\Sigma\cup\{b\}} (G \to b)$.

Formally we define functions g, h and f as follows. For all natural number s and for every concept representation G such that $|G| \leq s$, define $g(s, |\Sigma|, G) = G \to b$. For every Σ-interpretation I, define $f(s, |\Sigma|, I) = I$ and for every $\Sigma \cup \{b\}$-interpretation J define $h(s, |\Sigma|, J) = \top$ if $b \in J$ or $h(s, |\Sigma|, J) = J$ if $b \notin J$. These functions preserve conditions (1), (2)' and (3)' in Definition 1. ∎

The second step is based completely on the fact that monotone boolean circuits are as hard to predict as general boolean circuits. This result was proved in [6]. An exhaustive analysis of its proof allows us to ensure that monotone boolean formulas (denoted by $\mathcal{C}_{MBF}$) are as hard to predict as general ones.

Theorem 2. *[6] $\mathcal{C}_{BF} \leq_{pwm} \mathcal{C}_{MBF}$*

Next, we reduce the class of monotone boolean formulas to $GHorn^{\supset}$.

Theorem 3. *$\mathcal{C}_{MBF} \leq_{pwm} GHorn^{\supset}$*

Proof. Let C be a monotone boolean formula over signature Σ. From C we can construct, by induction, a goal C' over signature $\Sigma \cup \{b\}$, where $b \notin \Sigma$.

1. If C is a variable v, then C' is v
2. if C is of the form $C_1 \wedge \ldots \wedge C_k$, then C' is $C'_1 \wedge \ldots \wedge C'_k$
3. if C is of the form $C_1 \vee \ldots \vee C_k$, then C' is $((C'_1 \rightarrow b) \wedge \ldots \wedge (C'_k \rightarrow b)) \supset b$

The following proposition allows us to prove the theorem.

Proposition 2. *Given a signature Σ, a variable $b \notin \Sigma$, and a monotone boolean formula C over Σ: for all Σ-interpretation I, $I \Vdash_{\Sigma} C \iff I \Vdash_{\Sigma\cup\{b\}} C'$*

Proof. The proof is again by induction over the structure of C. The nontrivial case is when $C = C_1 \vee \ldots \vee C_k$ and we suppose the proposition holds for $C_1, \ldots, C_k$. Let us see that $I \Vdash_{\Sigma} C_1 \vee \ldots \vee C_k \iff I \Vdash_{\Sigma\cup\{b\}} ((C'_1 \rightarrow b) \wedge \ldots \wedge (C'_k \rightarrow b)) \supset b$

($\Longrightarrow$) Suppose $I \nVdash_{\Sigma\cup\{b\}} ((C'_1 \rightarrow b) \wedge \ldots \wedge (C'_k \rightarrow b)) \supset b$. There must exist J such that: $I \subseteq J$; for all $i \in \{1, \ldots, k\}$, $J \Vdash_{\Sigma\cup\{b\}} C'_i \rightarrow b$; and $J \nVdash_{\Sigma\cup\{b\}} b$. Therefore, for all $i \in \{1, \ldots, k\}$, $J \nVdash_{\Sigma\cup\{b\}} C'_i$. By persistence of goals, for all $i \in \{1, \ldots, k\}$, $I \nVdash_{\Sigma\cup\{b\}} C'_i$. Hence, by hypothesis of induction, for all $i \in \{1, \ldots, k\}$, $I \nVdash_{\Sigma} C_i$. Thus, $I \nVdash_{\Sigma} C_1 \vee \ldots \vee C_k$.

($\Longleftarrow$) Suppose $I \nVdash_{\Sigma} C_1 \vee \ldots \vee C_k$. That is, for all $i \in \{1, \ldots, k\}$, $I \nVdash_{\Sigma} C_i$. By hypothesis of induction, for all $i \in \{1, \ldots, k\}$, $I \nVdash_{\Sigma\cup\{b\}} C'_i$. Hence, for all $i \in \{1, \ldots, k\}$, $I \Vdash_{\Sigma\cup\{b\}} C'_i \rightarrow b$. Since $b \notin I$, $I \nVdash_{\Sigma\cup\{b\}} ((C'_1 \rightarrow b) \wedge \ldots \wedge (C'_k \rightarrow b)) \supset b$. ■

Now we define functions g, h, and f. For all natural number s and for every monotone boolean formula C over Σ, such that $|C| \leq s$, let C' be the $\Sigma\cup\{b\}$-goal obtained from C as described above, where $b \notin \Sigma$. We define $g(s, |\Sigma|, C) = C'$. For every Σ-interpretation I, $f(s, |\Sigma|, I) = I$ and for every $\Sigma\cup\{b\}$-interpretation J, $h(s, |\Sigma|, J) = \top$ if $b \in J$ or $h(s, |\Sigma|, J) = J$ if $b \notin J$. These functions preserve conditions (1), (2) and (3) in Definition 1. ■

Therefore, by Lemmas 2 and 3, if $DHorn^{\supset}$ is polynomially predictable with membership queries, then $\mathcal{C}_{BF}$ is polynomially predictable with membership queries. As a consequence, by Lemma 1, if $DHorn^{\supset}$ is polynomially learnable, then $\mathcal{C}_{BF}$ is polynomially predictable with membership queries. However, the well-known Angluin and Kharitonov's result below (see [3]) ensures us that predicting $\mathcal{C}_{BF}$ in polynomial time and using membership queries is a hard problem.

Theorem 4. *[3] If we assume the intractability of any of the following three problems: testing quadratic residues modulo a composite, inverting RSA encryption, or factoring Blum integers, then $\mathcal{C}_{BF}$ is not polynomially predictable with membership queries.*

6 Conclusions

In this paper, we have presented a pwm-reduction from boolean formulas to conjunctions of $Horn^{\supset}$ clauses. Consequently, this class is not predictable even with membership queries under cryptographic assumptions. This result gives rise to other questions: Could we give a negative result about the learnability of the class? Could we improve the relative result presented in this paper by showing that conjunctions of $Horn^{\supset}$ clauses are pwm-equivalent to boolean formulas?

References

1. Angluin, D. *Queries and Concept Learning.* Machine Learning, volume 2(4): 319–342, (1988).
2. Angluin, D., Frazier, M. and Pitt, L. *Learning Conjunctions of Horn Clauses.* Machine Learning, volume 9: 147–164, (1992).
3. Angluin, D. and Kharitonov, M. *When won't Membership Queries Help?.* Journal of Computer and System Sciences, 50(2): 336–355, April 1995.
4. Arruabarrena, R., Lucio P. and Navarro, M.*A Strong Logic Programming View for Static Embedded Implications.*In: Proc. of FOSSACS'99, Springer-Verlag Lect. Notes in Comput. Sciences 1578: 56–72 (1999).
5. Bugliesi, M., Lamma, E. and Mello, P. *Modularity in Logic Programming.* Journal of Logic Programming, (19-20): 443–502, (1994).
6. Dalmau, V. *A Dichotomy Theorem for Learning Quantified Boolean Formulas.* Machine Learning, volume 35(3): 207–224 (1999).
7. Gabbay, D. M. *N-Prolog: An Extension of Prolog with Hypothetical Implications. II. Logical Foundations and Negation as Failure.* Journal of Logic Programming 2(4): 251–283 (1985).
8. Giordano, L., and Martelli, A. *Structuring Logic Programs: A Modal Approach.* Journal of Logic Programming 21: 59–94 (1994).
9. Giordano, L., Martelli, A., and Rossi, G. *Extending Horn Clause Logic with Implication Goals.* Theoretical Computer Science 95: 43–74, (1992).
10. Kearns, M. and Valiant, L. *Criptographic Limitations on Learning Boolean Formulae and Finite Automata.* Journal of the ACM 41(1): 67–95, (1994).
11. Lucio, P.*Structured Sequent Calculi for Combining Intuitionistic and Classical First-Order Logic.* In: Proc. of FroCoSS'2000,Springer-Verlag Lect. Notes in Artificial Intelligence 1794: 88–104 (2000).
12. Meseguer, J. *Multiparadigm Logic Programming.* In: Proc. of ALP'92, Springer-Verlag Lect. Notes in Comput. Sciences 632: 158–200, (1992).
13. Miller, D.*A Logical Analysis of Modules in Logic Programming.* In: Journal of Logic Programming 6: 79–108, (1989).
14. Miller, D., Nadathur, G., Pfenning, F. and Scedrov, A. *Uniform Proofs as a Foundation for Logic Programming.* Annals of Pure and App. Logic 51: 125–157, (1991).
15. Monteiro, L., Porto, A. *Contextual Logic Programming.* In: Proc. 6th International Conf. on Logic Programming 284–299, (1989).
16. Moscowitz, Y., and Shapiro, E. *Lexical Logic Programs.* In: Proc. 8th International Conf. on Logic Programming 349–363, (1991).
17. Navarro, M. *From Modular Horn Programs to Flat Ones: a Formal Proof for the Propositional Case.* In: Proc. of ISIICT 2004 (Int. Symp. on Innovation in Information and Communication Technology), Amman, Jordan. April 2004.
18. Pitt, L. and Warmuth, M.K. *Prediction-Preserving Reducibility.* Journal of Computer and System Sciences, 41(3): 430–467 (1990)
19. Valiant L.G. *A Theory of the Learnable.* Communications of the ACM, 27: 1134–1142 (1984)

Defying Dimensions Modulo 6

Vince Grolmusz

Department of Computer Science, Eötvös University
H-1117 Budapest, Hungary
grolmusz@cs.elte.hu

Abstract. We consider here a certain modular representation of multi-linear polynomials. The modulo 6 representation of polynomial g is just any polynomial $g + 6e$. The 1-a-strong representation of g modulo 6 is polynomial $g + 3f + 4h$, where no two of g, f and h has common monomials.
Using this representation, we describe some surprising applications: we show that the n homogeneous linear polynomials $x_1, x_2, \ldots, x_n$ can be linearly transformed to $n^{o(1)}$ linear polynomials,[1] and from these linear polynomials we get back the 1-a-strong representations of the original ones, also with linear transformations. We define Probabilistic Memory Cells (PMC's), and show how to encode n bits into n PMC's, transform n PMC's to $n^{o(1)}$ PMC's (we call this form Hyperdense Coding), and we show how one can transform back these $n^{o(1)}$ PMC's to n PMC's, and from these we can get back the original bits, while from the hyperdense form we could have got back only $n^{o(1)}$ bits. We also show that $n \times n$ matrices can be converted to $n^{o(1)} \times n^{o(1)}$ matrices and from these tiny matrices we can retrieve 1-a-strong representations of the original ones, also with linear transformations. Applying PMC's to this case will return the original matrix, and not only the representation.
We also show that a 1-a-strong representation of the matrix-product can be computed with only $n^{o(1)}$ multiplications, significantly improving our earlier result.

1 Introduction

Let f be an n-variable, multi-linear polynomial (that is, every variable appears on the power of 0 or 1) with integer coefficients, for example $f(x_1, x_2, x_3) = 34x_1x_2 + 23x_1x_2x_3$. For any positive integer $m > 1$, we say that multi-linear polynomial g is a mod m representation of polynomial f, if the corresponding coefficients of the two polynomials are congruent modulo m; for example, $g(x_1, x_2, x_3) = 4x_1x_2 + 3x_1x_2x_3 + 5x_2$ is a mod 5 representation of the f in the previous example. If we choose a non-prime-power, composite modulus, say $m = 6$, then the modulo 6 representation of polynomial f is also a modulo 3 and modulo 2 representation at the same time. This means, that if we examine

[1] The quantity $o(1)$ here denotes a positive number which goes to 0 as n goes to infinity.

the properties of the modulo 6 representations of multi-linear polynomials (or, equivalently, multi-linear polynomials over ring Z_6), it is not probable that we get more interesting properties over Z_6 than over fields F_2 or F_3.

Over composite, non-prime-power moduli (say 6), however, we can consider different representations as well. We will define 1-a-strong representations of polynomials formally in the next section, but now it is enough to say that the 1-a-strong representation of multi-linear polynomial f modulo 6 is a polynomial $f + 3g + 4h$, where no two of g, f and h have common monomials. The last restriction is necessary, since otherwise the constant polynomial 0 would be the 1-a-strong representation modulo 6 of an arbitrary polynomial f, simply because $0 = f + 3f - 4f$.

A similar polynomial representation modulo non-prime-power composites was considered in [6]. There we proved, that a representation (what we call a 0-a-strong representation in the next section) of the elementary symmetric polynomials can be computed dramatically faster than over prime moduli. This result plays a main rôle in the proofs of the present work.

1.1 Our Results:

Let m be a non-prime power composite constant (that is, it is constant in n, e.g., $m = 6$).

(a) From the n variables $x_1, x_2, \dots, x_n$, (each seen as a 1-variable linear function,) we compute $t = n^{o(1)}$ linear functions $z_1, z_2, \dots, z_t$, and from these t linear functions again n linear functions $x'_1, x'_2, \dots, x'_n$, such that x'_i is a 1-a-strong representation of linear function (i.e., variable) x_i, for $i = 1, 2, \dots, n$. Both computations are linear transformations.

(b) We define Probabilistic Memory Cells (PMC's). By an observation of a PMC one can get a constant amount of information. We encode n bits into n PMC's: one bit into one PMC, and we use the first linear transformation in (a) to transform the n PMC's to $t = n^{o(1)}$ PMC's (observing these t PMC's would yield only $O(n^{o(1)})$ bits of information), and then we transform these t PMC's back to n PMC's, also with a linear transformation, and the observation of the resulting n PMC's will yield the original n bits. We call this phenomenon hyperdense coding modulo m.

(c) For any $n \times n$ matrix X with elements from set Z_m, we compute an $n^{o(1)} \times n^{o(1)}$ matrix Z with elements from set Z_m, such that from Z, one can retrieve the 1-a-strong representation of the $n \times n$ matrix X; here both operations (the computing and the retrieval) are simple linear transformations. Note, that this means that with $n = N^{100}$, even an $N^{100} \times N^{100}$ matrix X can be converted to $\sqrt[100]{N} \times \sqrt[100]{N}$ matrix Z, and back to an $N^{100} \times N^{100}$ matrix X' with linear transformations, for large enough N.

(d) Using Probabilistic Memory Cells for storing each entry of the binary matrix X in (c), matrix Z can be stored with $n^{o(1)}$ PMC's, from which we can compute the original $n \times n$ matrix X, by using the second linear transform of (c) and observations of the resulting n^2 PMC's. We call this phenomenon the dimension defying property of the 1-a-strong representation.

(e) For $n \times n$ matrices X and Y, with elements from set Z_m, we compute the 1-a-strong representation of the product matrix XY, with only $n^{o(1)}$ multiplications, significantly improving our earlier result of computing the 1-a-strong representation of the matrix-product with $n^{2+o(1)}$ multiplications [7].

2 Preliminaries

2.1 A-strong representations

In [6] we gave the definition of the *a-strong (i.e., alternative-strong) representation* of polynomials. Here we define the *alternative*, and the *0-a-strong* and the *1-a-strong* representations of polynomials. Note that the 0-a-strong representation, defined here, coincides with the a-strong representation of the paper [6]. Note also, that for prime or prime-power moduli, polynomials and their representations (defined below), coincide. This fact also motivates the examination of such representations.

Definition 1. *Let m be a composite number with prime-factorization $m = p_1^{e_1} p_2^{e_2} \cdots p_\ell^{e_\ell}$. Let Z_m denote the ring of modulo m integers. Let*

f be a multi-linear polynomial of n variables over Z_m: $f(x_1, x_2, \ldots, x_n) = \sum_{I \subset \{1,2,\ldots,n\}} a_I x_I$, where $a_I \in Z_m$, $x_I = \prod_{i \in I} x_i$. Then we say that $g(x_1, x_2, \ldots, x_n) = \sum_{I \subset \{1,2,\ldots,n\}} b_I x_I$, is an

– alternative representation *of f modulo m, if $\forall I \subset \{1,2,\ldots,n\}$ $\exists j \in \{1,2,\ldots,\ell\}$: $a_I \equiv b_I \pmod{p_j^{e_j}}$;*

– 0-a-strong representation *of f modulo m, if it is an alternative representation, and, furthermore, if for some i, $a_I \not\equiv b_I \pmod{p_i^{e_i}}$, then $b_I \equiv 0 \pmod{p_i^{e_i}}$;*

– 1-a-strong representation *of f modulo m, if it is an alternative representation, and, furthermore, if for some i, $a_I \not\equiv b_I \pmod{p_i^{e_i}}$, then $a_I \equiv 0 \pmod{m}$;*

Example 1. Let $m = 6$, and let $f(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + x_1x_3$, then $g(x_1, x_2, , x_3) = 3x_1x_2 + 4x_2x_3 + x_1x_3$ is a 0-a-strong representation of f modulo 6; $g(x_1, x_2, , x_3) = x_1x_2 + x_2x_3 + x_1x_3 + 3x_1^2 + 4x_2$ is a 1-a-strong representation of f modulo 6; $g(x_1, x_2, , x_3) = 3x_1x_2 + 4x_2x_3 + x_1x_3 + 3x_1^2 + 4x_2$ is an alternative representation modulo 6.

Example 2. Let $m = 6$. Then $0 = xy - 3xy + 2xy$ is **not** a 1-a-strong representation of xy. Similarly, polynomial $f + 2g + 3h$ is a mod 6 1-a-strong representation of polynomial f if and only if g and h do not have common monomials with f, and g does not have common monomials with h.

2.2 Previous results for a-strong representations

We considered elementary symmetric polynomials $S_n^k = \sum_{\substack{I\subset\{1,2,\ldots,n\}\\ |I|=k}} \prod_{i\in I} x_i$ in [6], and proved that for constant k's, 0-a-strong representations of elementary symmetric polynomials S_n^k can be computed dramatically faster over non-prime-power composites than over primes.

In [6], we proved the following theorem:

Theorem 1 ([6]). *Let the prime factorization of positive integer m be $m = p_1^{e_1} p_2^{e_2} \cdots p_\ell^{e_\ell}$, where $\ell > 1$. Then a degree-2 0-a-strong representation of*

$$S_n^2(x,y) = \sum_{\substack{i,j\in\{1,2,\ldots,n\}\\ i\neq j}} x_i y_j, \tag{1}$$

modulo m:

$$\sum_{\substack{i,j\in\{1,2,\ldots,n\}\\ i\neq j}} a_{ij} x_i y_j \tag{2}$$

can be computed as the following product: $\sum_{j=1}^{t-1} \left(\sum_{i=1}^n b'_{ij} x_i\right) \left(\sum_{i=1}^n c'_{ij} y_i\right)$ where $t = \exp(O(\sqrt[\ell]{\log n (\log\log n)^{\ell-1}})) = n^{o(1)}$. Moreover, this representation satisfies that $\forall i \neq j : a_{ij} = a_{ji}$.

□

The following result is the basis of our theorems in the present paper.

Theorem 2 ([7]). *Let $m = p_1^{e_1} p_2^{e_2} \cdots p_\ell^{e_\ell}$, where $\ell > 1$, and $p_1, p_2, \ldots, p_\ell$ are primes. Then a degree-2, 1-a-strong representation of the dot-product $f(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n) = \sum_{i=1}^n x_i y_i$ can be computed with $t = \exp(O(\sqrt[\ell]{\log n (\log\log n)^{\ell-1}})) = n^{o(1)}$ multiplications of the form*

$$\sum_{j=1}^{t} \left(\sum_{i=1}^{n} b_{ij} x_i\right) \left(\sum_{i=1}^{n} c_{ij} y_i\right) \tag{3}$$

Proof. Let $g(x,y) = g(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n)$ be the degree-2 polynomial from Theorem 1 which is a 0-a-strong representation of $S_n^2(x,y)$. Then consider polynomial

$$h(x,y) = (x_1 + x_2 + \ldots + x_n)(y_1 + y_2 + \ldots + y_n) - g(x,y). \tag{4}$$

In $h(x,y)$, the coefficients of monomials $x_i y_i$ are all 1's modulo m, and the coefficients of monomials $x_i y_j$, for $i \neq j$ are 0 at least for one prime-power divisor of m, and if it is not 0 for some prime divisor, then it is 1. Consequently, by Definition 1, $h(x,y)$ is a 1-a-strong representation of the dot-product $f(x,y)$.

3 Dimension-Defying: Linear Functions

For simplicity, let $m = 6$.

By Theorem 2, a 1-a-strong representation of the dot-product $\sum_{i=1}^{n} x_i y_i$ can be computed as

$$\sum_{i=1}^{n} x_i y_i + 3g(x,y) + 4h(x,y) = \sum_{j=1}^{t} \left(\sum_{i=1}^{n} b_{ij} x_i\right) \left(\sum_{i=1}^{n} c_{ij} y_i\right) \tag{5}$$

where $b_{ij}, c_{ij} \in \{0,1\}$ and where both g and h has the following form: $\sum_{i \neq j} a_{ij} x_i y_j$, and no term $x_i y_j$ appears in both f and g; and $t = \exp(O(\sqrt{\log n \log\log n})) = n^{o(1)}$. Note that every monomial $x_i y_j$, $i \neq j$ has really a coefficient which is a multiple of 3 or 4, since 1-4=3 and 1-4=3 modulo 6.

Now, let us observe that for each $j = 1, 2, \ldots, t$,

$$z_j = \sum_{i=1}^{n} b_{ij} x_i \tag{6}$$

is a linear combination of variables x_i.

Let these $t = n^{o(1)}$ linear forms be the encoding of the n 0-1 variables $x_i's$. The decoding is done also from (5): the 1-a-strong representation of x_i can be computed by plugging in

$$y^i = (0, 0, \ldots, \overbrace{1}^{i}, 0, \ldots, 0).$$

Obviously, on the LHS of (5) we get the 1-a-strong representation of x_i, and on the RHS we get a linear combination of the z_j's of (6).

By matrix-notation, if x is a length-n vector, and $B = \{b_{ij}\}$ is an $n \times t$ matrix with b_{ij}'s given in (5), and $C = \{c_{ij}\}$ is an $n \times t$ matrix with c_{ij}'s given in (5), then we can write that

$$z = xB, \text{ and } x' = zC^T = xBC^T.$$

Consequently, $x' = xBC^T$ is a length-n vector, such that for $i = 1, 2, \ldots, n$, $x'_i = x_i + 3g_i(x) + 4h_i(x)$ where $g(x)$ and $h(x)$ are integer linear combinations (that is, homogeneous linear functions) of the coordinates of x such that none of which contains x_i and they do not contain the same x_j with non-zero coefficients. The proof of this fact is obvious from (5). It is easy to see that we proved the following Theorem (stating for general m this time):

Theorem 3. *For any non-prime-power positive integer m, and positive integer n, there exist effectively computable constant $n \times t$ matrices B and C over Z_m, with $t = n^{o(1)}$, such that for any vector $x = (x_1, x_2, \ldots, x_n)$ with variables as coordinates, the coordinate i of the length-n vector xBC is a 1-a-strong representation of polynomial x_i modulo m, for $i = 1, 2, \ldots, n$.*

□

Note, that xB has t coordinates (linear functions), while xBC^T has again n coordinates (linear functions). Note that similar representation is *impossible* with m prime and $t < n$.

For an application of this striking observation we need the definition of Probabilistic Memory Cells.

4 Probabilistic Memory

The words "probabilistic" and "memory" are rarely mixed well: a probabilistically behaving memory element – typically – is not desirable in any computer. Here we consider 1-0 step functions on the real interval $[0, 1]$, describing some physical object changing its state from 1 to 0 in a random point of the interval $[0, 1]$. We assume that the distribution of this point is uniform in the the real interval $[0, 1]$. We also assume that the distribution of these random points are independent. The randomness will assure us that with high probability (more exactly, with probability 1) no two different functions have the state-change at the same moment. We intend to use integer linear combinations of these functions for dense data storage. The formal definition is as follows:

Definition 2. *An m-Probabilistic Memory Cell (m-PMC in short) is a step-function $\rho : [0, 1] \to Z_m$, such that $\rho(i)^{-1}$, for $i = 0, 1, \ldots, m-1$, is a finite union of subintervals of the interval $[0, 1]$. $a \in [0, 1]$ is a step-point of ρ if $\lim_{+a} \rho \neq \lim_{-a} \rho$. The step-value in step-point a is equal to $\lim_{+a} \rho - \lim_{-a} \rho$ modulo m. An m-PMC is simple, if there exists an $a \in [0, 1]$ such that $\rho^{-1}(1) = [0, a]$, and $\rho^{-1}(0) = (a, 1]$. A collection of m-PMC's $\rho_1, \rho_2, \ldots, \rho_n$ is called a proper-(n, m)-PMC, if*

- *every ρ_i is a simple m-PMC, and*
- *for all $i \neq j$, the step-points of ρ_i and ρ_j differ.*

The observation operator $\mathcal{O}(\rho)$ returns the (un-ordered) set of step-values, modulo m, in all the step-points of m-PMC ρ, that is, $\mathcal{O}(\rho) \subset \{0, 1, \ldots, m-1\}$, for any m-PMC ρ.

Note, that the set of the m-PMC's forms a module over the integer ring Z_m. Note also, that the set of step-points of an integer linear combination of several m-PMC's is a subset of the union of the step-points of the individual PMC's.

Fact. If the step-points are distributed uniformly and independently in each of the n simple m-PMC's, then their collection will form a proper-(n, m)-PMC with probability 1.

This is the reason that the word "Probabilistic" appears in Definition 2.

Example 1: On Figure 1, the linear combination of simple PMC's ρ and ξ, $2\rho + 3\xi$ is also a PMC, and $\mathcal{O}(2\rho + 3\xi) = \{-2, -3\} = \{4, 3\}$, with $m = 6$.

Example 2: The sum of the members of the proper-$(n, 6)$-PMC $\rho_1, \rho_2, \ldots, \rho_n$ is also a 6-PMC $\xi = \sum_{i=1}^{n} \rho_i$, and clearly, $\mathcal{O}(\xi) = \{5\}$.

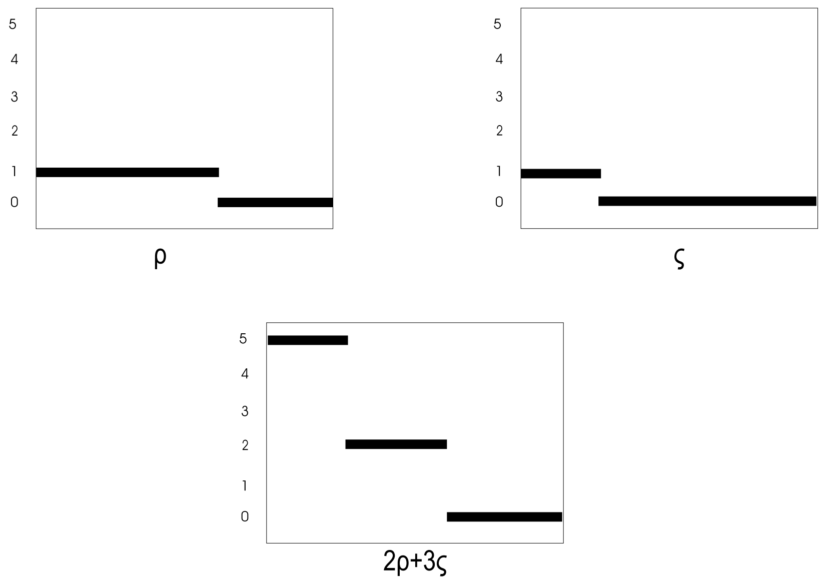


Fig. 1. Linear combination of PMC's ρ and ξ.

5 Hyperdense Coding

Let $h_1, h_2, \ldots, h_n$ be n bits. Let $\rho_1, \rho_2, \ldots, \rho_n$ be a proper $(n, 6)$ PMC. Now define $x_i = h_i \rho_i$, for $i = 1, 2, \ldots, n$, and let $x = (x_1, x_2, \ldots, x_n)$. Clearly, the x_i's are also PMC's. Now, let us use matrices B and C from Theorem 3. Let $z = xB$ be a vector, and each of the $t = n^{o(1)}$ coordinates of it is a PMC. Note, that observing any coordinate of z yields only $O(1)$ bits of information, $O(n^{o(1)})$ in total. However, if we do not observe the coordinates of z, but instead of that we apply the linear transform C^T to it, then we would get back the 1-a-strong representation of polynomials x_i in each coordinate of $zC^T = xBC^T$ in case of variables as $x_i's$, that is: $x_i' = x_i + 3g_i(x) + 4h_i(x)$. But now we have PMC's instead of linear functions.

What happens if we observe x_i'? Clearly, for $m = 6$, $h_i = 1 \iff 5 \in \mathcal{O}(x_i')$, since in case of $h_i = 0$ every step-value is a multiple of 2 or 3. That means that by observing the n PMC's in the coordinates of $zC^T = xBC^T$, we get back the n bits of $h_1, h_2, \ldots, h_n$.

Note, that the t coordinates of z also contained the information on the n input-bits, but with observations we were not able to recover it. We call z the hyperdense coding of bits $h_1, h_2, \ldots, h_n$. Consequently, we have proved (again stating for general m):

Theorem 4. *For any non-prime-power positive integer m, and positive integer n, there exist effectively computable constant $n \times t$ matrices B and C over Z_m, with $t = n^{o(1)}$, such that for any bit-sequence $h_1, h_2, \ldots, h_n$ can be encoded into n*

m-PMC's $x = (x_1, x_2, \ldots, x_n)$, and these m-PMC's can be linearly transformed into t m-PMC's $z = xB$, and these PMC's can be linearly transformed to n PMC's $x' = zC^T = xBC^T$, such that the observation of the PMC's in the coordinates of x' yields the original values of $h_1, h_2, \ldots, h_n$.

□

Note, that in a completely different model, Bennet and Wiesner [4], using Einstein-Podolski-Rosen entangled pairs, showed that n classic bits can be encoded by $\lceil n/2 \rceil$ quantum bits. They called their result superdense coding.

6 Our Result for Matrix Compression

Definition 3. *Let $X = \{x_{ij}\}$ be an $n \times n$ matrix with one-variable homogeneous linear functions (that is, $x'_{ij}s$) as entries. Then $Y = \{y_{ij}\}$ is a 1-a-strong representation of the matrix X modulo m if for $1 \le i, j \le n$, the polynomial y_{ij} of n^2 variables $\{x_{uv}\}$ is a 1-a-strong representation of polynomial x_{ij} modulo m.*

If we plug in column-vectors instead of just variables in the homogeneous linear forms of Theorem 3, then we will get linear combinations of the column-vectors. Consequently, we proved the following implication of Theorem 2:

Theorem 5. *For any non-prime-power positive integer m, and positive integer n, there exist effectively computable constant $n \times t$ matrices B and C, such that for any $n \times n$ matrix $X = \{x_{ij}\}$, XBC^T is a 1-a-strong representation of matrix X modulo m, where $t = n^{o(1)}$.*

The dimension-defying implication of Theorem 5 is that X is an $n \times n$ matrix, XB is an $n \times n^{o(1)}$ matrix, and XBC^T is again an $n \times n$ matrix.

An easy corollary of Theorem 5, that

Corollary 1. *With the notations of Theorem 5, CB^TX is a 1-a-strong representation of matrix X modulo m, where $t = n^{o(1)}$.*

Our main result in this section is the following implication of Corollary 1 and Theorem 5:

Theorem 6. *For any non-prime-power $m > 1$, there exist effectively computable constant $n \times t$ matrices B and C, such that for any matrix $X = \{x_{ij}\}$, B^TXB is a $t \times t$ matrix, where $t = n^{o(1)}$, and matrix CB^TXBC^T is a 1-a-strong representation of matrix X modulo m.*

The dimension-defying implication of Theorem 6 is that from the $n \times n$ matrix X with simple linear transformations we make the tiny $n^{o(1)} \times n^{o(1)}$ matrix B^TXB, and from this, again with simple linear transformations, $n \times n$ matrix CB^TXBC^T, where it is a 1-a-strong representation of matrix X modulo m.

7 Dimension Defying

Similarly as in Section 5, where we changed our result from linear functions to numbers with using PMC's, now we repeat the same method.

Theorem 7. *For any non-prime-power $m > 1$, and for any positive integer n, there exist effectively computable constant $n \times t$ matrices B and C, such that any $H = \{h_{ij}\}$ a 0-1 $n \times n$ matrix can be encoded into an $n \times n$ matrix $X = \{x_{ij}\}$ with n^2 PMC's as entries, applying two linear transforms to this matrix we get an $t \times t$ matrix $B^T XB$ which contains t^2 m-PMC's, and applying two further linear transforms, we get the $n \times n$ matrix $CB^T XBC^T$, with n^2 PMC's as entries, whose observation returns the original 0-1 values of the matrix H.*

Proof. Let $\rho_{11}, \rho_{12}, \dots, \rho_{nn}$ be a proper (n^2, m)-PMC, and let us define the $x_{ij} = h_{ij}\rho_{ij}$. Clearly, the entries of $CB^T XBC^T$ are 1-a-strong representations of x'_{ij}s, so by observing its (i, j) entry, x'_{ij} the following holds: $h_{ij} = 1 \iff m - 1 \in \mathcal{O}(x'_{ij})$.

8 Our result for matrix multiplication

The matrix multiplication is a basic operation in mathematics in applications in almost every branch of mathematics itself, and also in the science and engineering in general. An important problem is finding algorithms for fast matrix multiplication. The natural algorithm for computing the product of two $n \times n$ matrices uses n^3 multiplications. The first, surprising algorithm for fast matrix multiplication was the recursive method of Strassen [10], with $O(n^{2.81})$ multiplications. After a long line of results, the best known algorithm today was given by Coppersmith and Winograd [5], requiring only $O(n^{2.376})$ multiplications. Some of these methods can be applied successfully in practice for the multiplication of large matrices [1].

The best lower bounds for the number of needed multiplications are between $2.5n^2$ and $3n^2$, depending on the underlying fields (see [2], [3], [9]). A result of Raz [8] gives an $\Omega(n^2 \log n)$ lower bound for the number of multiplications, if only bounded scalar multipliers can be used in the algorithm.

In [7] we gave an algorithm with $n^{2+o(1)}$ multiplications for computing the 1-a-strong representation of the matrix product modulo non-prime power composite numbers (e.g., 6). The algorithm was an application of a method of computing a representation of the dot-product of two length-n vectors with only $n^{o(1)}$ multiplications.

In the present work, we significantly improve the results of [7], we give an algorithm for computing the 1-a-strong representation of the product of two $n \times n$ matrices with only $n^{o(1)}$ multiplications.

Definition 4. *Let $X = \{x_{ij}\}$ and $Y = \{y_{ij}\}$ be two $n \times n$ matrices with $2n^2$-variable homogeneous linear functions (that is, x'_{ij}s and y'_{ij}s as entries. We say that matrix $V = \{v_{ij}\}$ is a 1-a-strong representation of the product-matrix XY, if* *for $1 \leq i, j \leq n$, v_{ij}, as a $2n^2$-variable polynomial, is a 1-a-strong representation of polynomial $\sum_{k=1}^{n} x_{ik}y_{kj}$ modulo m.*

Note, that this definition is not implied by Definition 3. We need to define a sort of generalization of the matrix-product:

Definition 5. *$f : R^{2n} \to R$ is a homogeneous bilinear function over ring R if $f(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n) = \sum_{1 \leq i,j \leq n} a_{ij}x_iy_j$ for some $a_{i,j} \in R$. Let $U = \{u_{ij}\}$ be an $u \times n$ matrix over ring R, and let $V = \{v_{k\ell}\}$ be an $n \times v$ matrix over R. Then $U(f)V$ denotes the $u \times v$ matrix over R with entries $w_{i\ell}$, where $w_{i\ell} = f(u_{i1}, u_{i2}, \dots, u_{in}, v_{1\ell}, v_{2\ell}, \dots, v_{n\ell})$.*

Note, that if f is the dot-product, then $U(f)V$ is just the simple matrix-product. First we need a simple lemma, stating that the associativity of the matrix multiplication is satisfied also for the "strange" matrix-multiplication defined in Definition 5:

Lemma 1. *Let $f(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n) = \sum_{1 \leq i,j \leq n} a_{ij}x_iy_j$ and let $g(x_1, x_2, \dots, x_v, y_1, y_2, \dots, y_v) = \sum_{1 \leq i,j \leq v} b_{ij}x_iy_j$ be homogeneous bilinear functions over the ring R. Let $U = \{u_{ij}\}$ be an $u \times n$ matrix, and let $V = \{v_{k\ell}\}$ be an $n \times v$ matrix, and $W = \{w_{ij}\}$ be a $v \times w$ matrix over R, where u, n, w are positive integers. Then $(U(f)V)(g)W = U(f)(V(g)W)$, that is, the "strange" matrix-multiplication, given in Definition 5, is associative.*

Proof: The proof is obvious from the homogeneous bi-linearity of f and g.

Now we are in the position of stating and proving our main theorem for matrix multiplications:

Theorem 8. *Let X and Y two $n \times n$ matrices, and let $m > 1$ be a non-prime-power integer. Then the 1-a-strong representation of the matrix-product XY can be computed with $t^3 = n^{o(1)}$ non-scalar multiplications.*

Proof. We use Theorem 5 and Corollary 1. Let us consider $t \times n$ matrix B^TX and $t \times n$ matrix YB; these matrices can be computed without any multiplications from X and Y (we do not count multiplications by constants). Let $h(x, y)$ be the homogeneous bi-linear function (4). Then $B^TX(h)YB$ can be computed with $n^{o(1)}$ multiplications (Note, that because of Lemma 1, the associativity holds). Now compute matrix $CB^TX(f)YBC^T = (CB^TY)(f)(YBC^T)$ without any further (non-constant) multiplication. By Theorem 5 and Corollary 1, CB^TX and YBC^T is a 1-a-strong representations of X and Y respectively, and they are the linear combinations of the rows of X and columns of Y, respectively. Consequently, using Theorem 2, $CB^TX(f)YBC^T$ is a 1-a-strong representation of XY.

References

1. David H. Bailey. Extra high speed matrix multiplication on the Cray-2. *SIAM J. Sci. Statist. Comput.*, 9(3):603–607, 1988.
2. Markus Bläser. A $\frac{5}{2}n^2$-lower bound for the rank of $n \times n$-= matrix multiplication over arbitrary fields. In *40th Annual Symposium on Foundations of Computer Scienc= e (New York, 1999)*, pages 45–50. IEEE Computer Soc., Los Alamitos, CA, 1999.=
3. Nader H. Bshouty. A lower bound for matrix multiplication. *SIAM J. Comput.*, 18(4):759–765, 1989.
4. C.H. Bennet and S.J. Wiesner. Communication via one- and two particle operators on Einstein-Podolski-Rosen states. *Phys. Rev. Lett.*, 69:2881–2884, 1992.
5. Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Comput.*, 9(3):251–280, 1990.
6. Vince Grolmusz. Computing elementary symmetric polynomials with a subpolynomia= l number of multiplications. *SIAM Journal on Computing*, 32(6):1475–1487, 2003.
7. Vince Grolmusz. Near quadratic matrix multiplication modulo composites. Technical Report TR03-001, ECCC, 2003. ftp://ftp.eccc.uni-trier.de/pub/eccc/reports/2003/TR03-001/index.html.
8. Ran Raz. On the complexity of matrix product. In *Proceedings of the thirty-fourth annual ACM symposium o= n Theory of computing*. ACM Press, 2002.
9. Amir Shpilka. Lower bounds for matrix product. In *IEEE Symposium on Foundations of Computer Science*, p= ages 358–367, 2001.
10. V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

Solving SAT with membrane creation[⋆]

Miguel Ángel Gutiérrez-Naranjo, Mario J. Pérez-Jiménez, and Francisco José Romero-Campero

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012
Sevilla, Spain
{magutier,marper,fran}@us.es

Abstract. Membrane Computing is a branch of Natural Computing which starts from the assumption that the processes taking place in the compartmental structure of a living cell can be interpreted as computations. In this paper we present a solution to the SAT problem using Membrane Computing devices (P systems) where an exponential number of membranes can be created from objects in polynomial time.

1 Introduction

Membrane Computing is an emergent branch of Natural Computing introduced by Păun in [9]. Since then it has received important attention from the scientific community. In fact, Membrane Computing has been selected by the Institute for Scientific Information, USA, as a fast *Emerging Research Front* in Computer Science, and [8] was mentioned in [12] as a highly cited paper in October 2003.

This new non-deterministic model of computation starts from the assumption that the processes taking place in the compartmental structure of a living cell can be interpreted as computations. The devices of this model are called *P systems*.

Roughly speaking, a P system consists of a cell-like membrane structure, in the compartments of which one places multisets of objects which evolve according to given rules in a synchronous non-deterministic maximally parallel manner[1]. The representation of data as multisets is an abstraction from the way in which chemical compounds are found in living cells.

Membrane Computing is a cross-disciplinary field with contributions by computer scientists, biologists, formal linguists and complexity theoreticians, enriching each others with results, open problems and promising new research lines.

In this paper we present a contribution from the computational side. We introduce a family of P systems with *membrane creation*, constructed in an

⋆ This work is supported by Ministerio de Ciencia y Tecnología of Spain, by *Plan Nacional de I+D+I (2000–2003)* (TIC2002-04220-C03-01), cofinanced by FEDER funds, and by a FPI fellowship (of the first author) from the University of Sevilla.

1 A layman-oriented introduction can be found in [10] and further bibliography at [13].

uniform way, that solves the problem of determining for a given formula in conjunctive normal form whether it is *satisfiable* or not (the SAT problem).

The paper is organised as follows: first P systems with membrane creation are introduced in the next section. In section 3 recognizer P systems (devices that capture the intuitive idea underlying the concept of algorithm) are presented. The solution in the framework of *membrane creation* to the SAT problem is given in section 4. Finally, some formal details and conclusions are given.

2 P systems with membrane creation

Polynomial solutions to NP-complete problems in Membrane Computing is done by trading time by space. This is inspired from the capability of cells to produce an exponential number of new membranes (new workspace) in polynomial time. Basically there are two ways of producing new membranes in living cells: *mitosis* (membrane division) and *autopoiesis* (membrane creation), see [3]. Both ways of generating new membranes have given rise to different variants of P systems: *P systems with active membranes*, where the new workspace is generated by membrane division and *P systems with membrane creation*, where the new membranes are created from objects. Both models have been proved to be universal, but up to now there is no theoretical result proving that these models simulate each other in polynomial time. P systems with active membranes have been successfully used to design solutions to **NP**-complete problems, as SAT [7], Subset Sum [4], Knapsack [5], Bin Packing [6] and Partition [2], but as Gh. Păun pointed in [11] *"membrane division was much more carefully investigated than membrane creation as a way to obtain tractable solutions to hard problems"*.

In this paper we investigate the second variant mentioned above. Membranes are created in living cells, for instance, in the process of vesicle mediated transport and in order to keep molecules close to each other to facilitate their reactions. Membranes can also be created in a laboratory - see [3]. Here we abstract the operation of creation of new membranes under the influence of existing chemical substances to define P systems with membrane creation. Recall that a *P system with membrane creation* is a construct of the form $\Pi = (O, H, \mu, w_1, \dots, w_m, R)$ where:

1. $m \geq 1$ is the initial degree of the system; O is the alphabet of *objects* and H is a finite set of *labels* for membranes;
2. μ is a *membrane structure* consisting of m membranes labelled (not necessarily in a one-to-one manner) with elements of H and $w_1, \dots, w_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ;
3. R is a finite set of *rules*, of the following forms:
   (a) $[a \rightarrow v]_h$ where $h \in H$, $a \in O$ and v is a string over O describing a multiset of objects. These are *object evolution rules* associated with membranes and depending only on the label of the membrane.
   (b) $a[\,]_h \rightarrow [b]_h$ where $h \in H$, $a, b \in O$. These are *send-in communication rules.* An object is introduced in the membrane possibly modified.

(c) $[a]_h \rightarrow [\,]_h\, b$ where $h \in H$, $a, b \in O$. These are *send-out communication rules.* An object is sent out of the membrane possibly modified.
(d) $[a]_h \rightarrow b$ where $h \in H$, $a, b \in O$. These are *dissolution rules.* In reaction with an object, a membrane is dissolved, while the object specified in the rule can be modified.
(e) $[a \rightarrow [v]_{h_2}]_{h_1}$ where $h_1, h_2 \in H$, $a \in O$ and v is a string over O describing a multiset of objects. These are *creation rules.* In reaction with an object, a new membrane is created. This new membrane is placed inside of the membrane of the object which triggers the rule and has associated an initial multiset and a label.

Rules are applied according to the following principles:

– Rules from (a) to (d) are used as usual in the framework of membrane computing, that is, in a maximal parallel way. In one step, each object in a membrane can only be used for one rule (non deterministically chosen when there are several possibilities), but any object which can evolve by a rule of any form must do it (with the restrictions below indicated).
– Rules of type (e) are used also in a maximal parallel way. Each object a in a membrane labelled with h_1 produces a new membrane with label h_2 placing in it the multiset of objects described by the string v.
– If a membrane is dissolved, its content (multiset and interior membranes) becomes part of the immediately external one. The skin membrane is never dissolved.
– All the elements which are not involved in any of the operations to be applied remain unchanged.
– The rules associated with the label h are used for all membranes with this label, irrespective of whether or not the membrane is an initial one or it was obtained by creation.
– Several rules can be applied to different objects in the same membrane simultaneously. The exception are the rules of type (d) since a membrane can be dissolved only once.

3 Recognizer P systems with membrane creation

Recognizer P systems were introduced in [5] and are the natural framework to study and solve decision problems, since deciding whether an instance has an affirmative or negative answer is equivalent to deciding if a string belongs or not to the language associated with the problem.

In the literature, recognizer P systems are associated in a natural way with P systems with *input*. The data related to an instance of the decision problem has to be provided to the P system in order to compute the appropriate answer. This is done by codifying each instance as a multiset placed in an *input membrane.* The output of the computation (*yes* or *no*) is sent to the environment. In this way, P systems with input and external output are devices which can be seen as black boxes, in which the user provides the data before the computation starts and the

P system sends to the environment the output in the last step of the computation. Another important feature of P systems is the non-determinism. The design of a family of recognizer P system has to consider it, because all possibilities in the non-deterministic computations have to output the same answer. This can be summarized in the following definitions (taken from [1]).

Definition 1. *A* P system with input *is a tuple* (Π, Σ, i_Π)*, where: (a)* Π *is a P system, with working alphabet* Γ*, with* p *membranes labelled by* $1, \dots, p$*, and initial multisets* $w_1, \dots, w_p$ *associated with them; (b)* Σ *is an (input) alphabet strictly contained in* Γ*; the initial multisets are over* $\Gamma - \Sigma$*; and (c)* i_Π *is the label of a distinguished (input) membrane.*

Let m be a multiset over Σ. The *initial configuration of* (Π, Σ, i_Π) *with input* m is $(\mu, w_1, \dots, w_{i_\Pi} \cup m, \dots w_p)$.

Definition 2. *A* recognizer P system *is a P system with input,* (Π, Σ, i_Π)*, and with external output such that:*

1. *The working alphabet contains two distinguished elements yes, no.*
2. *All its computations halt.*
3. *If* $\mathcal{C}$ *is a computation of* Π*, then either some object yes or some object no (but not both) must have been released into the environment, and only in the last step of the computation. We say that* $\mathcal{C}$ *is an accepting computation (respectively, rejecting computation) if the object yes (respectively, no) appears in the external environment associated to the corresponding halting configuration of* $\mathcal{C}$*.*

In the next section we present a solution to the SAT problem in linear time in the sense of the following definition.

Definition 3. *Let* $\mathcal{F}$ *be a class of recognizer P systems. We say that a decision problem* $X = (I_X, \theta_X)$ *is solvable in polynomial time by a family* $\mathbf{\Pi} = (\Pi(n))_{n\in\mathbb{N}}$*, of* $\mathcal{F}$*, and we denote this by* $X \in \mathbf{PMC}_{\mathcal{F}}$*, if the following is true:*

- *The family* $\mathbf{\Pi}$ *is polynomially uniform by Turing machines; that is, there exists a deterministic Turing machine constructing* $\Pi(n)$ *from* $n \in \mathbb{N}$ *in polynomial time.*
- *There exists a pair* (cod, s) *of polynomial-time computable functions over* I_X *such that:*
  - *for each instance* $u \in I_X$*,* $s(u)$ *is a natural number and* $cod(u)$ *is an input multiset of the system* $\Pi(s(u))$*.*
  - *the family* $\mathbf{\Pi}$ *is polynomially bounded with regard to* (X, cod, s)*; that is, there exists a polynomial function* p*, such that for each* $u \in I_X$ *every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is halting and, moreover, it performs at most,* $p(|u|)$ *steps.*
  - *the family* $\mathbf{\Pi}$ *is sound with regard to* (X, cod, s)*; that is, for each* $u \in I_X$*, if there exists an accepting computation of* $\Pi(s(u))$ *with input* $cod(u)$*, then* $\theta_X(u) = 1$*.*

– *the family* $\mathbf{\Pi}$ *is complete with regard to* (X, cod, s)*; that is, for each* $u \in I_X$*, if* $\theta_X(u) = 1$*, then every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is an accepting one.*

In the above definition we have imposed to every P system $\Pi(n)$ to be *confluent*, in the following sense: every computation of a system with the *same* input must always give the *same* answer.

It can be proved that $\mathbf{PMC}_{\mathcal{F}}$ is closed under polynomial–time reduction and complement, see [7]. In this paper we will deal with the class $\mathcal{MC}$ of recognizer P systems with membrane creation.

4 Solving SAT in linear time with membrane creation

The SAT problem is the following: *Given a boolean formula in conjunctive normal form, to determine whether or not it is satisfiable, that is, whether there exists an assignment to its variables on which it evaluates true.*

In this section we describe a family of P systems which solves it. We will address the resolution via a brute force algorithm, in the framework of recognizer P systems with membrane creation, which consists in the following phases:

– Generation and Evaluation Stage: Using membrane creation we will generate all possible assignments associated with the formula and evaluate it on each one.
– Checking Stage: In each membrane we check whether or not the formula evaluates true on the assignment associated with it.
– Output Stage: The systems sends out to the environment the right answer according to the previous stage.

Let us consider the pair function $\langle\ ,\ \rangle$ defined by $\langle n, m\rangle = ((n+m)(n+m+1)/2) + n$. This function is polynomial-time computable (it is primitive recursive and bijective from $\mathbb{N}^2$ onto $\mathbb{N}$). For any given formula, $\varphi = C_1 \wedge \cdots \wedge C_m$, with n variables and m clauses we construct a P system $\Pi(\langle n, m\rangle)$ solving it. Therefore the family presented here is

$$\mathbf{\Pi} = \{(\Pi(\langle n, m\rangle), \Sigma(\langle n, m\rangle), i(\langle n, m\rangle)) \ : \ (n, m) \in \mathbb{N}^2\}$$

For each element of the family, the input alphabet is

$$\Sigma(\langle n, m\rangle) = \{x_{i,j}, \overline{x}_{i,j} \ : \ 1 \le i \le m, 1 \le j \le n\}$$

the input membrane is $i(\langle n, m\rangle) = t$, and the P system

$$\Pi(\langle n, m\rangle) = (\Gamma(\langle n, m\rangle), \{a, t, f, 1, \ldots, m\}, \mu, w_a, w_t, R(\langle n, m\rangle))$$

is define as follows:
• Working alphabet:

$$\begin{aligned}\Gamma(\langle n, m\rangle) = \Sigma(\langle n, m\rangle) &\cup \{x_{i,j,l}, \overline{x}_{i,j,l}, z_i, z_{i,l}, r_j, r_{j,l}, d_j \ : \ l = t, f,\ 1 \le i \le n,\ 1 \le j \le m\} \\ &\cup \{yes, no, yes_i, no_j \ : \ 0 \le i \le 9, 0 \le j \le 2n+11\} \\ &\cup \{q, k_0, k_1, k_2, t_0, t_1, t_2, t_3\}\end{aligned}$$

• Initial membrane structure: $\mu = [\,[\,]_a\,]_t$
• Initial Multisets: $w_a = \{no_0\}$ $w_t = \{z_{0,t},\, z_{0,f}\}$
• The set of evolution rules, $R(\langle n, m\rangle)$, consists of the following rules (recall that λ denotes the empty string):

1. $$\left.\begin{array}{l} [z_{j,t} \to [z_{j+1}\,k_0]_t]_l \\ [z_{j,f} \to [z_{j+1}\,k_0]_f]_l \end{array}\right\}\text{ for }\begin{array}{l} l = t, f \\ j = 0, \dots, n-2 \end{array}$$

The goal of these rules is to create one membrane for each assignment to the variables of the formula. The new membrane with label t, where the object z_{j+1} is placed, represents the assignment $x_{j+1} = true$; on the other hand the new membrane with label f, where the object z_{j+1} is placed represents the assignment $x_{j+1} = false$.

2. $$\left.\begin{array}{l} [x_{ij} \to x_{i,j,t}x_{i,j,f}]_l \\ [\overline{x}_{i,j} \to \overline{x}_{i,j,t}\overline{x}_{i,j,f}]_l \\ [r_i \to r_{i,t}r_{i,f}]_l \\ [z_k \to z_{k,t}\,z_{k,f}]_l \end{array}\right\}\text{ for }\begin{array}{l} l = t, f \\ k = 0, \dots, n-1 \\ i = 1, \dots, m \\ j = 1, \dots n \end{array}$$

These rules duplicate the objects representing the formula so it can be evaluated on the two possible assignments, $x_j = true$ $(x_{i,j,t}, \overline{x}_{i,j,t})$ and $x_j = false$ $(x_{i,j,f}, \overline{x}_{i,j,f})$. The objects r_i are also duplicated $(r_{i,t}, r_{i,f})$ in order to keep track of the clauses that evaluate true on the previous assignments to the variables. Finally the objects z_k produce the objects $z_{k,t}$ and $z_{k,f}$ which will create the new membranes representing the two possible assignments for the next variable.

3. $$\left.\begin{array}{l} x_{i,1,t}[\,]_t \to [r_i]_t,\ \ \overline{x}_{i,1,t}[\,]_t \to [\lambda]_t \\ x_{i,1,f}[\,]_f \to [\lambda]_f,\ \overline{x}_{i,1,f}[\,]_f \to [r_i]_f \end{array}\right\}\text{ for } i = 1, \dots, m$$

According to these rules the formula is evaluated in the two possible assignments for the variable that is being analysed. The objects $x_{i,1,t}$ (resp. $\overline{x}_{i,1,f}$) get into the membrane labelled with t (resp. f) being transformed into the objects r_i representing that the clause number i evaluates true on the assignment $x_{j+1} = true$ (resp. $x_{j+1} = false$). On the other hand the objects $\overline{x}_{i,1,t}$ (resp. $x_{i,1,t}$) get into the membrane labelled with f (resp. t) producing no objects. This represents that these objects do not make the clause true in the assignment $x_{j+1} = true$ (resp. $x_{j+1} = false$).

4. $$\left.\begin{array}{ll} x_{i,j,t}[\,]_t \to [x_{i,j-1}]_t, & \overline{x}_{i,j,t}[\,]_t \to [\overline{x}_{i,j-1}]_t \\ x_{i,j,f}[\,]_f \to [x_{i,j-1}]_f, & \overline{x}_{i,j,f}[\,]_f \to [\overline{x}_{i,j-1}]_f \\ r_{i,t}[\,]_t \to [r_i]_t, & r_{i,f}[\,]_f \to [r_i]_f \end{array}\right\}\text{ for }\begin{array}{l} i = 1, \dots, m \\ j = 2, \dots, n \end{array}$$

In order to analyse the next variable the second subscript of the objects $x_{i,j,l}$ and $\overline{x}_{i,j,l}$ are decreased when they are sent into the corresponding membrane labelled with l. Moreover, following the last rule, the objects $r_{i,l}$ get into the new membranes to keep track of the clauses that evaluate true on the previous assignments.

5. $$\left.\begin{array}{l} [k_s \to k_{s+1}]_l \\ [k_2]_l \to \lambda \end{array}\right\}\text{ for }\begin{array}{l} l = t, f \\ s = 0, 1 \end{array}$$

The objects k_i for $i = 0, 1, 2$ are counters that dissolve membranes when they are not useful anymore during the rest of the computation.

6. $$\left.\begin{array}{l}[z_{n-1,t} \rightarrow [z_n]_t]_l,\ \ [z_{n-1,f} \rightarrow [z_n]_f]_l \\ [z_n \rightarrow d_1 \ldots d_m q]_l\end{array}\right\} \text{ for } l = t, f$$

At the end of the generation stage the objects $z_{n-1,l}$ create two new membranes where the formula will be evaluated on the two possible assignments for the last variable x_n. The object z_n is placed in both membranes and will produce the objects $d_1, \ldots, d_m$ and yes_0, which will take part in the checking stage.

7. $$\left.\begin{array}{l}[d_i \rightarrow [t_0]_i]_l \\ r_{i,t}[\,]_i \rightarrow [r_i]_i,\ [r_i]_i \rightarrow \lambda \\ [t_s \rightarrow t_{s+1}]_i,\ \ [t_2]_i \rightarrow t_3\end{array}\right\} \text{ for } \begin{array}{l} i = 1, \ldots, m \\ s = 0, 1\end{array}$$

Following these rules each object d_i creates a new membrane with label i where the object t_0 is placed; this object will act as a counter. The object r_i gets into the membrane labelled with i and dissolves it preventing the counter, t_i, from reaching the object t_2. The fact that the object t_2 appears in a membrane with label i means that there is no object r_i, that is, the clause number i does not evaluate true on the assignment associated with the membrane; therefore neither does the formula evaluate true on the associated assignment.

8. $$\left.\begin{array}{ll}[q \rightarrow [yes_0]_a]_l & \\ t_3[\,]_a \rightarrow [t_3]_a & [t_3]_a \rightarrow \lambda \\ [yes_h \rightarrow yes_{h+1}]_a,\ [yes_5]_a \rightarrow yes_6 & \\ [yes_6]_l \rightarrow yes_7[\,]_l & \end{array}\right\} \text{ for } \begin{array}{l} l = t, f \\ h = 0, \ldots, 4\end{array}$$

The object q creates a membrane with label a where the object yes_0 is placed. The object yes_h evolves to the object yes_{h+1}; at the same time the objects t_3 can get into the membrane labelled with a and dissolve it preventing the object yes_6 from being sent out from this membrane.

9. $$\left.\begin{array}{l}[no_p \rightarrow no_{p+1}]_a,\ \ \ [no_{2n+10}]_a \rightarrow no_{2n+11} \\ [no_{2n+11}]_t \rightarrow no[\,]_t \\ yes_7[\,]_a \rightarrow [yes_8]_a,\ [yes_8]_a \rightarrow yes_9 \\ [yes_9]_t \rightarrow yes[\,]_t\end{array}\right\} \text{ for } p = 0, \ldots, 2n+9$$

From the beginning of the computation the object no_p evolves to the object no_{p+1} inside the membrane labelled with a. If any object yes_7 is produced during the computation, which means that the formula evaluates true on some assignment to its variables, it gets into this membrane and dissolved it producing the object yes_9 that will send out to the environment the object yes. On the other hand if no object yes_7 appears in the skin the object no_{2n+10} will dissolve the membrane labelled with a producing the object no_{2n+11} that will send out to the environment the object no.

4.1 An overview of the computation

First of all we define a polynomial encoding of the SAT problem in the family $\mathbf{\Pi}$ constructed in the previous section. Given a formula in CNF, $\varphi = C_1 \wedge \cdots \wedge C_m$ such that $Var(\varphi) = \{x_1, \ldots, x_n\}$ we define $s(\varphi) = \langle n, m \rangle$ (recall the bijection mentioned in the previous section) and the input multiset $cod(\varphi) = \{x_{i,j} \ : \ x_j \in C_i\} \cup \{\overline{x}_{i,j} \ : \ \neg x_{i,j} \in C_i\}$.

Next we describe informally how the recognizer P system with membrane creation $\Pi(s(\varphi))$ with input $cod(\varphi)$ works.

In the initial configuration we have on the one hand the input multiset $cod(\varphi)$ and the objects $z_{0,t}$ and $z_{0,f}$ placed in the skin (membrane labelled with t); and on the other hand we have in the membrane labelled with a the object no_0. This object evolves during the computation following the first rule in the set 9.

In the first step of the computation the object $z_{0,t}$ creates a new membrane with label t which represents the assignment $x_1 = true$ and the object $z_{0,f}$ creates a new membrane with label f which represents the assignment $x_1 = false$. In these two new membranes the objects z_1 and k_0 are placed. At the same time the input multiset representing the formula is duplicated following the two first rules in 2. In the next step, according to the rules in 3, the formula is evaluated on the two possible assignments for x_1. In the same step the rules in 4 decrease the second subscript of the objects representing the formula ($x_{i,j,l}, \overline{x}_{i,j,l}$ with $j \geq 2$) in order to analyse the next variable. Moreover, at the same time, the object z_1 produces the object $z_{1,t}$ and $z_{1,f}$ and the system is ready to analyse the next variable. And so the generation and evaluation stages goes until all the possible assignments to the variables are generated and the formula is evaluated on each one of them. Observe that it takes two steps to generate the possible assignments for a variable and evaluate the formula on them; therefore the generation and evaluation stages take $2n$ steps. Note that the object k_0 in the rules 5 is a counter that dissolves the membrane when the object k_2 appears; that is it dissolves the membrane once the membrane is not useful anymore in the rest of the computation.

The checking stage starts when the object z_n produces the objects $d_1, \ldots, d_m$ and the object q. In the first step of the checking stage each object d_i, for $i = 1, \ldots, m$ creates a new membrane labelled with i where the object t_0 is placed, and the object q creates a new membrane with label a placing the object yes_0 in it. The objects r_i, which represent that the clause number i evaluates true on the assignment associated with the membrane, are sent into the membranes by the last rule in 4 so the system keeps track of the clauses that are true. The objects $r_{i,t}$ get into the membrane with label i and dissolves it in the following two steps preventing the counter t_2 from dissolving the membrane and producing the object t_3 according to the last rule in 7. If for some i there is no object r_i, which means that the clause i does not evaluate true on the associated assignment, the object t_2 will dissolve the membrane labelled with i producing the object t_3 that will get into the membrane with label a where the object yes_h evolves following the rules in 8. The object t_3 dissolves the membrane preventing the production of the object yes_6. Therefore the checking stage takes 6 steps.

Finally the output stage takes place according to the rules in 9. On the one hand if some object yes_6 is present in any membrane (which represents that the formula evaluates true on the assignment associated with this membrane) it is sent out to the skin being transformed into the object yes_7. In the next step yes_7 gets into the membrane labelled with a being transformed into yes_8 then it dissolves the membrane producing the object yes_9. This dissolution prevents the object no_{2n+11} from being produced. And finally the object *yes* is sent out to the environment. On the other hand if there is no object yes_6 the membrane with label a is not dissolved, and then the object no_{2n+11} is produced and the object *no* is sent out to the environment. Observe that the output stage takes 5 steps if the answer is yes, and 6 steps if the answer is no.

5 Some formal details

In the previous section we have presented a family $\mathbf{\Pi}$ of recognizer P systems which solves the SAT problem. For each boolean formula a P system $\Pi(\langle n, m\rangle)$ is constructed, where n is the number of variables and m is the number of clauses. First of all, observe that the evolution rules of $\Pi(s(\varphi))$ are defined in a recursive manner from φ, in particular from n and m. Let us list the necessary resources to construct $\Pi(s(\varphi))$:

– Size of the alphabet: $6nm + 5n + 4m + 33 \in \Theta(nm)$
– Initial number of membranes: $2 \in \Theta(1)$
– Initial number of objects: $3 \in \Theta(1)$
– Sum of the lengths of the rules: $86nm + 84n + 144m + 121 \in \Theta(nm)$

Therefore a Turing machine can build $\Pi(s(\varphi))$ in polynomial time with respect to $s(\varphi)$. It can also be proved that the family $\mathbf{\Pi}$ solves the SAT problem in the sense of definition 3 in section 3.

Finally, we can prove, using a formal description of the computation, that the P system always halts and sends to the environment the object *yes* or *no* in the last step. The number of steps of the P system is $2n + 11$ if the output is *yes* and $2n + 12$ if the output is *no*, therefore there exists a linear bound for the number of steps of the computation.

From the above discussion we deduce that SAT belongs to $\mathbf{PMC}_{\mathcal{MC}}$, therefore this class is closed under polynomial-time reduction and complement we have $\mathbf{NP} \cup \mathbf{co\text{-}NP} \subseteq \mathbf{PMC}_{\mathcal{MC}}$.

6 Conclusions and Future Work

Membrane Computing is a new cross-disciplinary field of Natural Computing which has reached an important success in its short life. In these years many results have been presented related to the computational power of membrane devices, but up to now no implementation *in vivo* or *in vitro* has been carried out. This paper deals with the study of *algorithms* to solve well-known problems and in this sense it is placed between the theoretical results, mainly related to computational completeness and computational efficiency, and the real implementation of the devices. Moreover this paper represents a new step in the study of algorithms in the framework of P systems because it exploits membrane creation (a variant poorly studied) to solve **NP**-complete problems. Recently, we have proved that this variant is PSPACE powerful; this result will be published in a forthcoming paper.

References

1. Gutiérrez-Naranjo, M.A.; Pérez-Jiménez, M.J.; Riscos-Núñez,A.: Towards a programming language in cellular computing. Proceedings of the 11th Workshop on Logic, Language, Information and Computation (WoLLIC'2004), July 19-22, 2004, 1-16 Campus de Univ. Paris 12, Paris, France.
2. Gutiérrez-Naranjo, M.A.; Pérez-Jiménez, M.J.; Riscos-Núñez, A.: A fast P system for finding a balanced 2-partition, Soft Computing, in press.
3. Luisi, P.L.: The Chemical Implementation of Autopoiesis, *Self-Production of Supramolecular Structures* (G.R. Fleishaker et al., eds.), Kluwer, Dordrecht, 1994
4. Pérez-Jiménez, M.J.; Riscos-Núñez, A.: Solving the Subset-Sum problem by active membranes, *New Generation Computing*, in press.
5. Pérez-Jiménez, M.J.; Riscos-Núñez, A.: A linear solution for the Knapsack problem using active membranes, *Membrane Computing*, C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg and A. Salomaa (eds.), Lecture Notes in Computer Science, **2933**, 2004, 250–268.
6. Pérez-Jiménez, M.J.; Romero-Campero, F.J.: Solving the BIN PACKING problem by recognizer P systems with active membranes, *Proceedings of the Second Brainstorming Week on Membrane Computing*, Gh. Păun, A. Riscos, A. Romero and F. Sancho (eds.), Report RGNC 01/04, University of Seville, 2004, 414–430.
7. Pérez-Jiménez, M.J.; Romero-Jiménez, A.; Sancho-Caparrini, F.: A polynomial complexity class in P systems using membrane division, *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003*, E. Csuhaj-Varjú, C. Kintala, D. Wotschke and Gy. Vaszyl (eds.), 2003, 284-294.
8. Păun, A.; Păun, Gh.: The power of communication: P systems with symport/antiport, *New Generation Computing*, **20**, 3 (2002), 295–305
9. Păun, Gh.: Computing with membranes, *Journal of Computer and System Sciences*, **61**, 1 (2000), 108–143.
10. Păun, Gh.; Pérez-Jiménez, M.J.: Recent computing models inspired from biology: DNA and membrane computing, *Theoria*, **18**, 46 (2003), 72–84.
11. Păun, Gh.: Further Open Problems in Membrane Computing, *Proceedings of the Second Brainstorming Week on Membrane Computing*, Gh. Păun, A. Riscos, A. Romero and F. Sancho (eds.), Report RGNC 01/04, University of Seville, 2004, 354–365.
12. ISI web page `http://esi-topics.com/erf/october2003.html`
13. P systems web page `http://psystems.disco.unimib.it/`

Polynomial Time Enumeration Reducibility

Charles Harris

Department of Pure Mathematics,
University of Leeds,
Leeds, LS2 9JT, U.K.
http://perso.wanadoo.fr/charles.harris/
harris.charles@gmail.com

Polynomial time enumeration (pe) reducibility was defined by Alan Selman in 1978 as a polynomial time restricted analogue of enumeration reducibility. Selman showed that, over the class of computable sets, pe-reducility properly contains polynomial time non deterministic conjunctive (np-c) reducibility and that it gives rise to a degree structure $(\text{REC-pe}, \leq)$ whose zero element is NP. Unusually, there is no (known) effective listing of operators for this (i.e. pe) reducibility and so the properties of its degree structure are not directly provable using standard techniques for resource bounded reducibilities. Nevertheless a number of fundamental properties — for example density and the existence of certain lattice embeddings — can be established by a slight change of approach applied in conjunction with two basic results proved in Selman's original paper. Interestingly pe-reducibility and np-c-reducibility are equivalent over the class EXP. This suggest that the associated set of degrees EXP-pe (which is countably infinite provided EXP is different from NP), has a special status within the overall structure. On the other hand $(\text{REC-pe}, \leq)$ shares various characteristics of the structure of the Cook degrees; in particular, a similar lack of homogeneity. My present work seeks to determine the extent of this analogy between the two structures. This includes the question of whether certain sets of degrees, such as EXP-pe, are definable within $(\text{REC-pe}, \leq)$, and which fragments of the latter's theory are decidable.

On the Prediction of Recursive Real-Valued Functions [⋆]

Eiju Hirowatari[1], Kouichi Hirata[2], Tetsuhiro Miyahara[3], and Setsuo Arikawa[4]

[1] Department of Business Administration, 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

[4] Department of Informatics, Kyushu University, Fukuoka 812-8581, Japan
arikawa@i.kyushu-u.ac.jp

Abstract. In this paper, we investigate *prediction* of recursive real-valued functions. First, we introduce a new criterion RealNV for prediction of recursive real-valued functions, as an extension of the criterion NV for prediction of recursive functions. Then, we show that the set of all elementary functions with rational coefficients defined on a fixed rational closed interval is in RealNV. Furthermore, by comparing RealNV with other criteria RealEx, RealFin and RealNum! for inductive inference of recursive real-valued functions, we show that RealNV # (RealFin $\cap$ RealNum!) and RealNV # RealEx, in contrast to the well-known relationship that Num! $\subsetneq$ NV $\subsetneq$ Ex and NV # Fin, where $A \# B$ denotes that A and B are incomparable.

1 Introduction

The *prediction* or *extrapolation* of recursive functions [3, 4] is to predict the n-th element of the sequence given the first $n-1$ elements by a prediction machine. We call the predicted element a *next value*. 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.

The criterion NV (abbreviates 'next value') for prediction of recursive functions has been introduced by Bārzdiņš and Freivalds [3]. Then, by comparing NV with other criteria for inductive inference of recursive functions, Ex [5] corresponding to identification in the limit, Fin [5] to learning finitely, and Num! [2, 3] to learning by enumeration, the following relationships [3, 4] are

[⋆] This work is partially supported by Grant-in-Aid for Scientific Research 14780304, 15700137 and 16016275 from the Ministry of Education, Culture, Sports, Science and Technology, Japan, and 16500084 from Japan Society for the Promotion of Science.

known: NUM! $\subsetneq$ NV $\subsetneq$ EX and NV # FIN, where $A \# B$ denotes that A and B are incomparable. Also refer to Figure 1 (left).

On the other hand, a *recursive real-valued function* is one of the formulations for a computable real function [10–12], and formulated as a function that maps a sequence of intervals which converges to a real number to a sequence of intervals which converges to another real number. Inductive inference of recursive real-valued functions was first introduced by Hirowatari and Arikawa [6] and developed by their co-authors [1, 7–9]. In their works, the criteria REALEX, REALFIN and REALNUM! for inductive inference of recursive real-valued functions were introduced as extensions of EX, FIN and NUM!, respectively, and their interaction has been widely studied.

In this paper, we investigate prediction of *recursive real-valued functions*. First, we introduce a new criterion REALNV for prediction of recursive real-valued functions as an extension of NV. Since every recursive real-valued function is defined on a set of rational closed intervals, we assume the ordered *interval sequence* to define a next value in REALNV. Then, we show that the criterion REALNV is rich enough to predict the elementary functions with rational coefficients defined on a fixed rational closed interval. Furthermore, by comparing REALNV with other criteria REALEX, REALFIN and REALNUM!, we show the interaction of their criteria as Figure 1.

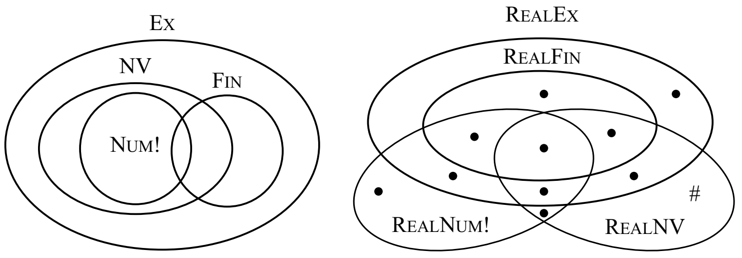


Fig. 1. The interaction of criteria for prediction and inductive inference of recursive functions (left) and of recursive real-valued functions (right). We will show in this paper the existence of the function in the place marked by •.

2 Recursive Real-Valued Functions

In this section, we prepare some notions for *recursive real-valued functions*, which is one of the formulations for a computable real function [10–12]. 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 $[a, b]$ (*resp*, (a, b)), we denote a *closed* (*resp.*, *open*) interval for each $a, b \in R$ such that $a < b$.

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) $\lim_{n\to\infty} g(n) = 0$.
(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.

In order to formulate a *recursive real-valued function*, we introduce the concepts of a *rationalized domain* and a *rationalized function*.

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 $\langle f(n), g(n)\rangle \in RD_h$.

We also call a rationalized function $\mathcal{A}_h$ of h an *algorithm which computes 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*.

Without loss of generality, we can assume that, for each real-valued function h, RD_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 $\langle p, \alpha\rangle \in Q \times Q^+$ such that $[p - \alpha, p + \alpha] \subseteq dom(h)$.

Definition 5. Let h and h_0 be recursive real-valued functions such that $dom(h) = S$ and $dom(h_0) = S_0$. Then, we say that h_0 is a *restriction* of h or h is an *extension* of h_0, denoted by $h_0 = h|_{S_0}$, if $S_0 \subseteq S$ and $h_0(x) = h(x)$ for each $x \in S_0$. Furthermore, for the set $\mathcal{T}$ of recursive real-valued functions, we call a restriction of a function in $\mathcal{T}$ a *restriction* in $\mathcal{T}$ simply.

We say that a set $\mathcal{T} \subseteq \mathcal{RRVF}$ is *recursively enumerable* if there is a recursive function Ψ such that a set of extensions of functions in $\mathcal{T}$ is equal to the set of all functions computed by algorithms $\Psi(0), \Psi(1), \cdots$.

3 Prediction of Recursive Real-Valued Functions

In this section, we introduce the criterion RealNV for the prediction of recursive real-valued functions, together with the criteria RealEx, RealFin and RealNum! [7–9] for inductive inference of recursive real-valued functions.

As an approximation of a real number x, we deal with a pair $\langle p, \alpha \rangle$ 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 6. An *example* of a 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.

We can imagine an example of h as a rectangular box $[p - \alpha, p + \alpha] \times [q - \beta, q + \beta]$. Then, a sequence $w_1, w_2, \ldots$ of such boxes is a presentation of h if each box contains a point $(x, h(x))$ on the graph of h, and for each point on the graph, there are arbitrarily small boxes w_k having the point in their interior.

Definition 7. For $S \subseteq R$, an *interval sequence* $I(S) = I_1, I_2, \ldots$ of S is an infinite sequence of rational closed intervals satisfying the following condition: For each $x \in S$ and $\epsilon \in Q^+$, there exists an $n \in N^+$ such that $x \in I_n$ and $I_n \subseteq [x - \epsilon, x + \epsilon]$.

For each recursive real-valued function h, there always exists an interval sequence $I(dom(h)) = I_1, I_2, \ldots$ of $dom(h)$. Then, we denote $I(dom(h))$ by $I(h)$ simply.

Definition 8. A *presentation* of a function $h : S \to R$ ($S \subseteq R$) is an infinite sequence $\sigma = w_1, w_2, \ldots$ of examples of h in which, for each real number x in the domain of 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 $\sigma[n]$, $\sigma[0]$ and $\sigma(n)$, we denote the initial segment of n examples in σ, an empty sequence and the n-th element of σ, respectively.

For each recursive real-valued function h and interval sequence $I(h) = I_1, I_2, \ldots$ of $dom(h)$, let $W_{I(h)} = w_1, w_2, \ldots$ be a presentation of h such that $w_i = \langle \langle p_i, \alpha_i \rangle, \langle q_i, \beta_i \rangle \rangle$ and $I_i = [p_i - \alpha_i, p_i + \alpha_i]$ ($i \in N^+$). We call $W_{I(h)}$ a *presentation of h w.r.t. $I(h)$*.

A *prediction machine* (*PM*, for short) is a procedure that requests a finite sequence of examples of a recursive real-valued function and a datum of a real number, and that may after some time delay output a datum of a real number. For a PM $\mathcal{N}$ and a finite sequence W of examples of a recursive real-valued function, by $\mathcal{N}(W)$, we denote the output $\langle\langle p, \alpha\rangle, \langle q, \beta\rangle\rangle \in (Q \times Q^+) \times (Q \times Q^+)$ of $\mathcal{N}$ after requesting W as inputs. In this paper, we assume that $\mathcal{N}(W)$ is defined for each finite sequence W of examples of recursive real-valued function. Let π denote a *projection* $(Q \times Q^+) \times (Q \times Q^+) \to Q \times Q^+$ such that $\pi(\langle\langle p, \alpha\rangle, \langle q, \beta\rangle\rangle) = \langle p, \alpha\rangle$.

Definition 9. Let h be a recursive real-valued function, $I(h) = I_1, I_2, \ldots$ an interval sequence of $dom(h)$ and $W_{I(h)}$ a presentation of h w.r.t. $I(h)$. For a PM $\mathcal{N}$, we say that $\mathcal{N}(W_{I(h)}[n])$ is a *next value of* h *w.r.t.* $I(h)$ if $\mathcal{N}(W_{I(h)}[n])$ is an example of h and $I_{n+1} = [p - \alpha, p + \alpha]$, where $\langle p, \alpha\rangle = \pi(\mathcal{N}(W_{I(h)}[n]))$.

Example 1. For $a, b \in Q$ such that $a < b$, let $\mathcal{T}^{[a,b]}$ be the set of all recursive real-valued functions h such that $a \leq h(x)$ and $h(x) \leq b$ for each $x \in dom(h)$.

For each $h \in \mathcal{T}^{[a,b]}$ and $x \in R$, $h(x)$ is either undefined or satisfies that $a \leq h(x)$ and $h(x) \leq b$. Let $I(h) = I_1, I_2, \ldots$ be an interval sequence of $dom(h)$ with $I_n = [p_n - \alpha_n, p_n + \alpha_n]$ for each $n \in N^+$, and $W_{I(h)} = I_1, I_2, \ldots$ be a presentation of h w.r.t. $I(h)$. Consider a PM $\mathcal{N}$ such that $\mathcal{N}(W_{I(h)}[n]) = \langle\langle p_{n+1}, \alpha_{n+1}\rangle, \langle \frac{a+b}{2}, \frac{b-a}{2}\rangle\rangle$ for each $n \in N^+$. Then, $\mathcal{N}$ always outputs a next value of h w.r.t. $I(h)$, that is, $\mathcal{N}(W_{I(h)}[n])$ is an example of h for each $n \in N^+$.

Hence, in order to formulate prediction of recursive real-valued functions, it is necessary to introduce not only a next value but also the condition for an error bound, which is corresponding to the statement (2) in Definition 10.

Definition 10. Let h be a recursive real-valued function. A PM $\mathcal{N}$ RealNV*-predicts* h, denoted by $h \in \text{RealNV}(\mathcal{N})$, if, for each interval sequence $I(h) = I_1, I_2, \ldots$ of $dom(h)$ with $I_k = [p_k - \alpha_k, p_k + \alpha_k]$ $(k \in N^+)$ and presentation $W_{I(h)}$ of h w.r.t. $I(h)$, there exists an algorithm $\mathcal{A}_h$ which computes an extension of h, satisfying the following condition: There exists a number $n_0 \in N$ such that the following conditions (1) and (2) hold for each $n \geq n_0$.

(1) $\mathcal{N}(W_{I(h)}[n])$ is a next value of h w.r.t. $I(h)$.
(2) If $\langle p_{n+1}, \alpha_{n+1}\rangle$ is in a rationalized domain RD_h of $dom(h)$, then it holds that $\mathcal{N}(W_{I(h)}[n]) = \langle\langle p_{n+1}, \alpha_{n+1}\rangle, \mathcal{A}_h(\langle p_{n+1}, \alpha_{n+1}\rangle)\rangle$.

Also let $\mathcal{T}$ be a set of recursive real-valued functions. Then, a PM $\mathcal{N}$ RealNV-*predicts* $\mathcal{T}$ if $\mathcal{N}$ RealNV-predicts every $h \in \mathcal{T}$, and $\mathcal{T}$ is RealNV*-predictable* if there exists a PM that RealNV-predicts $\mathcal{T}$. By RealNV, we denote *the class of all* RealNV*-predictable sets of recursive real-valued functions*.

Furthermore, we introduce the criteria for inductive inference of recursive real-valued functions [7–9].

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 11. 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 $\mathcal{A}_h$ if there exists a number $n_0 \in N$ such that $\mathcal{M}(\sigma[m])$ equals $\mathcal{A}_h$ for each $m \geq n_0$.

Definition 12. Let h be a recursive real-valued function and $\mathcal{T}$ a set of recursive real-valued functions.

(1) An IIM $\mathcal{M}$ RealEx-*infers* h, denoted by $h \in$ 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.
(2) An IIM $\mathcal{M}$ RealFin-*infers* h, denoted by $h \in$ 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.

Furthermore, let $\mathcal{T}$ be a set of recursive real-valued functions and $X \in \{\text{Ex}, \text{Fin}\}$. 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*.

Also by RealNum! [7], we denote *the class of all recursively enumerable sets of recursive real-valued functions*.

Example 2. Let $\mathcal{T}$ be the set of all polynomial functions defined on R with rational coefficients. Note that each function $h \in \mathcal{T}$ is defined by $h(x) = a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n$, where $n \in N$, $a_0, a_1, \ldots, a_n \in Q$.

Since $\mathcal{T}$ is recursively enumerable, $\mathcal{T}$ is RealEx-inferable [7]. Then, there exists an IIM $\mathcal{M}$ that RealEx-infers $\mathcal{T}$. Without loss of generality, $\mathcal{M}$ outputs for each input, and $\mathcal{M}(\sigma[n])$ is an algorithm which computes a function in $\mathcal{T}$ for each presentation σ of a function and $n \in N$.

For each $h \in \mathcal{T}$, let $I(h) = I_1, I_2, \ldots$ be an interval sequence of $dom(h)$ with $I_k = [p_k - \alpha_k, p_k + \alpha_k]$ for each $k \in N^+$, and $W_{I(h)}$ be a presentation of h w.r.t. $I(h)$. By using $\mathcal{M}$, for each $n \in N^+$, we can construct the PM $\mathcal{N}$ such that $\mathcal{N}(W_{I(h)}[n]) = \langle\langle p_{n+1}, \alpha_{n+1}\rangle, \mathcal{M}(W_{I(h)}[n])(\langle p_{n+1}, \alpha_{n+1}\rangle)\rangle$. Then, $\mathcal{N}$ RealNV-predicts $\mathcal{T}$.

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.

Definition 13. For the set $\mathcal{BF}$ of all basic functions, $\mathcal{EF}$ is 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 functions in $\mathcal{EF}$ *elementary functions*.

All elementary functions are recursive real-valued functions. Let $\mathcal{EF}_I$ be the set of all elementary functions with rational coefficients defined on a fixed rational closed interval I. Then, $\mathcal{EF}_I$ is recursively enumerable.

Theorem 1. *Let $\mathcal{T}_I$ be a recursively enumerable set of recursive real-valued functions defined on a fixed rational closed interval I. Then, $\mathcal{T}_I$ is* RealNV*-predictable.*

Proof. Since $\mathcal{T}$ is recursively enumerable, $\mathcal{T}$ is RealEx-inferable [7]. Then, there exists an IIM $\mathcal{M}$ such that $h \in \text{RealEx}(M)$ for each $h \in \mathcal{T}_I$.

For each $h \in \mathcal{T}_I$, let $I(h) = I_1, I_2, \ldots$ be an interval sequence of $dom(h)$ with $I_k = [p_k - \alpha_k, p_k + \alpha_k]$ for each $k \in N^+$, and $W_{I(h)}$ be a presentation of h w.r.t. $I(h)$. By using $\mathcal{M}$, for each $n \in N^+$, we can construct the following PM $\mathcal{N}$ that RealNV-predicts $\mathcal{T}_I$.

$$\mathcal{N}(W_{I(h)}[n]) = \begin{cases} \langle\langle p_{n+1}, \alpha_{n+1}\rangle, \mathcal{M}(\sigma[n])(\langle p_{n+1}, \alpha_{n+1}\rangle)\rangle \\ \qquad\qquad\qquad\qquad \text{if } [p_{n+1} - \alpha_{n+1}, p_{n+1} + \alpha_{n+1}] \subseteq I, \\ \langle\langle p_{n+1}, \alpha_{n+1}\rangle, \mathcal{M}(\sigma[n])(\langle r, \gamma\rangle)\rangle \\ \qquad\qquad\qquad\qquad \text{otherwise,} \end{cases}$$

where $\langle r, \gamma\rangle$ is an element in $Q \times Q^+$ such that $I = [r - \gamma, r + \gamma]$. Hence, it holds that $\mathcal{T}_I \in \text{RealNV}$, so the statement holds. □

Corollary 1. $\mathcal{EF}_I \in \text{RealNV}$.

Hence, we can conclude that the criterion RealNV is rich enough to predict every elementary function with rational coefficients defined on a fixed rational closed interval.

4 Interaction of the Criteria

In this section, we investigate the interaction of the criteria RealNV, RealEx, RealFin and RealNum!.

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 $\varphi \in \mathcal{R}$, the following function $h : [0, \infty) \to R$ is called the *line function* of φ.

$$h(x) = (\varphi(i+1) - \varphi(i))x + \varphi(i)(i+1) - \varphi(i+1)i.$$

For $\mathcal{T} \subseteq \mathcal{R}$, we call a line function of a function in $\mathcal{T}$ a *line function* in $\mathcal{T}$ simply.

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. $(\textsc{RealNV} \cap \textsc{RealNum!}) \setminus \textsc{RealEx} \neq \emptyset$.

Proof. For each $i \in N$, we construct the following three recursive real-valued functions h_i, $\hat{h}_i$ and $\hat{h}$.

$$h_i(x) = \begin{cases} 1 & \text{if } x \leq 0, \\ 0 & \text{if } x > \frac{1}{2^i}. \end{cases} \quad \hat{h}_i(x) = \begin{cases} 1 & \text{if } x < 0, \\ 0 & \text{if } x > \frac{1}{2^i}. \end{cases} \quad \hat{h}(x) = \begin{cases} 1 & \text{if } x < 0, \\ 0 & \text{if } x > 0. \end{cases}$$

Suppose that $\mathcal{H} = \{\hat{h}, h_0, h_1, h_2, \ldots\}$. Now let $\hat{\sigma}_i = \hat{w}^i_1, \hat{w}^i_2, \ldots$ be a presentation of $\hat{h}_i$ with $\hat{w}^i_{2j-1} = \langle\langle 0, \frac{1}{2^j}\rangle, \langle 1, \frac{1}{2^j}\rangle\rangle$ such that $j \in N^+$. Then, $\hat{\sigma}_i$ is also a presentation of h_i for each $i \in N$. For each $i \in N$, we can construct the following presentation $\sigma_i = w^i_1, w^i_2, w^i_3, \ldots$ of h_i.

$$w^i_j = \begin{cases} \hat{w}^0_j & \text{if } i = 0, \\ w^{i-1}_j & \text{if } i > 0 \text{ and } 1 \leq j \leq n_{i-1}, \\ w^{i-1}_{n_i+m} & \text{if } i > 0 \text{ and } j = n_{i-1} + 2m, \\ \hat{w}^i_m & \text{if } i > 0 \text{ and } j = n_{i-1} + 2m - 1. \end{cases}$$

Here, $m \in N^+$, and n_i is the least natural number such that $\mathcal{M}(\sigma_i[n]) = \mathcal{M}(\sigma_i[n_i])$ for each $n \geq n_i$. Let $\sigma_* = \lim_{i\to\infty} \sigma_i$. Since σ_i is also a presentation of $\hat{h}_i$ for each $i \in N$, σ_* is a presentation of $\hat{h}$. By the definition of σ_*, $\{M(\sigma_*[n])\}_{n\in N}$ does not converge. Then, $\mathcal{H} \notin \textsc{RealEx}$. On the other hand, it is obvious that $\mathcal{H} \in \textsc{RealNum!}$.

For each $h \in \mathcal{H}$, let $I(h) = I_1, I_2, \ldots$ be an interval sequence of $dom(h)$ with $I_k = [p_k - \alpha_k, p_k + \alpha_k]$ for each $k \in N^+$, and $W_{I(h)}$ be a presentation of h w.r.t. $I(h)$. We can construct the following PM $\mathcal{N}$ that RealNV-predicts $\mathcal{H}$.

$$\mathcal{N}(W_{I(h)}[n]) = \begin{cases} \langle\langle p_{n+1}, \alpha_{n+1}\rangle, \langle 1, \alpha_{n+1}\rangle\rangle & \text{if } p_{n+1} - \alpha_{n+1} \leq 0, \\ \langle\langle p_{n+1}, \alpha_{n+1}\rangle, \langle 0, \alpha_{n+1}\rangle\rangle & \text{otherwise,} \end{cases}$$

where $n \in N$. Then, it holds that $\mathcal{H} \in \textsc{RealNV}$, so the statement holds. □

Theorem 3. $(\textsc{RealNV} \cap \textsc{RealFin}) \setminus \textsc{RealNum!} \neq \emptyset$.

Proof. For a set $M \subsetneq N$ that is not recursively enumerable, let $\mathcal{C}_M$ be the set of all constant functions $c_m : [0,1] \to M$ such that $c_m(x) = m$ for each $m \in M$. Then, it holds that $\mathcal{C}_M \in \textsc{RealFin} \setminus \textsc{RealNum!}$.

For each $h \in \mathcal{C}_M$, let $I(h) = I_1, I_2, \ldots$ be an interval sequence of $dom(h)$ with $I_k = [p_k - \alpha_k, p_k + \alpha_k]$ for each $k \in N^+$, and $W_{I(h)}$ be a presentation of h w.r.t. $I(h)$. We can construct a PM $\mathcal{N}$ such that, for each $n \in N$,

$$\mathcal{N}(W_I I(h)[n]) = \langle\langle p_{n+1}, \alpha_{n+1}\rangle, \langle s, \alpha_{n+1}\rangle\rangle,$$

where $s \in N$ is the least natural number such that $q_i - \beta_i \leq s$ for each $i \in N$ $(1 \leq i \leq n)$. Then, $\mathcal{N}$ RealNV-predicts $\mathcal{C}_M$, so $\mathcal{C}_M \in \textsc{RealNV}$. □

Theorem 4. $(\textsc{RealNV} \cap \textsc{RealNum!} \cap \textsc{RealEx}) \setminus \textsc{RealFin} \neq \emptyset$.

Proof. For a $\varphi \in \mathcal{R}$, $\varphi^{-1}(0)$ denotes the set $\{n \in N \mid \varphi(n) = 0\}$. For a set $M \subseteq N$, $\|M\|$ denotes the cardinality of M. Then, let $\mathcal{R}_{\{0,1\}}$, $\mathcal{R}^m_{\{0,1\}}$ and $\mathcal{R}^*_{\{0,1\}}$ be the following sets.

$$\begin{aligned}\mathcal{R}_{\{0,1\}} &= \{\varphi : N \to \{0,1\} \mid \varphi \in \mathcal{R}\},\\ \mathcal{R}^m_{\{0,1\}} &= \{\varphi \in \mathcal{R}_{\{0,1\}} \mid \|\varphi^{-1}(0)\| \le m\},\\ \mathcal{R}^*_{\{0,1\}} &= \cup_{m \in N} \mathcal{R}^m_{\{0,1\}}.\end{aligned}$$

Let $\mathcal{T}^*_{\{0,1\}}$ be the set of all line functions in $\mathcal{R}^*_{\{0,1\}}$. Then, it holds that $\mathcal{T}^*_{\{0,1\}} \in (\textsc{RealEx} \cap \textsc{RealNum!}) \setminus \textsc{RealFin}$ [8], so there exists an IIM $\mathcal{M}$ that RealEx-infers $\mathcal{T}^*_{\{0,1\}}$. Without loss of generality, $\mathcal{M}$ outputs for each input, and $\mathcal{M}(\sigma[n])$ is an algorithm which computes a function in $\mathcal{T}^*_{\{0,1\}}$ for each presentation σ of a function and $n \in N$.

For each $h \in \mathcal{T}^*_{\{0,1\}}$, let $I(h) = I_1, I_2, \ldots$ be an interval sequence of $dom(h)$ with $I_k = [p_k - \alpha_k, p_k + \alpha_k]$ for each $k \in N^+$, and $W_{I(h)}$ be a presentation of h w.r.t. $I(h)$. We can construct the following PM $\mathcal{N}$ that RealNV-predicts $\mathcal{T}^*_{\{0,1\}}$.

$$\mathcal{N}(W_{I(h)}[n], \langle p, \alpha\rangle) = \begin{cases} \langle\langle p_{n+1}, \alpha_{n+1}\rangle, \mathcal{M}(W_{I(h)}[n])(\langle p_{n+1}, \alpha_{n+1}\rangle)\rangle \\ \qquad\qquad\qquad \text{if } p_{n+1} - \alpha_{n+1} \ge 0, \\ \langle\langle p_{n+1}, \alpha_{n+1}\rangle, \mathcal{M}(W_{I(h)}[n])(\langle \frac{\alpha_{n+1}}{2}, \frac{\alpha_{n+1}}{2}\rangle)\rangle \\ \qquad\qquad\qquad \text{otherwise,}\end{cases}$$

where $n \in N$. Hence, it holds that $\mathcal{T}^*_{\{0,1\}} \in \textsc{RealNV}$, so the statement holds. □

Theorem 5. *$(\textsc{RealNV} \cap \textsc{RealEx}) \setminus (\textsc{RealFin} \cup \textsc{RealNum!}) \neq \emptyset$.*

Proof. For each subset $F \subseteq N$, let φ_F be the following function: If $n \in F$, then $\varphi_F(n) = 0$, otherwise $\varphi_F(n) = 1$. Also let $M \subsetneq N$ be a set that is not recursively enumerable and $\mathcal{T}^*_{\{0,1\},M}$ the set of all line functions of φ_F such that F is a finite subset of M. Then, the set $\{j \in N \mid \varphi_j \in \mathcal{R}\}$ is an example of M. Furthermore, it holds that $\mathcal{T}^*_{\{0,1\},M} \notin \textsc{RealFin} \cup \textsc{RealNum!}$. For the set $\mathcal{T}^*_{\{0,1\}}$ in the proof of Theorem 4, since $\mathcal{T}^*_{\{0,1\},M} \subseteq \mathcal{T}^*_{\{0,1\}}$, it holds that $\mathcal{T}^*_{\{0,1\},M} \in \textsc{RealNV} \cap \textsc{RealEx}$. □

Theorem 6. *$(\textsc{RealFin} \cap \textsc{RealNum!}) \setminus \textsc{RealNV} \neq \emptyset$.*

Proof. Let $t(x)$ be the following function.

$$t(x) = \begin{cases} \left|\tan \pi \left(n - \frac{1}{2^m} - \frac{1}{2}\right)\right| & \text{if } x \in [n - \frac{1}{2^m}, n + \frac{1}{2^m}] \text{ for } n, m \in N^+ \\ & \text{ such that } \varphi_n(n) \text{ is defined and } \Phi_n(n) = m, \\ |\tan \pi (x - \frac{1}{2})| & \text{otherwise.}\end{cases}$$

Here, $\Phi_n(n)$ is a step counting function for computation of $\varphi_n(n)$. Then, $t(x)$ is a recursive real-valued function.

Consider the set $\{t(x)\}$. It is sufficient to show that $\{t(x)\} \notin \textsc{RealNV}$.

Suppose that $\{t(x)\} \in \textsc{RealNV}$. Then, there exists a PM $\mathcal{N}$ which RealNV-predicts $\{t(x)\}$. Let $I(t) = I_1, I_2, \ldots$ be an interval sequence of $dom(t)$ such that $I_{2^n 3^m} = [n - \frac{1}{2^m}, n + \frac{1}{2^m}]$ for each $n, m \in N^+$. Thus, there exists an $i \in N$

such that, for each $k \in N$ $(k \geq i)$ with $k = 2^n 3^m$, $\mathcal{N}(W_{I(t)}[k])$ outputs a next value $\langle\langle p_{k+1}, \alpha_{k+1}\rangle, \langle q, \beta\rangle\rangle$ of $t(x)$ and it holds that either $\Phi_n(n)$ is defined and $t([n - \frac{1}{2^m}, n + \frac{1}{2^m}]) \subseteq [q - \beta, q + \beta]$ or $\Phi_n(n)$ is not defined.

If there exists a number $m \in N^+$ such that $t(n - \frac{1}{2^m} - \frac{1}{2}) \leq q + \beta$ and $t(n - \frac{1}{2^m} - \frac{1}{2}) = t(n - \frac{1}{2^{m+1}} - \frac{1}{2})$, then it holds that $t([n - \frac{1}{2^m}, n + \frac{1}{2^m}]) \subseteq [q - \beta, q + \beta]$, that is, $\Phi_n(n) = m$, otherwise $\Phi_n(n)$ is not defined. Thus, we can determine whether or not $\varphi_n(n)$ is defined for each $n \in N$, which is a contradiction. Hence, it holds that $\{t(x)\} \notin$ RealNV. □

Theorem 6 implies the following corollary.

Corollary 2. $\mathcal{RRVF} \notin$ RealNV.

Note that $\mathcal{RRVF} \notin$ RealEx [8].

Furthermore, by incorporating Theorem 6 with our previous works [7], we can obtain the following corollary.

Corollary 3. *The following statements hold.*

(1) RealFin $\setminus$ (RealNV $\cup$ RealNum!) $\neq \emptyset$.
(2) (RealEx $\cup$ RealNum!) $\setminus$ (RealNV $\cup$ RealFin) $\neq \emptyset$.
(3) RealNum! $\setminus$ (RealNV $\cup$ RealEx) $\neq \emptyset$.
(4) RealNV $\cap$ RealFin $\cap$ RealNum! $\neq \emptyset$.
(5) RealEx $\setminus$ (RealNV $\cup$ RealFin $\cup$ RealNum!) $\neq \emptyset$.

Proof. We combine $\{t(x)\}$ in the proof of Theorem 6 to other sets of functions.

(1) Let U be the set of all recursive functions f from N to N such that $\varphi_{f(0)} = f$ and $\mathcal{T}_U$ the set of all stair functions in U. Since $\mathcal{T}_U \in$ RealFin$\setminus$RealNum! [7], it holds that $\{t(x)\} \cup \mathcal{T}_U \in$ RealFin $\setminus$ (RealNV $\cup$ 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 $r \in Q$. Since $\mathcal{C}_Q \in$ (RealEx $\cup$ RealNum!) $\setminus$ RealFin [7], it holds that $\{t(x)\} \cup \mathcal{C}_Q \in$ (RealEx $\cup$ RealNum!) $\setminus$ (RealNV $\cup$ RealFin).
(3) Note that $\mathcal{P}$ is the set of all partial recursive real-valued functions from N to N. Let $\mathcal{T}_\mathcal{P}$ be the set of all stair functions in $\mathcal{P}$. Since $\mathcal{T}_\mathcal{P} \in$ RealNum! $\setminus$ RealEx [7], it holds that $\{t(x)\} \cup \mathcal{T}_\mathcal{P} \in$ RealNum! $\setminus$ (RealNV $\cup$ RealEx).
(4) It is obvious that $\{h(x) = 0\} \in$ RealNV $\cap$ RealFin $\cap$ RealNum!.
(5) By the statements (1), (2) and (3), it holds that $\{t(x)\} \cup \mathcal{T}_U \cup \mathcal{C}_Q \in$ RealEx$\setminus$ (RealNV $\cup$ RealFin $\cup$ RealNum!).

□

5 Conclusion

In this paper, by introducing the criterion RealNV for *prediction* of recursive real-valued functions, we have shown that $\mathcal{EF}_I \in$ RealNV, where $\mathcal{EF}_I$ is the set of all elementary functions with rational coefficients defined on a fixed rational closed interval I. Then, we have obtained the interaction of RealNV and other criteria RealEx, RealFin and RealNum! as Figure 1. It is open whether or not $\textsc{RealNV} - (\textsc{RealEx} \cup \textsc{RealNum}!) = \emptyset$ (marked by # in Figure 1).

In this paper, we have not paid our attention to the domains of the target functions. It is a future work to discuss the prediction of recursive real-valued functions defined on *a fixed rational closed interval*, as similar as $\textsc{RealEx}_I$, $\textsc{RealFin}_I$ and $\textsc{RealNum}!_I$ [7], where I is a closed interval.

In prediction of recursive functions, the criteria NV' and NV'' stronger than NV have been introduced [3, 4]. In particular, NV'' coincides with the criterion BC for *behaviorally correct learning* [4]. It is a future work to formulate the criteria corresponding to NV', NV'' and BC for prediction and inductive inference of recursive real-valued functions, and investigate their interaction.

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, *Inductive inference of automata, functions and programs*, Proc. Int. Math. Congress, 771–776, 1974.
3. J. M. Bārzdiņš, R. V. Freivalds, *On the prediction of general recursive functions*, Soviet Mathematics Doklady **13**, 1224–1228, 1972.
4. J. Case, C. Smith, *Comparison of identification criteria for machine inductive inference*, Theoret. Comput. Sci. **25**, 193–220, 1983.
5. E. M. Gold, *Language identification in the limit*, Inform. Control **10**, 447–474, 1967.
6. E. Hirowatari, S. Arikawa, *Inferability of recursive real-valued functions*, Proc. 8th International Workshop on Algorithmic Learning Theory, LNAI **1316**, 1997.
7. E. Hirowatari, S. Arikawa, *A comparison of identification criteria for inductive inference of recursive real-valued functions*, Theoret. Comput. Sci. **268**, 351–366, 2001.
8. E. Hirowatari, K. Hirata, T. Miyahara, S. Arikawa, *Criteria for inductive inference with mind changes and anomalies of recursive real-valued functions*, IEICE Trans. Inf. Syt. **E86-D**, 219–227, 2003.
9. E. Hirowatari, K. Hirata, T. Miyahara, S. Arikawa, *Refutability and reliability for inductive inference on recursive real-valued functions*, IPSJ Trans. Mathematical Modeling and Its Applications, **46**, 1–11, 2005.
10. K. Ko. *Complexity theory of real functions*, Birkhäuser, 1991.
11. M. B. Pour-El, J. I. Richards, *Computability in analysis and physics*, Springer-Verlag, 1988.
12. K. Weihrauch, *Computable analysis – An introduction*, Springer-Verlag, 2000.

Alan Turing: Logical and Physical

Andrew Hodges

University of Oxford
Oxford, UK
andrew.hodges@wadham.oxford.ac.uk

Alan Turing (1912–1954), the founder of computability theory, is generally considered as a purely mathematical logician. But his ideas constantly involved the practical and physical implementation of logical structure. His scientific work both began and ended in theoretical physics. This talk will illustrate the surprisingly wide range of Alan Turing's work, and point towards the general question of the relationship between computation and the physical world.

Skolemization in Intuitionistic Logic.

Rosalie Iemhoff[1,2] and Matthias Baaz[1]

[1] Institute for Discrete Mathematics and Geometry E104
Technical University Vienna
Wiedner Hauptstrasse 8-10
1040 Vienna, Austria
[2] `http://www.logic.at/people/iemhoff`
`iemhoff@logic.at`

Skolemization, the replacement of strong quantifiers by functions, transfers formulas into formulas that contain only weak quantifiers and that are equiconsistent with the original formula. This method does not work for intuitionistic logic IQC in the sense that the skolemized formula may be provable while the original formula is not. The reason for this is that in intuitionistic logic constants cannot denote partial terms that may come into existence only at a later stage. On the other hand, quantifiers range over domains that may increase in going from one world to another in a Kripke model. We show that by allowing partial terms this dichotomy between quantifiers and constants disappears. That is, we show that for a conservative extension of IQC in which there is an existence predicate that indicates whether a term is partial or not, one can define an alternative Skolemization method called eSkolemization, which, for a large class of formulas, possesses many of the nice properties that classical Skolemization has. This class of formulas includes the formulas in which all strong quantifiers are existential. Based on the eSkolemization method one can easily prove an intuitionistic analog of the Herbrand theorem that holds for the class of formulas for which the eSkolemization method applies.

Labeled finite trees as ordinal diagrams

Herman Ruge Jervell

University of Oslo
Oslo, Norway
`http://folk.uio.no/herman`
`herman.jervell@iln.uio.no`

The paper is an extension of the contributed paper "Finite trees as ordinals".

Finite trees are labeled with labels from a well ordered set. The ordering is an extension of the ordering of finite trees with a gap condition using the labels. The ordering is proved to be a well ordering by a minimal bad sequence argument.

The ordering of finite trees gives the ordinals up to the Veblen ordinal. The extension here gives a version of the ordinal diagrams of Takeuti.

Entanglement and its Role in Shor's Algorithm

Vivien M. Kendon[1,3] and William J. Munro[2,3]

[1] School of Physics and Astronomy, University of Leeds, LS2 9JT, UK,
V.Kendon@leeds.ac.uk
[2] Hewlett-Packard Laboratories, Filton Road, Stoke Gifford, Bristol, BS34 8QZ, UK
[3] QOLS, Optics, Blackett Laboratory, Imperial College London, SW7 2BW, UK.

Abstract. Entanglement has been termed a critical resource for quantum information processing and is thought to be the reason that certain quantum algorithms, such as Shor's factoring algorithm, can achieve exponentially better performance than their classical counterparts. The nature of this resource is still not fully understood: here we use numerical simulation to investigate how entanglement between register qubits varies as Shor's algorithm is run on a quantum computer. The shifting patterns in the entanglement are found to relate to the choice of basis for the quantum Fourier transform.

1 Introduction

Quantum computation has the potential to provide significantly more powerful computers than classical computation – if we can build them. There are numerous possible routes forward for quantum hardware [1], however, progress in the development of algorithms has been slow, in part because we don't yet fully understand how the quantum advantage works. There are two key characteristics of the quantum resources used for computation. The first is that a general superposition of 2^n levels may be represented in n 2-level systems [3, 2], allowing the the physical resource to grow only *linearly* with n (quantum parallelism). The second aspect is best explained by considering the classical computational cost of simulating a typical step in a quantum computation. If entanglement is absent then the algorithm can be simulated with an equivalent amount of classical resources. In recent work, Jozsa and Linden [4] have proven that, if a quantum algorithm that cannot be simulated classically using resources only polynomial in the size of the input data, then it must have multipartite entanglement involving unboundedly many of its qubits – if it is run on a quantum computer using pure quantum states. However, the presence of multipartite entanglement is not a sufficient condition for a pure state quantum computer to be hard to simulate classically. If the quantum computer is described using stabilizer formalism [5, 6], there are many highly entangled states that have simple classical descriptions. Moreover, a quantum computer using mixed states may still require exponential classical resources to simulate even if its qubits are not entangled, and it is not known whether such states may be used to perform efficient computation. In any case, being hard to simulate classically doesn't imply the quantum process is doing any useful computation. If we want to understand quantum computation, we will have to look more closely at specific examples.

Few quantum algorithms provide an exponential speed up over classical algorithms, of those that do, Shor's algorithm (order-finding) [7] is perhaps the most important because it can be used to factor large numbers and hence has implications for classical security methods. There is no proof that an equally efficient classical algorithm cannot exist for Shor's algorithm[4], though for quantum walks an algorithm with a proven exponential speed up (w.r.t. an oracle) is known [8]. It is difficult to draw general conclusions about how such a speed up comes about with so few examples to work from. Given that multipartite entanglement is necessary (though not sufficient) for pure state quantum computation with an exponential speed up over classical computation, we next ask what the entanglement is doing during the computational process. We try to answer this question by investigating the level of entanglement within Shor's algorithm as it proceeds, gate by gate.

2 Shor's Algorithm

We begin with a brief overview of how Shor's algorithm works. We wish to factor a number $N = pq$ where p and q are prime numbers. Classical number theory provides a way to determine these primes with high probability (not unity generally) by finding the period r of the function $f_a(x) = a^x (\text{mod } N)$ where a is an integer chosen to be less than N and co-prime to it. It is efficient to check whether a is co-prime to N using Euclid's algorithm. If a happens not to be co-prime then their common factor gives a factor of N and the job is done, but this happens only rarely for large N. Once the period r is found, the numbers

$$m_{\pm} = a^{r/2} \pm 1 \tag{1}$$

generally share either p or q with N as a common factor. Not all choices of a give periods r which yields a factor p or q. For instance, sometimes the period r will be odd, whence the numbers from Eq. (1) can be non-integer. When the chosen a does not lead to a valid factor, the procedure can be repeated with a different choice until a factor is found.

The hard part of the algorithm is determining the period r of the function $f_a(x) = a^x (\text{mod } N)$. Shor found a very elegant and efficient means of doing this quantum mechanically, depicted schematically in Fig. 1. Consider that one has two quantum registers (one of size $2n$ where $n = \lceil \log_2 N \rceil$ qubits and the second of size n qubits. We will denote the basis states of a quantum register by $|x\rangle$, with $x \in \{0 \ldots 2n - 1\}$. The binary representation of x indicates which register qubits are in state $|0\rangle$ and which are in state $|1\rangle$. A general state of a $2n$ qubit

[4] A sub-exponential factoring algorithm is now known but there is still no classically efficient order-finding algorithm, which is the core of Shor's algorithm.

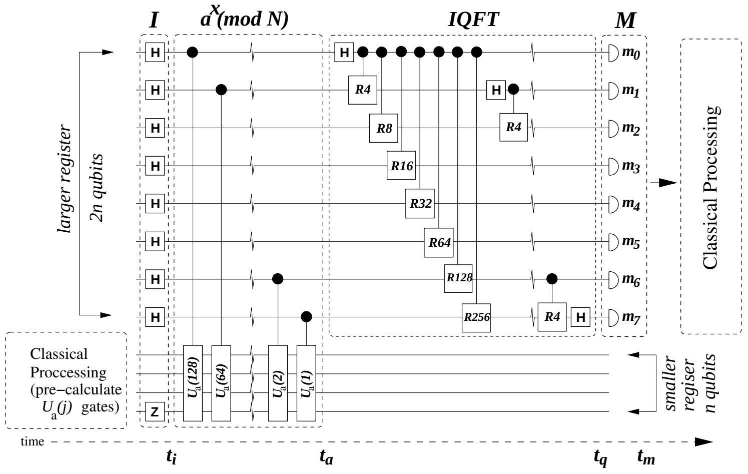


Fig. 1. Schematic circuit diagram of Shor's algorithm for factoring 15 implemented on a 12 qubit quantum register. The initialisation I is done with single qubit Hadamard (H) and bit-flip (Z) gates. Controlled-$U(j)$ gates are used to produce $a^x(\text{mod } N)$. The inverse quantum Fourier transform (IQFT) uses controlled rotations (Rm). The last quantum step is the measurement (M), which is followed by classical post-processing to obtain a factor of N.

register $|\Psi(t)\rangle$ at time t can thus be written as a superposition of basis states,

$$|\Psi(t)\rangle = \sum_{x=0}^{2n} \alpha_x(t)|x\rangle, \tag{2}$$

where $\alpha_x(t)$ is a complex number, normalised such that $\sum \alpha_x(t)^2 = 1$. The algorithm begins by preparing the larger quantum register in an equal superposition $\sum_{x=0}^{2^{2n}-1} |x\rangle$ of all possible $2n$ basis states while the smaller register is prepared in the definite state $|1\rangle$. The initial state of both registers is thus

$$|\Psi(t_i)\rangle = \frac{1}{2^n} \sum_{x=0}^{2^{2n}-1} |x\rangle|1\rangle \tag{3}$$

The next step is a unitary transformation which acts on both registers according to $U|x\rangle|b\rangle = |x\rangle|ba^x(\text{mod } N)\rangle$ giving the output state

$$|\Psi(t_a)\rangle = \frac{1}{2^n} \sum_{x=0}^{2^{2n}-1} |x\rangle|a^x(\text{mod } N)\rangle \tag{4}$$

Then an inverse quantum Fourier transform (IQFT) defined by

$$Q^{-1}|y\rangle = \frac{1}{2^n} \sum_{z=0}^{2^{2n}-1} e^{-2\pi i yz/2^{2n}}|z\rangle \tag{5}$$

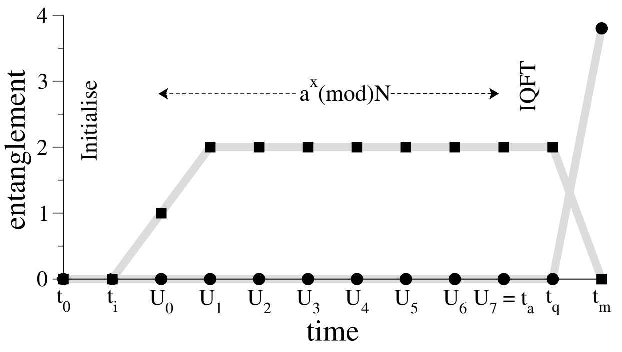


Fig. 2. Entanglement between the two registers (squares) and within the smaller register (circles) in Shor's 12 qubit algorithm as a function of gates sequence according to Fig (1) with the co-prime chosen as $a = 13$. The entanglement within the larger register is zero throughout.

is applied, which transform the state $|\Psi(t_a)\rangle$ from Eq. (4) into

$$|\Psi(t_q)\rangle = \frac{1}{2^{2n}} \sum_{x=0}^{2^{2n}-1} \sum_{z=0}^{2^{2n}-1} e^{-2\pi i x z/2^{2n}} |x\rangle |a^x (\text{mod } N)\rangle. \qquad (6)$$

By measuring the larger register in the computational basis we obtain an integer number c. Now $c/2^{2n}$ is closely approximated by the fraction j/r and so r can be obtained classically using continued fractions. Choosing the larger register to be $2n$ qubits provides a high enough accuracy for c such that r can be determined from a single measurement on all $2n$ qubits. It is possible to use fewer qubits in this first register but the probability of determining r decreases, and the algorithm may need to be repeated correspondingly many more times.

3 Factoring 15

We start our analysis of the entanglement by studying the circuit for factoring 15 (3×5), though it is not necessarily typical of factoring larger numbers. Since many gates make no change to the entanglement, rather than tracking the entanglement as each basic gate is applied, we choose to look at certain key points in the algorithm. We restrict our attention to controlled composite gates: the $U(j)$ gate which implements the operation $a^j (\text{mod } N)$ for $j \in \{1, 2 \ldots 2^{2n}\}$, and the rotations in the IQFT. Details of how to efficiently construct these composite gates from a universal set of one and two qubit gates may be found in, for example, [5]. There are 8 of the $U(j)$ gates (in general $2n$, one for each larger register qubit), which is manageable, but for the IQFT there are 27 (in general $(2n+1)(n-1)$ for a $2n$ qubit register) rotation gates: for out purposes in this paper it is sufficient to treat the whole IQFT as one unit. Along with single qubit gates as necessary, the circuit using these composite gates is depicted in Fig. 1.

As we are only considering the evolution of pure states we can measure the entanglement between the two registers using the entropy of the subsystems

$$E_c = -\sum_i \lambda_i \log \lambda_i, \tag{7}$$

where the $\{\lambda_i\}$ are the eigenvalues of the reduced density matrix of either of the registers (both have the same eigenvalues). To quantify the entanglement *within* each register is not so straightforward. To restrict our attention to the qubits within a single register, we first trace out the other register leaving a mixed state, ρ_L or ρ_S. Most entanglement measures for mixed states are computationally intractable in practice for more than a few qubits; we also need to consider all the possible divisions of the qubits into different subsets in order to locate all of the entanglement. A reasonable approximation to quantifying the entanglement within a register can be obtained by applying a partial transpose to each subset of qubits and calculating the negativity [9, 10] given by $\eta = \mathrm{Tr}|\rho^T| - 1$ i.e., the sum of the negative eigenvalues of the transposed matrix ρ_L^T or ρ_S^T. If the negativity is zero for all possible subsets of qubits in the register, then we can say that at most the register has bound entanglement [11], which is not generally considered useful for quantum information tasks (though see [12]). Non-zero negativity definitely implies the presence of entanglement. Finally, we use the entanglement of formation [13] to quantify the pairwise entanglement between two qubits. Quantum states can be highly entangled without containing any pairwise entanglement [13].

In Fig. 2 we plot the entanglement in Shor's algorithm using the entropy of the subsystem where possible (full state is pure) and the negativity where the single register state is mixed. The negativity turns out to be zero for both registers throughout the algorithm (except the measurement leaves the smaller register entangled, but this cannot be useful for the remaining classical steps of the algorithm). The entanglement between the registers builds up to a maximum during the first two $U(j)$ gates, then stays constant until the measurement. We also note (from calculating the entanglement of formation for appropriate pairs of qubits) that there is no pairwise entanglement between any pair of qubits at any of the sampled points in this instance of the algorithm.

Jozsa and Linden [4] showed that this and similar gate models for Shor's algorithm contain highly multipartite entangled states at this point in the algorithm, thus fulfilling their necessary (but not sufficient) criterion for pure state quantum computation with exponential speed up. Use of mixed states and different gate sequences may produce different entanglement patterns. Parker and Plenio [14] have presented a version of Shor's algorithm using only one pure qubit, the rest may start in any mixed state. They found that entanglement was present when the algorithm ran efficiently for factoring 15 and 21 (tested numerically). It is also clear from the circuit diagram that entanglement will not decrease during the IQFT, since each pair of qubits in the upper register has an entangling (2-qubit) gate applied to it only once during the algorithm. Since the IQFT is the crucial step for finding the period, we next examine how the entanglement distribution changes during the operation of the $IQFT$.

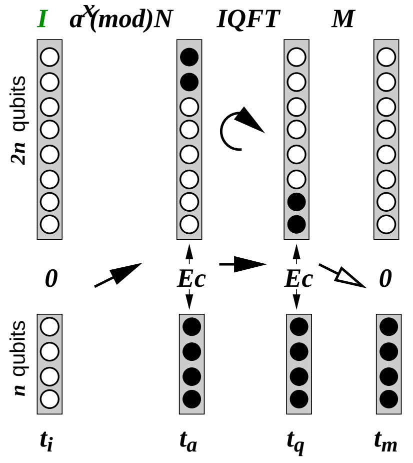


Fig. 3. Pattern of entanglement during Shor's algorithm factoring $N = 15$ with co-prime $a = 13$. After the $U(j)$ gates the top two qubits in the larger register (filled) are entangled with the four qubits in the smaller register. After the IQFT, the entanglement is transfered to the lower two qubits in the larger register. Qubits represented by open circles are not entangled. Time sequence corresponds to Fig. 1.

The entanglement between the registers, as measured by the entropy of the subsystems, does not change during the IQFT, since no entangling gates are applied between the two registers, yet the distribution of the entanglement between the individual qubits does change. If we return to our first example, factoring 15 with $a = 13$, and look at the entropy of subsets of qubits from the larger register, we can deduce that only two of the eight qubits are entangled with the four qubits in the smaller register. During the action of the IQFT, this entanglement is transfered from the top two qubits to the bottom two in the larger register. We represent this schematically in Fig. 3.

However, we should remember that 15 is actually extremely easy to factor. It is straightforward to see that at least one of $a^{r/2} \pm 1$ is divisible by 3 or 5 for nearly all choices of $a, r > 1$, regardless of whether a is co-prime to N or even whether r is the period of $x^a (\text{mod } N)$. We need to look at more examples.

4 Factoring 21

We next look at factoring 21 (3×7). To do this on a quantum computer in the same manner as the circuit for factoring 15 shown in Fig. 1 requires a total of 15 qubits, 10 in the larger register and 5 in the smaller. For co-prime $a = 13$, we find a similar pattern of entanglement to that shown in Fig. 3 for 15 with $a = 13$, except that for 21 there is only entanglement between one qubit in the larger register and two qubits in the lower register. Similarly, the IQFT step shifts the entanglement from the top qubit to the bottom qubit in the larger

size of subsystem	small register	large register after U	large register after IQFT	large register difference $\varDelta E$	large register negativity
1 qubit	0.811	1.000	0.938	-0.062	0.172
2 qubits	1.538	1.600	1.599	-0.001	0.397
3 qubits	2.151	1.843	2.020	+0.177	0.591
4 qubits	2.585	1.972	2.283	+0.311	0.678
5 qubits	2.585	2.081	2.447	+0.366	0.749
6 qubits		2.184	2.547	+0.363	
7 qubits		2.285	2.602	+0.318	
8 qubits		2.385	2.589	+0.204	
9 qubits		2.485	2.619	+0.134	

Table 1. Average entropy of subsystems for factoring 21 with $a = 2$, and average negativity (after the IQFT) for different sized cuts on the larger register.

register. Again, there is no pairwise entanglement, so these three qubits are in a GHZ type of state [15].

The larger register is now at the limit of our computational resources for calculating the full analysis of the negativity. By using random samples of cuts, instead of calculating all possible cuts, we confirm that for co-prime $a = 13$ there is no entanglement within either register, but for other choices of co-prime such as $a = 2$ and $a = 4$, entanglement is generated within the larger register during the IQFT. For these co-primes we also find a more complex pattern in the entropies of the subsystems: the entanglement now involves all of the register qubits. There is also a significant amount of pairwise entanglement of formation (average 0.261 per qubit before the IQFT) contributing to the total entanglement in the system. The details are shown in Table 1: essentially the entanglement becomes more multipartite: the average reduces slightly for one and two qubit cuts, while for larger cuts it increases. The average pairwise entanglement of formation also decreases slightly (from 0.261 to 0.242). We will provide an explanation for this observation in the next section where we examine larger examples.

5 Factoring larger numbers

We also studied semi-primes $32 < N < 64$ and $64 < N < 128$, which require 18 and 21 qubits respectively for the quantum registers. In these cases, though we cannot easily calculate a full entanglement analysis, we have calculated the entropy between one qubit and the rest of the qubits in both registers, this corresponds to the quantities in the top line of Table 1. The difference in the average entropy $\varDelta E_1$ before and after the IQFT (corresponding to the last column in Table 1), is shown in Table 2 grouped by the period r.

There is a clear pattern for $\varDelta E_1$: the closer the period r is to a power of 2, the smaller the value of $\varDelta E_1$. For $r = 2^m$, the IQFT is exact giving $\varDelta E_1 = 0$ in all cases. This can be understood by looking at the measurement results on the

N	r (number of co-primes with this r)								
$p \times q$	$-\langle \Delta E_1 \rangle$								
15	2 (3)	4 (4)							
3×5	0.0	0.0							
21	2 (3)	3 (2)	6 (6)						
3×7	0.0	0.706	0.624						
33	2 (3)	5 (4)	10 (12)						
3×11	0.0	0.285	0.256						
35	2 (3)	3 (2)	4 (4)	6 (6)	12 (8)				
5×7	0.0	0.869	0.0	0.788	0.706				
39	2 (3)	3 (2)	4 (4)	6 (6)	12 (8)				
3×13	0.0	0.869	0.0	0.788	0.706				
51	2 (3)	4 (4)	8 (8)	16 (16)					
3×17	0.0	0.0	0.0	0.0					
55	2 (3)	4 (4)	5 (4)	10 (12)	20 (16)				
5×11	0.0	0.0	0.285	0.256	0.226				
57	2 (3)	3 (2)	6 (6)	9 (6)	18 (18)				
3×19	0.0	0.869	0.788	0.080	0.071				
77	2 (3)	3 (2)	5 (4)	6 (6)	10 (12)	15 (8)	30 (24)		
7×11	0.0	1.033	0.343	0.951	0.314	0.034	0.031		
91	2 (3)	3 (8)	4 (4)	6 (24)	12 (32)				
7×13	0.0	1.033	0.0	0.951	0.869				
119	2 (3)	3 (2)	4 (4)	6 (6)	8 (8)	12 (8)	16 (16)	24 (16)	48 (32)
7×17	0.0	1.033	0.0	0.951	0.0	0.869	0.0	0.788	0.706

Table 2. Average decrease in entanglement $-\langle \Delta E_1 \rangle$ between one qubit and the rest during the IQFT.

larger register, from which the period r is calculated. Figure 4 shows the probability of measuring each possible number c in the larger register at the end of running the algorithm for two examples: factoring 119 with co-prime 92 (period $r = 16$), and with co-prime 93 (period $r = 24$). When the period is exactly a power of two, the fraction $c/2^{2n}$ gives the period r exactly, whereas when r is not a power of two, the peak probability tries to fall between two possible numbers and thus spreads the wavefunction over several adjacent numbers. This spread increases the entanglement in the upper register.

6 Discussion

We have shown that, as expected, the quantum registers become highly entangled during the order-finding part of Shor's algorithm. In particular, during the IQFT, it can become even more multipartite in nature if the period being found is not an exact power of two. Thus, if we performed the IQFT to some other base than two – for example, in base three, perhaps using a quantum register made

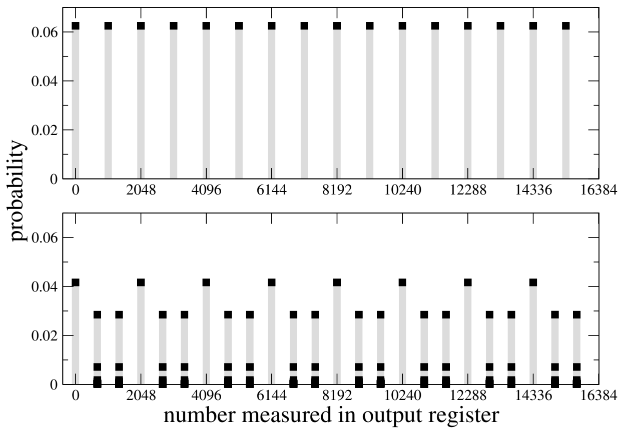


Fig. 4. Distribution of measurement outcomes for factoring 119 with $a = 92$ (upper) and $a = 93$ (lower) which have periods $r = 16$ and 24 respectively. Symbols show the probability of measuring the number on the ordinate as the outcome of the algorithm; drop lines are for clarity. The upper figure has just 16 peaks, while the lower figure shows a significant probability for measuring neighbouring numbers to the minor peaks.

up of qutrits (three-state quantum systems) rather than qubits – the entanglement pattern would change completely. The entanglement does not correlate in a quantitative way with the success of the algorithm, which is determined largely by classical factors such as whether r is even or odd. While entanglement is certainly generated in significant quantities during pure state quantum computation, this is best understood as a by-product of exploiting the full Hilbert space for quantum parallelism [3, 4, 2]: most of Hilbert space consists of highly entangled states [16, 17].

The fact that entanglement generated during the execution of the algorithm is not used up in a quantitative way to fuel the computation process is in complete contrast to quantum communications tasks where maximally entangled pairs of qubits can perform a specific amount of communication, during which the entanglement is consumed. Entanglement is also used quantitatively in many practical proposals for implementations of a quantum computer, notably [18], this use can be identified with carrying out communications tasks to move the quantum data around in the physical qubits.

We thank Stephen Parker, Martin Plenio and Ben Travaglione for valuable discussions. VK was funded by the UK Engineering and Physical Sciences Research Council grant number GR/N2507701 (to Sept 2003) and now by a Royal Society University Research Fellowship. A Royal Society Research Grant provided additional computing facilities. Partly funded by the European Union.

References

1. Braunstein, S., Lo, H.K., eds.: Scalable Quantum Computers: Paving the Way to Realization. Volume 48. Fortschr. Phys. (2000)
2. Blume-Kohout, R., Caves, C.M., Deutsch, I.H.: Climbing mount scalable: Physical-resource requirements for a scalable quantum computer. Found. Phys. **32** (2002) 1641–1670
3. Jozsa, R.: Entanglement and quantum computation. In Huggett, S.A., Mason, L.J., Tod, K.P., Tsou, S., Woodhouse, N.M.J., eds.: The Geometric Universe, Geometry, and the Work of Roger Penrose, Oxford University Press (1998) 369–379
4. Jozsa, R., Linden, N.: On the role of entanglement in quantum computational speed-up. Proc. Roy. Soc. Lond. A Mat. 459 **459** (2003) 2011–2032
5. Nielsen, M.A., Chuang, I.J.: Quantum Computation and Quantum Information. Cambridge University Press, Cambs. UK (2000)
6. Gottesman, D.: Stabiliser codes and quantum error correction. PhD thesis, California Institute of Technology, Pasadena, CA (1997)
7. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Sci Statist. Comput. **26** (1997) 1484
8. Childs, A.M., Cleve, R., Deotto, E., Farhi, E., Gutmann, S., Spielman, D.A.: Exponential algorithmic speedup by a quantum walk. In: Proc. 35th Annual ACM Symposium on Theory of Computing (STOC 2003), Assoc. for Comp. Machinery, New York (2003) 59–68
9. Peres, A.: Separability criterion for density matrices. Phys. Rev. Lett. **77** (1996) 1413–1415
10. Życzkowski, K., Horodecki, P., Sanpera, A., Lewenstein, M.: On the volume of the set of mixed entangled states. Phys. Rev. A **58** (1998) 883–892
11. Horodecki, M., Horodecki, P., Horodecki, R.: Mixed-state entanglement and distillation: is there a “bound” entanglement in nature? Phys. Rev. Lett. **80** (1998) 5239–5242
12. Horodecki, P., Horodecki, M., Horodecki, R.: Bound entanglement can be activated. Phys. Rev. Lett. **82** (1999) 1056–1059
13. Wootters, W.K.: Entanglement of formation of an arbitrary state of two qubits. Phys. Rev. Lett. **80** (1998) 2245–2248
14. Parker, S., Plenio, M.B.: Entanglement simulations of shor’s algorithm. J. Mod. Optic **49** (2001) 1325–1353
15. Greenberger, D.M., Horne, M., Zeilinger, A. In Kafatos, M., ed.: Bell’s Theorem, Quantum Theory, and Conceptions of the Universe, Kluwer (1989) 73
16. Kendon, V.M., Życzkowski, K., Munro, W.J.: Bounds on entanglement in qudit systems. Phys. Rev. A **66** (2002) 062310
17. Hayden, P., Leung, D.W., Winter, A.: Aspects of generic entanglement. quant-ph/0407049 (2004)
18. Raussendorf, R., Briegel, H.J.: A one-way quantum computer. Phys. Rev. Lett. **86** (2001) 5188–5191

Hypercomputability in Quantum Mechanics

Tien D. Kieu

Centre for Atom Optics and Ultrafast Spectroscopy,
Swinburne University of Technology, Victoria 3122, Australia
kieu@swin.edu.au

It is proposed that computation employing the principles of quantum mechanics (QM) is more powerful than Turing computation (which is embodied by present-day classical computers) in terms of computability: **quantum computation in its most general framework can compute some problems which are not computable by Turing machines**. Such a computation which transcends the Turing computation is also known as hypercomputation. This is in stark contrast to and more powerful than the well-known capability of the more "standard" quantum computation with qubits and quantum gates [1], which is capable of improving computational complexity of classical algorithms – that is, capable of speeding up some computation which is *also* computable by Turing machines.

We have recently proposed a quantum algorithm [2] exploiting the quantum adiabatic computation [3] to solve Hilbert's tenth problem and the mathematically equivalent Turing halting problem, both of which are known to be Turing non-computable and related to many other important mathematical problems [4]. We now would like to discuss and further extend this quantum adiabatic algorithm, which has received much implicit as well as some explicit attention in the literature, see [5] and [6].

We wish to cover in this presentation most, if not all, of the following points:

– We firstly establish a reformulation of Hilbert's tenth problem, a problem in integer arithmetics, into problems in solving two different classes of partial differential equations (PDEs), one of which is nonlinear and the other one is linear (namely, the Schrödinger equation) and related to the physical quantum adiabatic processes.
– Immediate consequences of the reformulation are some new classes of (linear and non-linear) PDEs that are Turing-noncomputable, due to the established Turing non-computability of Hilbert's tenth problem. These results are in the same spirit as those of noncomputable wave equations discovered by Pour-El and Richards [7], and others.
– But thanks to the relationship between the linear Schrödinger equations and physical processes, assuming that the so-far-never-failed QM is valid always, we then argue that Hilbert's tenth problem is computable in the framework of quantum adiabatic processes.
– We next briefly review the key points in QM, in particular the crucial quantum adiabatic theorem. Although this theorem by itself does not give rise to hypercomputability, we are, in combining the theorem with a new result, able to present an algorithm for Hilbert's tenth problem. The theorem asserts

that a particular eigenstate of a final-time Hamiltonian, even in dimensionally *infinite* spaces, could be found mathematically and/or physically *in a finite time*. This is a remarkable property, which is enabled by quantum interference and quantum tunnelling with complex-valued probability amplitudes, and allows us to find a needle in an infinite haystack, in principle! Such a property is clearly *not* available for recursive search in an unstructured infinite space, which, in general, cannot possibly be completed in a finite time, in contrast to the quantum scenario.

– Our quantum algorithm seems to be an infinite search through the positive integers in a finite time! This may sound strange unless one recognises that even though in a finite time the wave function of a system starting from an initial ground state can only cover essentially a finite domain of the underlying Hilbert space, the finite domain covered is indeed the relevant domain which contains the point of interest. That is, the initial ground state of our quantum adiabatic algorithm provides one end of the lead along which we can trace to the final ground state at the other end. The information extracted from this final ground state then gives us the answer to our original decision problem.
– Such an algorithm is also relevant to the inherent randomness of mathematics through the hypercomputation of Chaitin's Ω number, which is the probability that a randomly chosen Turing program will halt upon being fed with a randomly chosen input. (We have also been able to extend a key result concerning Chaitin's Ω – linking the halting of Turing programs, a computer science problem, to the parity of the number of solutions of Diophantine equations, a number theory problem. Our result, in turn, has recently been extended further by Matiyasevich [8].)
– Some misconceptions and prejudices against the algorithm and against its physical implementation are summarised and refuted [5].
– That still leaves some unanswered difficulties in the physical implementation of the algorithm. We explicitly list and discuss these difficulties, which will require further investigations in the future.
– Nevertheless, we point out that [9], contrary to the prejudice that the algorithm is not physically implementable, a particular instance of the algorithm has *already* been realised physically in the quantum phase transition [10] associated with the superfluid-Mott insulator transition! The first experiment for this transition has only been carried out very recently with Bose-Einstein condensates in optical lattices [11]. This and subsequent experiments can in fact be interpreted as implementations of the quantum adiabatic algorithm for the class of linear Diophantine equations!
– We stress the fact that the algorithm is probabilistic.
– We also emphasise that Turing machines equipped with true randomness could be capable of hypercomputation [12], contrary to the widely spread misconception that probabilistic computation has the same computability as Turing computation – which is only the case if the probabilities are themselves restricted to computable numbers (a fact pointed out by Shannon *et al.* in their seminal paper [13] but overlooked by many).

- Basing on that conclusion, we argue that Hilbert's tenth problem should also be computable with Turing machines equipped with some suitable measures of probability.
- We then generalise the quantum adiabatic algorithm for probabilistic searches (optimisation) in spaces of *infinite dimensions*! We explicitly present the principles involved for these Turing non-computable searches.
- Armed with this generalisation, we *speculate* about the computability of the problems in arithmetics that are finitely refutable (a term defined in [14]) and are beyond the equivalent class of Hilbert's tenth problem.

References

1. M.A. Nielsen and I.L. Chuang, *Quantum Computation and Quantum Information* (CUP, Cambridge, 2000).
2. T.D. Kieu, Minds and Machine 12, 541-61 (2002).
   T.D. Kieu, Contemporary Physics 44, 51-71 (2003).
   T.D. Kieu, Int. J. Theo. Phys. 42, 1461-1478 (2003).
   T.D. Kieu, Proc. Roy. Soc. A 460, 1535 (2004).
   T.D. Kieu, Theoretical Computer Science 317, 93-104 (2004).
   T.D. Kieu, in Proceedings of SPIE Vol. 5105 Quantum Information and Computation, edited by Eric Donkor, Andrew R. Pirich, Howard E. Brandt, (SPIE, Bellingham, WA, 2003), pp. 89-95 (2003).
   T.D. Kieu, Int. J. Quantum Info. 3, 177-182 (2005).
   T.D. Kieu, Quantum adiabatic algorithm for Hilbert's tenth problem: I. The algorithm, `quant-ph/0310052` (2003).
3. E. Farhi, J. Goldstone, S. Gutmann and M. Sipser, Quantum computation by adiabatic evolution, `quant-ph/0001106` (2000).
4. Y. Matiyasevich, *Hilbert's Tenth Problem*, (MIT Press, Cambridge, Massachussetts, 1993).
5. B. Tsirelson, The quantum algorithm of Kieu does not solve the Hilbert's tenth problem, `quant-ph/0111009` (2001).
   T.D. Kieu, Reply to 'The quantum algorithm of Kieu does not solve the Hilbert's tenth problem, `quant-ph/0111020` (2001).
   R. Srikanth, Computable functions, the Church-Turing Thesis and the quantum measurement problem, `quant-ph/0402128` (2004).
   T.D. Kieu, Finiteness of the universe and computation beyond Turing computability, `quant-ph/0403045` (2004).
6. A. Sicard, M. Vélez and J. Ospina, Computing a Turing-incomputable problem from quantum computing, `quant-ph/0309198` (2003).
   A. Sicard, M. Vélez and J. Ospina, Hypercomputation based on quantum computing, `quant-ph/0406137` (2004).
   M. Ziegler, Does quantum mechanics allow for infinite parallelism?, `quant-ph/0410141` (2004).
   R. Srinivasan and H.P. Raghunandan, On the existence of truly autonomic computing systems and the link with quantum computing, `cs.LO/0411094` (2004).
7. M.B. Pour-El and J.I. Richards, *Computability in Analysis and Physics* (Sringer-Verlag, Berlin, Heidelberg, 1989).

8. T. Ord and T.D. Kieu, Fundamenta Informaticae 56, 273-284 (2003).
G. Chaitin, *Meta Math! The Quest for Ω* (Pantheons, to be published in 2005).
Y. Matiyasevich, Diophantine flavour of Kolmogorov complexity, `http://at.yorku.ca/cgi-bin/amca/cani-11` (2004).
9. T.D. Kieu, in preparation (2005).
10. S. Sachdev, *Quantum Phase Transitions* (CUP, Cambridge, 1999).
11. M. Greiner, O. Mandel, T.W. Hänsch and I. Bloch, Nature 419, 51-54 (2002).
12. T. Ord and T.D. Kieu, Brit. J. Phil. Sci. 56, 159-168 (2005).
T. Ord and T.D. Kieu, Using biased coins as oracles, `cs.OH/0401019` (2004).
13. K. de Leeuw, E.F. Moore, C.E. Shannon and N. Shapiro, in Automata Studies Annals of Mathematics Studies, No. 34 (Princeton U. Press, Princeton, 1956).
14. C.S. Calude, *Information and Randomness, 2nd Ed.* (Springer-Verlag, Berlin, Heidelberg, 2002)

Almost everywhere domination and K-triviality

Bjørn Kjos-Hanssen

Department of Mathematics
University of Connecticut
Storrs, USA
`http://www.math.uconn.edu/~bjorn`
`bjorn@math.uconn.edu`

A function is almost everywhere dominating (AED) if it dominates every function computed by almost every oracle.

Dobrinen and Simpson introduced this notion in the context of reverse mathematics. It turns out that there are AED functions of degree $< 0'$ (Cholak, Greenberg) but they must all be truth-table high (Binns, Kjos-Hanssen, Lerman, Solomon).

A real is K-trivial if the Kolmogorov complexity of its initial segments is as low as possible (up to a constant).

I will describe work with Hirschfeldt on a connection between K-triviality and AED functions.

Universality of Walking Automata on a Class of Geometric Environments

Oleksiy Kurganskyy[1] * and Igor Potapov[2] **

[1] Institute of Applied Mathematics and Mechanics
Ukrainian National Academy of Sciences
74 R. Luxemburg St, Donetsk, Ukraine
kurgansk@gmx.de
[2] Department of Computer Science
University of Liverpool, Chadwick Building
Peach St, Liverpool L69 7ZF, UK
igor@csc.liv.ac.uk

Abstract. In this paper we study the model of a finite state automaton interacting with two-dimensional geometric environment. We consider a kind of well-known 4-way finite state automaton introduced by Blum and Hewitt and the geometric environments defined as a subspace of two dimensional integer grid bounded by integer value functions. As a main result of this paper we characterize a wide class of geometric environments where a finite automaton exhibits universal behaviour. We leave the problem of reachability for automaton interacting with environments bounded by other functions as an open question. Our conjecture is that the problem is undecidable if the infinite environment in the half plane is limited by any unbounded nondecreasing function. Though we show that bounds on increase rate of the environment width does not have any influence on universality, it is not clear what would be the cornerstones of undecidability for some specific environments (e.g. when environment is limited by a logarithmic function).

1 Introduction

Finite state automata arose as models of transducers of discrete information, i.e. models interacting with their environments [10]. Automata on picture languages [3, 19], automata in labyrinth [2, 11, 12], communicating automata [16], counters automata [17, 9], a model of a computer in the form of interaction of a control automaton [5] are examples of such an interaction.

The fundamental problem for systems where the automaton interacts with infinite environment is the reachability problem: “Does a state S belong to the set of states reachable from an initial configuration”. Reachability has connections to many classical problems in automata theory such as diagnostic problems, distinguishability problems, searching in labyrinths, etc. One of the standard

* Work partially supported by RDF grant RDF/4417
** Work partially supported by The Nuffield Foundation grant NAL/00684/G.

methods to show the undecidablity of the reachability problem for some model is to prove that this computational model is universal.

In this paper we consider the computational power of reactive system, where an input/output automaton interacts with two-dimensional geometric environments. In particular we consider the reactive system that includes a finite state automaton (FSA) A and an infinite environment E where

– an automaton A is a kind of well known 4-way finite state automaton introduced by Blum and Hewitt [1] that read the symbol from the environment and reacts by changing its internal state and moving up, down, left or right to a neighbouring cell of an environment E;
– an environment E is a two dimensional language over one-letter alphabet [14, 3] which is defined as a subspace of two dimensional integer grid bounded by a number of integer value functions.

It was shown in [13, 14] that the problem of checking indistinguishable states for two finite automata in two dimensional language E over one-letter alphabet (geometric environments) is decidable if and only if the reachability problem for a finite state automaton in E is decidable. A geometric environment is called *efficient* if the reachability problem for this environment is decidable and *non-efficient* otherwise. The set of rectangles of fixed height that can be represented by a regular or context-free expression are efficient [6, 7, 14]. There are other classes of efficient environments that can be constructed by substitution of any cell in efficient environment by another environment [14].

It was proved in [2] that the set of input-output words generated by automaton interacting with an infinite geometric environment is a context-sensitive language, in general. Thus the reachability problem for finite automata interacting with geometric environments is fundamentally difficult and algorithmically unsolvable, in the general case. In spite of known undecidability results it is interesting to identify new classes of non-effective environments, i.e. where a model of finite automaton interacting with these environments is universal. Recently, we initiated the research on non-efficient environments by investigating several special cases: quadrant of the plane extended by power function, polynomial function and linear function [15].

As a main result of this paper we characterize a wide class of geometric environments where a finite automaton can exhibit universal behaviour. We leave the problem of reachability for an automaton interacting with environments bounded by other functions as an open question. Our conjecture is that the problem is undecidable if the infinite environment in the half plane is defined by any unbounded nondecreasing function.

The motivation for investigation of walking automaton dynamics is that the different constraints on the environments can be seen as constraints on the values of counters in counter automata or other computational models and vice versa. The main results of this paper define so-called *essentially two-dimensional environments*, i.e. environments where a walking automaton can exhibit universal behaviour and cannot be modelled by any one-dimensional systems, like push-down or one-counter automata. Thus, we consider the analysis of two-dimensional geometric environments as a new look on the classical topic about the border between decidability and undecidability.

2 Automata interacting with two dimension environments

In what follows we use traditional denotations $\mathbb{N}, \mathbb{Z}$ and $\mathbb{Z}_n$ for the sets of naturals $\{1, 2, 3, \dots\}$, set of integers $\{\dots, -2, -1, 0, 1, 2, \dots\}$ and set of bounded integers $\{-n, \dots, -2, -1, 0, 1, 2, \dots, n\}$ respectively.

Definition 1. *Let $A = (S_A, I, O, \delta_A, \lambda_A, s_0)$ be a finite deterministic everywhere defined Mealy automaton, where S_A, I and O are the sets of states, input symbols, and output symbols, respectively, and $\delta_A : S \times I \rightarrow S$ and $\lambda_A : S \times I \rightarrow O$ are transition function and function of outputs. respectively and $s_0 \in S$ is an initial state.*

Definition 2. *The geometric environment is defined by possibly infinite (countable) Moore automaton $E = (D, O, I, \delta_E, \lambda_E)$, where $D \subseteq \mathbb{Z} \times \mathbb{Z}$. is a set of states, $O = \mathbb{Z}_1 \times \mathbb{Z}_1$ is a set of input symbols, $I = 2^{\mathbb{Z}_1 \times \mathbb{Z}_1}$ is a set of output symbols, $\delta_E : D \times O \rightarrow D$ such that*

$$\delta_E((x, y), (d_1, d_2)) = \begin{cases} (x + d_1, y + d_2), & (x + d_1, y + d_2) \in D \\ \quad undefined, & (x + d_1, y + d_2) \notin D \end{cases}$$

is the partial transition function and $\lambda_E : D \rightarrow I$ such that

$$\lambda_E(x, y) = \{(d_1, d_2) \in \mathbb{Z}_1 \times \mathbb{Z}_1 | (x + d_1, y + d_2) \in D\}$$

is the function of outputs.

We call the set of states D - the nodes of the environment, the symbols of the output alphabet I - the labels of nodes, the function of outputs λ_E - the function of labels of nodes and the words in alphabet O - the movements of automaton in the environment. From Definition 2 follows that two-dimensional geometric environment is represented by an automaton E. According to the fact that the set D uniquely defines the environment E, we identify D and E in the rest of the paper.

Two nodes of the environment (x_1, y_1) and (x_2, y_2) are the neighbours iff $(x_2 - x_1, y_2 - y_1) \in \mathbb{Z}_1 \times \mathbb{Z}_1$. All neighbours of the node (x, y) is the Moore neighbourhood of range 1.

We can also consider finite automaton in two-dimensional environment as a kind of 4-way finite state automaton introduced by Blum and Hewitt [1] that read the symbol (Moore neighbourhood of range 1) from the environment and reacts by changing its internal state and moving up, down, left or right to a neighbouring cell of an environment E.

Let an automaton A and an environment E interact with each other, then an output signal of each of them coincides with an input signal of the other

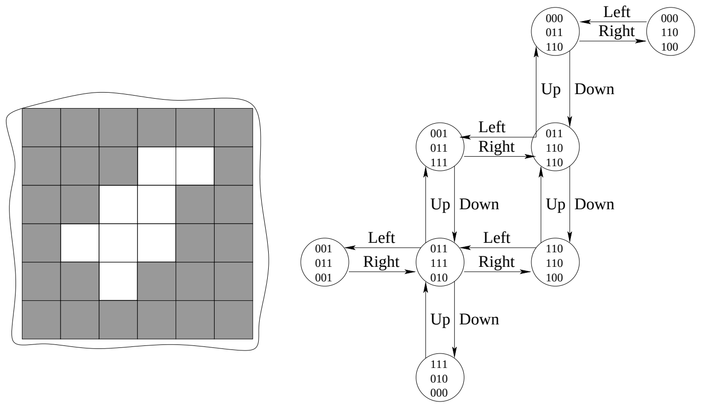


Fig. 1. Geometric environment represented by Moore automaton E.

at each instant of time. So in case of 4-way finite state automaton the set of output symbols of an automaton A (which is also a set of input symbols of an environment E) is $\{Left = (-1, 0), Right = (1, 0), Up = (0, 1), Down = (0, -1)\}$ and the input symbols of A (a set of output symbols of E) is a matrix $O^{3\times3}$ that represents the Moore neighbourhood of range 1 for a position (x, y):

$$\begin{pmatrix} o_{-1,1} & o_{0,1} & o_{1,1} \\ o_{-1,0} & o_{0,0} & o_{1,0} \\ o_{-1,-1} & o_{0,-1} & o_{1,-1} \end{pmatrix},$$

where $o_{i,j} = 1$ if $(x + i, y + j) \in D$, otherwise $o_{i,j} = 0$ and $i, j \in \mathbb{Z}_1$.

Let us describe the process of interaction between automaton and environment. An automaton A initiates the interaction with an environment E starting from the state s_0 and a node $r \in D$. Let A be in state s and node r, then automaton moves to the state $\delta_A(s, \lambda_E(r))$ and the node $\delta_E(r, \lambda(s, \lambda_A(s, \lambda_E(r))))$.

By pair (s, r), where s is state of an automaton, and r is a node of the environment, we denote a *configuration* of the automaton in the environment. We also say that configuration (s', r') is directly reachable from (s, r) if $s' = \delta_A(s, \lambda_E(r))$ and $r' = \delta_E(r, \lambda_A(s, \lambda_E(r)))$ and denote it by $(s, r) \to (s', r')$.

3 Geometric representation of Minsky Machine

It is easy to see that the behaviour of 2-counter machine with a zero test (Minsky machine) can be interpreted as a 4-way finite state automaton that interacts with (or moves in) the quadrant of the plane $D_Q = \{(x, y) \subseteq \mathbb{Z} \times \mathbb{Z} | x \geq 0, y \geq 0\}$, where the border of the quadrant is the following set of nodes: $G_Q = \{(x, y)|x \geq 0, y = 0, x \in \mathbb{N}, y \in \mathbb{N}\ \} \cup \{(x, y)|x = 0, y \geq 0, x \in \mathbb{N}, y \in \mathbb{N}\ \}$. The node (x, y) of the environment $E_Q = (D_Q, O, I, \delta_E, \lambda_E)$ represent the values x and y of two counters and the empty counters of Minsky machine corresponds to the situation when an automaton is on the borders of the geometric environment E_Q.

Now we explain a well known trick to get an equivalent model of two counter machine where one of the counters is used as a scratchpad. Another, counter holds an integer whose prime factorization is $2^a \cdot 3^b$. The exponents a, b can be thought of as two virtual counters that are being simulated. When the real counter is set to one, it is equivalent to setting all the virtual counters to zero. If the real counter is doubled, that is equivalent to incrementing a, and if it is halved, that is equivalent to decrementing a. By a similar procedure, it can be multiplied or divided by 3, which is equivalent to incrementing or decrementing b. To check if a virtual counter such as a (b) is equal to zero, just divide the real counter by 2 (3), see what the remainder is, then multiply by 2 (3) and add back the remainder. That leaves the real counter unchanged. The remainder will have been nonzero if and only if a (b) was zero.

In the rest of the paper we are going to use the universal model of 2-counter machine M_{scrt} with one scratchpad counter to show several undecdiability results for walking automaton in different geometric environments.

The simple extention of machine M_{scrt} shows that the environment limited by two linear functions in the half plane is non-efficient. Since the integral line defines the regular shifts of the borders, we only need to amortise this shifts by adding a fixed number of right or left moves according to the chosen direction to each transition of original automata. So we can directly simulate multiplication and division that corresponds to original operation of machine M_{scrt}. In the next section we show a wide class of environments where the reachability for walking automaton is undecidable but the direct multiplication and division are not implementable.

4 Non-efficient Environments

Let us consider a function $f : \mathbb{N} \to \mathbb{N}$ such that $f(n) \geq n$ and function $g : \mathbb{N} \to \mathbb{N}$ such that $g(1) = 1$, $g(n) = g(n-1) + f(n-1)$. We define now a geometric environment $E(g)$, that is defined by the set of vertices $\{(x, y) \subseteq \mathbb{N} \times \mathbb{N} | x \geq g(y)\}$. The example of an environment is shown on Figure 2.

Theorem 1. *Given a number $p \in \mathbb{N}$. If for all $n > p$ a function f satisfies the condition $f(n) \equiv p \pmod{n}$ then an environment $E(g)$ is non-efficient.*

Proof. The proof is done by reduction of the reachability problem for 2-counter automaton (Minsky machine) to the reachability problem for walking finite state automaton in the above environment. In particular we show how to perform operation of multiplication and division in the environment $E(g)$ that allow us to perform the proposed reduction. We show the simulation of multiplication by 2 in the environment $E(g)$ in details. It is easy to apply this method to simulate the multiplication by 3 and division by 2 or 3 by walking automaton in $E(g)$.

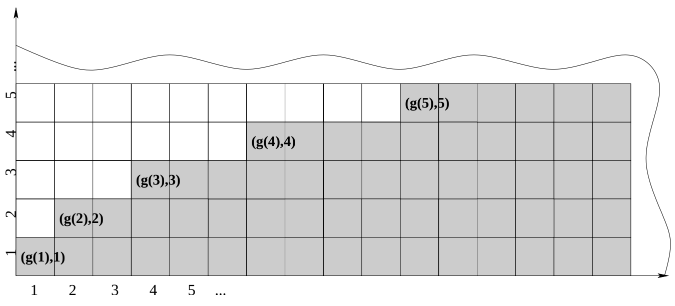


Fig. 2. The example of an environment represented by function $f(x) = x$.

Let us assume that an automaton starts from a point $(g(y), y)$ in the environment $E(g)$. The vertex of environment $E(g)$ in this point has a label

$$\begin{pmatrix} 0\,0\,0 \\ 0\,1\,1 \\ 1\,1\,1 \end{pmatrix}.$$

Now we show that a finite state automaton on environment $E(g)$ can reach a vertex $(g(2y), 2y)$ from a vertex $(g(y), y)$ and stop on it. Note that all movements (computations) of a finite state automaton that interacts with an environment $E(g)$ is done using only labels of environments that are available to our walking automaton and its finite number of states.

By $S_n = (1, 0)^n \in O^*$, $n \in N$ we denote the movement of automaton from a vertex (x, y) to a vertex $(x + n, y)$, that is n movements to the right.

Another kind of movements that we are going to use for simulation are movements of type $C : C_1$ or C_0.

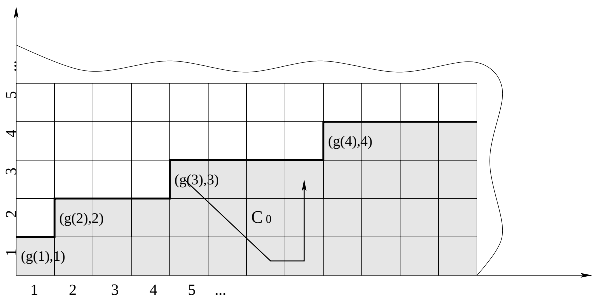


Fig. 3. The example of a movement C_0.

If $g(y) \le x < g(y + 1) - y$ by $C_0 = (1, -1)^{y-1}(1, 0)(0, 1)^{y-1} \in O^*$ we denote the movement of an automaton from a vertex (x, y) to a vertex $(x + y, y)$.

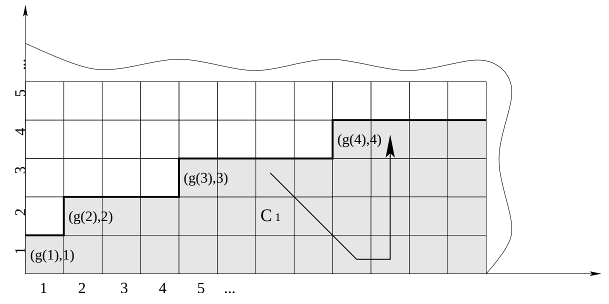


Fig. 4. The example of a movement C_1.

If $g(y+1)-y \leq x < g(y+1)$ by $C_1 = (1,-1)^{y-1}(1,0)(0,1)^y \in O^*$ we denote the movement of automaton from vertex (x, y) to a vertex $(x+y, y+1)$. The examples of movements C_0 and C_1 are shown on Figures 3, 4.

There are several useful properties of C-movements. Let us assume that a walking automaton is in the bounded vertex (x, y) of an environment $E(g)$, where $g(y) \leq x < g(y+1)$, then it can make only a finite number of consecutive moves of type C_0. If $f(y) = iy + p$ (note that p is a constant), then the number of consecutive moves of type C_0 is bounded by i. So finally after a finite number of C_0 movements a walking automaton does C_1 movement. It can also determine the fact of changing from C_0 to C_1 by checking evenness/oddness of second component of the vertex coordinate using a finite number of its internal states.

Next type of movements that we use for simulation is T. Let a walking automaton is at the boundary vertex (x, y), $g(y) \leq x < g(y)+y$ of an environment $E(g)$, in which the following property $f(y) = iy + p$ holds. By T we denote the sequence of movements from a vertex (x, y) to $(x+p, y)$ by S_p and then a finite number of C_0 movements that end by C_1 . Let us prove now that T will move an automaton from a vertex (x, y) to a vertex $(x', y+1)$, where

$$x - g(y) = x' - g(y+1) \tag{1}$$

Since we have that $f(y) = iy + p$, a walking automaton should make exactly $i-1$ movements of type C_0 to reach vertex $(x+(i-1)y+p, y)$ from vertex $(x+p, y)$. Then after movement C_1 it reaches vertex $(x+(i-1)y+y+p, y+1)$ that is $(x+iy+p, y+1)$. From it follows that $x' - g(y+1) = x+iy+p-g(y+1) = x+iy+p-g(y)-f(y) = x-g(y)$. In other words, we proved that after a movement T the distance from a corner $(g(y), y)$ of the environment to a vertex (x, y) is the same as the distance from a vertex $(x', y+1)$ to a corner $(g(y+1), y+1)$. So we can keep some information of the computations during the walk of automaton as a distance from its location to the nearest left corner of the environment.

Let us show how to move an automaton from a vertex $(g(y), y)$ to a vertex $(g(2y), 2y)$ that corresponds to the multiplication of y by 2. The automaton starts in a vertex $(g(y), y)$ by making a movement $(TS_2)^y T$ until the automaton reaches a vertex $(g(2y+1), 2y+1)$, i.e. the first vertex with the label

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

and then by making a movement $(0,-1)(-1,0)^{f(2y)}$ to the first vertex with the label

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

This is exactly the vertex $(g(2y), 2y)$.

Let us track the sequence of above movements. The first movement TS_2 will change the position of automaton from $(g(y), y)$ to $(g(y+1)+2, y+1)$. The position of automaton after next movements TS_2 will be $(g(y+2)+4, y+2)$. Thus, the position of automaton will be changed to $(g(y+y)+2y, y+y) = (g(2y)+2y, 2y)$ exactly after y of such movements. After the last movement T of the movement $(TS_2)^y T$ the position of automaton will be changed exactly to $(g(2y+1), 2y+1)$ because of the equation 1. The example of such movements that correspond to the multiplication by 2 is shown on Figure 5.

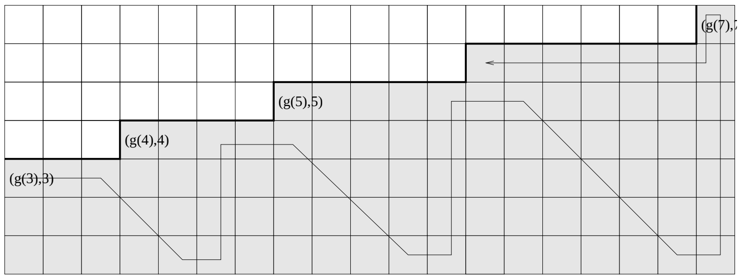


Fig. 5. The example of a multiplication $y \times 2$, where $y = 3$, in the environment represented by function $f(x) = x$.

By analogy with a case of multiplication by 2 we can show that a sequence of movements

$$(TS_3)^{2y} T(0,-1)(-1,0)^{f(3y)}$$

corresponds to multiplication by 3 and changes the location of automaton from $(g(y), y)$ to $(g(3y), 3y)$. The sequence of movements that guarantees the division by 2 and 3 can be easy derived from the operation of multiplications. It follows that the walking automaton interacting with environment $E(g)$ can simulate the computation of two counter machine where one of the counters is used as a scratchpad. Thus the reachability problem of a walking automaton in environment $E(g)$ is undecidable and the model is universal.

Corollary 1. *Let $g : \mathbb{N} \to \mathbb{N}$ be an integral polynomial function of degree n, where $n \geq 2$. The environment $E(g)$ is non-efficient.*

Proof. Let $g(x) = a_n x^n + \ldots + a_1 x + a_0$ and f(x)= g(x+1)-g(x). Since f is also polynomial, i.e. $f(x) = b_n x^n + \ldots + b_1 x + b_0$, then $f(x) \equiv b_0 \pmod{x}$. It follows form Theorem 1 that the environment $E(g)$ is non-efficient.

Moreover we can change the main theorem, but using the same ideas of computations with respect to some modulo.

Corollary 2. *Given a constant $p \in \mathbb{N}$. Let function f fulfil condition $f(n) \equiv n - p \pmod{n}$ for all $n > p$ then an environment $E(g)$ is non-efficient.*

Corollary 3. *Given a constant $p \in \mathbb{N}$. Let function f fulfil condition $f(n) \equiv \lfloor \frac{n}{2} \rfloor \pmod{n}$ for all $n \in \mathbb{N}$ then an environment $E(g)$ is non-efficient.*

Corollary 4. *Given a function $f : \mathbb{N} \to \mathbb{N}$ that can be represented as $f_1(x) \cdot f_2(x)$, where $f_1(n) \equiv 0 \pmod{n}$ and $f_2(n)$ is any nondecreasing function. An environment $E(g)$ is non-efficient.*

We can also use Theorem 1 to show that for any environment limited by unbounded nondecreasing function there exist a sub-environment where a finite automaton exhibits universal behaviour:

Corollary 5. *Let $E(h) = \{(x, y) \subseteq \mathbb{N} \times \mathbb{N} | x \geq h(y)\}$ and $h : \mathbb{N} \to \mathbb{N}$ is an unbounded integral nondecreasing function. Then there is a function $g : \mathbb{N} \to \mathbb{N}$, such that $E(g) \subseteq E(h)$ and $E(g)$ is non-efficient.*

Proof. Let $g : \mathbb{N} \to \mathbb{N}$ is a function such that $g(1) = 1$, $g(x) = g(x-1) + f(x-1)$ and let $f(x) = x \cdot c(x)$, where $c(x) : \mathbb{N} \to \mathbb{N}$ is large enough to satisfy to the following property:

$$f(x) > h(x+1) - h(x).$$

From the above construction follows that $E(g) \subseteq E(h)$. On the other hand the fact that $f(x) \equiv 0 \pmod{x}$ gives us that the environment $E(g)$ is non-efficient by Theorem 1.

5 Conclusion

The decidability of reachability problem for automaton interacting with environments defined by other functions is still an open question. Our conjecture is that the reachability problem for $E(g)$ is undecidable for any everywhere defined function $f : \mathbb{N} \to \mathbb{N}$, (i.e., not only for those where $f(n) \equiv \text{constant} \pmod{n}$).

Though we show in Corollary 5 that bounds on increase rate of the environment width does not have any influence on universality, it is not clear what would be the cornerstones of undecidability for some specific environment, e.g. when environment is limited by a logarithmic function:

$$E_{log} = \{(x, y) \subseteq \mathbb{N} \times \mathbb{N} | y \leq log(x)\}.$$

References

1. M. Blum and C. Hewitt: Automata on a 2-dimensional tape, Proc. 8th Annual Symp. on Switching and Automata Theory, 1967, 155-160.
2. V.I. Grunskaya. Some properties of trajectories of automata in labyrinths. PhD Thesis. Moscow State University, Moscow, 1994.
3. Dora Giammarresi: Finite State Recognizability for Two-Dimensional Languages: A Brief Survey. Developments in Language Theory 1995, 299-308.
4. H.D. Ebbinghaus: Undecidability of some domino connectability problems, Zeitschr. f. math. Logik und Grundl. der Math. 28, 1982, 331-336.
5. V.M. Glushkov, G.E. Tseitlin and E.L.Yushchenko. Algebra, Languages and Programming [in Russian], Naukova Dumka, Kiev, 1989.
6. Grunsky I.S., Kurganskyy A.N. Indistinguishability of finite automata interacting with an environment. Dokl. AN Ukraine, 11(1993), 31-33.
7. Grunsky I.S., Kurganskyy A.N. Indistinguishability of finite automata with constrained behaviour, Cybernetics and systems analysis, 5(1996), 58-72.
8. Katsushi Inoue, Itsuo Takanami. A survey of two-dimensional automata theory. Inf. Sci. 55(1-3) (1991) 99-121
9. M. Jantzen and O. Kurganskyy. Refining the hierarchy of blind multicounter languages and twist-closed trios. *Information and Computation*, 185, 2003, 159-181.
10. V.B. Kudryavtsev, S.V. Alyoshin and A.S.Podkolzin. An introduction to the Theory of Automata [in Russian], Nauka, Moscow, 1985.
11. V. B. Kudryavtsev, Sh. Ushchumlich, and G. Kilibarda. On the behavior of automata in labyrinths. Discrete Math. and Applications, 3, 1993, 1-28.
12. G. Kilibarda, V. B. Kudryavtsev and Sh. Ushchumlich. Collectives of automata in labyrinths Discrete Mathematics and Applications 13 (5), 2003, 429-466.
13. Kurganskyy A.N. Indistinguishability of the finite automata interacting with an environment, Thesis, Saratov State University, Russia, 1997.
14. Kurganskij A.N. Indistinguishability of finite-state automata with respect to some environments. (Russian, English) [J] 37, No.1, 33-41, 2001; translation from Kibern. Sist. Anal. 2001, No.1, 43-55 2001.
15. A. Kurganskyy, I. Potapov. On the computation power of finite automata in two-dimensional environments. Developments in Language Theory 2004, LNCS 3340, 261-271.
16. A. Kurganskyy, I. Potapov. On the bound of algorithmic resolvability of correctness problems of automaton interaction through communication channels. *Cybernetics and System Analysis*, 3, 1999, 49-57.
17. Maurice Mergenstern. Frontier between decidability and undecidability: a survey. Theoretical Computer Science, v.231, 2000, 217-251.
18. M. Minsky. Computation: Finite and Infinite Machines. Englewood Cliffs, N.J.: Prentice-Hall, 1967.
19. C.Moore and J. Kari. New Results on Alternating and Non-Deterministic Two-Dimensional Finite-State Automata. International Symposium on Theoretical Aspects of Computer Science 2001, LNCS 2010, 396-406.
20. Tokio Okazaki, Katsushi Inoue, Akira Ito, Yue Wang. Space Hierarchies of Two-Dimensional Alternating Turing Machines, Pushdown Automata and Counter Automata. IJPRAI 13(4), 1999, 503-521.

On an embedding of N_5 lattice into the Turing degrees

Igor Kuznetsov

Ulyanovsk State Univerity
Ulyanovsk, Russia
igor_ku@pisem.net

We study a problem of an embedding of nondistributive lattice N_5 as initial segment of the Turing degrees. The main problem of such embedding is that the lattice N_5 has no a finite fine presentation. This involves an inclusion in the construction a growing forest of k-splitting trees with a larger and larger k.

We show that a standard procedure of the embedding method by Lachlan and Lerman is not completely adequate in the case of N_5.

Complexity in a Type-1 Framework for Computable Analysis

Branimir Lambov

BRICS
Department of Computer Science
University of Aarhus, Denmark
`http://www.brics.dk/~barnie`
`barnie@brics.dk`

Abstract. Implementations of real number computations have largely been unusable in practice because of the very bad performance ratio between them and machine precision floating point numbers. Since a significant portion of the problem is the type-2 nature of the computable analysis frameworks usually employed, this paper tries to address this by creating a type-1 framework similar to the domain theoretic approach, with extensions that permit us to reason about the complexity of real numbers and functions.

1 Introduction

Using real numbers with a computer is a very difficult matter. Although the theoretical background for exact real number computation has been around for some time, it is very rarely if ever used in practice.

One of the most important reasons for this is the serious performance discrepancy between programs working with machine precision floating point numbers and ones working with computable reals. The ratio is huge, ranging from thousands to no less than 100 times even in cases where higher precisions are not actually needed, and even though attempts to implement packages for real number computations exist ([3, 12, 15]), they could never be used in practice because of this frightening and unjustified slowdown.

In our attempts to implement a good real number system, we discovered that a large portion, well over half, of the time in a low-precision computation is spent because of the type-2 character of the theoretical framework. More specifically, the need to maintain a history of a real number computation and perform the actual evaluations at a point in time different from the user's programming imposes several significant performance problems:

– slow *memory management* is needed to build and destroy (or garbage collect) expression dags
– when a term's value is requested, *recursive evaluation* has to traverse the graph, which requires extra storage proportional to the term's depth

- the information for the *lifetime and locality* of each variable, explicitly given by a programmer or compiler in a program, is irrevocably lost in the term representation
- the *preferred ordering* of the computations is lost
- *compiler optimizations* cannot be performed on the expression.

All of these items are uninteresting from a theoretical point of view, but in practice they contribute to such a large portion of the execution time of a real number program, that one could not expect to significantly improve the machine precision/real numbers performance ratio without handling them.

To do so, one has to take a theoretical approach that describes real functions as type-1 objects. In this paper we are describing such an approach which is useful in practice and is similar to Grzegorczyk's definition from [6] and the domain theoretic framework for Computable Analysis by Edalat and Sünderhauf [4].

The main problem this paper addresses is the question of complexity in a type-1 model. In such a model, complexity cannot be simply introduced as a restriction on the class of functions used in the system, because the system contains an implicit unbounded search which makes all complexity classes that can define the minimization normal form, down to polynomials over the integers, equivalent in terms of expressive power.

The framework described in this paper is used in an actual implementation[1] of an exact real numbers package. Recently implemented features of the package demonstrate the practical effect of using the theoretical framework: with this system it is now possible to write programs with real numbers for which the machine precision/real numbers performance ratio ranges from eight-nine to one for computations that do not require extra precision. In other words, the difference is small enough to make real number computations useable.

2 Prerequisites

In this paper we will be referring to different levels of the finite type hierarchy. Type 0 is the natural numbers and every other finite type is a function that takes argument of a lower type. Type 1 is the type of the functions over the natural numbers and Type 2 is the set of the functionals that take at least one type-1 function as argument.

Let $\mathbb{V}$ be an enumerable dense subset of the reals that contains the dyadic numbers. We will be using the following generally accepted definition as the established notion of computable real numbers and functions to which we will compare our model:

Definition 1. *A Cauchy function representation (CF-representation) of a real number α is a function $a : \mathbb{N} \to \mathbb{V}$, such that* $\forall n \in \mathbb{N}\ \left(|a(n) - \alpha| < 2^{-\mathrm{lth}(n)}\right)$

[1] RealLib, available at `http://www.brics.dk/~barnie/RealLib/` See also [11] for a description of an earlier version.

(we use $\mathrm{lth}(n) = \lfloor \log_2 n \rfloor + 1$ instead of simply n in the exponent as the latter would not allow us to define the class of feasible real numbers as the ones having a poly-time CF-representation)

Definition 2. *A Cauchy function representation of a partial function* $\phi : \mathbb{R} \to \mathbb{R}$ *is a partial functional* $\Phi : (\mathbb{N} \to \mathbb{V}) \times \mathbb{N} \to \mathbb{V}$*, such that*

$$\forall \alpha \in \text{ dom } \phi, \forall a - \text{CF} - \text{representations of } \alpha$$
$$\forall n \in \mathbb{N} \left((a,n) \in \text{ dom } \Phi \wedge |\Phi(a,n) - \phi(\alpha)| < 2^{-\mathrm{lth}(n)} \right)$$

Definition 3. *A real number or a real function is computable in a class C of computable functions, resp. functionals, iff there exists a representation in C for it in the sense of Definitions 1 or 2 respectively.*

To assess the complexity of real numbers we can use restrictions on the class of functions used in Definition 1. The complexity notions for real functions are somewhat more complicated, thus we have decided to compare our approach to two different complexity measures: Ko's notion of feasibility for Computable Analysis and type-2 complexity by restriction of the class of functionals used in Definition 2.

A well studied notion for feasibility of type-2 functionals is the class of the Basic Feasible Functionals (**BFF**, defined in [1]). They can also be defined as the type-2 restriction of the Basic Feasible Functionals of finite type ($\mathbf{BFF}^\omega$, [2]), and the latter can be defined as the higher-type functions that can be written as terms containing only

- constants 0 for every finite type
- variables for every finite type
- constants for every poly-time function
- the Σ and Π combinators for every combination of finite types
- bounded recursion on notation R_{bn}:

$$R_{bn}(x,y,g,h) = \begin{cases} y, & \text{if } x = 0 \\ \min\left(g(x, R_{bn}(\lfloor x/2 \rfloor, y, g, h), h(x)\right), & \text{otherwise} \end{cases}$$

Two essential properties that we will be using in the complexity part of our paper are the facts that $\mathbf{BFF}^\omega$restricted to Type 1 coincides with the poly-time functions, and $\mathbf{BFF}^\omega$restricted to Type 2 (i.e. **BFF**) coincides with the functionals that are computable by an oracle Turing machine in time which is a second order polynomial to the lengths of the inputs ([8]). We will also use the fact that **BFF** can define the first normal form for type-2 functionals ([8]).

We will use this definition for feasibility of real functions in the usual sense for Computable Analysis:

Definition 4 (Ko, [10]). *A function* ϕ *is poly-time computable on* $[a,b] \subseteq$ dom ϕ *in Ko's sense iff there is an oracle Turing machine computing it in the dyadic representation, which runs in time polynomial to the precision given in unary notation.*

This definition of complexity is equivalent to the signed digit one defined in [14] among others, and, using the next theorem, can be easily translated to the setting of **BFF**.

Theorem 1. *A partial real function is poly-time computable on $[a, b]$ in the sense of Ko iff it is* ***BFF****-computable on the same interval.*

Proof. A functional is in **BFF** if and only if there exists an oracle Turing machine computing it, running in time which is a second-order polynomial in the length of the inputs. In a compact interval, there exist representations of any real number that satisfy $\mathrm{lth}(B(k)) \leq p(\mathrm{lth}(k))$ for a polynomial p (using dyadic representations cut after the $\mathrm{lth}(k)$'th digit), and therefore a second order polynomial in $\mathrm{lth}(k), \mathrm{lth}(B)$ does not give more power than simply a polynomial in $\mathrm{lth}(k)$. □

Our investigation of the connections between complexity in our model and type-2 and Computable Analysis complexity also require a very useful tool from Proof Theory, the concept of majorizability:

Definition 5 (W.A. Howard [7]). *We define x^* maj_ρ x for a finite type ρ by induction on the type:*

$$x^* \ \mathrm{maj}_0 \ x := x^* \geq x,$$

$$x^* \ \mathrm{maj}_{\tau \to \rho} \ x := \forall y^*, y \left(y^* \ \mathrm{maj}_\tau \ y \to x^*(y^*) \ \mathrm{maj}_\rho \ x(y) \right).$$

We will say that a class of function(al)s C is majorizable, if for every function(al) f in C there exists $f^* \in C$ with f^* maj f, where the majorization relation is of the appropriate type.

The majorizability relation defines monotonicity in higher types, and for us the most interesting instantiation of the definition is in the case $\rho \equiv 0$ used in the form $x^* \ \mathrm{maj}_{\tau \to 0} \ x \wedge y^* \ \mathrm{maj}_\tau \ y \to x^*(y^*) \geq x(y)$, giving us the possibility to bound the result of the application of one higher-type functional to another as a number.

It is not hard to see that the poly-time functions and $\mathbf{BFF}^\omega$ are majorizable classes (detailed proof can be found in [11]). Other majorizable classes are the class of the primitive recursive functionals of finite type (in the sense of Kleene S1-S8 ([9]) as well as in the sense of Gödel ([5]) and any specific level in Gödel's primitive recursive hierarchy), any level of the Grzegorczyk hierarchy and many others, but not the class of all type-2 functionals (e.g. if the function

$$F(f) = \begin{cases} 0, & \text{if } \forall x (f(x) = 0) \\ \mu x[f(x) = 0], & \text{otherwise} \end{cases}$$

were majorized by F^*, then $F^*(\lambda x.1)$ would bound F applied to all zero/one functions which is not possible) or the class of the partial computable functionals if the notion is extended in a suitable way to accommodate partiality.

3 Model

The basic objects we are going to use contain approximation information and estimation of the amount of error in this approximation. To be able to define a class of real functions equivalent to the computable ones in the sense of Definition 2, a totally indeterminate value has to be allowed (otherwise e.g. division cannot be defined, see [13]). We do this by allowing an infinitely large value for the error.

Let $\mathbb{E}$ be a subset of the positive rational numbers which contains 2^{-n} for every natural n, to which the special value ∞ is added, and which has a poly-time encoding and a poly-time comparison operator that respects ∞. It is possible to define encodings of pairs $\mathbb{V} \times \mathbb{E}$ with the following properties:

- $\mathrm{lth}(\langle a, b\rangle)$ is polynomial in $\max(\mathrm{lth}(\langle a\rangle_{\mathbb{V}}), \mathrm{lth}(\langle b\rangle_{\mathbb{E}}))$
- $\langle a, b\rangle \geq \langle a\rangle_{\mathbb{V}}$ and $\langle a, b\rangle \geq \langle b\rangle_{\mathbb{E}}$
- $\langle 2^{-n}\rangle_{\mathbb{E}} \geq 2^n$
- there exist poly-time functions that convert between the encodings of $\mathbb{V}$ and $\mathbb{E}$, rounding up if a number in $\mathbb{V}$ cannot be represented in $\mathbb{E}$
- multiplication and division by 2 are poly-time (and thus also multiplication by $2^{\pm \mathrm{lth}(k)}$) in both $\mathbb{V}$ and $\mathbb{E}$
- addition and the floor function $\lfloor\cdot\rfloor$ in $\mathbb{V}$ are poly-time
- there exists a function $\mathrm{dya}(n, d)$ that selects a code for the dyadic number $n2^{-d}$, such that whenever a, b, c, d are positive integers, $a \leq c \wedge b \leq d \rightarrow \mathrm{dya}(a, b) \leq \mathrm{dya}(c, d)$
- the absolute value operator on the codes is such that $\langle v\rangle_{\mathbb{V}} \leq \langle |v|\rangle_{\mathbb{V}}$ for any $v \in \mathbb{V}$

These properties can be satisfied e.g. for $\mathbb{V} = \mathbb{Q}$ and $\mathbb{E} = \mathbb{Q}^{+}_{\infty}$ by the Cantor pairing, the encoding of rational numbers q as $\Pi(n, d)$, such that

$$q = (-1)^n \frac{\lfloor (n+1)/2 \rfloor}{d},$$

and the encoding of ∞ as $\Pi(0, 0)$.

Having the distinction between the sets $\mathbb{V}$ and $\mathbb{E}$ is prompted by the need to include ∞, but also closely follows the choice one would often make when an actual implementation is developed as one might prefer a simpler (and thus more efficient) representation of the error information.

3.1 Partial approximation representations

Definition 6. *A* partial approximation *to a real number α is a pair (v, e) of type $\mathbb{V} \times \mathbb{E}$, such that $|v - \alpha| < e$. We will denote the class of partial approximations to α with $\mathbb{A}_\alpha$, and the class of partial approximations to any real with $\mathbb{A}_{\mathbb{R}} = \cup_{\alpha \in \mathbb{R}} \mathbb{A}_\alpha$. If $a \in \mathbb{A}_{\mathbb{R}}$ we will use a_v, a_e to denote respectively the value and error in a.*

Definition 7. *A* partial approximation representation, p.a.r., *of a real number α is a function $A : \mathbb{N} \to \mathbb{A}_\alpha$, for which $\forall k \exists n ((A(n))_e \leq 2^{-k})$.*

If a real number is computable, then it certainly has a computable p.a.r.: if B is a representation of α, then $\lambda n.(B(n), 2^{-\mathrm{lth}(n)})$ is one of its p.a.r.'s. Conversely, if a is a p.a.r. of α, then

$$\lambda k.A(\mu n[A(n)_e \leq 2^{-\mathrm{lth}(k)}])_v \tag{1}$$

is a valid CF-representation for it.

This equivalence does not hold for restrictions of the notion of computability. Because of the unbounded search in (1), it is possible to define all computable reals using p.a.r.'s in subrecursive classes such as primitive recursive, elementary or poly-time functions. For a proof of this, see [13].

For real functions, we want to have objects that operate on partial approximations instead of the full representations. They will have to convert approximations to an input to approximations to the result of the application of the function, and also we need to require that the precision of the output approximations gets arbitrarily good as the precision of the input increases. In other words,

Definition 8. *A* partial approximation representation *of a partial function $\phi : \mathbb{R} \to \mathbb{R}$ is a partial function $F : \mathbb{A}_\mathbb{R} \to \mathbb{A}_\mathbb{R}$, such that for any choice of $\alpha \in \operatorname{dom} \phi$ and a partial approximation representation A of α, $\lambda n.F(A(n))$ is a partial approximation representation of $\phi(\alpha)$.*

Remark 1. This definition implies $a \in \mathbb{A}_\alpha \to F(a) \in \mathbb{A}_{\phi(\alpha)}$ for $\alpha \in \operatorname{dom} \phi$, because for any p.a.r. A of α we can create

$$B(n) := \begin{cases} a, & \text{if } n = 0 \\ A(n-1), & \text{otherwise} \end{cases},$$

which is another p.a.r. of α that has $B(0) = a$, hence $F(a)$ has to be a partial approximation to $\phi(\alpha)$.

Remark 2. In contrast to the domain theoretic approach ([4]), where successive applications of the representation function to the same interval are required to give better and better approximation to the image interval, our requirement is to only be able to provide a superset of the image interval and to only make it smaller if the input interval gets smaller.

3.2 Computability

We have severely restricted the information to which the function object has access; nevertheless, the following theorem proves we have not restricted the class of real functions that are computable:

Theorem 2. *A partial function $\phi : \mathbb{R} \to \mathbb{R}$ is computable iff it has a computable p.a.r.*

Proof. ($\leftarrow$) If we have a p.a.r. F of a function ϕ, and $\alpha \in \operatorname{dom}\phi$, then the functional

$$\Phi(B,n) := F(B(m), 2^{-\mathrm{lth}(m)}) \text{ with } m = \mu p \left[(F(B(p), 2^{-\mathrm{lth}(p)})_e \leq 2^{-\mathrm{lth}(n)} \right] \quad (2)$$

is total in n for any CF-representation B of α since from Definitions 8 and 7 the minimization will always stop, and Definition 6 together with Remark 1 ensures $|\Phi(a,n) - \phi(\alpha)| < 2^{-\mathrm{lth}(n)}$. □

Proof. ($\rightarrow$) Fix a CF-representation Φ for ϕ.

For any $a \in \mathbb{A}_{\mathbb{R}}$ with $a_e < 1$, we can effectively find the largest natural number m with the property $2^m a_e < 1$. If $a_e \geq 1$, we take $m = 0$. Define the function

$$b := \lambda n.2^{-\mathrm{lth}(n)} \lfloor 2^{\mathrm{lth}(n)} a_v + 1/2 \rfloor. \quad (3)$$

For $0 \leq \mathrm{lth}(n) < m$ we have that if $\alpha \in \mathbb{A}_\alpha$

$$|b(n) - \alpha| \leq |a_v - \alpha| + 2^{-(\mathrm{lth}(n)+1)} \leq 2^{-m} + 2^{-(\mathrm{lth}(n)+1)} \leq 2^{-\mathrm{lth}(n)}$$

Using the fact that exceptions are computable and that given the code of a computable functional Φ, we can construct an equivalent one $\Phi^\dagger$ that honors a new exception x, we can create a function

$$b\lceil m := \lambda n. \begin{cases} b(n), & \text{if } n < m \\ \textbf{raise } x, & \text{otherwise} \end{cases}$$

and then define

$$\Phi^\ddagger(B,n,m) := \textbf{try } \Phi^\dagger(B\lceil m, n) + 1 \textbf{ catch}(x)\ 0. \quad (4)$$

(i.e. $\Phi^\ddagger$ will return $\Phi(b,n)+1$ if b restricted to length m was sufficient to compute it, and 0 otherwise)

We will now prove that the function

$$F(a) := \begin{cases} (0,\infty), & \text{if } l = 0 \\ (\Phi^\ddagger(b, l-1, m) - 1, 2^{-\mathrm{lth}(l-1)}), & \text{otherwise} \end{cases}$$

for

$$l = \mu p \leq m \left[\Phi^\ddagger(b,p,m) = 0 \right] \quad (5)$$

is the required p.a.r. of ϕ. To do this, we need to prove that $G = \lambda n.F(A(n))$ is a p.a.r. of $\phi(\alpha)$ for any p.a.r. A of α.

The first condition, $F(a) \in \mathbb{A}_{\phi(\alpha)}$ for any $a \in \mathbb{A}_\alpha$, follows from the requirement for Φ and the fact that there is a CF-representation for α that starts with $b(0), b(1), \ldots, b(m-1)$.

For the second condition, we need to prove the existence of 2^{-k}-approximations to $\phi(\alpha)$ among $G(n)$ for any k. The sequence defined by

$$c := \lambda n.2^{-\mathrm{lth}(n)} \lfloor 2^{\mathrm{lth}(n)} \alpha + 1/2 \rfloor$$

is a proper CF-name for α. If α is not a dyadic number, then for an arbitrary n, $|\alpha - c(n)| < 2^{-\mathrm{lth}(n)-1}$. There exists q depending on n, such that $|\alpha - c(n)| \leq 2^{-\mathrm{lth}(n)}(1/2 - 2^{-(q-\mathrm{lth}(n))})$, and for all partial approximations a with $a_e < 2^{-q}$ we have $2^{\mathrm{lth}(i)}|a_v - c(i)| < 1/2$ for all $0 \leq i \leq n$. But this implies that the sequence obtained by (3) coincides with c on the first $n+1$ elements.

Now, since Φ would look at finitely many elements of c to produce a value with any precision 2^{-k}, using that count in the procedure described above, we can come up with a q supplying a long enough sequence. Combining this with a requirement that the minimization (5) reaches the target precision, we have $(F(a))_e \leq 2^{-k}$ for all a's with $a_e \leq 2^{-\max(q,k)}$, and since A has arbitrarily close approximations, this can be satisfied for $a = A(n)$ for some n.

If α is a dyadic number, i.e. $\exists n(c(n) = \alpha)$, then there are only finitely many variations of b that can exists, because they have to coincide after the first $n+1$ positions. Then there exists a maximum m for the number of lookups Φ can make to any of these b's in order to get a 2^{-k}-precise result. Hence $a_e \leq 2^{-\max(m,k)}$ suffices to get the required precision for $F(a)$. □

As in the case of real numbers, this equivalence does not hold for subclasses of the type-2 computable functions. To define all computable functions, it suffices to use severely restricted type-1 computability subclasses:

Theorem 3. *A partial real function is computable iff it has a p.a.r. in any subrecursive class C that contains the poly-time functions.*

Proof. ($\rightarrow$) It suffices to change the definition of $\Phi^{\ddagger}$ to a version bounded in execution time:

$$\Phi^{\ddagger}(B, n, m) := \mathbf{try}\ \Phi^{m}(B\lceil m, n) + 1\ \mathbf{catch}(x)\ 0.$$

where by Φ^m we denote $\Phi^{\dagger}$ executed for m steps throwing an exception if Φ did not halt, which can be done in a basic feasible functional (because **BFF** can define the first normal form for type-2 functionals).

Since m is of the order of $\mathrm{lth}(a)$ for the encoding of a it is possible to do all required steps in time polynomial to $\mathrm{lth}(a)$. The proof of the existence of good approximations can be carried out here as well, the only difference being the need to satisfy a condition in the form $a_e \leq 2^{-max(q,k,s)}$ for s being the number of steps it takes for Φ to complete its evaluation on b of length q.

The p.a.r. is type-1 basic feasible, therefore it is poly-time. □

Proof. ($\leftarrow$) Follows from the previous theorem. □

3.3 Complexity

Real numbers. In order to be able to speak about different complexity classes of real numbers, we must make a definition which requests more from our functions in order to avoid the minimization in (1). This gives rise to the following definitions and equivalence property:

Definition 9. *A* modulus *for a p.a.r. A of a real number α is a function $m : \mathbb{N} \to \mathbb{N}$, such that for all k, $(A(m(k)))_e \le 2^{-\mathrm{lth}(k)}$.*

Definition 10. *We will say that a real number is p.a.r.-computable in a given class C of computable functions, if there exist both a p.a.r. and a modulus for it in C.*

Theorem 4. *A real number is computable in a subrecursive class C that contains the poly-time functions and is closed under composition if and only if it is p.a.r.-computable in C.*

Proof. Take $A := \lambda n.(B(n), 2^{-\mathrm{lth}(n)})$ and $m := \lambda n.n$, or for the other direction $B := \lambda k.A(m(k))_v$. □

On the level of feasible functions, poly-time p.a.r. computability coincides with Ko's notion of poly-time computable real numbers [10] (Ko speaks about numbers given in unary notation, which is equivalent to the $\mathrm{lth}(n)$ parameter used in our definitions).

Type-2 complexity for functions. Again taking the p.a.r. of a real function we lose all complexity information about that function. To talk about complexity classes, we define a function that can replace the minimization in (2):

Definition 11. *A* modulus *for a p.a.r. F of a partial real function ϕ is a partial functional $M : (\mathbb{N} \to \mathbb{A}_{\mathbb{R}}) \times (\mathbb{N} \to \mathbb{N}) \times \mathbb{N} \longrightarrow \mathbb{N}$, such that for all $\alpha \in$ dom ϕ, p.a.r. A of α, moduli m for A,*

$$\forall k((F(A(M(A, m, k))))_e \le 2^{-\mathrm{lth}(k)}). \tag{6}$$

Note that even though the actual function object is a type-1 object, we now introduce a type-2 operation to characterize it. However, some extra flexibility comes from the separation of these two objects: to implement e.g. a feasible real function you do not have to implement a feasible type-2 object, but only need to prove that it exists. Moreover, if a CF-representation of a function needs extra information to be in a certain class (e.g. division needs evidence that the denominator is non-zero to be primitive recursive), it will in general only be needed for the modulus.

Definition 12. *We will say that a real function is p.a.r.-computable in a given class C of computable type-2 functionals, if both a computable p.a.r. and its modulus can be found in C.*

Theorem 5. *If a function is p.a.r.-computable in a given class C that contains* ***BFF*** *and is closed under functional substitution, then it is computable in the same class.*

Proof. For $\phi : \mathbb{R} \to \mathbb{R}$, $\alpha \in$ dom ϕ, F- p.a.r. of ϕ, M-modulus for F, and B - CF-representation of α, take

$$\Phi(B, n) := (F(A(M(A, \lambda p.p, n))))_v$$

where

$$A := \lambda p.(B(p), 2^{-\mathrm{lth}(p)}).$$

A is a p.a.r. for α with a modulus $\lambda p.p$, and hence from M being a modulus to F, we have $|\Phi(B, n) - \phi(\alpha)| < 2^{-\mathrm{lth}(n)}$. Φ is a basic feasible functional relative to F and M, therefore it is in C. □

The other direction is more complicated. First we will verify that p.a.r.-computability coincides with CF-computability:

Theorem 6. *If a partial function $\phi : \mathbb{R} \to \mathbb{R}$ is computable, then it is p.a.r.-computable.*

Proof. We've already proved in Theorem 2 that there exists a computable p.a.r. to every computable real function. If it is F, then

$$M(A, m, n) := \mu p[F(A(p))_e \leq 2^{-\mathrm{lth}(n)}]$$

is a modulus for F. □

This modulus does not even use the modulus for the real number. This is true, because in the presence of minimization brute force search makes the moduli redundant.

This is not the case for restricted complexity classes. To prove the equivalence between p.a.r. and CF-computability on some of them, we need the higher-type monotonicity we have in the majorizable classes and the following lemma:

Lemma 1. *Let b be defined as*

$$b(n) := \mathrm{dya}(\lfloor 2^{\mathrm{lth}(n)} a_v + 1/2 \rfloor, \mathrm{lth}(n)). \tag{7}$$

Then for all $\alpha \leq a_0, a \in \mathbb{A}_\alpha, b$, created by (7) for a with $a_e \leq 1$,

$$J(a_0) \ \mathrm{maj}_1 \ b$$

where

$$J(a_0) = \lambda n.\mathrm{dya}(1 + \lfloor 2^{\mathrm{lth}(n)} a_0 + 1/2 \rfloor, \mathrm{lth}(n)). \tag{8}$$

Proof. Since $a_e \leq 1$, we have $|a_v| < a_0 + 1$ and therefore by the properties of the encoding $J(a_0)(n) \geq b(n)$, and also, since when n is increased both the numerator and denominator in (8) do not decrease, we have $\forall k \leq n(J(a_0)(n) \geq J(a_0)(k) \geq b(k))$, which means $J(a_0)$ maj$_1$ b. □

Theorem 7. *If a partial real function is computable in a majorizable class of type-2 functionals that contains **BFF**and is closed under functional substitution, then it is p.a.r.-computable in that class.*

Proof. We will use the proof of Theorem 2, substituting the definition (3) of b with (7). All operations used in the generation of F can be done without leaving the class of Φ. Hence F is in the class. We now need to find a modulus for it.

In the class of Φ there exists a functional Ψ that does exactly the same job as Φ, but instead of returning the approximation it gives the largest k to which B was applied. Since the class contains this functional and is majorizable, it also contains a majorizer Ψ^* for it. The modulus for A gives us means to bound the absolute value of the real number described by it, therefore, with the previous lemma, there is a poly-time function $b^* := J(|A(m(0))| + 1)$ which majorizes all functions b generated by partial approximations with error less than 1.

Hence $\Psi^*(b^*, n) \geq \Psi(b, n)$ for all good b's, in particular for the one (call it b_0) generated by $a_0 = A(m(\Psi^*(b^*, n)))$, which means $\Phi^\dagger(b_0, n)$ will not raise an exception, and $F(a_0)$ will give a result with the required precision.

Hence $M(A, m, n) = m(\Psi^*(J(|A(m(0))| + 1), n))$ is a modulus for F. □

Real number complexity for functions. In the previous subsection we found correspondence between complexities in this model and type-2 complexity. As this is not the complexity measure normally used for real functions, we also define notions which are more closely related to the latter by defining type-1 moduli on closed subsets of the domain:

Definition 13. *A* uniform modulus *on* $[a,b] \subseteq \operatorname{dom}\phi$ *of a p.a.r. F of a real function ϕ is a function $U : \mathbb{N} \to \mathbb{N}$, such that*

$$\forall \alpha \in [a,b] \forall A - p.a.r.\ of\ \alpha \forall k \forall n (A(n)_e \leq 2^{-\mathrm{lth}(U(k))} \to (F(A(n)))_e \leq 2^{-\mathrm{lth}(k)})$$

Theorem 8. *A partial real function ϕ is computable in a majorizable class of type-2 functionals on $[a,b] \subseteq \operatorname{dom}\phi$ if and only if it has a p.a.r. and a uniform modulus in the same class.*

Proof. ($\to$) Use a and b to find an upper bound for the absolute value of α, then apply the same reasoning as in the previous proof. □

Proof. ($\leftarrow$) $M(A, m, k) = m(U(k))$ is a modulus for all A's representing reals in the interval, thus ϕ is p.a.r.-computable in the class. □

With this definition we're back at the type-1 level, and we also have a few important equivalences:

Corollary 1. *A partial real function ϕ is primitive recursive in the sense of Kleene ([9]) on $[a,b] \subseteq \operatorname{dom}\phi$ iff it has a primitive recursive p.a.r. and a primitive recursive uniform modulus on $[a,b]$.*

Corollary 2. *A partial real function ϕ is **BFF**-computable on $[a,b] \subseteq \operatorname{dom}\phi$ iff it has a poly-time p.a.r. and a poly-time uniform modulus on $[a,b]$.*

Combined with Theorem 1, the latter allows us to state the following equivalence property:

Corollary 3. *A partial real function ϕ is feasible in the sense of Ko on $[a,b] \subseteq \operatorname{dom}\phi$ if and only if it has a poly-time p.a.r. and a poly-time uniform modulus on $[a,b]$.*

4 An example: reciprocal of a real number

The task in this section will be to define a suitable p.a.r. of the function $1/\alpha : \mathbb{R} \to \mathbb{R}$ and to inspect its properties.

Theorem 9. *The poly-time function F defined as*

$$F(a) = \begin{cases} (\frac{1}{a_v}, \frac{a_e}{|a_v|(|a_v|-a_e)}), & \text{if } a_e < |a_v| \\ (0, \infty), & \text{if } a_e \geq |a_v| \end{cases} \tag{9}$$

is a p.a.r. of the reciprocal function on the reals.

Proof. Given a fixed $\alpha \neq 0$, if a is a partial approximation to α with $a_e < a_v$, then

$$\left|\frac{1}{\alpha} - \frac{1}{a}\right| \leq \frac{1}{|a_v| - a_e} - \frac{1}{|a_v|} \leq \frac{|a_v| - (|a_v| - a_e)}{|a_v|(|a_v| - a_e)} = \frac{a_e}{|a_v|(|a_v| - a_e)},$$

which means F converts partial approximations to α into partial approximations to $1/\alpha$. If a_0 is a positive rational number smaller than $|\alpha|$, then to get $(F(a))_e \leq \varepsilon$, it is enough to supply a partial approximation with $a_e < a_0^2\varepsilon/2$ (assuming w.l.o.g. $a_0, \varepsilon \leq 1$). $\square$

It is a well known fact that the reciprocal function on the reals is not even primitive recursively computable. However, it is poly-time computable on every closed interval that does not contain 0. How does this translate to our framework?

Having a closed interval that does not contain 0 is equivalent to having witness information for the strict positivity of $|\alpha|$. But this extra information is enough to allow us to define

$$M(a_0, A, m, k) = m(\langle a_0\rangle_{\mathbb{E}} \# \langle a_0\rangle_{\mathbb{E}} \# k \# 2)$$

$$U(a_0, k) = \langle a_0\rangle_{\mathbb{E}} \# \langle a_0\rangle_{\mathbb{E}} \# k \# 2$$

(with # being the smash function, $x \# y = 2^{\mathrm{lth}(x)\mathrm{lth}(y)}$) which are, respectively, a modulus for F and an uniform modulus for F on every closed interval of the real line parameterized by the witness a_0. Therefore, F defined in (9) is uniformly linear in the error of the approximation a_e and the requested precision k, and quadratic in the value of the approximation a_v and the witness a_0 on the full real line.

Note that we needed the witness only to define and prove a property of the p.a.r. The representation itself is not changed by whether it can be found or not.

References

1. Cook, S., Urquart, A., *Functional interpretations of feasibly constructive arithmetic.* Ann. Pure Applied Logic **63**, pp. 103-200 (1993).

2. Cook, Stephen A. *Computability and complexity of higher type functions.* Logic from computer science (Berkeley, CA, 1989), 51–72, Math. Sci. Res. Inst. Publ., **21**, Springer, New York, 1992.
3. Edalat, A., *Exact Real Number Computation Using Linear Fractional Transformations*, Final Report on EPSRC grant GR/L43077/01
available at `http://www.doc.ic.ac.uk/~ae/exact-computation/exactarithmeticfinal.ps.gz`
4. Edalat, Abbas; Sünderhauf, Philipp *A domain-theoretic approach to computability on the real line.* Theoret. Comput. Sci. **210** (1999), no. 1, 73–98.
5. Gödel, K., *Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes.* Dialectica **12** (1958), 280–287.
6. Grzegorczyk, A., *On the definitions of computable real continuous functions*, Fundamenta Matematicae **44** (1957), 61–67.
7. Howard, W.A., *Hereditarily majorizable functionals of finite type.* In: Troelstra (ed.), Metamathematical investigation of intuitionistic arithmetic and analysis, pp. 454-461. Springer LNM **344** (1973).
8. Kapron, B. M.; Cook, S. A. *A new characterization of type-2 feasibility.* SIAM J. Comput. **25** (1996), no. 1, 117–132.
9. Kleene, S.C., *Recursive Functionals and Quantifiers of Finite Types I*, Trans. Amer. Math. Soc. **91** (1959), 1–52.
10. Ko, K.-I., *Complexity theory of real functions.* Birkhäuser, Boston-Basel-Berlin (1991), x+309 pp.
11. Lambov, B., *A two-layer approach to the computability and complexity of real functions.* Computability and complexity in analysis (Cincinnati, 2003), 279–302, Informatik Berichte, **302** (8/2003), Fernuniversität Hagen, 2003
see also `http://www.brics.dk/~barnie/`
12. Müller, N., *The iRRAM: Exact arithmetic in C++.* Computability and complexity in analysis. (Swansea, 2000). Lecture Notes in Computer Science **2064**. Springer (2001).
see also `http://www.informatik.uni-trier.de/iRRAM/`
13. Skordev, D., *Characterization of the computable real numbers by means of primitive recursive functions.* Computability and complexity in analysis (Swansea, 2000), 296–309, Lecture Notes in Computer Science **2064**, Springer (2001).
14. Weihrauch, K., *Computable Analysis.*, Springer (2000).
15. Yap, Chee, *Towards Exact Geometric Computation.* Computational Geometry : Theory and application, **3-23**, Sep 1997.
see also `http://www.cs.nyu.edu/exact/core/`

RealLib: an efficient implementation of exact real numbers

Branimir Lambov

BRICS,
University of Aarhus
`http://www.brics.dk/~barnie`
`barnie@brics.dk`

Exact real number arithmetic has had the reputation of being excessively slow in comparison to standard machine precision floating point. Even in cases where high precisions are not neccessary, real number systems tend to take hundreds of times longer to compute their results.

RealLib is a system that changes this. It uses an approach that treats functions on real numbers as lower-type objects, which allows it to run very close to the computer hardware. As a result, real number computations with RealLib can run nearly as fast as floating point computations with double precision.

This talk is a demonstration of the features of the library and its performance.

For more information and the current version of RealLib, visit `http://www.brics.dk/~barnie/RealLib/`.

A Programming Language for Automated Average-Case Time Analysis

Joseph Manning[1,2] and Michel Schellekens[1]

[1] Centre for Efficiency-Oriented Languages (CEOL)
Department of Computer Science
University College Cork
Cork, UK
[2] `http://www.cs.ucc.ie/staff/jmanning.html`
`manning@cs.ucc.ie`

This talk presents an overview of $\mathcal{ACETT}$ (Average-Case Execution Time Tool), a new programming language with the distinguishing feature that the average-case running time of its programs can be determined in a (semi-)automated fashion.

The language is built around a carefully-selected and integrated suite of data-structuring operations, each of which has the key property of being "randomness-preserving". Essentially, this means that the relative distribution of all possible inputs to an operation is preserved and reflected in the distribution of its outputs. We have proved that programs constructed from such operations are guaranteed to have average-case running times that are linearly-compositional, and so the average time of an entire program can be readily deduced from those of its constituent parts.

In this regard, $\mathcal{ACETT}$ is unique among languages, and appears to offer a fundamental new contribution to the field of programming.

Proof-Theoretical Methods in Combinatory Logic and $\boldsymbol{\lambda}$-Calculus

Pierluigi Minari

Department of Philosophy, University of Florence
Florence, Italy
minari@unifi.it

Abstract. The paper deals with 'analytic', *transitivity-free* proof-systems for combinatory logic **CL** and generalizations thereof, called "combinatory systems", recently introduced by the Author ([10]). We shall review some applications (to weak reduction and to decidability of certain subsystems of **CL**+ *ext*) of the main theorem about the eliminability of transitivity in these systems, showing how the latter considerably simplify the proof-theory of the corresponding equational theories. Finally, we will present some new (partial) results concerning the application of proof-theoretical methods to the equational theory of $\boldsymbol{\lambda}$-calculus.

1 Introduction

An *equational calculus* **E** over an equational language $\mathcal{L}_{\mathbf{E}}$ is usually presented as consisting of: (i) certain *specific axioms*, namely a set AX(**E**), closed under substitutions, of $\mathcal{L}_{\mathbf{E}}$-equations; (ii) the common deductive apparatus of equational logic: the axiom schema of *reflexivity* of equality, and the inference rules saying that equality is a *symmetrical* and *transitive* relation and a f-*congruence* w.r. to every functor f in $\mathcal{L}_{\mathbf{E}}$ — the substitution rule being superfluous in view of the closure requirement on AX(**E**). Then, by Birkhoff's *Completeness theorem*, one has: $\vdash_{\mathbf{E}} t = s$ iff $\mathrm{AX}(\mathbf{E}) \models t = s$.

These standard presentations are for most purposes very convenient. Yet the transitivity rule (which obviously cannot be dispensed with here, except that in trivial cases) possesses an inherently *synthetic* character in combining derivations (like *modus ponens* in Hilbert-style calculi for first-order logic), which frequently renders *naive* proof-theoretic arguments impossible. For instance, imagine we are interested in getting a *syntactic* consistency proof of a given **E**, for which we do not have a natural model ready at hand. We cannot, arguing by induction on the length of the derivations, conclude that $\nvdash_{\mathbf{E}} x = y$ (x, y two distinct variables) because $x = y$ neither is a specific axiom (assume it isn't) neither (by I.H.) can be obtained as a conclusion of one of the inference rules: this naive argument trivially breaks down when the case of the transitivity rule is considered (and in fact in this case only).

So, given that standard, 'synthetic' equational calculi do not lend themselves directly to proof-theoretical analysis, one might ask the question whether there are significant cases in which it is both *possible* and *useful* to turn a 'synthetic' equational calculus into an equivalent 'analytic' calculus, where the transitivity rule is actually *redundant.*

We will primarily consider one specific equational calculus, *combinatory logic* **CL** (or *the calculus of weak equality*; see e.g. [1], [2], [3], [4], [8]), which is perhaps the simplest undecidable theory.

The language of **CL** comprises the individual constants K and S, and the binary functor *ap* (application); as usual, ts is short for $ap(t, s)$, tsr for $(ts)r$, etc. AX(**CL**) consists of the instances of the two schemas:

$$\mathsf{K}ts = t \qquad \text{and} \qquad \mathsf{S}tsr = tr(sr) \ .$$

It is well known that so-called "mathematical" models of **CL** appeared only by the end of the Sixties. Until then only *term models* were known: and these were, as it is often said, *proof-theoretically* obtained by introducing a relation of *reduction* between terms, whose analysis (hinging on the fundamental *Church-Rosser theorem* and its corollary on the *uniqueness of normal form*) provides a satisfactory interpretation of provable equality and consequently a consistency proof for **CL**. Yet the reduction calculus is no more an equational calculus in the strict sense; perhaps, instead of speaking of a *proof-theory* of **CL**, it would be more appropriate to speak of an *operational semantics* for **CL** (see e.g. [5]).

In a recent paper of ours ([10]) new formal systems for combinatory logic **CL** and for more general systems of combinators (called "combinatory systems", see Sect. 1) are introduced. The new formal system for **CL** is obtained from the standard synthetic one as follows:

– the axioms on the primitive combinators K and S are replaced by the pair of *introduction* rules (on the left and on the right):

$$\frac{tp_1 \dots p_n = s}{\mathsf{K}trp_1 \dots p_n = s}\,\mathsf{K}_l \qquad \frac{t = sp_1 \dots p_n}{t = \mathsf{K}srp_1 \dots p_n}\,\mathsf{K}_r \qquad (n \geq 0) \ ,$$

$$\frac{tq(rq)p_1 \dots p_n = s}{\mathsf{S}trqp_1 \dots p_n = s}\,\mathsf{S}_l \qquad \frac{t = sq(rq)p_1 \dots p_n}{t = \mathsf{S}srqp_1 \dots p_n}\,\mathsf{S}_r \qquad (n \geq 0) \ ,$$

which in a sense exploit the natural left/right asymmetry of the combinatory axioms;
– *reflexivity* and *transitivity* are retained, while
– *symmetry* is dropped and *ap-congruence* is taken in its *parallel* formulation

$$\frac{t_1 = s_1 \qquad t_2 = s_2}{t_1t_2 = s_1s_2}\,{\scriptstyle App} \ .$$

The above system $\mathbf{G}[\mathcal{C}]$ turns out to be both equivalent to **CL** and *analytic*, since we can (constructively) prove that it admits the *elimination of transitivity.* Now the *consistency* of $\mathbf{G}[\mathcal{C}]$, and so also of **CL**, is an immediate consequence; moreover, all these results hold in general for arbitrary combinatory systems $\mathcal{X}$ and their associated synthetic and analytic calculi $\mathbf{C}[\mathcal{X}]$, resp. $\mathbf{G}[\mathcal{X}]$ — in certain cases even when an *extensionality rule* is added. Finally, owing to the analyticity of our calculi, the proof-theory of the respective equational theories is drastically simplified. By exploiting the natural analogy between *cut-elimination* in sequent-style logical calculi, and *transitivity elimination* in our analytic combinatory calculi, we are able to give new demonstrations, by a purely proof-theoretical analysis, of well known results concerning **CL** and weak reduction (e.g. the *Church-Rosser theorem* and the *leftmost reduction theorem*) and to generalize these results to combinatory systems as well. Also, some new (as far as we know) results concerning the decidability of certain fragments of **CL**+ *extensionality* can be obtained in the same style.

In the present paper, we shall give a survey of the results so far discussed (unless otherwise specified, detailed proofs are to be found in [10]), and we shall hint as well at some new (still partial) results concerning the proof-theory of $\boldsymbol{\lambda}$-calculus.

2 Combinatory Systems and Analytic Combinatory Calculi

Given a non-empty, possibly countably infinite set $\mathbb{X} = \{\mathsf{F}, \mathsf{G}, \mathsf{H}, \dots\}$ of individual constants (*combinators*), we denote by $\mathcal{T}_{\mathbb{X}}$ the set of all terms that are inductively generated from variables in $V = \{v_1, v_2, v_3, \dots\}$ and combinators in $\mathbb{X}$ by means of the binary function symbol *ap* for *application.* We let $x, y, z, \dots$, resp. $t, s, r, \dots$ (occasionally $\Phi, \Psi, \dots$) vary over V, resp. $\mathcal{T}_{\mathbb{X}}$. The standard conventions for writing application terms are adopted throughout: ts is an abbreviation of $ap(t, s)$, and missing parentheses are to be restored by associating to the *left.* Further notational conventions include:

- $V(t) :=$ the set of all variables occurring in t;
- $\mathcal{T}_{\mathbb{X}}^n :=$ the set of all terms t such that $V(t) \subseteq \{v_1, \dots, v_n\}$ (the set of all closed terms, if $n = 0$);
- $t[x_1/s_1, \dots x_n/s_n] :=$ the term resulting from t by simultaneous substitution of s_i for x_i $(1 \leq i \leq n)$;
- for $\Phi \in \mathcal{T}_{\mathbb{X}}^n$, $\Phi[s_1, \dots, s_n] := \Phi[v_1/s_1, \dots v_n/s_n]$.

The symbol $\equiv$ denotes syntactic equality between terms, and $\|t\|$ denotes the *length* of t, i.e. the number of occurrences of individual variables and individual constants in t.

A *combinatory system* $\mathcal{X}$ over $\mathbb{X}$ is defined as a **CL**-like system built over the set $\mathbb{X}$ of primitive combinators. Formally, $\mathcal{X}$ is a map associating to each combinator $\mathsf{F} \in \mathbb{X}$ a pair $\langle k_\mathsf{F}, \Phi_\mathsf{F} \rangle$, where $k_\mathsf{F} \geq 0$ (the *index* of F under $\mathcal{X}$) and $\Phi_\mathsf{F} \in \mathcal{T}_{\mathbb{X}}^{k_\mathsf{F}}$ (the *definition* of F under $\mathcal{X}$). Intuitively, the pair $\mathcal{X}(\mathsf{F})$ describes the intended *reduction* for the combinator F in the system $\mathcal{X}$:

$$\mathsf{F} t_1 \dots t_{k_\mathsf{F}} \rightarrow \Phi_\mathsf{F}[t_1, \dots, t_{k_\mathsf{F}}] \ .$$

Accordingly, **CL** may be represented as the combinatory system $\mathcal{C}$ over $\mathcal{L}_{\{\mathsf{K},\mathsf{S}\}}$ such that:

$$\mathcal{C}(\mathsf{K}) = \langle 2, v_1 \rangle \ \text{ and } \ \mathcal{C}(\mathsf{S}) = \langle 3, v_1 v_3 (v_2 v_3) \rangle \ .$$

A combinatory system $\mathcal{X}$ over $\mathbb{X}$ is said to be *linear*, resp. *pure*, iff in the definition Φ_{F} of an arbitrary $\mathsf{F} \in \mathbb{X}$ each variable has *at most one* occurrence, resp. no combinator does occur. It is said to be *recursive* iff, being $\mathbb{X} = \{\mathsf{F}_0, \mathsf{F}_1, \ldots\}$, the map $n \mapsto \langle k_{\mathsf{F}}, \Phi_{\mathsf{F}} \rangle$ is recursive. Note that in combinatory systems which are *both* pure and linear each combinator F satisfies, for every $t_1, \ldots, t_{k_{\mathsf{F}}}$: $\|\mathsf{F}t_1 \ldots t_{k_{\mathsf{F}}}\| > \|\Phi_{\mathsf{F}}[t_1, \ldots, t_{k_{\mathsf{F}}}]\|$.

We are going to associate to each combinatory system $\mathcal{X}$ a standard *synthetic* calculus $\mathbf{C}[\mathcal{X}]$ and an *analytic* calculus $\mathbf{G}[\mathcal{X}]$. To this aim let, for $\mathsf{F} \in \mathbb{X}$, $k = k_{\mathsf{F}}$ and $m \geq k$:

- $[\mathrm{Ax} - \mathsf{F}]_{\mathcal{X}}$ be the equation schema:

$$\mathsf{F}t_1 \ldots t_k = \Phi_{\mathsf{F}}[t_1, \ldots, t_k] \ ;$$

- $[\mathsf{F}_r]_{\mathcal{X}}$ and $[\mathsf{F}_l]_{\mathcal{X}}$ be the two inference rules (*right* and *left* F-*introducton*):

$$\frac{s = \Phi_{\mathsf{F}}[t_1, \ldots, t_k]t_{k+1} \ldots t_m}{s = \mathsf{F}t_1 \ldots t_m}\,[\mathsf{F}_r]_{\mathcal{X}} \qquad \frac{\Phi_{\mathsf{F}}[t_1 \ldots t_k]t_{k+1} \ldots t_m = s}{\mathsf{F}t_1 \ldots t_m = s}\,[\mathsf{F}_l]_{\mathcal{X}} \ .$$

The terms $t_{k+1}, \ldots, t_m$ are said to form the *context* of the introduction rule (which is empty in case $k = m$).

Next, let $[\varrho], [\sigma], [\tau]$ and $[App]$ be the inference rules of *reflexivity*, *symmetry*, *transitivity* and *ap-congruence* of equality (reflexivity being conveniently treated as a 0-premises rule):

$$\frac{}{t = t}\,[\varrho] \qquad \frac{t = s}{s = t}\,[\sigma] \qquad \frac{t = s \qquad s = r}{t = r}\,[\tau] \qquad \frac{t_1 = s_1 \qquad t_2 = s_2}{t_1t_2 = s_1s_2}\,[App] \ .$$

Now, the synthetic proof-system $\mathbf{C}[\mathcal{X}]$ is defined as the equational calculus determined by the axiom schemas $[\mathrm{Ax} - \mathsf{F}]_{\mathcal{X}}$ (for each $\mathsf{F} \in \mathbb{X}$) and the inference rules $[\varrho], [\sigma], [\tau]$ and $[App]$; and the analytic proof-system $\mathbf{G}[\mathcal{X}]$ as the equational calculus determined by the *"combinatory"* introduction rules $[\mathsf{F}_r]_{\mathcal{X}}$ and $[\mathsf{F}_l]_{\mathcal{X}}$ (for each $\mathsf{F} \in \mathbb{X}$), plus the *"structural"* inference rules $[\varrho]$, *restricted to atomic terms*, $[\tau]$ and $[App]$.

Furthermore, by $\mathbf{C_{ext}}[\mathcal{X}]$ and $\mathbf{G_{ext}}[\mathcal{X}]$ we shall denote the extension of $\mathbf{C}[\mathcal{X}]$, resp. $\mathbf{G}[\mathcal{X}]$, by means of the *extensionality* rule:

$$\frac{tx = sx}{t = s}\,[Ext] \qquad (x \notin V(ts)) \ .$$

Note that the *symmetry* rule *is not* a primitive rule of analytic calculi, yet the latter are readily seen to be closed under $[\sigma]$. As a consequence, it can be easily verified that the $\mathbf{C}$- and $\mathbf{G}$-calculi are equivalent.

Proposition 1. *For every combinatory system $\mathcal{X}$ and every equation E,*

$$\vdash_{\mathbf{C_{(ext)}}[\mathcal{X}]} E \quad \textit{iff} \quad \vdash_{\mathbf{G_{(ext)}}[\mathcal{X}]} E \ .$$

As a concluding remark, we call the reader's attention to the fact that, as far as equational deductive power is concerned, *"almost every" calculus* $\mathbf{C}[\mathcal{X}]$ *can be embedded in* **CL**. More precisely, given an arbitrary *recursive* combinatory system $\mathcal{X}$ over $\mathbb{X} = \{\mathsf{F}_0, \mathsf{F}_1, \ldots\}$ we can prove (by applying the *multiple fixed point theorem* for **CL** in the simplest case in which $\mathbb{X}$ is finite, and Klop's *infinite fixed point theorem*, see e.g. [1], p. 184, in the general case) that there exists a set $\mathbb{X}' = \{\mathsf{F}'_0, \mathsf{F}'_1 \ldots\}$ of *closed* $\mathcal{T}_{\{\mathsf{K},\mathsf{S}\}}$*-terms* such that, for every $n \geq 0$ and every list $t_1, \ldots, t_{k_{\mathsf{F}_n}}$ of $\mathcal{T}_{\{\mathsf{K},\mathsf{S}\}}$*-terms:*

$$\vdash_{\mathbf{CL}} \mathsf{F}'_n t_1 \ldots t_{k_{\mathsf{F}_n}} = \Phi'_n[t_1, \ldots, t_{k_{\mathsf{F}_n}}] \ ,$$

where Φ'_n is obtained from Φ_n by replacing every F_i with the corresponding F'_i. By contrast, it is evident that $\mathcal{X}$*-reduction* (to be defined in the next section) cannot, in general, be faithfully simulated by means of standard **CL** *w(eak)-reduction.*

3 Elimination of Transitivity, with Applications

In the first part of this section, we will concentrate on **G**-calculi *without extensionality.* The central result, actually that which justifies the attribute 'analytic' we have used so far, is:

Main Theorem 1 (Elimination). *Let $\mathcal{X}$ be a (recursive) combinatory system: every derivation $\mathcal{D}$ in* $\mathbf{G}[\mathcal{X}]$ *can (effectively) be transformed into a* τ-free *derivation $\mathcal{D}^*$ having the same end–equation as $\mathcal{D}$.*

This theorem follows as an immediate consequence of a syntactic *Reduction* lemma, saying that any given derivation $\mathcal{D}$ containing a single and final application of the transitivity rule $[\tau]$ can be transformed into a τ-free derivation of the same end-equation as $\mathcal{D}$. Its proof, in turn, runs by ω^3-induction, specifically by triple induction on the following numerical parameters (in the order) associated to each derivation $\mathcal{D}$: *height* (maximum number of applications of combinatory inferences along a path of $\mathcal{D}$), *size* (overall number of applications of combinatory inferences in $\mathcal{D}$), and *rank* (sum of the lengths of all terms r occurring as a *cut-term* in some application of $[\tau]$ in $\mathcal{D}$).

Owing to the Elimination theorem, we now know that $\mathbf{G}[\mathcal{X}]$ and $\mathbf{G}^-[\mathcal{X}]$ ($\mathbf{G}[\mathcal{X}]$ minus $[\tau]$) are equivalent. On the other side, given two distinct variables x and y it is evident that $\mathbf{G}^-[\mathcal{X}] \nvdash x = y$, the 'naive' argument alluded to in Sect. 1 being well available once $[\tau]$ has been removed; see also the more general property (R.4) below. We so have, together with Proposition 1, syntactical *consistency* as a bonus:

Proposition 2. *For all $\mathcal{X}$, the calculi* $\mathbf{G}^-[\mathcal{X}]$, $\mathbf{G}[\mathcal{X}]$, $\mathbf{C}[\mathcal{X}]$ *are equivalent and consistent.*

A second, easy consequence of the elimination theorem concerns the existence of a terminating proof-searching strategy in $\mathbf{G}^-[\mathcal{X}]$, whenever the combinatory system $\mathcal{X}$ is *pure, linear and recursive* (like for instance BCI and BCK): on the one side we do not have the transitivity rule, and on the other side (due to pureness and linearity) the complexity of a conclusion $t = s$ (i.e. $\|ts\|$) of a combinatory rule is greater than that of its premise. Hence a direct, purely proof-theoretical demonstration of:

Proposition 3. *If $\mathcal{X}$ is pure, linear and recursive, then $\mathbf{C}[\mathcal{X}]$ is decidable.*

In view of further applications let us consider, given an arbitrary combinatory system $\mathcal{X}$, the (straightforward) relativizations to $\mathcal{X}$ of the standard notions of *redex* and *contractum*, *one step reduction*, *(weak) reduction*, *(weak) normal form* etc. from **CL**. E.g., we call a term t a *$\mathcal{X}$-redex*, and t' its *$\mathcal{X}$-contractum*, iff $t \equiv \mathsf{F}s_1 \dots s_{k_\mathsf{F}}$ and $t' \equiv \Phi_\mathsf{F}[s_1, \dots, s_{k_\mathsf{F}}]$ for some $\mathsf{F} \in \mathbb{X}$. The induced relations of *one-step $\mathcal{X}$-reduction* and *$\mathcal{X}$-reduction* (the reflexive and transitive closure of the former) are denoted by $\rightarrow_\mathcal{X}$, resp. $\twoheadrightarrow_\mathcal{X}$, and the set of $\mathcal{T}_\mathbb{X}$-terms in *$\mathcal{X}$-normal form* by $\mathrm{NF}_\mathcal{X}$ (subsequently, the subscript $\mathcal{X}$ will be dropped throughout whenever possible).

$\mathcal{X}$-reducibility and $\mathbf{G}[\mathcal{X}]$-derivability are easily seen to be related as expected:

(R.1) $\quad t \twoheadrightarrow_\mathcal{X} s \quad \Rightarrow \quad \vdash_{\mathbf{G}[\mathcal{X}]} t = s$.

Moreover, by a straightforward induction on the length of τ-free derivations, we can also prove:

(R.2) $\quad \vdash_{\mathbf{G}^-[\mathcal{X}]} t = s \ \Rightarrow \ \exists r(t \twoheadrightarrow r \twoheadleftarrow s)$;

(R.3) *if $\mathcal{D} \vdash_{\mathbf{G}^-[\mathcal{X}]} t = s$ and the derivation $\mathcal{D}$ contains* at least one *occurrence of a left (right) combinatory introduction rule, then the term t (resp.: s) contains* at least one *occurrence of a $\mathcal{X}$* -redex.

The latter (a kind of *subterm property* of τ-free derivations) immediately yields, in turn:

(R.4) *if $\mathcal{D} \vdash_{\mathbf{G}^-[\mathcal{X}]} t = s$ and $t, s \in \mathrm{NF}_\mathcal{X}$, then $t \equiv s$* .

Now, simply by combining (R.1), (R.2) and the elimination theorem, we get:

Theorem 2 (CR($\mathcal{X}$)). *For every combinatory system $\mathcal{X}$, the relation $\twoheadrightarrow_\mathcal{X}$ is Church-Rosser.*

As usual, CR($\mathcal{X}$) implies *uniqueness of $\mathcal{X}$-normal forms.* Note however that in the present context an alternative proof of that property, which does not pass through CR, immediately follows from (R.1), (R.4) and Theorem 1. Also, observe that if $\mathcal{X}$ is *pure and linear* we trivially have (i) *strong normalization* for $\mathcal{X}$ (for every term t, the $\mathcal{X}$-reduction graph of t is finite) and so, directly and without using CR, (ii) every $\mathcal{L}_\mathbb{X}$ term t has *exactly one* $\mathcal{X}$-normal form.

Another interesting consequence of τ-elimination and the subterm property (R.3) consists in a new, proof-theoretical demonstration of the *Leftmost reduction theorem* for **CL** (usually proved as a corollary to the stronger *Standardization theorem*, see e.g. [1] and [8]), here generalized to arbitrary combinatory systems. The theorem says that the leftmost reduction strategy ($\overset{l}{\twoheadrightarrow}_\mathcal{X}$), consisting in reducing step by step ($\overset{l}{\rightarrow}_\mathcal{X}$) always the *leftmost* redex occurrence, is normalizing.

Theorem 3 (Leftmost reduction). *For every $\mathcal{X}$ and every term t, if t has the $\mathcal{X}$-normal form s, then $t \overset{l}{\twoheadrightarrow}_{\mathcal{X}} s$.*

The intuitive idea underlying our proof is simple (although the technical details require some non trivial work, cf. [10]): given a term t having the normal form s, property (R.1) and Theorem 1 provide us with a τ-free $\mathbf{G}[\mathcal{X}]$-derivation $\mathcal{D}$ of $t = s$ which, by (R.3), doesn't contain occurrences of *right* introduction rules, so allowing the *extraction* of a leftmost reduction path going from t to s. Specifically, this leftmost path corresponds to the succession of occurrences of combinatory inferences which are encountered by "visiting" each node in $\mathcal{D}$'s tree starting from the bottommost node and following an upward-leftward direction, with backtracking to the the leftmost not yet visited node when a terminal node is reached.

Once the *extensionality* rule is added to the **G**-calculi, the problem concerning the eliminability of transitivity becomes much more complicated. Indeed, in this case — by a suitable adaptation of the methods employed in the proof of Theorem 1 — we are presently able to prove τ-elimination *only* for a restricted class of combinatory systems, *not including* **CL** (cf. [10]).

Theorem 4 (Restricted elimination). *Let $\mathcal{X}$ be a* linear *combinatory system: every derivation $\mathcal{D}$ in $\mathbf{G_{ext}}[\mathcal{X}]$ can (effectively) be transformed into a τ-*free *derivation $\mathcal{D}^*$ having the same end–equation as $\mathcal{D}$.*

It is an *open problem* whether the above result can be extended to *every* combinatory system $\mathcal{X}$, and in particular to the combinatory system $\mathcal{C}$. As we showed in [10], a syntactical proof of eliminability of transitivity for $\mathbf{G_{ext}}[\mathcal{C}]$, the analytic version of **CL** + *ext*, would have a relevant consequence consisting in a *direct* proof of the Church-Rosser property for combinatory *strong* reduction $\twoheadrightarrow_s$. We recall that $\twoheadrightarrow_s$ is defined by the same rules as *weak* reduction $\twoheadrightarrow_w$ (i.e. $\twoheadrightarrow_{\mathcal{C}}$), with the addition of

$$\frac{t \twoheadrightarrow_s s}{\lambda^* x.t \twoheadrightarrow_s \lambda^* x.s} \ ,$$

where, for a **CL**-term r and a variable x, the associated *abstraction* **CL**-term $\lambda^* x.r$ is defined as usual, see e.g. [8], p. 25. Indeed, $\twoheadrightarrow_s$ *is* Church-Rosser but, as far as we know, all the available proofs of CR($\twoheadrightarrow_s$) are *indirect*, hinging on the confluence of $\boldsymbol{\lambda\beta\eta}$-reduction and the equivalence between **CL**+*ext* and $\boldsymbol{\lambda\beta\eta}$ calculus (cf. [1], p. 156; [3], Sect. 11E; [7], Chapt. 7; [8], Sect. 8B; see also [6], [9]).

As a first step towards a positive solution of the conjectured eliminability of $[\tau]$ in $\mathbf{G_{ext}}[\mathcal{C}]$, we have recently proved that a *necessary* condition for the equivalence of $\mathbf{G_{ext}}[\mathcal{C}]$ and $\mathbf{G^-_{ext}}[\mathcal{C}]$ (cf. Open Problem 6.3 of [10]), saying that

$$(*) \ \ \mathbf{G^-_{ext}}[\mathcal{C}] \textit{ is equivalent to } \mathbf{G}^-[\mathcal{C}] + \frac{t = s}{\lambda^* x.t = \lambda^* x.s}\,\xi^* \ ,$$

does hold. Actually, $\mathbf{G}^-[\mathcal{C}] + [\xi^*]$ is easily seen to be closed under [*Ext*], while we can now prove the converse direction of (*), i.e. the closure of $\mathbf{G^-_{ext}}[\mathcal{C}]$ under

$[\xi^*]$, by means of a key lemma stating that each $\mathbf{G}^-_{\mathbf{ext}}[\mathcal{C}]$-derivation $\mathcal{D}$ of an equation $p[x/t[y/r]] = q$ (with the variable x occurring exactly once in p) can be transformed into a $\mathbf{G}^-_{\mathbf{ext}}[\mathcal{C}]$-derivation $\mathcal{D}^*$ of $p[x/(\lambda^* y.t)r] = q$.

We conclude this section by mentioning an interesting application of the *restricted* elimination theorem, which enables us to show, by purely proof-theoretical analysis, that certain extensional subsystems of **CL**+ *ext* (for instance BCK+ *ext*) are decidable.

Theorem 5. *For every pure, linear and recursive combinatory system* $\mathcal{X}$, $\mathbf{C}_{\mathbf{ext}}[\mathcal{X}]$ *is decidable.*

The key point consists in the proof of a *Boundedness* lemma saying, for combinatory systems $\mathcal{X}$ satisfying the above assumptions, that an arbitrary equation E is τ-free derivable in $\mathbf{G}_{\mathbf{ext}}[\mathcal{X}]$ iff there exists a τ-free $\mathbf{G}_{\mathbf{ext}}[\mathcal{X}]$-derivation $\mathcal{D}$ of E whose *length* is less than a suitable numerical bound depending uniformly and effectively on E. This clearly provides a *terminating* backward proof-searching strategy for $\mathbf{G}^-_{\mathbf{ext}}[\mathcal{X}]$ which in turn yields, because of the equivalence of $\mathbf{G}_{\mathbf{ext}}[\mathcal{X}]$ and $\mathbf{G}^-_{\mathbf{ext}}[\mathcal{X}]$ given by the restricted elimination theorem, the decidability of the two equivalent calculi $\mathbf{G}_{\mathbf{ext}}[\mathcal{X}]$ and $\mathbf{C}_{\mathbf{ext}}[\mathcal{X}]$.

4 An Analytic Proof-System for λ-Calculus

The feasibility of applying proof-theoretical methods also to the equational theory of **λ**-calculus was considered in [10]. In particular, in strict analogy with the peculiar formulation of analytic combinatory calculi, we introduced the calculus $\mathbf{G}[\beta]$ as a possible candidate to represent the *analytic* counterpart to the standard, synthetic presentation of the above theory. $\mathbf{G}[\beta]$ is determined by the following inference rules:

(i) *Left* and *right introduction rules* (*β-rules):*

$$\frac{t[x/r]p_1 \dots p_n = s}{(\lambda x.t)rp_1 \dots p_n = s}\,[\beta_l] \qquad\qquad \frac{t = s[x/r]p_1 \dots p_n}{t = (\lambda x.s)rp_1 \dots p_n}\,[\beta_r] \qquad (n \geq 0)\ ;$$

(ii) *Structural rules:* $[\varrho]$ (restricted to variables), $[App]$, $[\tau]$ and

$$\frac{t = s}{\lambda x.t = \lambda x.s}\,[\xi]\ .$$

Note that α-congruent λ-terms (λ-terms, as usual, are generated starting from individual variables by means of *application* and λ *-abstraction*) are here identified, and that we adopt the standard "variable convention" in writing terms and in doing substitutions (see e.g. [1], p. 26).

The calculus $\mathbf{G}[\beta]$ is clearly equivalent to the standard synthetic equational calculus $\mathbf{C}[\beta]$ determined by $[\varrho], [App], [\sigma], [\tau], [\xi]$ and the β-conversion schema $(\lambda x.t)s = t[x/s]$. The problem concerning the eliminability of transitivity in $\mathbf{G}[\beta]$ was left open in [10]. Indeed, it turns out that in trying to adapt to $\mathbf{G}[\beta]$ the proof of Theorem 1 one encounters similar difficulties as in trying to prove

Theorem 4 for arbitrary combinatory systems $\mathcal{X}$: *(weak) extensionality* (here the rule $[\xi]$), when combined with a form of *non-linearity* (here the obvious fact that in an abstraction term $\lambda x.t$ the variable x may have more than one occurrence), seems to resist inductive proof-strategies based on natural complexity measures of derivations.

However, we can now state:

Theorem 6. $\mathbf{G}[\beta]$ *admits τ-elimination.*

This result, although encouraging, is not completely satisfactory. Indeed, the only demonstration of it that we presently have is not entirely proof-theoretical, in so far as it makes essential use of the Church-Rosser theorem for $\lambda\beta$-reduction $\twoheadrightarrow_\beta$.

Here is a sketch of the proof. Let us say that a $\mathbf{G}[\beta]$-derivation $\mathcal{D}$ of $t = s$ is a *left*-derivation, in symbols $\mathcal{D} \vdash_L t = s$ (a *right*-derivation, $\mathcal{D} \vdash_R t = s$) whenever there are no applications of the rule $[\beta_r]$ (of the rule $[\beta_l]$) in $\mathcal{D}$. We also write $\mathcal{D} \vdash^- t = s$ to mean that $\mathcal{D}$ is a τ-free derivation of $t = s$.
Thanks to the left-right symmetry of the calculus, we trivially have:

$$(\beta.1) \qquad \vdash_L^{(-)} t = s \ \Leftrightarrow \vdash_R^{(-)} s = t \ .$$

Also, it is easily verified that:

$$(\beta.2) \qquad t \twoheadrightarrow_\beta s \ \Rightarrow \vdash_L t = s \ .$$

Next, by induction on the length of derivations, we can prove the *β-inversions*:

$$(\beta.3) \qquad \mathcal{D} \vdash_L^- t = (\lambda x.s)rp_1 \dots p_n \ \Rightarrow \ \exists \mathcal{D}^*(\ \mathcal{D}^* \vdash_L^- t = s[x/r]p_1 \dots p_n \ ) \ ,$$

$$\mathcal{D} \vdash_R^- (\lambda x.s)rp_1 \dots p_n = t \ \Rightarrow \ \exists \mathcal{D}^*(\ \mathcal{D}^* \vdash_R^- s[x/r]p_1 \dots p_n = t \ ) \ .$$

Let now $\mathtt{s}(\mathcal{D})$ denote the overall number of applications of β-rules in $\mathcal{D}$. For any given derivations $\mathcal{D}_1 \vdash_L^- t = s$ and $\mathcal{D}_2 \vdash_L^- s = r$ we are able to construct a derivation $\mathcal{D}^* \vdash_L^- t = r$, arguing by induction on $\omega^2 \cdot \mathtt{s}(\mathcal{D}_2) + \omega \cdot \mathtt{s}(\mathcal{D}_1) + \|s\|$ and using $(\beta.3)$. Hence:

$$(\beta.4) \qquad \mathcal{D} \vdash_L t = s \ \Rightarrow \ \exists \mathcal{D}^*(\ \mathcal{D}^* \vdash_L^- t = s \ ) \quad (\textit{left } \tau\textit{-elimination}).$$

Of course, the symmetrical *right τ-elimination* follows by $(\beta.1)$.
A last lemma is needed:

$$(\beta.5) \qquad \mathcal{D}_1 \vdash_L^- t = s \ \textit{ and } \ \mathcal{D}_2 \vdash_R^- s = r \ \Rightarrow \ \exists \mathcal{D}^*(\ \mathcal{D}^* \vdash^- t = r) \ ,$$

which is easily proved by induction on $\omega \cdot (\mathtt{s}(\mathcal{D}_1) + \mathtt{s}(\mathcal{D}_2)) + \|s\|$.

To finally prove Theorem 6, assume that $\mathbf{G}[\beta] \vdash t = s$. Then $\mathbf{C}[\beta] \vdash t = s$ (that is: t and s are *β-convertible*) and, *by the Church-Rosser theorem for* $\twoheadrightarrow_\beta$, we have that $t \twoheadrightarrow_\beta r$ and $s \twoheadrightarrow_\beta r$ for some term r. So, by $(\beta.2)$, $(\beta.4)$ and $(\beta.1)$:

$$\vdash_L^- t = r \ \textit{ and } \ \vdash_R^- r = s \ ,$$

whence $\mathbf{G}[\beta] \vdash^- t = s$ follows by $(\beta.5)$.

We think it would be interesting to find an alternative, more direct proof of the τ-elimination theorem for $\mathbf{G}[\beta]$, as well as to exploit this result towards nice applications, as we did for combinatory systems. Another open problem, still to be tackled, concerns the eliminability of the transitivity rule in the calculus $\mathbf{G}[\beta]+ext$ corresponding to the equational theory of $\boldsymbol{\lambda\beta\eta}$-reduction.

References

1. Barendregt, H.P.: The Lambda Calculus, its Syntax and Semantics. Revised edition, North Holland, Amsterdam 1984
2. Curry, H.B., Feys, R.: Combinatory Logic. Vol. **I**, North Holland, Amsterdam 1958
3. Curry, H.B., Hindley, J.R., Seldin, J.P.: Combinatory Logic. Vol. **II**, North Holland, Amsterdam 1972
4. Engeler, E.: Foundations of Mathematics: Questions of Analysis, Geometry & Algorithmics. Engl. Transl., Springer, Berlin 1983
5. Girard, J-Y., Taylor, P., Lafont, Y.: Proofs and Types. Revised edition, Cambridge University Press, Cambridge 1990
6. Hindley, J.R.: Axioms for strong reduction in combinatory logic. The Journal of Symbolic Logic **32** (1967) 224–236
7. Hindley, J.R., Lercher, B., Seldin, J.P.: Introduction to Combinatory Logic. Cambridge University Press, London 1972
8. Hindley, J.R., Seldin, J.P.: Introduction to Combinators and λ–Calculus. Cambridge University Press, London 1986
9. Lercher, B.: The decidability of Hindley's axioms for strong reduction. The Journal of Symbolic Logic **32** (1967) 237–239
10. Minari, P.: Analytic combinatory calculi and the elimination of transitivity. Archive for Mathematical Logic **43** (2004) 159–191

Potential Infinity and the Church Thesis

Marcin Mostowski

Department of Logic
Institute of Philosophy, Warsaw University
00–047 Warszawa, Krakowskie Przedmieście 3, Poland
m.mostowski@uw.edu.pl

Abstract. In this paper we consider a “mathematical” proof of the Church Thesis. The proof is based on very weak assumptions about *intuitive computability* and *the* FM*–representability theorem* from [1]. It develops and improves the argument from [2]. Our argument essentially depends on the mathematical model of the world we are in.

1 Introduction

We discuss here a justification of the Church Thesis, called also the Church–Turing Thesis.[1,2]

The thesis can be formulated as follows.

Thesis 1 (The Church Thesis, the first formulation, [3] and [4]). [3] *Our intuitive notion of computability is equivalent to the notion of computability defined by means of Turing machines.*

The main difficulty in justifying this thesis is proving that Turing machines can compute every intuitively computable notion. Therefore we concentrate on this part.

We use various forms of words “computable” and “recursive” in the following sense:

To compute means physical or abstract activity of carrying out intuitive algorithms, *computable* means the same as *intuitively computable*, that is something which can be computed;

Recursive means something which can be represented by means of any equivalent mathematical model of computations, such as: Turing machines, programs for register machines, or μ–recursive functions.

According to this terminology, the Church Thesis is the general claim that exactly those functions (sets, relations) are computable which are recursive.

[1] This paper owes a lot to Cezary Cieśliński, Leszek Kołodziejczyk, and Konrad Zdanowski. The majority of the ideas of this paper were formulated in discussions with them.

[2] A sketchy presentation of the argument has been given in [2].

[3] We use terms “THESIS” and “CLAIM” for statements with some non–mathematical content. They are numbered independently of purely mathematical statements.

1.1 The scheme of our justification

The general scheme of our argument is the following:

1. We start with an ontological part by giving a mathematical model of the world without actual infinity – finite but potentially infinite. Our argument is essentially based on the assumption that this is a good mathematical model of the world in which we live.
2. We characterize the class of concepts which can be meaningfully described in such a world – FM–representable relations. This gives our first upper bound for computable notions.
3. Then we consider an epistemological criterion separating effectively computable relations from all other FM–representable relations. The criterion says that, for a given query of the form "is a given tuple $(a_1, \ldots, a_n)$ in a relation R?", the answer should be known to us at some stage.
4. We assume relatively weak mathematical explication for *to know the answer*, by observing that our knowledge has to be expressible in our language and we should be able to express our knowledge when knowing the answer. We obtain on this basis the conclusion that all intuitively computable notions are computable in the standard mathematical sense.

1.2 The assumptions of our justification

Our justification is essentially based on three assumptions.

1. We assume that *a finite but potentially infinite* world is a good mathematical model of our reality. This assumption is discusses in sections 3 and 4.
2. We assume that if we know something then we can express this knowledge in our language. This assumption seems to be rather obvious.
3. We assume that our language has the following property: the relation between finite models and sentences of this language is recursive. As a matter of fact, a stronger version seems to be trivially true. Namely, each sentence of our language is equivalent to some sentence of finite order. This assumption is needed for applicability of theorem 3 in our justification. Let us emphasize that we do not need the assumption of recursivity of syntax, but only of the semantics in finite models.

It seems that only the first of the three assumptions is essential – in the sense that it can be considered as doubtful. Therefore we would say that the proof is given relatively to the first assumption.

2 Computability and recursivity

In 1900, many years before we had at our disposal any serious mathematical model of computation, Hilbert formulated his tenth problem "*Give a method which, using finitely many operations, will decide whether a given equation can* *be solved in integers.*"[4]. Obviously, the problem would be meaningless if we did not have any intuitive notion of computability. It is interesting that Hilbert asks for a positive solution, probably having no idea how a negative one would look. Even now, we do not know any method of giving negative answers to such questions which does not involve some variant of the Church Thesis. It seems obvious that Hilbert and other mathematicians in 1900 had an intuitive notion of decidability. Otherwise Hilbert's formulation of the tenth problem would be meaningless.

Let us observe that no essential doubt was raised against the effectivity of decision methods computable by means of e.g. Turing machines.[5] The analysis of computability given by Turing in [4] very convincingly supports the claim that every computable notion is recursive.[6] Therefore, as a rule, the following is generally accepted.

Thesis 2 (Soundness Thesis). *All recursively decidable sets and relations are intuitively decidable.*

Let us observe that thesis 2 gives a partial answer to the problem of an upper bound for intuitive computability. In what follows we assume the soundness thesis usually without explicit reference to it.

3 Finite but potentially infinite world

The notion of potential infinity was formulated by Aristotle more than 2300 years ago in *Physics* [9] and it is still understood in a similar way. Objects, which are not actually infinite can nevertheless be potentially infinite in the sense that they can always be enlarged.

Let us put things in a slightly more concrete way. In our practice of computing technology we never use an actually infinite set of natural numbers. In each concrete implementation we use only finitely many of them. Nevertheless always – if it is needed – we can enlarge this set, e.g. by choosing a new representation using more bits. Using Aristotelian terminology, we say that the set of natural numbers used in computing technology is only potentially infinite.

Looking at our world we can also consider it as finite but potentially infinite. Let us observe that modern physics supplies contradictory arguments here, on the one hand, estimating the number of elementary particles, and on the other hand using continuous values for describing physical relationships. It seems that

[4] "*Man soll ein Verfahren angeben, nach welchem sich mittels einer endlichen Anzahl von Operationen entscheiden läßt, ob die Gleichung in ganzen rationalen Zahlen lösbar ist.*", lecture delivered by D. Hilbert at Mathematical Congress in Paris 1900, quotation after [5].

[5] Sometimes people misleadingly identify *computability* with *practical computability.* The last notion is beyond our interest in this paper. A short discussion of practical computability from the point of view of our epistemic abilities can be found in [6].

[6] A lot of historical comments about the Church Thesis can be found in [7] and [8]. In these works also an extensive bibliography can be found.

it can be explained similarly as successful applications of analytical methods in combinatorics.

In what follows we assume that our world is finite but potentially infinite. To simplify our arguments we also assume that the only entities available are natural numbers. If the infinite world can be imagined as the domain of natural numbers $\mathcal{N} = (\omega, +, \times, s, 0)$ then a finite but potentially infinite world can be imagined as *the* FM–*domain* $\mathrm{FM}(\mathcal{N}) = \{\mathcal{N}_k : k = 1, 2, 3, \ldots\}$, where $\mathcal{N}_k$ is the restriction of $\mathcal{N}$ to the set $\{0, 1, \ldots k-1\}$. In this case the successor function is treated as a binary relation.

Let us stress here that our way of rejecting actual infinity is very distant from so called strict finitism. It is much closer to the approach presented by Jan Mycielski in [10].

4 Other worlds

We give our justification of thesis 1 relatively to the mathematical model of our world discussed in the previous section. So, if we reject soundness of this model then the justification is invalid. In this section we consider shortly some different hipotheses which would refute the Church Thesis.

Firstly, let us assume that our world consists of Euclidean space–time with arbitrary small material objects. Time is infinitely divisible similarly as space. Additionally, there is no limit for speed. It seems that the picture can be loosely considered as describing the Newtonian world.

In such a world we can imagine the possibility of performing infinitely many steps in bounded time. For instance, the first step is carried out in 1 second, the second one in $\frac{1}{2}$ second, the next one in $\frac{1}{4}$ second, and so on. In general the next step is carried out two times faster than the previous one. In this way countably many steps can be performed in 2 seconds. Thus we can check any Π^0_1 property of natural numbers in bounded time.

Secondly, let us consider an atomistic model of our reallity. Now again time and space are infinitely divisible and the space contains a number of nondivisible objects called atoms, which geometrically can be precisely represented as points.

Having any four points a, b, c, d at any time t we can describe the real number $f(t, a, b, c, d)$ representing the ratio of the distance between a and b to the distance between c and d at the moment t. So having any method of generating time points $t_0, t_1, t_2, \ldots$ we can define the function $g : \omega \longrightarrow \mathbb{R}$ as $g(n) = f(t_n, a, b, c, d)$. Now we can compute the function $h : \omega \longrightarrow \omega$ such that $h(n) = n$–th digit of $g(n)$. It can happen that h is not recursive.[7]

Let us observe that while the first example is incompatible with our current knowledge about the physical world, this is not so obvious in the case of the second one.

Both examples discussed in this section essentially assume actual infinity. We discuss them here only to emphasize the relative character of our justification

[7] This is a simplified version of an argument against the Church thesis, probably the most popular in the recent literature, see the discusion in [7].

of the Church thesis. In a sense the question whether our model of a finite but potentially infinite world is sound cannot be decided easily. Nevertheless, it seems that it describes quite well the world of digital computers.

5 FM–representability

In this section we will discuss the notion of FM–representability. Originally it was motivated by an attempt to transfer the Tarskian method of truth definitions – as a tool for classifying finite order notions – to finite models.[8]

In [1] the following problem is considered:

Which infinite classes or relations of natural numbers can be meaningfully described in such finite but potentially infinite domains?

The explication of *being meaningfully described* proposed in that paper is given by the notion of FM–representability. Here and in what follows by a formula we mean a first order arithmetical formula and by a finite model we mean an element of the FM–domain $\mathrm{FM}(\mathcal{N})$.

Definition 1. *Let $R \subseteq \omega^n$. Then R is* FM*–represented by the formula $\varphi(x_1, \ldots, x_n)$ if and only if for each $a_1, \ldots, a_n \in \omega$, $\varphi(a_1, \ldots, a_n)$ is true in almost all finite models exactly when $R(a_1, \ldots, a_n)$ and $\varphi(a_1, \ldots, a_n)$ is false in almost all finite models exactly when $R(a_1, \ldots, a_n)$ does not hold.*

R is FM*–representable if it is* FM*–represented by some formula.*

Equivalently, this definition can be formulated as follows: $\varphi(x_1, \ldots, x_n)$ FM–*represents R if $\varphi(x_1, \ldots, x_n)$ correctly describes each finite piece of the characteristic function of R in almost all finite models.*

In [1] the class of FM–representable notions is characterized as follows:

Theorem 3 (FM–**representability Theorem).** *Let $R \subseteq \omega^n$. Then R is* FM–*representable if and only if R is of degree $\leq \mathbf{0}'$ (recursive with some recursively enumerable oracle), or equivalently Δ^0_2 in the arithmetical hierarchy.*

Let us observe that theorem 3 does not depend on the underlying logic. It holds for any logic extending elementary logic with a decidable truth relation. We say that a logic L has a decidable truth relation if and only if the relation "$M \models \varphi$" is decidable, for a finite model M and an L–sentence φ.

It was observed recently by Michał Krynicki and Konrad Zdanowski [13] that the arithmetic of multiplication restricted to finite models would also be sufficient as an underlying theory. This result has been improved in [14] by showing that the divisibility relation is sufficient here. Some variants of the notion of FM–representability are discussed in [15].

[8] The notion of *truth definitions in finite models* was formulated in [1], and then investigated in [2], [11], [12].

6 FM–representability and recursivity

In this section we give our explication for *to know* when a formula $\varphi(x_1, \ldots, x_n)$ correctly answers a question of the form "$R(a_1, \ldots, a_n)$?".

We say that the formula $\psi(x_1, \ldots, x_n)$ is a testing formula for $\varphi(x_1, \ldots, x_n)$ and $R \subseteq \omega^n$ if and only if the following conditions are satisfied:

- for each $a_1, \ldots, a_n \in \omega$ there is n_0 such that for each finite model M, $M \models \psi(a_1, \ldots, a_n)$ if and only if $card(M) \geq n_0$;
- for each $a_1, \ldots, a_n \in \omega$ and each finite model M, if $M \models \psi(a_1, \ldots, a_n)$ then $R(a_1, \ldots, a_n)$ if and only if $M \models \varphi(a_1, \ldots, a_n)$.

A testing formula for $\varphi(x_1, \ldots, x_n)$ and $R \subseteq \omega^n$ simply says for given natural numbers $a_1, \ldots, a_n$ whether we know the answer to the question "$(a_1, \ldots, a_n) \in R$?" or not. Then according to our explication we would *know* the answer for each question of the above form if we have a testing formula. We easily get the following:

Theorem 4. *Let $R \subseteq \omega^n$, then R is recursively decidable if and only if there are formulae $\varphi(x_1, \ldots, x_n)$, $\psi(x_1, \ldots, x_n)$ such that $\psi(x_1, \ldots, x_n)$ is a testing formula for $\varphi(x_1, \ldots, x_n)$ and R.*

7 The proof of the Church Thesis

We consider here the version of the Church Thesis for sets and relations, stating that:

Thesis 5 (The Church Thesis for sets and relations). *Our intuitive notion of decidability is equivalent to the notion of recursive decidability.*

In one direction this thesis follows from the soundness thesis.

Let us consider the following obvious property of intuitively decidable relations $R \subseteq \omega^n$:

Claim. If R is intuitively decidable then each query of the form "$R(a_1, \ldots, a_n)$?" can be solved in some sufficiently large finite world in such a way that we would know whether our solution is conclusive or not.

Because we have assumed the existence of a testing formula as an explication for *to know* then we can replace claim 7 by a more precise version.

Claim (More mathematical version of claim 7). If R is intuitively decidable then

- there is a formula $\varphi(x_1, \ldots, x_n)$ FM–representing R,
- there is a formula $\psi(x_1, \ldots, x_n)$ which is a testing formula for $\varphi(x_1, \ldots, x_n)$ and R.

Proof (of thesis 5). Let $R \subseteq \omega^n$. By claim 7 we see that R should be FM–represented by some formula $\varphi(x_1, \ldots, x_n)$. Moreover, we have a formula $\psi(x_1, \ldots, x_n)$, which a testing formula for $\varphi(x_1, \ldots, x_n)$ and R. Then by theorem 4 the relation R is recursively decidable.

Since the reverse implication follows from thesis 2, we have justified thesis 5.

8 Concluding remarks

In this paper we have given a "mathematical" proof of the Church Thesis. Of course such a proof needs some mathematically well–defined knowledge about intuitive computability. The explanatory value of the proof depends on philosophical plausibility of this knowledge.

A standard approach to justifying the Church Thesis is considering some general conditions which should be satisfied by possible computations, and then proving that these conditions imply the thesis. This is exactly the way chosen by Gandy in [16].

Our mathematical model is rather the model of reality and our knowledge about results of computations than just another mathematical concept of computations. It seems that our proof meets to the requirements for a proof of the Church thesis proposed by Gödel.[9] According to them the thesis should be justified on the basis of some properties of intuitive computability which would be generally accepted as its properties. However, the thesis would be false if we assumed other mathematical models of our reality. Then the properties of the notion of intuitive computability would depend on properties of our world.

Finally, let us consider another aspect of our justification. The FM–representability theorem holds only for logics with a recursive truth relation. Hence somebody could accuse the justification of containing vicious circle, because the recursivity in the conclusion follows from recursivity in one of the premises. However, the assumption is not so strong as its falsity would mean that there are possible finite computational devices which cannot be correctly described in our language. Nevertheless we should agree that the justification proposed is based not only on the assumption about the world, but also about possible languages used for describing it.

References

1. Mostowski, M.: On representing concepts in finite models. Mathematical Logic Quarterly **47** (2001) 513–523
2. Mostowski, M.: On representing semantics in finite models. In Rojszczak†, A., Cachro, J., Kurczewski, G., eds.: Philosophical Dimensions of Logic and Science, Kluwer Academic Publishers (2003) 15–28
3. Church, A.: An unsolvable problem of elementary theory. Am. J. Math. **58** (1936) 345–363
4. Turing, A.M.: On computable numbers, with an application to the Enscheidungsproblem. Proc. London Math. Soc. **42** (1936–7) 230–265
5. Alexandrov, P.S.: Die Hilbertischen Probleme. Akademische Verlagsgesellschaft Geest & Portig K.–G. (1983)
6. Mostowski, M., Wojtyniak, D.: Computational complexity of some natural language constructions. Annals of Pure and Applied Logic **127(2)** (2004) 219–227
7. Odifreddi, P.: Classical Recursion Theory. North–Holland Pub. Co. (1989)

[9] See [8].

8. Sieg, W.: Step by recursive step: Church's analysis of effective calculability. The Bulletin of Symbolic Logic **3** (1997) 154–180
9. Aristotle: Physics. The Internet Classics Archive (written 350 B.C.) translated by R. P. Hardie and R. K. Gaye. available at: `http://classics.mit.edu/Aristotle/physics.html`.
10. Mycielski, J.: Analysis without actual infinity. Journal of Symbolic Logic **46** (1981) 625–633
11. Kołodziejczyk, L.: Truth definitions in finite models. The Journal of Symbolic Logic **69** (2004) 183–200
12. Kołodziejczyk, L.: A finite model-theoretical proof of a property of bounded query classes within PH. The Journal of Symbolic Logic **69** (2004) 1105–1116
13. Krynicki, M., Zdanowski, K.: Theories of arithmetics in finite models. Journal of Symbolic Logic **70(1)** (2005) 1–28
14. Mostowski, M., Wasilewska, A.E.: Arithmetic of divisibility in finite models. Mathematical Logic Quarterly **50(2)** (2004) 169–174
15. Mostowski, M., Zdanowski, K.: FM–representability and beyond. in this volume (2005)
16. Gandy, R.: Church's thesis and principles for mechanism. In Barwise, J., Kunen, K., eds.: The Kleene Symposium, North–Holland Publishing Company (1980) 123–148

An Interval-valued Computing Device

Benedek Nagy

Department of Computer Science, Institute of Informatics, University of Debrecen
Hungary H-4010 Debrecen, PO Box 12. Hungary
Research Group on Mathematical Linguistics, Rovira i Virgili University
Tarragona, Spain
nbenedek@inf.unideb.hu

Abstract. In this paper we introduce a computing system which based on computations on intervals over [0,1]. We present some natural reasons why this method can be useful and interesting. Our interval-values are built up from points and atomic intervals. We extend the Boolean operators to these values in a natural way. Changing the bytes of the traditional computing to interval-values one gets an interval-valued computing device. We investigate some other operators on interval-values as well, such as shift and product operators. With a simulation we show that the interval-valued computing device is an extension of the classical computing devices working on bits in a byte. We show that, theoretically, this device is more effective than the traditional computers, because of the possibility of continuum parallelism. A method is presented to solve the SAT problem in linear time using the interval-valued approach. Our device is a fundamental computing device using the usual concept of intervals to make the computations in a more effective way.

1 Introduction

In the end of the 20th century many so-called natural computing device have appeared in the (theoretical) computer science. They are, for instance the DNA-computing devices (insertion-deletion systems, splicing systems), the quantum-computing, and the membrane-computing [2]. To show the efficiency of these systems, there are shown that for example the SAT problem can be solved by polynomial time with some of their help. It is a very strong argument for these systems because there is not known algorithm to solve the SAT problem (or any other NP-complete problem) in polynomial time with a classical computing device.

In this paper we investigate a so-called interval-valued computing device. The intervals are very common. When one is drawing with a pencil on a paper, (s)he uses points and (approximate) intervals. We would like to use this natural concept for a computing device. In a traditional computer there are some bits (usually the number of them is a power of 2) in a byte and the computing process uses a/some byte(s) in a computing step. One byte can refer a natural number in the interval between 0 and $2^k - 1$. Theoretically the efficiency of the computer is highly depend on the value of k. In our investigation a so-called byte can refer not only a finite number of integers and not for real numbers in a given interval, but for interval-values, which mean very high efficiency in computing. To show the efficiency of our system we show a method to simulate the traditional – finitely many bits in a byte – computing and we solve the SAT problem in linear time.

The structure of the paper is as follows: after giving some natural motivations, first we recall some basic concepts of the classical propositional logic and the work of a central processing unit of a traditional computer. After this we present our new system, in which the computing is going with intervals. We show how our system can simulate a classical computing device and why this system is more general. We will show how the interval-valued computing device can solve the SAT problem in linear time, as well. Finally we will summarize our results.

2 Motivations from the Nature

As it is already mentioned when one use a pencil and (s)he draws on a paper then the figure contains points and lines. Restricting the paper to a 1-dimensional line the figure contains intervals (and points). So the intervals are very common for humans and using them looks very natural.

In this part we have some arguments why and how interval-values can be represented in the nature. Everybody knows that in our world almost every material objects are built up by atoms. The size of the atoms is approximately an Angstrom (å), which is 10^{-10} meter. In physics and in chemistry there are materials and methods that can be used up to the measure-mistake is comparable with the size of the atom. Well-known, for instance that using traditional photo-cameras the resolution of the photo is extremely high. Films for photography are coated with a number of very thin light-sensitive emulsion layers consisting of silver salts and/or dyes. The resolution is very high because the used chemical materials, especially the silver-colloids make the reaction by molecule [3].

In some TFT and LCD devices the polarization of the molecule can be changed by ones. So using a sequence of atoms or molecule and changing some measurable properties of any of them can be represented with a device, which uses interval-values. (With enough large devices with enough large resolution one can imagine continuous intervals, as a pencil leaves a 'continuous' graphite line on the paper.) We are going to represent and use an abstract device in this paper; and now we do not care about the technical details of a possible machine which works in the practice with interval-values in its registers and how the calculation with these interval-values can be realised in physical level.

3 Preliminaries: the Traditional Computing (with Classical Logic)

In this section we recall some basic definitions and facts of the classical logic [1] and the "classical" computing devices [7, 8]. We are going to use and extend them.

3.1 The Classical Logic

In classical logic we have statements, each of them either true or false. We use notation 1 for true and sign 0 for false. From the atomic statements, the complex statements can be built by the connectives. See Table 1 for the basic ones.

There are some more logical operators in classical logic, but we can define them by using the basic operators.
(equivalence: $A \equiv B =_{def} (A \to B) \wedge (B \to A)$, 'xor': $A \oplus B =_{def} A \equiv \neg B$, 'nand': $A \underline{\&} B =_{def} \neg A \vee \neg B$, 'nor': $A|B =_{def} \neg A \wedge \neg B$.)

Table 1. Truth-table of basic logical operators

Name	1st variable	2nd variable	negation	conjunction	disjunction	implication
Sign	A	B	¬A	A∧B	A∨B	A→B
values	0	0	1	0	0	1
	0	1	1	0	1	1
	1	0	0	0	1	0
	1	1	0	1	1	1

3.2 Traditional Computers: Bits and Operations

We call bit the unit of information, which is the answer of a yes-no question. This concept is widely used in information theory and in statistical physics also. The Central Processing Unit in a computer uses some bits – called a byte – to calculate in the same time. The number of bits in a byte first was 4, later was 8 in 1980 (for example in Commodore 64), after a decade 16, after this 32 etc. So we can say that the higher the number of bits in a byte, the higher the level of the CPU in a computer. (In computers they use only powers of two.)

The Arithmetical-Logical Unit is a part of the CPU which can produce some logical and non-logical operators (add, shift etc.). First we are dealing with logical operators of the ALU. The basic logical operators are negation, conjunction and disjunction. In fact they use 'xor', 'nand' and 'nor' operations also, but they are not independent from our basic operators, therefore one can express them by the basic operators. These operators work on bytes, but they work on each bit of the bytes (see Table 2). Note, that sometimes the decimal or the hexadecimal representation of a byte is used, but they are only simple translations from the binary form.

As we can see the logical operations are going in a parallel way on each bit. It is one of the reasons why a computer using more bits in a byte can be faster (in theoretically). The calculations of a classical computer based on the classical logic and on some other operations. Now, some non-logical (usually they called arithmetical) operators will be detailed.

We underline non-logical operators: the shift operators. They are the left- and the right-shift. They work on only one argument, cancelling one of the terminal

Table 2. The basic logical operators on a byte (bits), where the value of each x_j and y_i is either 0 or 1, actually

value of A	negation of A	value of B	conjunction of A and B	disjunction of A and B
$x_1 \quad x_2 \ldots x_n$	$\neg x_1 \neg x_2 \ldots \neg x_n$	$y_1 \quad y_2 \ldots y_n$	$x_1 \quad x_2 \ldots x_n$ $\wedge \quad \wedge \ldots \quad \wedge$ $y_1 \quad y_2 \ldots y_n$	$x_1 \quad x_2 \ldots x_n$ $\vee \quad \vee \ldots \quad \vee$ $y_1 \quad y_2 \ldots y_n$

bits and writing 0 to the other end of the byte (see Table 3). Mathematically these operations mean the double (forgetting the first bit) or the (integer part of the) half of the operand.

Table 3. Shifting bits in a byte

value of A in binary code	left-shift A	right-shift A
$x_1 \quad x_2 \ldots x_n$	$x_2 \quad x_3 \ldots x_n$ 0	0 $x_1 \quad x_2 \ldots x_{n-1}$

The add operator is one of the most important operators of bits, which is more complex than the previous ones and it can be calculated with the previous operators. The following construction can add two numbers represented by n bits. The i-th index means the bit representing the coefficient of the value 2^{n-i}. We will construct the value C by induction, which are the overflow bits. (The ALU uses logical gates to get sum of two numbers.) The calculation starts with the last bits and going step by step to the first one.

Let $C_n = A_n \wedge B_n$.

After this: $C_i = (A_i \wedge B_i) \vee (A_i \wedge C_{i+1}) \vee (B_i \wedge C_{i+1})$. $(n > i \geq 1)$

The bit values of the result are: $R_n = A_n \oplus B_n$, and

$R_i = (A_i \oplus B_i) \oplus C_{i+1}$. $(n > i \geq 1)$ and

C_1 is the overflow bit of the result.

This calculation is the basis of the computations of traditional computers, at the physical level the calculation is based on signs 0's and 1's.

We reviewed the basic operators of traditional computing as well as the basic operators of the classical propositional logic.

4 The Interval-valued Computing Device

In this section we are presenting our new system. First we define what the interval-values mean in a mathematical way. After this we extend some logical operators to the interval-valued system (based on [5, 6]). Some non-logical operators are also shown. In the second subsection of the section we show that our system is an extension of the traditional computing devices. The interval-valued device (using some restrictions for the used interval-values) can simulate the traditional computing. Finally, in this section a method is presented how our device solves the SAT problem in efficient way.

4.1 Mathematical Description of the System

In this section, – using mathematical accuracy – we describe our model of computation. The computing device is almost similar to the processing unit of a traditional computer, but using interval-values in the computation instead of bytes.

The Interval-values In the interval-valued computing system we will use interval-values above $[0,1]$. We give an iterative definition:
Each interval $[a,b]$ with $0 \leq a \leq b \leq 1$ is an atomic interval which contains all the points x between a and b ($a \leq x \leq b$). With special choosing $a = b$ we receive that each point of the interval $[0,1]$ is an atomic interval itself.
The union and the difference of two interval-values are interval-values. (the union of two interval-values consists of each point which occurs at least one of the original one, while the difference of the interval-values A and B contains each point which is in A but not in the interval-value B)

Each interval-value can be constructed from atomic interval-values using only operations union and difference.

We have two special interval-values: the empty and the full interval. We will refer them with the signs $\perp$ and $\mathbf{T}$, respectively. They can be seen in Figure 1. $\mathbf{T}$ is an atomic interval: $[0,1]$ and $\perp$ can be obtained as the difference of any interval-value with itself (or difference of any interval-value and $\mathbf{T}$).

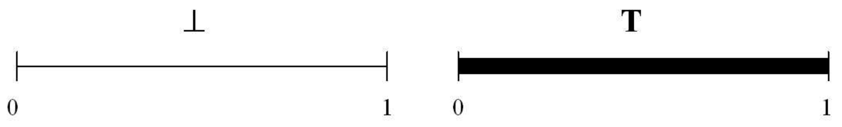


Fig. 1. Graphical representation of the interval-values $\perp$ and $\mathbf{T}$

Logical Operations with Interval-values In this part we present a natural extension of the Boolean operators of classical logic to the interval-valued system [5]. Each point of $[0,1]$ is either in the interval-value or not. We use the basic connectives of the interval-values as the connectives work in classical logic for each of the points of the interval $[0,1]$. More precisely, let $A(x)$ the characteristic function of the interval-value A, i.e. $A(x) = 1$ if $x \in A$ and $A(x) = 0$ if $x \notin A$.

Let the characteristic function of interval-value B be $B(x)$. Then the interval-value resulted by a logical operator acting on interval-values A, B, contains all the points x which have value 1 using the same logical operations between the characteristic values of the interval-values at point x. Table 4 shows what this procedure give for the basic logical connectives.

Table 4. The basic logical operators for interval-values

Name	negation	conjunction	disjunction	implication
Sign	¬A	A ∧ B	A ∨ B	A → B
Value	T \ A	A ∩ B	A ∪ B	¬(A\B) = ¬A ∪ B
Set theoretically:	complement	intersection	union	

It is trivial that the negation of an interval-value is an interval-value (the result is the difference of **T** and the interval-value). The disjunction and the implication of two interval-values are interval-values as well, as Table 4 shows.

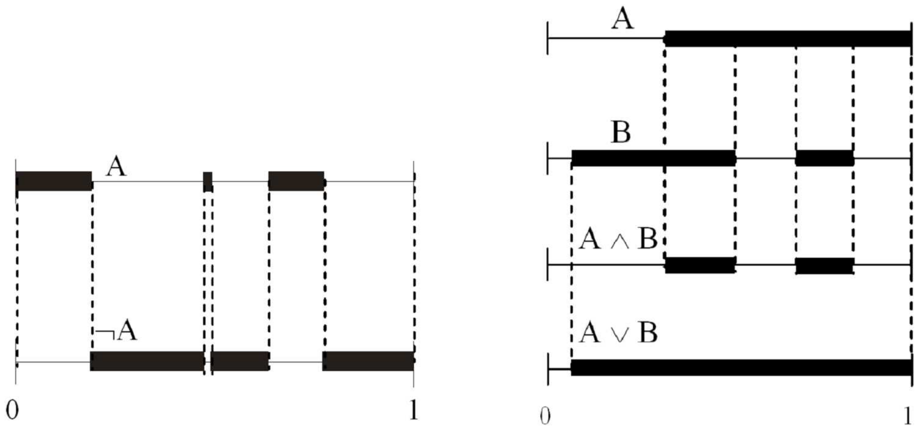


Fig. 2. Examples for logical operations with interval-values, negation (left), conjunction and disjunction (right)

Now, we are proving it for the conjunction, by using the set-theoretical De-Morgan's law. $A \cap B$ is equal to $\neg(\neg A \cup \neg B)$ therefore $A \wedge B$ is an interval-value when A and B are interval-values.

Figure 2 shows some examples for acting logical connectives among interval-values. Theoretically, the logical operations of interval-values act in continuum many points in a parallel way. Note, that in graphical representation – for the sake of simplicity – we deal with points only when they play an important role, elsewhere we represents only the interval components with non-zero length.

Non-logical Operators Similarly as the traditional computer has more operations, to use our system as a general computing device we need to introduce some other operations. To make our system more efficient we will investigate the shift operators in two different ways.

But, first we introduce the first-length function which will help us. This function assigns a real number to each interval-value, namely the length of its first component. More precisely:

$\mathit{Flength}(A) = b - a$, if A contains the open interval (a, b) and A does not contain any interval (a, c) with $c \geq b$, moreover the difference of A and $[a, b]$ does not contain any point x with $x < a$.

$\mathit{Flength}(A) = 0$, in other cases.

By the help of the previous function we investigate two kinds of shift operators. Both of them have two operands. The left-shift operator will shift the first interval-value by first-length of the second operand to the left, and removing the part which is shifted out of the interval $[0, 1]$. Opposite to this, we define the right-shift operator in a circular way, i.e. the parts shifted above 1 will appear at the other end of $[0, 1]$. Mathematically, let $\text{Lshift}(A, B)$ and $\text{Rshift}(A, B)$ be the interval-values given after A was shifted to the left and to the right by $\mathit{Flength}(B)$. (Similarly, we use the notation as the characteristic function of the interval-value.) Then,

$\text{Lshift}(A, B)(x) = A(x + \mathit{Flength}(B))$ if $0 \leq x + \mathit{Flength}(B) \leq 1$, and

$\text{Lshift}(A, B)(x) = 0$ in other cases.

$\text{Rshift}(A, B)(x) = A(\mathit{frac}(x - \mathit{Flength}(B)))$, where $x < 1$. The function *frac* gives the fractional part of a real, i.e. $\mathit{frac}(x) = x - \mathit{int}(x)$, where $\mathit{int}(x)$ is the greatest integer which is not greater than x. Finally, for completeness let

$\text{Rshift}(A, B)(1) = \text{Rshift}(A, B)(0)$.

In Figure 3 some examples are shown both for operations Rshift and Lshift. The second (ancillary-) operands are shown with grey colour to help understanding, but they are not real parts of the resulted interval-values.

Note, that with Lshift one loose all the information which go outside of the interval $[0, 1]$. Opposite to this case, using Rshift only the information at the point 1 will loose.

These operators are very effective ones, with combined use of them one can move or delete any part of an interval-value. The deletion can be in the following way. Use the Rshift to shift the interval-value such a way that the part – which should be deleted – begins at 0. Then, using Lshift the chosen part $[0, a)$ will be deleted. After this cutting, using Rshift again the remained part can be shifted back to its original place.

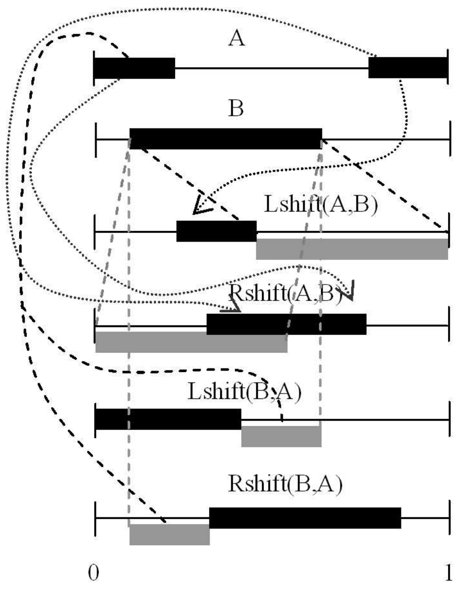


Fig. 3. Examples of shift operators with interval-values

Now we are presenting another operator with which one can multiply the interval-values, namely the (Fractalian) Product. Let A contain k interval components with ends $a_{i,1}, a_{i,2} (1 \leq i \leq k)$ and B contain l components with ends $b_{i,1}, b_{i,2} (1 \leq i \leq l)$. Then let the value of $C = A * B$ be the following: let the number of components of C be $k \cdot l$. For this process we can use double indexes for the components of C. Let the starting and ending point of the ij-th component be $a_{i1} + b_{j1}(a_{i2} - a_{i1})$ and $a_{i1} + b_{j2}(a_{i2} - a_{i1})$ respectively. An endpoint belongs to the interval iff the original endpoints belong to the interval-values of the original interval-values A and B.

As we can see in the definition, this product looks like the Cartesian product; the indexes works in the same way. The component ij of C comes from the i-th component of A and the j-th component of B. Moreover, it is easy to show, that the sum of the lengths of the components of $A * B$ is exactly the same as the product of the sum of the length of the components of A and the sum of the length of B. Note that the (Fractalian) Product is not a symmetric operator.

In Figure 4 there are examples shown.

4.2 Modelling Traditional Computing Devices with the Interval-valued Device

In this part we show that the interval-valued system can easily simulate a traditional computing device. Suppose that the traditional device use n bits in a

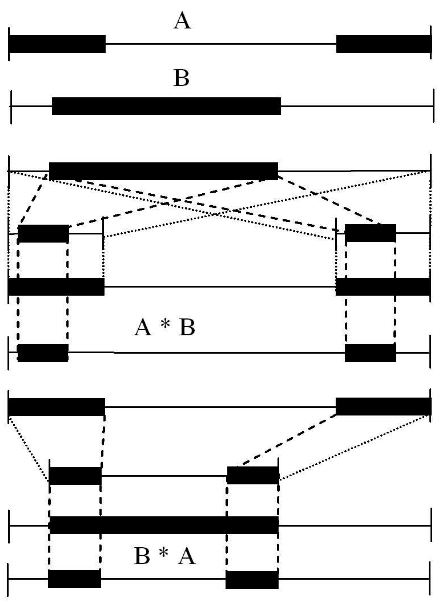


Fig. 4. Examples for Product of the interval-values

byte. Take the interval-values $[m/n, (m+1)/n]$ (for $0 \leq m \leq n-1$), each of them represents the $(m+1)$st bit. When the ALU of a classical computer works on some bytes then the interval-valued device uses the interval-values representing the bytes.

Modelling logical operators is trivial therefore we omit them. The shift operators need for example the atomic interval-value $P = [0, 1/n]$. Left-shift can also be simulated in a trivial way. We show the simulation method of right-shift: using Rshift is not enough in its pure form, because of its circular property. It is easy to show that Rshift(Lshift(Rshift(A, P)$, P$)$, P$) gives the desired result shifting the interval-value to the right with eliminating the shifted-out part.

Now we introduce an abbreviation: let Rshift$(A, 2B)$ mean that we use twice the operator with the same second operand: Rshift(Rshift$(A, B), B)$. We will use this abbreviation with any positive integers (here up to n). Similarly: Lshift$(A, kB) =_{def}$ Lshift(Lshift(… (Lshift(A, B), …), B), B) using it k times.

The simulation of the Add operation is going in the following way:

Let A and B be two interval-values which represent numbers as bit-values. Using the shift operators we can translate the Add operator: First we construct the interval-value of overflow C. The last bit of C is

$$C_n = \text{Rshift}(\text{Lshift}(A \wedge B, \neg P), \neg P).$$

Note that here Lshift$(A \wedge B, \neg P)$ has the same meaning as shifting by the value $(n-1)P$. Similar statement holds for the Rshift.

In the iteration method (we gather the bits and add a new one in each step) i goes from $n-1$ to 2 decreasing by 1 in each step:

$$C_i = C_{i+1} \vee \mathrm{Lshift}(\mathrm{Rshift}(\mathrm{Rshift}(\mathrm{Lshift}(A \wedge B, (i-1)P), (i-1)P) \vee$$

$$\mathrm{Rshift}(\mathrm{Lshift}(A \wedge \mathrm{Lshift}(C_{i+1}, P), (i-1)P), (i-1)P) \vee$$

$$\mathrm{Rshift}(\mathrm{Lshift}(B \wedge \mathrm{Lshift}(C_{i-1}, P), (i-1)P), (i-1)P), (n-i)P), (n-i)P).$$

(The last value, C_2 contains all the overflow bits that we need to the calculation. The first overflow bit – which is the overflow bit actually, – would be given by only the next step of these iteration with the value of C_1 which we do not need to the operation.) And now we use 'xor' to get the sum of A and B:

$$A + B = (A \oplus B) \oplus \mathrm{Lshift}(C_2, P).$$

As we showed the interval-valued system can simulate the classical computing with finitely many bits in a byte. To the simulation we used only restricted interval-values, which can be obtained from the atomic interval-value $P = [0, 1/n]$ by logical operations and shifts. Therefore one can say that the interval-valued device is an extension of the classical one. Moreover, in logical computation our system looks like a system with continuum-many bits in a byte: each point of $[0, 1]$ can represent a bit.

4.3 Solving SAT Problem with the Interval-valued Device

As one of our last results, showing the theoretical efficiency of the interval-valued computing device we solve the SAT problem. The SAT problem is the following: given a formula in propositional logic, decide whether it is satisfiable or not (it means logical falsity when it is unsatisfiable). This problem is an NP-complete problem and plays very important role in complexity-theory.

Usually only formulae in conjunctive (or in k-ary conjunctive) normal form are used, but in this paper we do not use any restriction about the forms of them.

Assume that there are number n variables occur in the formula. Let the interval-value be assigned to the i-th variable

$$A_i = \bigcup_{j=0}^{2^{i-1}-1} \left[\frac{j}{2^{i-1}}, \frac{2j+1}{2^i}\right).$$

One can compute them using the product operator: $A_{i+1} = A_i * [0, 0.5) \vee \neg A_i * [0, 0.5)$.

Then one can evaluate the logical formula using interval-values A_i. If the result has the value $\perp$, then the formula is unsatisfiable. Elsewhere the formula is satisfiable, moreover if the first-length of the result is 1 then the formula is a Boolean tautology. We show that the formula can be evaluated to true when the result using interval-values differs with $\perp$. Assume that the characteristic function of the result is 1 at x (such a value must exist in an interval-value which is not equal to $\perp$). In this case let the values of the formula evaluated by Boolean variables be the following: let the value of the i-th Boolean variable be the characteristic value $A_i(x)$. With these values the propositional formula must evaluate to true, similarly with the interval-value at point x. The process can be seen in the next example (see Fig. 5). Let the formula be $\neg((B \wedge C) \rightarrow \neg(C \vee (D \rightarrow \neg B)))$. The figure shows that this formula is satisfiable (it is true when B and C are both true), but it is not a Boolean tautology.

One can see that to check the satisfiability of a formula we need interval-values not more than the length of the formula. More precisely, counting the number of variables and logical connectives, their sum gives an upper bound to the number of interval-values are needed for the calculation, because without using lazy evaluation we need the interval-value of all the variables, and of each subformula of the original.

Calculating the satisfiability of a formula with interval-values looks like to the evaluation going in parallel way, building the Boolean truth-table of the formula by lines.

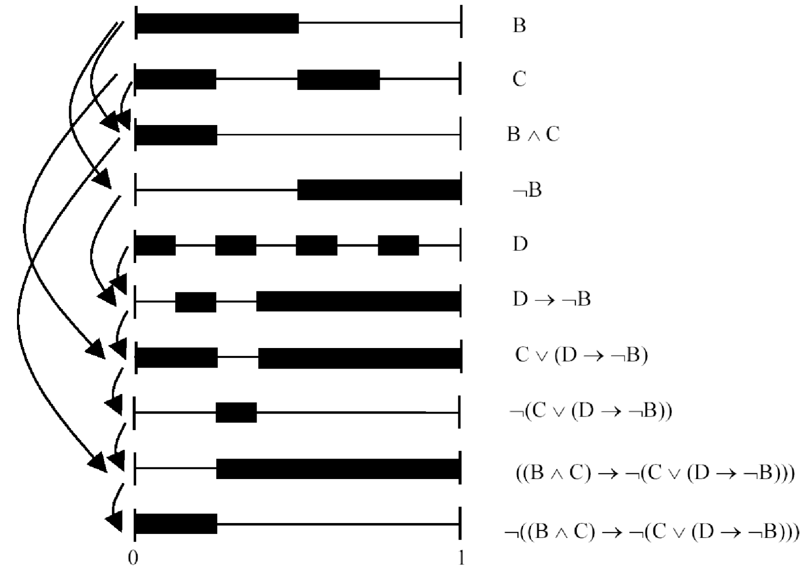


Fig. 5. Solving the satisfiability of the formula $\neg((B \wedge C) \rightarrow \neg(C \vee (D \rightarrow \neg B)))$.

5 Conclusion

In our paper we investigated a new theoretical model of computing. Our naturally motivated device works with interval-values. The interval-values are used as bytes in traditional computers. An abstract mathematical description of the system is given. Beside the logical operators, some non-logical operators are also defined. The interval-valued computing device is an extension of the classical computing devices. To present its applicability and efficiency we showed how it can solve the SAT problem. The key-point is that the system use parallelism in a natural way because its architecture.

Note, that in [5] some other non-logical operators were introduced, such as mirror of intervals. Using them the computation can be in a faster way. Note also, that for simplicity in practice one can use only closed intervals by closing the result after each operation.

It is a matter of a future work to analyze the practical applicability of the interval-valued system (or one of its restricted forms). Both physical realisation and the way of possible programming of the system can be interesting. Interesting questions about the system, that how efficient it can be. It may solve EXPTIME or PSPACE problems in polynomial time. By simulating classical computing the computing power should be at least the level of Turing machines, but in this system one can use continuum many values (moreover the number of possible interval-values is more than continuum), therefore the power can be more. How one can use it in a more efficient way?

References

1. Bell, J.L. and Machover, M., *A course in mathematical logic*, Elsevier Science, Amsterdam, New York, Oxford, North-Holland, 1977.
2. Calude, C.S. and Paun, G., *Computing with Cells & Atoms: An Introduction to Quantum, DNA & Membrane Computing*, Taylor & Francis, London, 2001
3. Davidson, M.W. and Rill, R.L., "Photomicrography: Common Ground for Science and Art," *Microscopy and Analysis 11*, pp. 7-12, 1989.
4. Hájek, P., *Metamathematics of Fuzzy Logic*, (Trends in Logic **4**), Kluwer Academic Publ., Dordrecht 1998.
5. Nagy, B., "Interval-valued logic as a generalization of many valued logics," Technical Report 2002/20., Institute of Mathematics and Informatics, University of Debrecen, submitted to publication
6. Nagy, B., "Intervallum-logika," Thesis (in Hungarian), University of Debrecen, 1998. (earlier version is presented at Hungarian Scientific Student Conference, Eger, Hungary, 1997.)
7. Patterson, D. and Hennessy, J., *Computer Organization and Design: The Hardware/Software Interface*, Morgan Kaufmann, 1997.
8. Tomek, I., *The Foundations of Computer Architecture and Organization*, Computer Science Press, New York, 1990.

On the notion of ∀-definedness of non-deterministic programs

Stela K. Nikolova

Faculty of Mathematics and Computer Science,
Sofia University,
5 James Bourchier Blvd.,
1164 Sofia, Bulgaria,
`stenik@fmi.uni-sofia.bg`

1 Introduction

We consider a kind of while programs over an abstract data type, that along with the usual assignment and loop statements has also a non-deterministic choice statement $\mathtt{x_i} := \mathtt{arbitrary(A)}$, where A is a subset of the basic set. After the execution of such a statement, the register $\mathtt{x_i}$ is evaluated with an arbitrary element of A. The choice of this element is completely arbitrary – it does not depend on the input, the current configuration, etc. Due to this non-deterministic operator, one and the same input may lead to different computational paths, both finite and infinite. By analogy with the non-deterministic Turing machines, we can speak about the set, *accepted* by a non-deterministic program P – this is the set $L(P)$ of those inputs, for which *there exists* a finite (accepting) computation. The set of all inputs I, such that *every* computation, starting from I is finite, is the set $D(P)$ of the so-called *points of* ∀*-definedness* of P (cf. Manna [1]). It is clear that the set of all points of ∀-definedness of a non-deterministic Turing machine is again Turing semi-recognizable (although at exponential prise). However, this is no longer true for the type of non-determinism that we consider here ([2], [3]). For example, in [2] it is shown that for the case of non-deterministic programs over the natural numbers $\mathbb{N}$ with a choice operator $\mathtt{x_i} := \mathtt{arbitrary}(\mathbb{N})$, the sets $D(P)$ coincide with the Π^1_1 sets, while the sets $L(P)$ are exactly the recursively enumerable sets. In [3] a syntactical characterization of the sets $L(P)$ and $D(P)$ for the most general case of non-deterministic programs with counters and stacks is obtained.

In the present work we study the sets $D(P)$ and $L(P)$ for a non-deterministic program P over an admissible abstract structure $\mathfrak{A}$. We give both explicite and inductive characterization of these sets. In the next section we introduce the appropriate definitions and formulate the precise results.

2 Preliminaries

Given an arbitrary total structure $\mathfrak{A}_0 = (B; f_1, \dots, f_a; P_1, \dots, P_b)$ (the case f_i is a constant is also possible), its least acceptable extention $\mathfrak{A}$ [4] is determined as follows: Take an object $O \notin B$ and let $\langle\ \rangle$ be a pairing operation, such that no element of $B_0 = B \cup \{O\}$ is an ordered pair. Let B^* be the least set, that contains B_0 and is closed under $\langle\ \rangle$. Denote by L and R the left and right decoding functions for the mapping $\langle\ \rangle$ (assume that $L(s) = R(s) = O$ for $s \in B_0$). The initial functions and predicates of $\mathfrak{A}_0$ are extended on B^* by the equalities $f_i(s_1, \dots, s_n) = O$ for $(s_1, \dots, s_n) \notin B^n$ and $P_i(s_1, \dots, s_n) =$ "*falsity*" for $(s_1, \dots, s_n) \notin B^n$.

Now put $\mathfrak{A} = (B^*; O, \langle\ \rangle, L, R, f_1, \dots, f_a; B, P_1, \dots, P_b)$. We shall suppose that the equality relation is among the basic predicates of $\mathfrak{A}$. Throughout the paper we shall assume this structure $\mathfrak{A}$ fixed. As customary, the natural numbers will be identified with the elements of the set $Nat = \{O, \langle O, O\rangle, \langle\langle O, O\rangle, O\rangle, \dots\}$. Using the coding function $\langle\ \rangle$, we define a coding $\ll\ \gg$ of all finite sequences from B^*, putting $\ll\ \gg = O, \ll s_1, \dots, s_{n+1} \gg = \langle \ll s_1, \dots, s_n \gg, s_{n+1}\rangle$.

A *non-deterministic program P over* $\mathfrak{A}$ is a construction of the type $\mathtt{input}(\mathtt{x_1}, \dots, \mathtt{x_k}), S.$, where $\mathtt{x_1}, \dots, \mathtt{x_k}$ are the *input variables* of P and S is a *statement*. Here a statement is defined inductively as follows: initial statements are $\mathtt{x_i} := \tau$ (assignment) and $\mathtt{x_i} := \mathtt{arbitrary}(\mathtt{B}^*)$ (choice statement); if S_1 and S_2 are statements, then $S_1; S_2$ (composition) and $\mathtt{while}\ C\ \mathtt{do}\ S\ \mathtt{od}$ (loop) are statements (here τ and C are a term and a quantifier-free formula in the language $\mathcal{L}_{\mathfrak{A}}$ of $\mathfrak{A}$). The semantics of the assignment, the composition and the loop statements is the usual one. As we said above, the execution of the choice statement $\mathtt{x_i} := \mathtt{arbitrary}(\mathtt{B}^*)$ assigns to the register $\mathtt{x_i}$ an arbitrary element of B^*. Let us notice that, since B^* is infinite, the computational trees are infinite branching, hence $D(P)$ turns out to be much more complex than $L(P)$. The precise definitions, concerning the sets $L(P)$ and $D(P)$, are in the next section.

We will need infinite sequences $\{\Phi^n\}_n$ of quantifier-free formulas in $L_{\mathfrak{A}}$ for the explicite description of the sets $D(P)$ and $L(P)$. Let us call such a sequence *primitive recursive*, if the function, which assigns to each n the code of Φ^n is primitive recursive.

We next remind some basic notations and results concerning inductive definability (cf. for example [4]). Let $\varphi(x_1, \dots, x_k, X)$ be a first-order formula in $\mathcal{L}_{\mathfrak{A}}$, in which the relational variable X occurs only positively. Then φ determines the mapping $\Gamma_\varphi : \mathcal{P}((B^*)^k) \to \mathcal{P}((B^*)^k)$, defined with $\Gamma_\varphi(A) = \{(s_1, \dots, s_k) |$ $\mathfrak{A} \models \varphi(s_1, \dots, s_k, A)\}$. The sets I^ξ_φ are defined by transfinite induction on ξ: $I^\xi_\varphi = \Gamma_\varphi(\bigcup_{\eta<\xi} I^\eta_\varphi)$. Then the set $I_\varphi = \bigcup_\xi I^\xi_\varphi$ is the least fixed point of Γ_φ. Finally, for every $\bar{s} \in I_\varphi$ put $|\bar{s}|_\varphi = \min\{\xi | s \in I^\xi_\varphi\}$. It is convenient to consider that for all $\bar{s} \notin I_\varphi$, $|\bar{s}|_\varphi = \zeta$, where ζ is any ordinal, greater than $sup\{|\bar{s}|_\varphi : \bar{s} \in I_\varphi\}$. A set $A \subseteq (B^*)^m$ is *inductively definable* (by φ on $\mathfrak{A}$) if A is I_φ or a section of I_φ for some X-positive formula φ.

For the inductive characterization we will need formulas in the first-order language of an extention $\mathfrak{A}^+$ of $\mathfrak{A}$. The structure $\mathfrak{A}^+$ is $(\mathfrak{A}; Nat, Seq, U, \phi)$, where $U(a, s)$ is some fixed universal relation for all quantifier-free formulas with one free variable in $\mathcal{L}_{\mathfrak{A}}$, and $\phi(a, n)$ is some fixed universal function for all unary primitive recursive functions.

Now we are ready to formulate our main result.

Theorem 1. *Let D and L be subsets of $(B^*)^k$. Then the following statements are equivalent:*

(i) There exists a non-deterministic program P over $\mathfrak{A}$ such that $D = D(P)$ and $L = L(P)$.

(ii) There exists a primitive recursive sequence of quantifier-free formulas $\{\Phi^n\}_n$ in $\mathcal{L}_{\mathfrak{A}}$ with variables among $x_1, \dots, x_k, y$ such that

$$L = \{(s_1, \dots, s_k)|\ \exists\alpha_{\alpha:\mathbb{N}\to B^*}\exists n\ \mathfrak{A} \models \Phi^n(s_1, \dots, s_k, \bar{\alpha}(n))\},$$

$$D = \{(s_1, \dots, s_k)|\ \forall\alpha_{\alpha:\mathbb{N}\to B^*}\exists n\ \mathfrak{A} \models \Phi^n(s_1, \dots, s_k, \bar{\alpha}(n))\}.$$

(iii) There exists an X-positive quantifier-free formula $\varphi(x_1, \dots, x_k, y, z, t, X)$ in $\mathcal{L}_{\mathfrak{A}^+}$ such that

$$L = \{(s_1, \dots, s_k)|(s_1, \dots, s_k, O, O) \in I_{\exists y\varphi}\},$$

$$D = \{(s_1, \dots, s_k)|(s_1, \dots, s_k, O, O) \in I_{\forall y\varphi}\}.$$

Here, as usual, $\bar{\alpha}(n)$ stands for $\ll \alpha(1), \dots, \alpha(n) \gg$. We can imagine the sequence $\alpha = \alpha(1), \alpha(2), \dots$ in (ii) as the sequence of the successive values, returned by the choice statement in the course of the computation. Clearly, if we have these values in advance, the execution of P (no matter finite or infinite) over a fixed input $(s_1, \dots, s_k)$ is uniquely determined. Our proposition (ii) says that this execution can be carried out in some canonical way: compute successively $\Phi^0(\bar{s}, \bar{\alpha}(0))$, $\Phi^1(\bar{s}, \bar{\alpha}(1)), \dots$ until you find the first n for which $\Phi^n(\bar{s}, \bar{\alpha}(n))$ holds. (Here and further, when saying that a formula Φ holds, we will mean that it holds in the relevant structure – $\mathfrak{A}$ or $\mathfrak{A}'$.) Another way to formulate (ii) is to say that some absolutely prime computable (cf. [5]) on $\mathfrak{A}$ predicate R exists, such that $L = \{\bar{s}|\exists\alpha\exists n R(\bar{s}, \bar{\alpha}(n))\}$ and $D = \{\bar{s}|\forall\alpha\exists n R(\bar{s}, \bar{\alpha}(n))\}$.

In [6] it is shown that the relations Seq, Nat, U, and the graph of ϕ are Δ^0_1-positively inductively definable on $\mathfrak{A}$. Hence the sets $I_{\forall y\varphi}$ are in fact Π^0_1-positive inductive on $\mathfrak{A}$. Thus the equivalence between (i) and (iii) gives us also a computational characterization of the Π^0_1-positive inductive definitions on $\mathfrak{A}$. As for the sets of the type $I_{\exists y\varphi}$, i.e., the sets that are Σ^0_1-positive inductively definable on $\mathfrak{A}$ – in [7] it is shown that they coincide with the absolutely search computable relations.

In the next section we prove the implication $(i) \Rightarrow (ii)$ of Theorem 1, in our last section are the proofs of $(ii) \Rightarrow (iii)$ and $(iii) \Rightarrow (i)$.

3 Syntactical characterization of the sets $L(P)$ and $D(P)$

We will need a detailed description of the input-output relation of P, which comprises the whole memory of P (i.e. all the registers that occur in P). Set $\mathrm{mem}(S) = \max\{n|\mathtt{x_n}$ occurs in $S\}$ and take an arbitrary $m \geq \mathrm{mem}(S)$. We can view the statement S as a nondeterministic transducer, that takes as an input a tuple $(s_1, \dots, s_m)$ and, if halts, returns as an output the current memory state

– say some $(t_1, \ldots, t_m)$ (of course, we will have $t_i = s_i$ for the variables x_i, that do not occur in S). The set of all $(s_1, \ldots, s_m, t_1, \ldots, t_m)$, thus obtained, is the *input-output relation* $R_m(S)$ of the statement S. We define also a set $D_m(S)$ of *all points of* $\forall$*-definedness* of S as the collection of all inputs $(s_1, \ldots, s_m)$, such that every computation of S, starting with input $(s_1, \ldots, s_m)$, is finite. We will describe the sets $R_m(S)$ and $D_m(S)$ by sequences of em clauses. A clause is a syntactical expression of the form $\Phi \to (\tau_1, \ldots, \tau_m)$, where Φ is a quantifier-free formula and $\tau_1, \ldots, \tau_m$ are terms in $\mathcal{L}_{\mathfrak{A}}$ (the tuple $(\tau_1, \ldots, \tau_m)$ will sometimes be contracted to $\bar{\tau}$). We will consider recursive sequences $\{\Phi^n \to \bar{\tau}^n\}_n$, whose variables are among $x_1, \ldots, x_m, y_1, y_2, \ldots$. Our intention is x_i to hold the current value of the register $\mathtt{x_i}$ of S, while the y-variables are for the additional input, coming from the choice statement. More precisely, we will place in y_n the value, returned after the n-th invocation of the choice statement.

Let E be an expression with variables among $x_1, \ldots, x_m, y_1, \ldots, y_n$. We shall write $E(s_1, \ldots, s_m, \alpha)$ for the value

$$E(x_1/s_1, \ldots, x_m/s_m, y_1/\alpha(1), \ldots, y_n/\alpha(n))$$

of E in $\mathfrak{A}$. Now suppose that S is a statement with registers among $\mathtt{x_1}, \ldots, \mathtt{x_m}$ and $\Sigma = \{\Phi^n \to (\tau_1^n, \ldots, \tau_m^n)\}_n$ is a sequence of clauses with variables among $x_1, \ldots, x_m, y_1, y_2, \ldots$ Say that Σ *represents* $R_m(S)$ and $D_m(S)$, iff for all $(\bar{s}, \bar{t})$:

$$(\bar{s}, \bar{t}) \in R_m(S) \iff \exists\alpha\exists n(\Phi^n(\bar{s}, \alpha) \& \ \tau_1^n(\bar{s}, \alpha) = t_1 \ \& \ldots \& \ \tau_m^n(\bar{s}, \alpha) = t_m) \quad (1)$$

$$\bar{s} \in D_m(S) \iff \forall\alpha\exists n \Phi^n(\bar{s}, \alpha). \quad (2)$$

A sequence $\Sigma = \{\Phi^n \to \bar{\tau}^n\}_n$ is *treelike* (on $\mathfrak{A}$), if for every $\bar{s}, \alpha$ and n: $\Phi^n(\bar{s}, \alpha) \Rightarrow \forall k \neq n \neg\Phi^k(\bar{s}, \alpha)$. Say that Σ is *primitive recursive*, if the function $\lambda n.$"the code of $\Phi^n \to \bar{\tau}^n$" is primitive recursive. Finally, let us say that S is *representable*, if there exists a primitive recursive treelike Σ, that represents $L_m(S)$ and $D_m(S)$ for $m = mem(S)$. Using induction on the definition of S, in the next three propositions we prove that every statement is representable.

Proposition 1. *Assignment and choice statements are representable.*

Proof. The assignment $\mathtt{x_i} := \tau$ is obviously representable by the treelike sequence $\Sigma = \{\Phi^n \to \bar{\tau}^n\}_n$, where Φ^0 is $x_1 = x_1$ and for $n > 0$ Φ^n is $\neg(x_1 = x_1)$; $\tau_i^0 = \tau$, $\tau_j^0 = x_j$ for $i \neq j$ and for $n > 0$ $\tau_j^n = x_j$, $1 \leq j \leq m$. For the choice statement $\mathtt{x_i} := \mathtt{arbitrary}(\mathtt{B}^*)$ the sequence Σ is almost the same – the unique difference is that $\tau_i^0 = y_1$ instead or τ. □

If E is an expression (a term or a quantifier-free formula) with variables among $x_1, \ldots, x_m, y_1, \ldots, y_n$, and $\tau_1, \ldots, \tau_m$ are terms and $p \in \mathbb{N}$, then

$$E[x_1|\tau_1 \ldots, x_m|\tau_m, y_1|y_{1+p}, \ldots, y_n|y_{n+p}] \text{ (or } E[\bar{x}|\bar{\tau}, \bar{y}|\bar{y} + p] \text{ for short)}$$

will denote the expression, obtained from E after the simultaneous replacement of x_i by τ_i and y_i by y_{i+p}. We have the following substitution property:

$$E[\bar{x}|\bar{\tau}, \bar{y}|\bar{y}+p](s_1, \dots, s_m, \alpha) = E(s'_1, \dots, s'_m, \alpha'), \tag{3}$$

where $s'_i = \tau_i(\bar{s}, \alpha), 1 \le i \le m$, and $\alpha'(i) = \alpha(i+p), i \ge 1$.

Let us fix some primitive recursive coding π of the pairs of natural numbers with decoding functions l and r, such that $r(n) < n$ for all $n > 0$.

Proposition 2. *If S_1 and S_2 are representable, then so is $S = S_1; S_2$.*

Proof. Let $mem(S) = m$ and suppose that the sequences $\{\Gamma^n \to (\gamma_1^n, \dots, \gamma_m^n)\}_n$ and $\{\Delta^n \to (\delta_1^n, \dots, \delta_m^n)\}_n$ represent S_1 and S_2, resp. Let p_n be the maximal index of an y-variable, that occurs in $\Gamma^n \to (\gamma_1^n, \dots, \gamma_m^n)$. For $n = \pi(k, l)$ set

$$\Phi^n = \Gamma^k \& \Delta^l[x_1|\gamma_1^k, \dots, x_m|\gamma_m^k, \bar{y}|\bar{y}+p_k],$$

$$\tau_i^n = \delta_i^l[x_1|\gamma_1^k, \dots, x_m|\gamma_m^k, \bar{y}|\bar{y}+p_k], 1 \le i \le m.$$

Clearly, $\{\Phi^n \to \bar{\tau}^n\}_n$ is primitive recursive and one can easily check that it is also treelike. We are going to show that it represents $R_m(S)$. Let us notice that

$$(s_1, \dots, s_m, t_1, \dots, t_m) \in R_m(S) \Leftrightarrow \exists q_1 \dots \exists q_m((\bar{s}, \bar{q}) \in R_m(S_1) \& (\bar{q}, \bar{t}) \in R_m(S_2))$$

$$\Leftrightarrow \exists \beta \exists k(\Gamma^k(\bar{s}, \beta) \ \& \ \gamma_1^k(\bar{s}, \beta) = q_1 \ \& \ \dots \ \& \ \gamma_m^k(\bar{s}, \beta) = q_m) \ \&$$
$$\exists \beta' \exists l(\Delta^l(\bar{q}, \beta') \ \& \ \delta_1^l(\bar{q}, \beta') = t_1 \ \& \ \dots \ \& \ \delta_m^l(\bar{q}, \beta') = t_m).$$

Now suppose that $(\bar{s}, \bar{t}) \in R_m(S)$ and take β, β', k and l as above. Put $n = \pi(k, l)$, $\alpha(i) = \beta(i)$, $1 \le i \le p_k$ and $\alpha(p_k + i) = \beta'(i)$ for $i \ge 1$. Then using (3) we get

$$\Phi^n(\bar{s}, \alpha) \Leftrightarrow \Gamma^k(\bar{s}, \alpha) \ \& \ \Delta^l[\bar{x}|\bar{\gamma}^k, \bar{y}|\bar{y}+p_k](\bar{s}, \alpha) \Leftrightarrow \Gamma^k(\bar{s}, \beta) \ \& \ \Delta^l(\bar{q}, \beta'),$$

hence $\Phi^n(\bar{s}, \alpha)$ holds. Further, $\tau_i^n(\bar{s}, \alpha) = \delta_i^l[\bar{x}|\bar{\gamma}^k, \bar{y}|\bar{y}+p_k](\bar{s}, \alpha) = \delta_i^l(\bar{q}, \beta') = t_i$ for $1 \le i \le m$. So we established the first direction of (1). For the opposite direction proceed in a similar way.

Towards proving the equivalence (2), let us make the following observation:

$$\bar{s} \in D_m(S) \Leftrightarrow \bar{s} \in D_m(S_1) \ \& \ \forall \bar{t}((\bar{s}, \bar{t}) \in R_m(S_1) \Rightarrow \bar{t} \in D_m(S_2)). \tag{4}$$

Suppose that $\bar{s} \in D_m(S)$ and take an arbitrary α. According to (4), $\bar{s} \in D_m(S_1)$, hence for some $k : \Gamma^k(\bar{s}, \alpha)$. Let $t_i = \gamma_i^k(\bar{s}, \alpha), 1 \le i \le m$. Then $(\bar{s}, \bar{t}) \in R_m(S_1)$ and again by (4), $\bar{t} \in D_m(S_2)$. Put $\beta(i) = \alpha(i + p_k), i > 0$. Since $\{\Delta^n \to \bar{\delta}^n\}_n$ represent S_2 and $\bar{t} \in D_m(S_2)$, there is an l with $\Delta^l(\bar{t}, \beta)$. From here by (3) we get $\Delta^l[\bar{x}|\bar{\gamma}^k, \bar{y}|\bar{y}+p_k](\bar{s}, \alpha)$, which together with $\Gamma^k(\bar{s}, \alpha)$ yields $\Phi^n(\bar{s}, \alpha)$ for $n = \pi(k, l)$.

Conversely, assume that $\forall \alpha \exists n \Phi^n(\bar{s}, \alpha)$. We will check that the right-hand side of (4) holds, hence $\bar{s} \in D_m(S)$. Indeed, $\forall \alpha \exists n \Phi^n(\bar{s}, \alpha)$ implies $\forall \alpha \exists n \Gamma^n(\bar{s}, \alpha)$, so $\bar{s} \in D_m(S_1)$. Take $\bar{t}$ such that $(\bar{s}, \bar{t}) \in R_m(S_1)$. Then there exist k and $q_1, \dots, q_{p_k}$ such that $\Gamma^k(\bar{s}, \bar{q})$ and $t_i = \gamma^k(\bar{s}, \bar{q}), 1 \le i \le m$. We have to see that $\bar{t} \in D_m(S_2)$, or equivalently, $\forall \beta \exists l \Delta^l(\bar{t}, \beta)$. Indeed, take an arbitrary β and put $\alpha(i) = q_i, 1 \le i \le p_k$, and $\alpha(p_k + i) = \beta(i), i \ge 1$. For this α there is an n with

$\Phi^n(\bar{s},\alpha)$, i.e. $\Gamma^{l(n)}(\bar{s},\alpha)$ and $\Delta^{r(n)}[\bar{x}|\bar{\gamma}^{l(n)},\bar{y}|\bar{y}+p_{l(n)}](\bar{s},\alpha)$. We have, however, that $\Gamma^k(\bar{s},\bar{q})$, or written with $\alpha, \Gamma^k(\bar{s},\alpha)$. Since $\{\Gamma^n \rightarrow \bar{\gamma}^n\}_n$ is treelike, this yields $l(n)=k$. Hence $\Delta^{r(n)}[\bar{x}|\bar{\gamma}^{l(n)},\bar{y}|\bar{y}+p_{l(n)}]$ is in fact $\Delta^{r(n)}[\bar{x}|\bar{\gamma}^{k},\bar{y}|\bar{y}+p_{k}]$, so $\Delta^{r(n)}[\bar{x}|\bar{\gamma}^{k},\bar{y}|\bar{y}+p_{k}](\bar{s},\alpha)$ is true, and by (3), $\Delta^{r(n)}(\bar{t},\beta)$ is true. □

Proposition 3. *If S_1 is representable, then so is $S=$* `while` *C* `do` *S_1* `od`.

Proof. Denote $mem(S)=m$ and suppose that $\{\Gamma^n \rightarrow (\gamma_1^n,\dots,\gamma_m^n)\}_n$ represents S_1. Let again p_n be the maximal index of an y-variable, that occurs in $\Gamma^n \rightarrow \bar{\gamma}^n$. We define the sequence $\{\Phi^n \rightarrow \bar{\tau}^n\}_n$ by induction on n as follows:

$$\Phi^0=\neg C,\ \tau_i^0=\ x_i \textit{ for } 1\le i\le m, \textit{and for } n>0$$

$$\Phi^n=C\&\Gamma^{l(n)}\&\ \Phi^{r(n)}[\bar{x}|\bar{\gamma}^{l(n)},\bar{y}|\bar{y}+p_{l(n)}],\ \tau_i^n=\gamma_i^{r(n)}[\bar{x}|\bar{\gamma}^{l(n)},\bar{y}|\bar{y}+p_{l(n)}].$$

We have that $r(n)<n$ for $n>0$, so the definition is correct. Clearly $\{\Phi^n \rightarrow \bar{\tau}^n\}_n$ is primitive recursive. The fact that it is treelike follows by an easy induction on n.

According to the semantics of the while-loop, $R_m(S)$ is the least set satisfying

$$(\bar{s},\bar{t})\in\ R_m(S)\Leftrightarrow\neg C(\bar{s})\&\bar{t}=\bar{s}\vee C(\bar{s})\&\exists\bar{q}((\bar{s},\bar{q})\in R_m(S_1)\&(\bar{q},\bar{t})\in R_m(S)) \tag{5}$$

i.e. $R_m(S)$ is I_φ for $\varphi(\bar{x},\bar{y},X):\neg C(\bar{x})\&\bar{y}=\bar{x}\vee C(\bar{x})\&\exists\bar{z}((\bar{x},\bar{z})\in R_m(S_1)\&(\bar{z},\bar{y})\in X)$. By transfinite induction on $|(\bar{s},\bar{t})|_\varphi$ we will show that

$$(\bar{s},\bar{t})\in R_m(S)\Rightarrow\exists\alpha\exists n(\Phi^n(\bar{s},\alpha)\&\ \bar{\tau}^n(\bar{s},\alpha)=\bar{t}). \tag{6}$$

If $|(\bar{s},\bar{t})|_\varphi=0$, then $\neg C(\bar{s})$ and clearly (6) holds for $n=0$ and α – arbitrary. Suppose that $|(\bar{s},\bar{t})|_\varphi>0$. Then $C(\bar{s})$ and there exists $\bar{q}:(\bar{s},\bar{q})\in R_m(S_1),(\bar{q},\bar{t})\in R_m(S)$ and $|(\bar{q},\bar{t})|_\varphi<|(\bar{s},\bar{t})|_\varphi$. By induction hypothesis, there exists β' and l such that $\Phi^l(\bar{q},\beta')$ and $\tau^l(\bar{q},\beta')=\bar{t}$. We have $(\bar{s},\bar{q})\in R_m(S_1)$, hence for some β and k: $\Gamma^k(\bar{s},\beta)$ and $\Gamma^k(\bar{s},\beta)=\bar{q}$. Now put $n=\pi(k,l)$, $\alpha(i)=\beta(i)$ for $1\le i\le p_k$ and $\alpha(p_k+i)=\beta'(i)$ for $i\ge1$. The proof of the fact that $\Phi^n(\bar{s},\alpha)$ and $\bar{\tau}^n(\bar{s},\alpha)=\bar{t}$ is similar to the proof of the previous proposition. Thus we established (6), which is the first half of (1). For the second half, using induction on n, we are going to prove t hat

$$\forall\bar{s}\forall\bar{t}\forall\alpha(\Phi^n(\bar{s},\alpha)\ \&\ \bar{\tau}^n(\bar{s},\alpha)=\bar{t}\Rightarrow(\bar{s},\bar{t})\in R_m(S)). \tag{7}$$

Skipping the obvious case $n=0$, take some $n=\pi(k,l)>0$ and suppose that $\Phi^n(\bar{s},\alpha)$ and $\tau^n(\bar{s},\alpha)=\bar{t}$. From here $C(\bar{s})$ and $\Gamma^k(\bar{s},\alpha)$. Put $\bar{q}=\gamma^k(\bar{s},\alpha)$. Then clearly $(\bar{s},\bar{t})\in R_m(S_1)$, therefore in order to get $(\bar{s},\bar{t})\in R_m(S)$ it is sufficient, according to (5), to get $(\bar{q},\bar{t})\in R_m(S)$. Indeed, let $\beta(i)=\alpha(i+p_k), i\ge1$. We have $\Phi^n(\bar{s},\alpha)$, in particular, $\Phi^l[\bar{x}|\bar{\gamma}^k,\bar{y}|\bar{y}+p_k](\bar{s},\alpha)$, and applying (3), we get $\Phi^l(\bar{q},\beta)$ and $\bar{t}=\tau^l[\bar{x}|\bar{\gamma}^k,\bar{y}|\bar{y}+p_k](\bar{s},\alpha)=\tau^l(\bar{q},\beta)$. Now the induction hypothesis for $l<n$ gives us the desired $(\bar{q},\bar{t})\in R_m(S)$.

Using the definition of $D_m(S)$, one can easily check that

$$\bar{s}\in D_m(S)\Leftrightarrow\neg C(\bar{s})\vee\ \bar{s}\in D_m(S_1)\ \&\ \forall\bar{q}((\bar{s},\bar{q})\in R_m(S_1)\Rightarrow\bar{q}\in D_m(S)). \tag{8}$$

Actually, $D_m(S)$ is the least set, satisfying (8), i.e. $D_m(S) = I_\psi$ for $\psi(\bar{x}, X)$: $\neg C(\bar{x}) \ \vee \ \bar{x} \in D_m(S_1) \ \& \ \forall \bar{z}((\bar{x}, \bar{z}) \in R_m(S_1) \Rightarrow \bar{z} \in X)$.

Using induction on $|\bar{s}|_\psi$, we will prove that $\bar{s} \in D_m(S) \Rightarrow \forall \alpha \exists n \Phi^n(\bar{s}, \alpha)$. The case $|\bar{s}|_\psi = 0$ is again trivial. Take $\bar{s} \in D_m(S)$ with $|\bar{s}|_\psi > 0$ and arbitrary α. According to (8), $\bar{s} \in D_m(S_1)$, hence there is a k with $\Gamma^k(\bar{s}, \alpha)$. Set $\bar{q} = \bar{\gamma}^k(\bar{s}, \alpha)$. Further, $(\bar{s}, \bar{q}) \in R_m(S_1)$ and applying again (8), we get $\bar{q} \in D_m(S)$. Clearly $|\bar{q}|_\psi < |\bar{s}_\psi|$. Put $\beta(i) = \alpha(i+p_k), i \geq 1$. By induction hypothesis for $\bar{q}$ there exists l: $\Phi^l(\bar{q}, \beta)$. Now using (3), we get $\Phi^l[\bar{x}|\bar{\gamma}^k, \bar{y}|\bar{y} + p_k](\bar{s}, \alpha)$, hence $\Phi^{\pi(k,l)}(\bar{s}, \alpha)$.

Suppose finally that $\forall \alpha \exists n \Phi^n(\bar{s}, \alpha)$ and towards contradiction, assume that $\bar{s} \notin D_m(S)$. Let us notice first that $\forall \alpha \exists n \Gamma^n(\bar{s}, \alpha)$. Indeed, take an α. Then there is an n with $\Phi^n(\bar{s}, \alpha)$. Clearly $n > 0$ (otherwise $\neg C(\bar{s})$ and we would have $\bar{s} \in D_m(S)$). Therefore $\Gamma^{l(n)}(\bar{s}, \alpha)$, so $\forall \alpha \exists n \Gamma^n(\bar{s}, \alpha)$. From here, $\bar{s} \in D_m(S_1)$. However, $\bar{s} \notin D_m(S)$, which means, according to (8), that there exists $\bar{s}^1$: $(\bar{s}, \bar{s}^1) \in R_m(S_1)$ and $\bar{s}^1 \notin D_m(S)$. Since $\{\Gamma^n \to \bar{\gamma}^n\}_n$ represents S_1, there exist n_1 and $q_1^1, \ldots, q_{p_{n_1}}^1$ such that, putting $\bar{q}^1 = (q_1^1, \ldots, q_{p_{n_1}}^1)$, we will have $\Gamma^{n_1}(\bar{s}, \bar{q}^1)$ and $\gamma^{n_1}(\bar{s}, \bar{q}^1) = \bar{s}^1$. Since $\bar{s}^1 \notin D_m(S)$, $C(\bar{s})$ holds, hence $n_1 > 0$. Let us check that, similarly to $\bar{s}$, $\forall \alpha \exists n \Phi^n(\bar{s}^1, \alpha)$. Indeed, take an arbitrary α and set $\beta(i) = q_i, 1 \leq i \leq n_1$ and $\beta(i + p_n) = \alpha(i)$ for $i \geq 1$. We know that there exists n (and $n > 0$) with $\Phi^n(\bar{s}, \beta)$. From here, $\Gamma^{l(n)}(\bar{s}, \beta)$, and since we have also $\Gamma^{n_1}(\bar{s}, \beta)$, $l(n) = n_1$. Further, $\Phi^{r(n)}[\bar{x}|\bar{\gamma}^{l(n)}, \bar{y}|\bar{y} + p_{l(n)}](\bar{s}, \beta) = \Phi^{r(n)}[\bar{x}|\bar{\gamma}^{n_1}, \bar{y}|\bar{y}+p_{n_1})](\bar{s}, \beta) = \Phi^{r(n)}(\bar{s}^1, \alpha)$, so $\forall \alpha \exists n \Phi^n(\bar{s}^1, \alpha)$ does hold. Iterating this construction, we get for every $i \geq 1$ an index $n_i > 0$ and elements $\bar{q}^i = (q_1^i, \ldots, q_{p_{n_i}}^i)$ and $\bar{s}^i$ such that (putting $\bar{s}^0 = \bar{s}$):

$$\Gamma^{n_i}(\bar{s}^{i-1}, \bar{q}^i) \ \& \ \gamma^{n_i}(\bar{s}^{i-1}, \bar{q}^i) = \bar{s}^i \ \& \ \bar{s}^i \notin D_m(S).$$

Now put $\alpha^i = q_1^i, \ldots, q_{p_{n_i}}^i, q_1^{i+1}, \ldots, q_{p_{n_{i+1}}}^{i+1}, \ldots$. So we have $\Gamma^{n_i}(\bar{s}^{i-1}, \alpha^i)$ and $\gamma^{n_i}(\bar{s}^{i-1}, \alpha^i) = \bar{s}^i$. For the sequence α^1 there exists an $N > 0$ such that $\Phi^N(\bar{s}, \alpha^1)$. Hence $\Gamma^{l(N)}(\bar{s}, \alpha^1)$ and $\Gamma^{n_1}(\bar{s}, \alpha^1)$, so $l(N) = n_1$ and $\Phi^{r(N)}[\bar{x}|\bar{\gamma}^{l(N)}, \bar{y}|\bar{y} + p_{l(N)}]$ $(\bar{s}, \alpha^1) = \Phi^{r(N)}(\bar{s}^1, \alpha^2)$ and $\bar{s}^1 \notin D_m(S)$, hence $r(N) > 0$, thus $\Gamma^{l(r(N))}(\bar{s}^1, \alpha^2)$, which combined with $\Gamma^{n_2}(\bar{s}^1, \alpha^2)$ gives us $l(r(N)) = n_2$. Repeating this argument we get that for every i, $l(r^{i-1}(N)) = n_i$ and $n_i > 0$, which is impossible. □

Now take a program P: `input(x1,...,xk);S`. Let the primitive recursive sequence $\Sigma = \{\Gamma^n \to (\gamma_1^n, \ldots, \gamma_m^n)\}_n$ represents S. We may assume that the y-variables of the clause $\Gamma^n \to \bar{\gamma}^n$ are among $y_1, \ldots, y_n$ (otherwise insert a sufficient number of dummy clauses $\neg(x_1 = x_1) \to (x_1, \ldots, x_m)$ in Σ.) Put

$$\Phi^n = \Gamma^n[x_{k+1}|x_k, \ldots, x_m|x_k, y_1|L^{n-1}(y), y_2|L^{n-2}(R(y)), \ldots, y_n|L^0(R(y))].$$

Clearly $\Gamma^n(s_1, \ldots, s_k, \ldots, s_k, \alpha) \Leftrightarrow \Phi^n(s_1, \ldots, s_k, \bar{\alpha}(n))$. From here, having in mind that Σ represents S, we conclude that the sequence $\{\Phi^n\}_n$ satisfies the statement (ii) of Theorem 1.

4 Inductive definability of the sets $L(P)$ and $D(P)$

In this section we are going to check the implications $(ii) \Rightarrow (iii)$ and (iii) $\Rightarrow (i)$ of Theorem 1. For the first one, take a primitive recursive sequence $\{\Phi^n\}_n$ of quantifier free formulas with variables among $x_1, \dots, x_k, y$. Set

$$L = \{\bar{s}|\ \exists\alpha\exists n\Phi^n(\bar{s}, \bar{\alpha}(n))\},\ D = \{\bar{s}|\ \forall\alpha\exists n\Phi^n(\bar{s}, \bar{\alpha}(n))\}.$$

Define the relation $R \subseteq (B^*)^{k+3} \times \mathcal{P}((B^*)^{k+2})$ as follows:

$$R(\bar{s}, q, n, t, A) \Leftrightarrow Nat(n)\ \&\ Seq(t)\ \&\ (\Phi^n(\bar{s}, t) \vee (s_1, \dots, s_k, n{+}1, t{*}q) \in A). \quad (9)$$

Here as usual $t * q$ stands for $\ll t_1, \dots, t_n, q \gg$, provided $t = \ll t_1, \dots, t_n \gg$. We shall write $\alpha \succ t$ to denote that α extends t, i.e. $\alpha(i) = t_i$ for $i = 1, \dots, n$. Let us first check that the relation R can be represented by a quantifier-free formula in $L_{\mathfrak{A}^+}$. Indeed, let a be an index of the primitive recursive function, that codes $\{\Phi^n\}_n$, i.e. such that for every n, $\phi(a, n)$ is the code of Φ^n. Having in mind our definition of the universal relation U, we may claim that

$$\Phi^n(s_1, \dots, s_k, t) \Leftrightarrow U(\phi(a, n), \ll s_1, \dots, s_k, t \gg).$$

Hence the right-hand side of (9) can be rewritten as $Nat(n)\ \&\ Seq(t) \& (U(\phi(a, n),$ $\ll s_1, \dots, s_k, t \gg) \vee (s_1, \dots, s_k, \langle n, O\rangle, \langle t, q\rangle) \in A)$, which means that R is definable on $\mathfrak{A}^+$ by the following formula $\varphi(x_1, \dots, x_k, y, z, t, X)$:

$$Nat(z)\&Seq(t)\&(U(\phi(a, z), \ll x_1, \dots, x_k, t \gg) \vee (x_1, \dots, x_k, \langle z, O\rangle, \langle t, y\rangle) \in X).$$

We claim that the sets L and D are sections of $I_{\exists y\varphi}$ and $I_{\forall y\varphi}$ for the above φ.

Proposition 4. $L = \{\bar{s}|(\bar{s}, O, O) \in I_{\exists y\varphi}\}, D = \{\bar{s}|(\bar{s}, O, O) \in I_{\forall y\varphi}\}$.

Proof. For the inclusion $L \supseteq \{\bar{s}|(\bar{s}, O, O) \in I_{\exists y\varphi}\}$ let us check the following more general statement (where $lh(t)$ stands for the length of t):

$$((s_1, \dots, s_k, n, t) \in I_{\exists y\varphi}\ \&\ lh(t) = n)\ \Rightarrow \exists\alpha_{\alpha\succ t}\exists m\Phi^m(\bar{s}, \bar{\alpha}(m)). \quad (10)$$

The proof is by induction on $|(\bar{s}, n, t)|_{\exists y\varphi}$. If $|(\bar{s}, n, t)|_{\exists y\varphi} = 0$, then $\Phi^n(\bar{s}, t)$ holds, hence for any $\alpha \succ t$, $\Phi^n(\bar{s}, \bar{\alpha}(n))$. When $|(\bar{s}, n, t)|_{\exists y\varphi} > 0$, there exists q such that $|(\bar{s}, n + 1, t * q)| < |(\bar{s}, n, t)|$ and by induction hypothesis $\Phi^m(\bar{s}, \bar{\alpha}(m))$ holds for some m and $\alpha \succ t * q$, hence for $\alpha \succ t$. Now putting $n = O$ and $t = \ll\gg = O$ in (10) we get $(\bar{s}, O, O) \in I_{\exists y\varphi} \Rightarrow \exists\alpha\exists n\Phi^n(\bar{s}, \bar{\alpha}(n)) \Leftrightarrow \bar{s} \in L$.

For the opposite implication, towards contradiction, assume that there exists $\bar{s} \in L$ such that $(\bar{s}, O, O) \notin I_{\exists y\varphi}$. Take α and n with $\Phi^n(\bar{s}, \bar{\alpha}(n))$ and put $q_i = \alpha(i)$. We have $(\bar{s}, O, O) \notin I_{\exists y\varphi}$, hence $\neg\Phi^0(\bar{s}, \ll\gg)$ and $\forall q(\bar{s}, 1, \ll q \gg) \notin I_{\exists y\varphi}$. From here, for $q_1 = \alpha(1)$ we get $\neg\Phi^1(\bar{s}, \ll q_1 \gg)$ and $\forall q(\bar{s}, 2, \ll q_1, q \gg) \notin I_{\exists y\varphi}$. Iterating this construction we obtain that for every k, $\neg\Phi^k(\bar{s}, \ll q_1, \dots, q_k \gg)$ – a contradiction.

The equality $D = \{\bar{s}|(\bar{s}, O, O) \in I_{\forall y\varphi}\}$ is verified in a very similar way: for the first inclusion use the implication $(\bar{s}, n, t) \in I_{\forall y\varphi}\ \&\ lh(n) = t \Rightarrow$ $\forall\alpha_{\alpha\succ t}\exists m\Phi^m(\bar{s}, \bar{\alpha}(m))$, and a contraposition – for the second one. □

Thus we have proved the implication $(ii) \Rightarrow (iii)$ of Theorem 1. It remains to establish the validity of $(iii) \Rightarrow (i)$. To do this, take an arbitrary quantifier-free formula $\varphi(x_1, \dots, x_k, y, X)$ in $\mathcal{L}_{\mathfrak{A}^+}$, in which the k-ary relational variable X occurs positively. Clearly, for the implication $(iii) \Rightarrow (i)$ it is sufficient to construct a non-deterministic program P such that $L(P) = I_{\exists y\varphi}$ and $D(P) = I_{\forall y\varphi}$. To simplify the construction, we will assume that $k = 1$, hence φ, written in disjunctive normal form, is equivalent to

$$\chi(x,y) \vee \chi_1(x,y) \& \tau_1^1 \in X \& \dots \& \tau_{j_1}^1 \in X \vee \dots \vee \chi_n(x,y) \& \tau_1^n \in X \& \dots \& \tau_{j_n}^n \in X.$$

We will consider the case $n = 2$, since it is sufficiently representative. Without essential loss of generality we may assume also that $j_1 = j_2 = 1$. Finally, we will omit the formulas χ_1 and χ_2 in φ (dropping the formula χ, however, trivializes our task). So φ takes the form

$$\chi(x,y) \ \vee \ \tau(x,y) \in X \ \vee \ \mu(x,y) \in X.$$

where τ, μ and χ are terms and a quantifier-free formula in $\mathcal{L}_{\mathfrak{A}^+}$.

Below we define a nondeterministic program Q over $\mathfrak{A}^+$, that works on strings, i.e. on those $t \in B^*$, such that $Seq(t)$.
Q : $\mathtt{input(t); q := arbitrary(B^*); z :=}$ $head(\mathtt{t})$; $\mathtt{while}\neg\chi(\mathtt{z},\mathtt{q})$ $\mathtt{do}$
$\mathtt{t} :=$ $append(tail(\mathtt{t}), \ll \tau(\mathtt{z},\mathtt{q}), \mu(\mathtt{z},\mathtt{q}) \gg)$; $\mathtt{q := arbitrary(B^*); z :=}$ $head(\mathtt{t})$) $\mathtt{od}$.

Here *head*, *tail*, and *append* are the usual string-transforming operations. It is an easy exercise to show that these operations, as well as the relations Seq, Nat, U and the universal function ϕ can be computed by means of (deterministic) while-programs over $\mathfrak{A}$. Hence the program Q may be thought of as a non-deterministic program over $\mathfrak{A}$.

Put $L = L(Q) \cap Seq$ and $D = D(Q) \cap Seq$. Clearly, for every $t = \ll t_1, \dots, t_n \gg$:

$$\ll t_1, \dots, t_n \gg \in L \Leftrightarrow \exists q(\chi(t_1, q) \vee \ \ll t_2, \dots, t_n, \tau(t_1, q), \mu(t_1, q) \gg \in L)$$

$$\ll t_1, \dots, t_n \gg \in D \Leftrightarrow \forall q(\chi(t_1, q) \vee \ll t_2, \dots, t_n, \tau(t_1, q), \mu(t_1, q) \gg \in D). \quad (11)$$

Now let P be the program $\mathtt{input(x); t :=} \ll \mathtt{x} \gg$; Q_0, where Q_0 is the body of the program Q, i.e. Q_0 is Q without its input statement. We claim that for this program P it is true that $L(P) = I_{\exists y\varphi}$ and $D(P) = I_{\forall y\varphi}$. It is clear from the construction of P, that $s \in L(P) \Leftrightarrow \ll s \gg \in L(Q) \Leftrightarrow \ll s \gg \in L$ and $s \in D(P) \Leftrightarrow \ll s \gg \in D(Q) \Leftrightarrow \ll s \gg \in D$. So, we have to prove that $I_{\exists y\varphi} = \{s | \ll s \gg \in L\}$ and $I_{\forall y\varphi} = \{s | \ll s \gg \in D\}$. Half of these equalities is a direct consequence of the next more general statement.

Proposition 5. *For every $s \in B^*$:*
(i) $s \in I_{\exists y\varphi} \Rightarrow \forall t(t = \ll t_1, \dots, s, \dots, t_n \gg \Rightarrow t \in L)$,
(ii) $s \in I_{\forall y\varphi} \Rightarrow \forall t(t = \ll t_1, \dots, s, \dots, t_n \gg \Rightarrow t \in D)$.

Proof. (i) Induction on $|s|_{\exists y\varphi}$. If it is 0, then there exists q: $\chi(s, q)$. We continue with an inner induction on $pos(s, t)$ – the position of s in $t = \ll t_1, \dots, s, \dots, t_n \gg$. If $pos(s, t) = 1$, i.e. $t = \ll s, t_2, \dots, t_n \gg$, then $t \in L$, since $\chi(s, q)$ (and, of course, since L satisfies (11)). If $pos(s,t) = i > 1$, then $pos(s,t') = i-1$ for $t' = \ll t_2 \ldots, s, \ldots, t_n, \tau(t_1,q), \mu(t_1,q) \gg$. So by induction hypothesis $t' \in L$, hence using again (11), we get that $t \in L$. Now suppose that $|s|_{\exists y\varphi} > 0$. Then there exists $q : min(|\tau(s,q)|_{\exists y\varphi}, |\mu(s,q)|_{\exists y\varphi}) < |s|_{\exists y\varphi}$. By induction hypothesis, $t'' = \ll t_2, \ldots, t_n, \tau(s,q), \mu(s,q) \gg \in L$. We will need an induction on $i = pos(s,t)$ again. Indeed, if $i = 1$, then $t = \ll s, t_2, \ldots, t_n \gg$ and since $t'' \in L$, then $t \in L$, too. If $i > 1$, then $pos(s,t') = i-1$ for $t' = \ll t_2 \ldots, s, \ldots, t_n, \tau(t_1,q), \mu(t_1,q) \gg$. So by induction hypothesis for $i-1$ we get $t' \in L$, hence $t \in L$ as well.

For the part (ii) of the proposition, proceed by induction on $|s|_{\forall y\varphi}$, using this time the fact that $|s|_{\forall y\varphi} = sup\{min(|\tau(s,q)|_{\forall y\varphi}, |\mu(s,q)|_{\forall y\varphi}) + 1 : \neg\chi(s,q)\}$. □

It remains to prove the inclusions $\{s | \ll s \gg \in L\} \subseteq I_{\exists y\varphi}$ and $\{s | \ll s \gg \in D\} \subseteq I_{\forall y\varphi}$. Again they are partial cases of the following more general proposition.

Proposition 6. *For any $t_1, \ldots, t_n$ in B^*:*
(i) $\ll t_1, \ldots, t_n \gg \in L \Rightarrow \exists i(t_i \in I_{\exists y\varphi})$,
(ii) $\ll t_1, \ldots, t_n \gg \in D \Rightarrow \exists i(t_i \in I_{\forall y\varphi})$.

Proof. We will use the observation that L and D are in fact the least sets, satisfying (11), in other words, $L = I_{\exists y\psi}$ and $D = I_{\forall y\psi}$, where $\psi(x,y,X)$ is:

$$Seq(x) \;\&\; (\chi(head(x),y) \vee append(tail(x), \ll \tau(head(x),y), \mu(head(x),y) \gg) \in X).$$

The proofs of (i) and (ii) are again very similar. Let us check this time the second one. It is by induction on $|t|_{\forall y\psi}$ for $t = \ll t_1, \ldots, t_n \gg \in D$. If $|t|_{\forall y\psi} = 0$, then $\forall q \chi(t_1,q)$ holds, hence $t_1 \in I_{\forall y\varphi}$. Assume now that $|t|_{\forall y\psi} > 0$. Hence $|t|_{\forall y\psi} = sup\{| \ll t_2 \ldots, t_n, \tau(t_1,q), \mu(t_1,q) \gg |_{\forall y\psi} + 1 : \neg\chi(t_1,q)\}$. If there exists t_i with $i \in \{2, \ldots, n\}$, such that $t_i \in I_{\forall y\varphi}$, (ii) is verified. Otherwise for every $q : \neg\chi(t_1,q)$ we must have, according to the induction hypothesis for $\ll t_2 \ldots, t_n, \tau(t_1,q), \mu(t_1,q) \gg$, that $\tau(t_1,q) \in I_{\forall y\varphi}$ or $\mu(t_1,q) \in I_{\forall y\varphi}$. This means, however, that $t_1 \in I_{\forall y\varphi}$. □

References

1. Manna, Z.: Mathematical theory of partial correctness. Lect. Notes in Math. **188** (1971) 252 – 269
2. Skordev, D.: Computability in Combinatory Spaces. Kluwer Academic Publishers, Dordrecht–Boston–London (1992)
3. Pchelarov, I.: Characterization of non-deterministic programs in abstract structures. Master's thesis, Sofia University, Dept. of Maths and Inf. (2003)
4. Moschovakis, Y. N.: Elementary induction on abstract structures. North - Holland, Amsterdam (1974)
5. Moschovakis, Y. N.: Abstract first order computability I. Trans. Amer. Math. Soc. **138** (1969) 427–464
6. Nikolova, S. K.: Π^0_1-positive inductive definability on abstract structures. Ann. Univ. Sofia **90** (1996) 91 – 108
7. Grilliot, T. G.: Inductive definitions and computability I. Trans. Amer. Math. Soc. **151** (1971) 309–317

Computable Principles in Admissible Structures

Vadim Puzarenko

Sobolev mathematical institute SB RAS
Novosibirsk, Russia
vagrig@math.nsc.ru

1 Preliminaries

This paper is devoted to a computability approach which is developed by Yu. Ershov last time. Using this approach S. Goncharov, Yu. Ershov and D. Sviridenko proposed theory of semantic programming in 1985 [1, 2]. Here the exact correspondences between computability on admissible sets and properties of enumeration ideals are established. Furthermore, one reducibility on admissible sets is introduced. Some lattice theoretical properties of this reducibility are studied. In particular, it is showed that there are minimal elements in proper sublattices under the reducibility. The properties from descriptive set theory are studied in these elements and in enumeration ideals. The technique developed here is used in description of admissible sets in which the field of reals is definable.

Let $\sigma \supseteq \{\mathrm{U}, \in, \varnothing\}$ be a fixing signature where $\mathrm{U}, \in, \varnothing$ are interpreted in models as urelements, the membership relation and the empty set respectively. As usual, $\exists y \in x\varphi$ and $\forall y \in x\varphi$ are shortenings of $\exists y((y \in x) \wedge \varphi)$ and $\forall y((y \in x) \to \varphi)$ respectively, where φ is a formula in signature σ.

Axioms of KPU_σ :
1. existence of empty set: $(\neg\mathrm{U}(\varnothing) \wedge \forall y \in \varnothing\neg(y = y))$;
2. extensionality: $\forall x\forall y((\neg\mathrm{U}(x) \wedge \neg\mathrm{U}(y)) \to (\forall z((z \in x) \leftrightarrow (z \in y)) \to (x = y)))$;
3. pair: $\forall x\forall y\exists z((x \in z) \wedge (y \in z))$;
4. union: $\forall x\exists y\forall z \in x\forall t \in z(t \in y)$;
5. foundation (for every formula $\varphi(x, z_1, \ldots, z_n)$ in σ):
$\forall z_1 \ldots \forall z_n(\exists x\varphi(x, z_1, \ldots, z_n) \to (\exists x\varphi(x, z_1, \ldots, z_n) \wedge \forall y \in x\neg\varphi(y, z_1, \ldots, z_n)))$;
6. set structure: $\forall x(\exists y(y \in x) \to \neg\mathrm{U}(x))$;
A class of Δ_0 *formulas* is the least class containing atomic formulas and closed under $\wedge$, $\vee$, $\to$, $\neg$ and $\forall y \in x$, $\exists y \in x$.
7. Δ_0 separation (for every Δ_0 formula $\varphi(x, z_1, \ldots, z_n)$ in σ which doesn't contain y free): $\forall z_1 \ldots \forall z_n\forall x\exists y((z \in y) \leftrightarrow ((z \in x) \wedge \varphi))$;
8. Δ_0 collection (for every Δ_0 formula $\varphi(x, z_1, \ldots, z_n)$ in σ which doesn't contain y free): $\forall z_1 \ldots \forall z_n(\forall z \in x\exists u\varphi \to \exists y\forall z \in x\exists u \in y\varphi)$.

Further, we will consider models in some finite signature only.

Ordinals in models of KPU are defined as in the set theory. A set a is called *transitive* iff $\forall y \in a\forall z \in y(z \in a)$. A transitive set a is called an *ordinal* if it contains transitive sets only.

An *ordinal* or a *height* of a model $\mathfrak{A}_{\mathfrak{M}}$ of KPU is the collection of all ordinals of this model (we denote this collection as $\mathrm{Ord}(\mathfrak{A}_{\mathfrak{M}})$). A model $\mathfrak{A}_{\mathfrak{M}}$ of KPU is called an *admissible set* if $\mathrm{Ord}(\mathfrak{A}_{\mathfrak{M}})$ is standard, i.e. it is a well ordered set.
Examples.

1. Every model of the theory ZF can be considered as a model of KPU.
2. Let $\mathbb{HYP}(\mathfrak{N})$ be the least under inclusion admissible set which contains a structure $\mathfrak{N}$ as its element. E.g if $\mathfrak{N}$ is the standard model of arithmetics then $\mathrm{Ord}(\mathbb{HYP}(\mathfrak{N})) = \omega_1^{CK}$. Moreover, any countable admissible ordinal greater than ω is represented as $\mathrm{Ord}(\mathbb{HYP}(\mathfrak{N}, P))$ for a proper $P \subseteq \omega$.
3. Let M be a collection of urelements equipped with a structure $\mathfrak{M}$ and let $HF(M)$ be the least set which is closed under $\cup$, $x \mapsto \{x\}$ and contains $M \cup \{\varnothing\}$ as a subset. Then $\mathbb{HF}(\mathfrak{M})$ is the least admissible set over $\mathfrak{M}$. It is called a *hereditarily finite set* over $\mathfrak{M}$. Notice that $\mathrm{Ord}(\mathbb{HF}(\mathfrak{M})) = \omega$.
4. Let D be a nonprincipal ultrafilter over ω. Then ${}^{(\mathbb{HF}(\varnothing))^{\omega}}\!\big/_{D}$ is a model of KPU which has continuum many natural ordinals and hence it is not admissible set.
5. Let D be a nonprincipal ultrafilter over ω, let S_n be an n-element set, and let $\mathbb{HF}(S_n)$ be a hereditarily finite set over S_n. Then $\prod\limits_{D} \mathbb{HF}(S_n)$ is a model of KPU which has continuum many natural ordinals, too. However, $\mathrm{Th}(\prod\limits_{D} \mathbb{HF}(S_n))$ has no admissible set.

From now on, we will denote the standard model of arithmetics by $\mathfrak{N}$. Generally we can define the hierarchy of all definable sets on arbitrary admissible set.

Σ_n and Π_n formulas have forms $\exists x_1 \forall x_2 \dots Q x_n \varphi$ and $\forall x_1 \exists x_2 \dots Q x_n \varphi$, respectively, where φ is Δ_0 in both cases.
Let $\mathbb{A}$ be an admissible set and let R be a relation on $\mathrm{dom}(\mathbb{A})$. We say that $R \in \Sigma_n^{\mathbb{A}}$ ($R \in \Pi_n^{\mathbb{A}}$) if R is definable by some $\Sigma_n (\Pi_n)$ formula in $\mathbb{A}$ (possibly with parameters). We say that $R \in \Delta_n^{\mathbb{A}}$ if $R \in \Pi_n^{\mathbb{A}} \cap \Sigma_n^{\mathbb{A}}$.
R is called $\mathbb{A}$-*computably enumerable* ($\mathbb{A}$-c.e.) if $R \in \Sigma_1^{\mathbb{A}}$. R is called $\mathbb{A}$-*computable* ($\mathbb{A}$-c.) if $R \in \Delta_1^{\mathbb{A}}$.
Notice that $\omega \subseteq \mathrm{Ord}(\mathbb{A})$ and ω is $\mathbb{A}$-c., for any admissible set $\mathbb{A}$.

Theorem 1. *Let $\mathbb{A}$ be an admissible set. Then the following holds:*

$$\begin{tikzcd}
 & \Sigma_1^{\mathbb{A}} \arrow[dr, phantom, sloped, "\subseteq"] & & \Sigma_2^{\mathbb{A}} & \dots \\
\Delta_1^{\mathbb{A}} \arrow[ur, phantom, sloped, "\subseteq"] \arrow[dr, phantom, sloped, "\subseteq"] & & \Delta_2^{\mathbb{A}} \arrow[ur, phantom, sloped, "\subseteq"] \arrow[dr, phantom, sloped, "\subseteq"] & & \\
 & \Pi_1^{\mathbb{A}} \arrow[ur, phantom, sloped, "\subseteq"] & & \Pi_2^{\mathbb{A}} & \dots
\end{tikzcd}$$

Moreover, all the inclusions are strong.

Proposition 1. *Let $\mathbb{A}$ be an admissible set. Then the following conditions hold for every $m \geqslant 1$:*

1. *for any $n \geqslant 1$, there exists a $n+1$-ary $\Sigma_m^{\mathbb{A}}$ predicate which is universal for the class of all n-ary $\Sigma_m^{\mathbb{A}}$ predicates;*
2. *for any $n \geqslant 1$, there exists a $n+1$-ary $\Pi_m^{\mathbb{A}}$ predicate which is universal for the class of all n-ary $\Pi_m^{\mathbb{A}}$ predicates;*
3. *for any $n \geqslant 1$, there is no $n+1$-ary $\Delta_m^{\mathbb{A}}$ predicate which is universal for the class of all n-ary $\Delta_m^{\mathbb{A}}$ predicates;*
4. *$R(x_1, \dots, x_n) \in \Sigma_m^{\mathbb{A}}$ iff there exists $Q(y, y_1, \dots, y_n) \in \Delta_m^{\mathbb{A}}$ such that $R = \{\langle a_1, \dots, a_n\rangle \mid \exists y\, Q(y, a_1, \dots, a_n)\}$, $n \geqslant 1$;*
5. *$R(x_1, \dots, x_n) \in \Pi_m^{\mathbb{A}}$ iff there exists $Q(y, y_1, \dots, y_n) \in \Delta_m^{\mathbb{A}}$ such that $R = \{\langle a_1, \dots, a_n\rangle \mid \forall y\, Q(y, a_1, \dots, a_n)\}$, $n \geqslant 1$.*
6. *$\Sigma_m^{\mathbb{A}}$ is closed under $\wedge$, $\vee$, $\exists x$, $\exists x \in a$ and it is not closed under $\neg$;*
7. *$\Pi_m^{\mathbb{A}}$ is closed under $\wedge$, $\vee$, $\forall x$, $\forall x \in a$ and it is not closed under $\neg$;*
8. *$\Delta_m^{\mathbb{A}}$ is closed under $\wedge$, $\vee$, $\neg$.*

Examples.

1. Let $\mathbb{A}$ be an admissible set. Then $\Sigma_1^{\mathbb{A}}$ is closed under $\forall x \in a$, by Σ reflection principle.
2. Let $\mathbb{A}$ be a hereditarily finite set. Then $\Sigma_n^{\mathbb{A}}$ is closed under $\forall x \in a$.
3. The closure of $\Sigma_2^{\mathbb{HYP}(\mathfrak{N})}$ under $\forall x \in a$, $\vee$, $\wedge$, $\exists x$ coincides with $\bigcup_n \Sigma_n^{\mathbb{HYP}(\mathfrak{N})}$.

Let $A, B \subseteq \omega$. We say that $A \leqslant_e B$ iff
$$\forall n \in \omega (n \in A \Leftrightarrow \exists u[\langle m, u\rangle \in W \& (D_u \subseteq B)])$$
for some c.e. W where D_u is the canonical numbering of finite subsets of ω.
$A \equiv_e B \leftrightharpoons (A \leqslant_e B \& B \leqslant_e A)$.
$L_e \leftrightharpoons {}^{\mathcal{P}(\omega)}\!/_{\equiv_e}$. $\langle L_e, \leqslant\rangle$ is an upper semilattice with the least element. Moreover, $d_e(A) \sqcup d_e(B) = d_e(A \oplus B)$ where
$$A \oplus B \leftrightharpoons \{2n \mid n \in A\} \cup \{2n+1 \mid n \in B\}.$$
$\mathcal{I} \subseteq L_e$ is called an *ideal* if the following conditions hold:
• $\mathbf{0} \in \mathcal{I}$;
• $\mathbf{a}, \mathbf{b} \in \mathcal{I} \Rightarrow \mathbf{a} \sqcup \mathbf{b} \in \mathcal{I}$;
• $\mathbf{b} \in \mathcal{I}, \mathbf{a} \leqslant \mathbf{b} \Rightarrow \mathbf{a} \in \mathcal{I}$.
Let $X \subseteq L_e$. Then $\langle X\rangle$ is an ideal generated by X. An ideal is called *principal* if it is generated by one element. We denote $\langle\{\mathbf{x}\}\rangle$ by $\widehat{\mathbf{x}}$.

Theorem 2. (A. Morozov, V. Puzarenko; [6]) *1. Given an admissible set $\mathbb{A}$, the collection of all $\mathbb{A}$-c.e. subsets of ω is closed under $\oplus$ and e-reducibility.*
2. Given an ideal $\mathcal{I}$, there exists an admissible set $\mathbb{A}_{\mathcal{I}}$ such that the collection of e-degrees of all $\mathbb{A}_{\mathcal{I}}$-c.e. subsets of ω is equal to $\mathcal{I}$.

Let $\mathcal{I}_e(\mathbb{A})$ be the collection of e-degrees of all $\mathbb{A}$-c.e. subsets of ω.
$\mathrm{Th}_{\Sigma}(\mathbb{A}) \leftrightharpoons \{\Phi \mid \Phi \text{ is } \Sigma_1 \text{ sentence}, \mathbb{A} \models \Phi\}$.

Proposition 2. [8] $\mathcal{I}_e(\mathbb{A}) = \langle d_e(\mathrm{Th}_{\Sigma}(\mathbb{A}, a)) \mid a \in \mathrm{dom}(\mathbb{A})\rangle$.

$\mathrm{Th}_{\exists}(\mathfrak{M}) \leftrightharpoons \{\Phi \mid \Phi \text{ is } \exists \text{ sentence}, \mathfrak{M} \models \Phi\}$.

Proposition 3. [6] $\mathcal{I}_e(\mathbb{HF}(\mathfrak{M})) = \langle d_e(\mathrm{Th}_\exists(\mathfrak{M}, a)) \mid a \in \mathrm{dom}(\mathfrak{M})^{<\omega} \rangle$.

Let $\mathcal{S}$ be a class of sets and binary relations containing $\varnothing$, ω and the universal set U, i.e. $A \subseteq U$, for every $A \in S$. We say that $\mathcal{S}$ satisfies

– the *total extension* property iff, for any partial function $\varphi \in \mathcal{S}$, there exists a total function $f \in \mathcal{S}$ such that $\varphi \subseteq f$;
– the *separation* property iff, for any disjoint sets $A_0, A_1 \in \mathcal{S}$, there exists $D \in \mathcal{S}$ such that $U \setminus D \in \mathcal{S}$, $A_0 \subseteq D$ and $A_1 \subseteq D$;
– the *existence of a universal function* iff there exists a universal function for the class of all unary functions from $\mathcal{S}$;
– the *reduction* property iff, for any $A_0, A_1 \in \mathcal{S}$, there exist disjoint sets $B_0, B_1 \in \mathcal{S}$ such that $B_0 \subseteq A_0$, $B_1 \subseteq A_1$ and $B_0 \cup B_1 = A_0 \cup A_1$;
– the *uniformization* property iff, for every binary relation $R \in \mathcal{S}$, there exists a function $\rho \in \mathcal{S}$ such that $\rho \subseteq R$ and $\delta\rho = \mathrm{Pr}_1(R)$;
– the *enumeration (via ω)* property iff, for any non-empty $A \in \mathcal{S}$, there exists a function $\psi \in \mathcal{S}$ which acts from ω onto A;
– the *quasi-projectibility (in ω)* property iff, for any $A \in \mathcal{S}$, there exists a partial function $\psi \in \mathcal{S}$ which acts from some subset $D \subseteq \omega$ onto A.

Here we denote the domain of φ and the projection on the first coordinate of R by $\delta\varphi$ and $\mathrm{Pr}_1(R)$, respectively.

We will say that an admissible set $\mathbb{A}$ satisfies a property P iff $\Sigma_1^{\mathbb{A}}$ satisfies this property. Let $\mathcal{J}$ be an enumeration ideal and let P be one of properties considered above except of existence of a universal function. Then we will say that $\mathcal{J}$ satisfies P iff $\bigcup \mathcal{J}$ satisfies this property. Notice that an ideal $\mathcal{J}$ satisfies the enumeration property iff it is generated by total degrees. If an admissible set $\mathbb{A}$ has the height ω and satisfies the enumeration property then it is recursively listed.

2 Reducibilities on Admissible Sets

Let $\mathbb{A}$ be an admissible set and let S be arbitrary set. A map $\nu : \mathrm{dom}(\mathbb{A}) \twoheadrightarrow S$ (onto S) is called $\mathbb{A}$-*numbering* of S.

Let $S \subseteq \mathcal{P}(\mathrm{dom}(\mathbb{A}))$. $\mathbb{A}$-numbering ν is called *computable* if $\Gamma_\nu^* \leftrightharpoons \{\langle x, y\rangle \in \mathrm{dom}(\mathbb{A})^2 \mid y \in \nu(x)\} \in \Sigma_1^{\mathbb{A}}$.

$\mathbb{A}$-numbering ν is called *definable* if $\Gamma_\nu^* \in \bigcup_n \Sigma_n^{\mathbb{A}}$.

We will denote the class of all computable in $\mathbb{A}$ collections (which have some computable $\mathbb{A}$-numbering) of subsets of ω as $\mathcal{S}_\omega(\mathbb{A})$.

(1985; Yu. Ershov) Let $\mathfrak{M} = \langle M, Q_1, \ldots, Q_k\rangle$ be an arbitrary model in some finite relation language. We say that $\mathfrak{M}$ is Σ *definable* in an admissible set $\mathbb{A}$ (and we denote it as $\mathfrak{M} \leqslant_\Sigma \mathbb{A}$) if there exists an $\mathbb{A}$-numbering ν of M such that $\nu^{-1}(=)$, $\nu^{-1}(Q_1)$, …, $\nu^{-1}(Q_k)$ are $\mathbb{A}$-c.

We say that $\mathfrak{M}$ is *definable* in $\mathbb{A}$ (and we denote it as $\mathfrak{M} \leqslant \mathbb{A}$) if there exists an $\mathbb{A}$-numbering ν of M such that $\nu^{-1}(=)$, $\nu^{-1}(Q_1)$, …, $\nu^{-1}(Q_k)$ are in $\bigcup_n \Sigma_n^{\mathbb{A}}$.

(2002; A. Morozov) We say that an admissible set $\mathbb{A}$ is Σ*-reducible* to an admissible set $\mathbb{B}$ (and we denote it as $\mathbb{A} \sqsubseteq_\Sigma \mathbb{B}$) if there exists a $\mathbb{B}$-numbering ν of

dom($\mathbb{A}$) such that $\{\langle x, y\rangle \mid \nu(x) = \{\nu(z) \mid z \in y\}\}$ is $\mathbb{B}$-c., as it was in definition of $\mathbb{A} \leqslant_\Sigma \mathbb{B}$
(2004) We say that an admissible set $\mathbb{A}$ is *enumerably reducible* to an admissible set $\mathbb{B}$ (and we denote it as $\mathbb{A} \sqsubseteq_E \mathbb{B}$) if there exists a $\mathbb{B}$-numbering ν of dom($\mathbb{A}$) such that $\nu^{-1}(C)$ is $\mathbb{B}$-c.e., for every binary $\mathbb{A}$-c.e. predicate C.
We say that p-reducibility ($\leqslant_p \in \{\leqslant_\Sigma, \sqsubseteq_\Sigma, \sqsubseteq_E\}$) on admissible sets *preserves* P if $\mathbb{A} \leqslant_p \mathbb{B}$ implies $P(\mathbb{A}) \subseteq P(\mathbb{B})$.

	$\mathcal{I}_e$	$\mathcal{S}_\omega$	Ord	Set structure
$\leqslant_\Sigma$	$-$	$-$	$-$	$-$
$\sqsubseteq_E$	$+$	$+$	$-$	$-$
$\sqsubseteq_\Sigma$	$+$	$+$	$+$	$+$

Examples.

1. $\mathbb{HYP}(\mathfrak{N}) \leqslant_\Sigma \mathbb{HF}(\mathbb{HYP}(\mathfrak{N}))$. However, $\Pi^1_1 = \mathcal{I}_e(\mathbb{HYP}(\mathfrak{N})) \nleqslant \mathcal{I}_e(\mathbb{HF}(\mathbb{HYP}(\mathfrak{N}))) = \Delta^1_1$.
2. $\mathbb{HYP}(S) \sqsubseteq_E \mathbb{HF}(S)$, where S is some infinite model in the empty signature.

Theorem 3. (Yu. Ershov [3]) $\mathfrak{M} \leqslant_\Sigma \mathbb{A} \Leftrightarrow \mathbb{HF}(\mathfrak{M}) \leqslant_\Sigma \mathbb{A} \Leftrightarrow \mathbb{HF}(\mathfrak{M}) \sqsubseteq_\Sigma \mathbb{A}$.

Proposition 4. $\mathbb{HF}(\varnothing) \sqsubseteq_E \mathbb{A}$*, for every admissible set* $\mathbb{A}$.

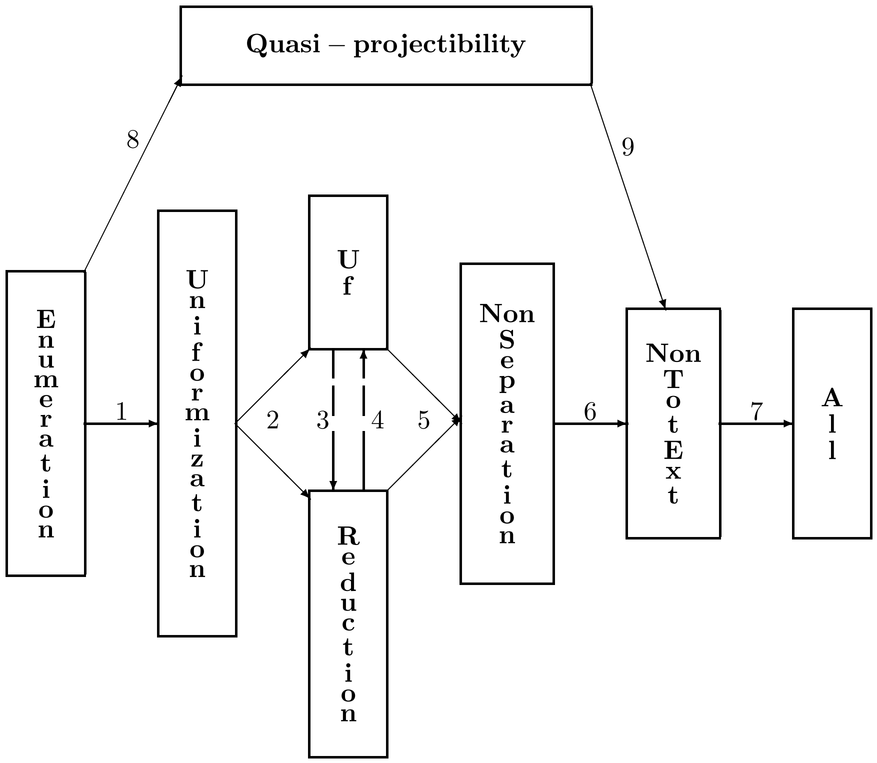
Proposition 5. *Let* $\mathbb{A}$ *and* $\mathbb{B}$ *be admissible sets and* ν *be a map such that* $\nu : \mathbb{A} \sqsubseteq_E \mathbb{B}$*. Then* $\nu^{-1}(\Sigma^{\mathbb{A}}_m) \subseteq \Sigma^{\mathbb{B}}_m$, $\nu^{-1}(\Pi^{\mathbb{A}}_m) \subseteq \Pi^{\mathbb{B}}_m$, $\nu^{-1}(\Delta^{\mathbb{A}}_m) \subseteq \Delta^{\mathbb{B}}_m$*, for any* $m \geqslant 1$.

• Let $\mathbb{A}$ be an admissible set. An admissible set $J(\mathbb{A})$ is called a *jump* if there are surjective maps $\nu_0 : \text{dom}(\mathbb{A}) \twoheadrightarrow \text{dom}(J(\mathbb{A}))$ and $\nu_1 : \text{dom}(J(\mathbb{A})) \twoheadrightarrow \text{dom}(\mathbb{A})$ such that $\nu_0^{-1}(\Sigma_1^{J(\mathbb{A})}) \subseteq \Sigma_2^{\mathbb{A}}, \quad \nu_1^{-1}(\Sigma_2^{\mathbb{A}}) \subseteq \Sigma_1^{J(\mathbb{A})}$.
Given an admissible set $\mathbb{A}$, there exists a jump $J(\mathbb{A})$ which is unique up to E-equivalence.

Proposition 6. *Let* $\mathbb{A}$, $\mathbb{B}$ *be admissible sets and let* $\mathfrak{M}$ *be a model in some finite signature. Then the following conditions hold:*
1. $\mathbb{A} \sqsubseteq_E J(\mathbb{A})$;
2. $\mathbb{A} \sqsubseteq_E \mathbb{B} \Rightarrow J(\mathbb{A}) \sqsubseteq_E J(\mathbb{B})$;
3. $\mathfrak{M} \leqslant \mathbb{A}$ *iff* $\mathfrak{M} \leqslant_\Sigma J^n(\mathbb{A})$ *for some* $n \in \omega$.

3 Computable Principles on Admissible Sets

Theorem 4. *We have the following correspondences between principles on admissible sets:*



Moreover, all the transitions are strong for exception maybe of 3.

4 Construction

Theorem 5. [8] *Let $\mathbb{A}$ be an admissible set and let P be one of the following properties: uniformization, reduction, separation, total extension, existence of a universal function, enumerability via ω, quasi-projectibility in ω. Then there exists a model $\mathfrak{M}_{\mathbb{A}}$ in the signature $\{Q^4\}$ such that the following conditions hold:*
1. $\mathbb{A} \equiv_E \mathbb{HF}(\mathfrak{M}_{\mathbb{A}})$;
2. $P(\mathbb{A}) \Leftrightarrow P(\mathbb{HF}(\mathfrak{M}_{\mathbb{A}}))$.

Example. $\mathbb{HYP}(\mathfrak{N})$ satisfies quasi-projectibility in ω, uniformization, reduction, existence of a universal Σ function and does not satisfy other principles. In particular, it is not enumerable via ω, nevertheless, it is recursively listed. By theorem 2, there exists a hereditarily finite set satisfying the same properties and being E-equivalent to $\mathbb{HYP}(\mathfrak{N})$.

5 Semilattices under E-Reducibility

Let $D_E(\mathbb{A})$ be the class of all admissible sets which are E-equivalent to $\mathbb{A}$.
$\mathcal{R}_\alpha = \{D_E(\mathbb{A}) \mid \mathbb{A} \text{ is admissible, card}(\mathbb{A}) = \alpha\}$,

$\mathcal{R}_{\leqslant\alpha} = \{D_E(\mathbb{A}) \mid \mathbb{A} \text{ is admissible }, \mathrm{card}(\mathbb{A}) \leqslant \alpha\}$,
$\mathcal{R}_{\alpha,\mathcal{I}} = \{D_E(\mathbb{A}) \mid \mathbb{A} \text{ is admissible}, \mathrm{card}(\mathbb{A}) = \alpha, \mathcal{I}_e(\mathbb{A}) = \mathcal{I}\}$,
$\mathcal{R}_{\leqslant\alpha,\mathcal{I}} = \{D_E(\mathbb{A}) \mid \mathbb{A} \text{ is admissible}, \mathrm{card}(\mathbb{A}) \leqslant \alpha, \mathcal{I}_e(\mathbb{A}) = \mathcal{I}\}$,
where $\mathcal{I}$ is an ideal of e-degrees and α is an infinite cardinal.
An upper semilattice $\langle L, \leqslant\rangle$ is called *distributive* if $x \leqslant y \sqcup z \Rightarrow \exists y_0 \leqslant y \exists z_0 \leqslant z[x = y_0 \sqcup z_0]$ for any $x, y, z \in L$.

Theorem 6. *Let $\mathcal{I}$ be an e-ideal and α be an infinite cardinal such that $\alpha \geqslant \mathrm{card}(\mathcal{I})$. Then the following conditions hold:*

1. *$\langle\mathcal{R}_\alpha, \sqsubseteq_E\rangle$, $\langle\mathcal{R}_{\leqslant\alpha}, \sqsubseteq_E\rangle$, $\langle\mathcal{R}_{\leqslant\alpha,\mathcal{I}}, \sqsubseteq_E\rangle$, $\langle\mathcal{R}_{\alpha,\mathcal{I}}, \sqsubseteq_E\rangle$ are upper semilattices of cardinality not greater than 2^α;*
2. *if $\mathcal{I}$ is at most countable ideal then $\mathrm{card}(\mathcal{R}_{\omega,\mathcal{I}}) = 2^\omega$ and hence $\mathrm{card}(\mathcal{R}_\omega) = 2^\omega$;*
3. *$\langle\mathcal{R}_\alpha, \sqsubseteq_E\rangle$, $\langle\mathcal{R}_{\leqslant\alpha}, \sqsubseteq_E\rangle$ and $\langle\mathcal{R}_{\alpha,\mathcal{I}}, \sqsubseteq_E\rangle$ have the least elements;*
4. *$\langle\mathcal{R}_{\leqslant\alpha,\mathcal{I}}, \sqsubseteq_E\rangle$ has the least element iff $\mathcal{I}$ is a principal ideal or $\alpha = \omega$;*
5. *if β is a cardinal such that $\alpha < \beta$ then there exist embeddings*
   $\langle\mathcal{R}_{\leqslant\alpha}, \sqsubseteq_E, \sqcup, \mathbf{0}\rangle \hookrightarrow \langle\mathcal{R}_{\leqslant\beta}, \sqsubseteq_E, \sqcup, \mathbf{0}\rangle$,
   $\langle\mathcal{R}_\alpha, \sqsubseteq_E, \sqcup, \mathbf{0}\rangle \hookrightarrow \langle\mathcal{R}_\beta, \sqsubseteq_E, \sqcup, \mathbf{0}\rangle$,
   $\langle\mathcal{R}_{\leqslant\alpha,\mathcal{I}}, \sqsubseteq_E, \sqcup\rangle \hookrightarrow \langle\mathcal{R}_{\leqslant\beta,\mathcal{I}}, \sqsubseteq_E, \sqcup\rangle$ *and*
   $\langle\mathcal{R}_{\alpha,\mathcal{I}}, \sqsubseteq_E, \sqcup, \mathbf{0}\rangle \hookrightarrow \langle\mathcal{R}_{\beta,\mathcal{I}}, \sqsubseteq_E, \sqcup, \mathbf{0}\rangle$;
6. *$\langle L_e, \leqslant, \sqcup, \backslash, \mathbf{0}\rangle \hookrightarrow \langle\mathcal{R}_\omega, \sqsubseteq_E, \sqcup, J, \mathbf{0}\rangle$ and hence $\langle\mathcal{R}_{\leqslant\alpha}, \sqsubseteq_E\rangle$ and $\langle\mathcal{R}_\alpha, \sqsubseteq_E\rangle$ are not distributive;*
7. *if $\alpha > \omega$ then $\langle\mathcal{R}_{\leqslant\alpha}, \sqsubseteq_E\rangle$ is not a lattice; furthermore, if $\mathcal{I}$ is nonprincipal then $\langle\mathcal{R}_{\leqslant\alpha,\mathcal{I}}, \sqsubseteq_E\rangle$ is not a lattice, too.*

Let $\mathbf{0}_{\alpha,\mathcal{I}}$ be the least element of $\langle\mathcal{R}_{\alpha,\mathcal{I}}, \sqsubseteq_E\rangle$, where $\mathcal{I}$ is an ideal and α is an infinite cardinal with $\alpha \geqslant \mathrm{card}(\mathcal{I})$.

6 Minimal Elements: General Case

Let $\mathcal{I}$ and α be an ideal and an infinite cardinal such that $\alpha \geqslant \mathrm{card}(\mathcal{I})$. Construct a model $\mathfrak{M}_{\alpha,\mathcal{I}}$ in the signature $\{P^2\}$ as a disjunct union of trees of kind $\mathcal{D}_S$, $S \in \bigcup\mathcal{I}$ (we take exactly α trees of kind $\mathcal{D}_S$, for every such S):
• every vertex has either one or two immediate successors;
• a vertex x has two immediate successors iff $\mathrm{height}(x) \in S$;

$$\bullet P(x,y) \Leftrightarrow \begin{cases} x \text{ is a root}\&(x = y); \\ x \text{ is not a root}\&(x \text{ is an immediate} \\ \text{successor of } y). \end{cases}$$

Then $\mathbb{HF}(\mathfrak{M}_{\alpha,\mathcal{I}}) \in \mathbf{0}_{\alpha,\mathcal{I}}$.

Theorem 7. [6] *Let $\mathbb{A}$ be an admissible set such that $\mathbb{A} \in \mathbf{0}_{\alpha,\mathcal{I}}$. Then $S \subseteq \mathcal{P}(\omega)$ is computable in $\mathbb{A}$ iff $S \cup \{\varnothing\} = \{\Theta_n(R, A) \mid n \in \omega, d_e(R) \in \mathcal{I}\}$ for some computable sequence of enumeration operators $\{\Theta_n\}_{n\in\omega}$ and $A \subseteq \omega$ with $d_e(A) \in \mathcal{I}$.*

We say that $\mathcal{I}$ satisfies the property of universal function existence if collection of all the graphs of functions f with $d_e(\Gamma_f) \in \mathcal{I}$ is computable in admissible sets from $\mathbf{0}_{\alpha,\mathcal{I}}$.

Theorem 8. [8] *Let P be one of the following properties: existence of a universal function, separation, total extension. Then there exists an admissible set $\mathbb{A} \in \mathbf{0}_{\alpha,\mathcal{I}}$ such that $P(\mathbb{A}) \Leftrightarrow P(\mathcal{I})$.*

7 Minimal Elements: Case of Non-principal Ideal

Let $\mathcal{I}$ be a nonprincipal ideal and α be an infinite cardinal such that $\alpha \geqslant \operatorname{card}(\mathcal{I})$.

Theorem 9. *(2004) Let $\mathbb{A}$ be an admissible set such that $\mathbb{A} \in \mathbf{0}_{\alpha,\mathcal{I}}$. Then the reduction principle does not hold in $\mathbb{A}$.*

Let $\mathbb{A}$ be an admissible set and let ν be an $\mathbb{A}$-numbering of S. ν is called *positive* if $\eta_\nu \rightleftharpoons \{\langle x, y\rangle \in \operatorname{dom}(\mathbb{A})^2 \mid \nu(x) = \nu(y)\}$ is $\mathbb{A}$-c.e. ν is called *decidable* if η_ν is $\mathbb{A}$-c. ν is called *single-valued* if there is a total $\mathbb{A}$-c. function f such that $\eta_\nu = \{\langle x, y\rangle \mid f(x) = f(y)\}$. ν is called *definable* if $\eta_\nu \in \bigcup_n \Sigma_n^{\mathbb{A}}$.

Theorem 10. [8] *Let $\mathbb{A}$ be an admissible set such that $\mathbb{A} \in \mathbf{0}_{\alpha,\mathcal{I}}$. Then there is no positive computable $\mathbb{A}$-numbering of the collection of all $\mathbb{A}$-c.e. sets.*

Theorem 11. [4] *1. There exists an admissible set $\mathbb{A}$ such that there is a single-valued computable $\mathbb{A}$-numbering of the collection of all $\mathbb{A}$-c.e. sets.*
2. There exists an admissible set $\mathbb{A}$ such that there is a decidable computable $\mathbb{A}$-numbering of the collection of all $\mathbb{A}$-c.e. sets but there is no single-valued computable $\mathbb{A}$-numbering of this collection.

8 Minimal Elements: Case of Principal Ideal

Let $C \subseteq \omega$. We construct a model $\mathfrak{N}_C$ in the signature $\{0, s^2, f^2\}$ in the following way:
$\operatorname{dom}(\mathfrak{N}_C) = \omega \uplus \{z_n \mid n \in C\}$, where $z_n \neq z_{n'}$ when $n \neq n'$;
$0 \in \omega$ is the least element of ω and s is the successor function on ω;
f maps n to z_n when $n \in C$. Notice that $\mathbb{HF}(\mathfrak{N}_C) \in \mathbf{0}_{\omega,\mathcal{I}}$ if $\mathcal{I} = \widehat{\mathrm{d}_e(C)}$.

Proposition 7. *Let $B, C \subseteq \omega$. Then $B \leqslant_e C$ iff $\mathbb{HF}(\mathfrak{N}_B) \sqsubseteq_E \mathbb{HF}(\mathfrak{N}_C)$.*

Proposition 8. *Let $C \subseteq \omega$. Then $\mathbb{HF}(\mathfrak{N}_{J_e(C)}) \equiv_E J(\mathbb{HF}(\mathfrak{N}_C))$.*

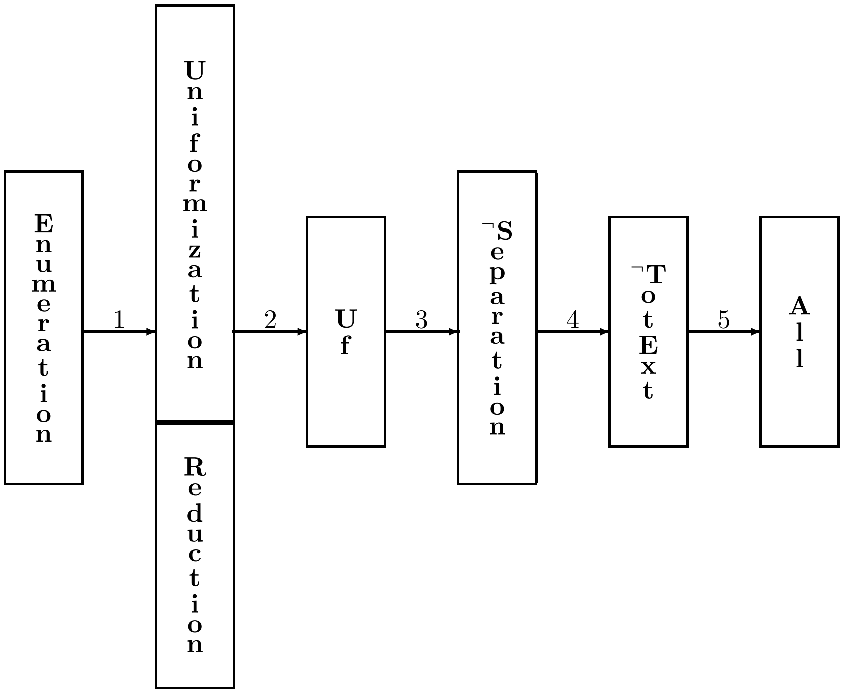
Proposition 9. *Let $\mathbb{A}$ be an admissible set which is quasi-projectible in ω. Then the following conditions hold:*

1. *$\mathcal{I}_e(\mathbb{A})$ is a principal ideal.*
2. *If $C \subseteq \omega$ and $\mathcal{I}_e(\mathbb{A}) = \widehat{\mathrm{d}_e(C)}$ then $\mathbb{A} \equiv_E \mathbb{HF}(\mathfrak{N}_C)$. In particular, $\mathbb{A} \in \mathbf{0}_{\omega,\mathcal{I}_e(\mathbb{A})}$.*

3. $\mathbb{A}$ does not satisfy the total extension property.

Theorem 12. [7] *Let $C \subseteq \omega$ and let P be one of the following properties: enumerability via ω, uniformization, reduction, existence of a universal function, separation. Then $P(\mathbb{HF}(\mathfrak{N}_C)) \Leftrightarrow P(\widehat{\mathrm{d}_e(C)})$.*

Theorem 13. (I. Kalimullin, V. Puzarenko; [7]) *We have the following correspondences between the principal ideals:*



9 Some Applications

This approach allows to give a natural description of all the admissible sets in which the field of reals is definable.

Theorem 14. [5] *Let $\mathbb{R}$ be a field of reals and let $\mathbb{A}$ be an admissible set. Then $\mathbb{R}$ is definable in $\mathbb{A}$ iff $\mathcal{P}(\omega)$ is definable in $\mathbb{A}$, i.e. there is a definable $\mathbb{A}$-numbering of $\mathcal{P}(\omega)$.*

As corollary, we can prove the following

Theorem 15. (Yu. Ershov [3]) *Let* T *be one of the following theories: infinite sets without structures, dense linear order, algebraically closed fields. Then $\mathbb{R}$ is not definable in hereditarily finite sets $\mathbb{HF}(\mathfrak{M})$ over $\mathfrak{M} \models \mathrm{T}$.*

Theorem 16. [5] *Let $\mathbb{R}$ be a field of reals and let $\mathbb{A}$ be an admissible set with $\mathrm{Ord}(\mathbb{A}) > \omega$. Then $\mathbb{R}$ is Σ-definable in $\mathbb{A}$ iff $\mathcal{P}(\omega) \subseteq \mathrm{dom}(\mathbb{A})$.*

Theorem 17. [8] *There exists an admissible set $\mathbb{A}$ such that $\mathbb{R} \leqslant \mathbb{A}$ but $\mathbb{R} \nleqslant_{\Sigma} \mathbb{A}$.*

References

1. Goncharov, S.S.; Ershov, Yu.L.; Sviridenko, D.I. Semantic programming. (English) Information processing, Proc. IFIP 10th World Comput. Congr., Dublin/Irel. 1986, IFIP Congr. Ser. 10, 1113-1120 (1986).
2. Goncharov, S.S.; Sviridenko, D.I. Σ-programming. (Russian, English) Transl., Ser. 2, Am. Math. Soc. 142, 101-121 (1989); translation from Vychisl. Sist. 107, 3-29 (1985).
3. Ershov, Yu.L. Definability and computability. Transl. from the Russian. (English) Siberian School of Algebra and Logic. New York, NY: Consultants Bureau (1996).
4. Puzarenko V. Decidable computable $\mathbb{A}$-numberings//Algebra and logic, **41**, No. 5(2002).
5. Puzarenko V. General numberings and definability of $\mathbb{R}$ in admissible sets// Vestnik NGU, Ser.: mathem., mechanics, inform., **3**, No. 2(2003), 107–117 (in russian).
6. Morozov A., Puzarenko V. Σ-subsets of naturals// Algebra and logic, **43**, No. 3(2004), 162–178.
7. Kalimullin I., Puzarenko V. Computable principles on admissible sets// Matematicheskie trudy, 7, No. 2(2004), 35–71 (in Russian).
8. Puzarenko V. On computability on special models// Siberian mathematical journal, 46, No. 1(2005).

A computability notion for locally finite lattices

Dimiter Skordev

Sofia University, Faculty of Mathematics and Informatics, Sofia, Bulgaria,
skordev@fmi.uni-sofia.bg,
http://www.fmi.uni-sofia.bg/fmi/logic/skordev/

Abstract. A rather restricted kind of computability called calculability is considered on locally finite lattices. Besides arbitrary given functions, its definition uses as an initial function also a four-argument one whose value equals the third or the fourth argument depending on whether the first argument is dominated by the second one. New functions are constructed by means of substitution and of supremum and infimum attained when one of the arguments ranges between varying limits. In one of the examples the calculable functions have a certain relation to the Bounded Arithmetic of S. R. Buss, and in another one they coincide with the so-called elementary definable functions considered by the present author in 1967. Under some assumptions that are satisfied in these two examples, the calculable functions on the natural numbers are characterized through definability by formulas with bounded quantifiers and domination by termal functions.

1 Introduction

The study of restricted kinds of computability is often used for achieving a better applicability of the theory of computability to problems about complexity of computations. In the present paper one such restricted kind called calculability will be examined. It is a sort of relative computability that makes sense not only on the natural numbers, but on any non-degenerate locally finite lattice. An example to the definition of calculability will be shown to have a certain relation to the Bounded Arithmetic introduced in [1].

2 The main definition and some examples to it

Let L be a non-degenerate lattice that is locally finite, i.e. $\{y \in L \mid x \leq y \leq z\}$ is a finite set for any x, z in L. For instance, L could consist of all finite subsets of some infinite set and be partially ordered by the inclusion relation, or could be the set of the positive integers partially ordered by the divisibility relation. In fact we are interested mainly in the case when L is some subset of the set $\mathbb{Z}$ of the integers and L is linearly ordered in the usual way (the most important case will be the one when L is the set $\mathbb{N}$ of the natural numbers, but there are also other cases that deserve some interest – for example the case of $L = \mathbb{Z}$ and the case of $L = \{0, 1\}$).

Each function considered in this paper will be, except explicitly said something else, a total function of some non-zero number of arguments in the set L, i.e. a mapping of L^n into L for some positive integer n. We shall denote by $\mathcal{P}$ the set of the projection functions, i.e. of the functions $\lambda x_1 \ldots x_n .\, x_i$ with $1 \leq i \leq n$. The function $\lambda xyst.\,(\text{if } x \leq y \text{ then } s \text{ else } t)$ will be denoted by δ_L. Whenever f is an m-ary function, and $g_1, \ldots, g_m$ are n-ary functions, the function

$$\lambda x_1 \ldots x_n .\, f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))$$

will be called their *composition*. For any non-negative integer n and any $(n+1)$-ary function f one can consider the two $(n+2)$-ary functions

$$\lambda x_1 \ldots x_n uv.\ \bigvee_{y \in [u..v]_L} f(x_1, \ldots, x_n, y)\,, \quad \lambda x_1 \ldots x_n uv.\ \bigwedge_{y \in [u..v]_L} f(x_1, \ldots, x_n, y)\,,$$

where $[u..v]_L = \{y \in L \,|\, u \wedge v \leq y \leq u \vee v\}$ (the subscript L will be usually omitted). These two functions will be denoted by f^+ and f^-, respectively.

Let $\mathcal{A}$ be a set of functions. The functions *termal in* $\mathcal{A}$ form the smallest class of functions containing $\mathcal{A} \cup \mathcal{P}$ and closed under compositions. The functions that are termal in $\mathcal{A} \cup \{\delta_L\}$ will be called *piecewise termal in* $\mathcal{A}$. Our main subject will be the smallest class of functions containing $\mathcal{A} \cup \{\delta_L\} \cup \mathcal{P}$ that is closed under compositions and contains f^+ and f^- for any f in it. The functions belonging to this class will be called *calculable in* $\mathcal{A}$ or $\mathcal{A}$*-calculable*, for short. Obviously, all functions termal in $\mathcal{A}$ are piecewise termal in $\mathcal{A}$, and all functions piecewise termal in $\mathcal{A}$ are $\mathcal{A}$-calculable. (The converse statements are not true in the general case.)

Remark 1. Each $\mathcal{A}$-calculable function is absolutely search computable (in the sense of [2]) in $\lambda xy.\ x \vee y$, $\lambda xy.\, x \wedge y$, $\lambda xz.\,\text{card}\{y \in L \,|\, x \leq y \leq z\}$ and the functions from some finite subset of $\mathcal{A}$.

Remark 2. For any positive integer n the functions $\lambda x_1 \ldots x_n .\, x_1 \vee \ldots \vee x_n$ and $\lambda x_1 \ldots x_n .\, x_1 \wedge \ldots \wedge x_n$ are $\mathcal{A}$-calculable, and they are even piecewise termal in $\mathcal{A}$ if L is linearly ordered. This statement is trivial for $n = 1$. For $n = 2$ one can use the equalities

$$u \vee v = \bigvee_{y \in [u..v]} y\,, \quad u \wedge v = \bigwedge_{y \in [u..v]} y\,,$$

in the general case and the equalities

$$u \vee v = \delta_L(u, v, v, u)\,, \quad u \wedge v = \delta_L(u, v, u, v)$$

in the case of linearly ordered L. The general statement can be proved by induction.

The following denotations will be used sometimes in the sequel:

$$f_a^+ = \lambda x_1 \ldots x_n v.\ f^+(x_1, \ldots, x_n, a, v)\,,$$
$$f_{a,b}^+ = \lambda x_1 \ldots x_n .\ f^+(x_1, \ldots, x_n, a, b)$$

for any natural number n, any $(n+1)$-ary function f and any a, b in L.

Remark 3. Suppose L is linearly ordered, there is a least element a in L, and the constant a regarded as a one-argument function is piecewise termal in $\mathcal{A}$. Then the class of the $\mathcal{A}$-calculable functions is the smallest class of functions containing $\mathcal{A} \cup \{\delta_L\} \cup \mathcal{P}$ that is closed under compositions and contains f_a^+ for any f in it. To prove this, it is sufficient to prove the following two statements:

(i) For any $\mathcal{A}$-calculable function f the function f_a^+ is also $\mathcal{A}$-calculable.
(ii) Whenever $\mathcal{C}$ is a class of functions with the properties formulated above, and f is a function belonging to $\mathcal{C}$, the functions f^+ and f^- also belong to $\mathcal{C}$.

The statement (i) is obvious, since f_a^+ is obtained from f^+ by means of a substitution of a into it. As to the statement (ii), it follows (assuming f is $(n+1)$-ary) from the equalities

$$\begin{aligned} f^+(\bar{x},u,v) &= \delta_L(u,v,g(\bar{x},u,v),g(\bar{x},v,u))\,, \\ f^-(\bar{x},u,v) &= h_a^+(\bar{x},u,v,f(\bar{x},u))\,, \end{aligned}$$

where $\bar{x}$ stands for $x_1,\ldots,x_n$,

$$g(\bar{x},s,t) = \bigvee_{y\in[a..t]} \delta_L(y,s,f(\bar{x},s),f(\bar{x},y))\,,$$

and h is defined by consecutively setting

$$\begin{aligned} l(\bar{x},u,v,z) &= \bigvee_{y\in[u..v]} \delta_L(z,f(\bar{x},y),a,z)\,, \\ h(\bar{x},u,v,z) &= \delta_L(l(\bar{x},u,v,z),a,z,a) \end{aligned}$$

(*comment*: for any given $\bar{x},u,v$ the inequality $l(\bar{x},u,v,z) \le a$ is equivalent to the equality $l(\bar{x},u,v,z) = a$, and it is satisfied by an element z of L if and only if $z \le f(\bar{x},y)$ for all y in $[u..v]$, therefore $\{\, h(\bar{x},u,v,z) \,|\, z \in [a..f(\bar{x},u)]\}$ is the set of all such z).

Remark 4. Suppose L is linearly ordered, there are a least element a and a greatest element b in L, and the constants a and b regarded as one-argument functions are piecewise termal in $\mathcal{A}$. Then the class of the $\mathcal{A}$-calculable functions is the smallest class of functions containing $\mathcal{A} \cup \{\delta_L\} \cup \mathcal{P}$ that is closed under compositions and contains $f_{a,b}^+$ for any f in it. Indeed, $f_{a,b}^+$ is $\mathcal{A}$-calculable for any $\mathcal{A}$-calculable function f, and, on the other hand, if f is an $(n+1)$-ary function for some non-negative integer n, then $f_a^+ = g_{a,b}^+$, where g is defined by means of the equality

$$g(\bar{x},v,y) = \delta_L(y,v,f(\bar{x},y),f(\bar{x},v))$$

($\bar{x}$ stands again for $x_1,\ldots,x_n$), hence any class of functions with the properties formulated in the present remark has also the properties formulated in Remark 3.

Remark 5. If L is linearly ordered, there are a least and a greatest element in L, and all constants regarded as one-argument functions are piecewise termal in $\mathcal{A}$, then all $\mathcal{A}$-calculable functions are piecewise termal in $\mathcal{A}$. This can be seen by using Remarks 2 and 4.

Remark 6. All $\mathcal{A}$-calculable functions are piecewise termal in $\mathcal{A}$ also in the case when L has exactly two elements (hence L is linearly ordered). Indeed, then $[u..v] = \{u, v\}$, therefore

$$f^+(\bar{x}, u, v) \ = \ f(\bar{x}, u) \vee f(\bar{x}, v)\,, \quad f^-(\bar{x}, u, v) \ = \ f(\bar{x}, u) \wedge f(\bar{x}, v)$$

for all $\bar{x}, u, v$, and we may use Remark 2.

Now we shall consider some examples to the definition of calculability. In each of them except for the last one, L will be a subset of $\mathbb{Z}$, and the usual linear ordering will be tacitly assumed to be the partial order on L (according to Remark 6 the examples with $L = \{0, 1\}$ concern in fact the ordinary theory of Boolean functions).

Example 1. Let $L = \mathbb{N}$, $\mathcal{A} = \{\lambda x.\, 0,\ \lambda x.\, 1,\ \lambda xy.\, x + y,\ \lambda xy.\, x \cdot y\}$. Then the $\mathcal{A}$-calculable functions will be shown to coincide with the class of functions considered in [4] and called elementary definable there.[1] By its definition (slightly paraphrased), the class of the elementary definable functions is the smallest class of functions in $\mathbb{N}$ containing $\lambda xy.\,(\text{if } x = y \text{ then } 1 \text{ else } 0)$, $\lambda xy.\, x + y$, $\lambda xy.\, x \cdot y$ and the projection functions, that is closed under compositions and contains f_0^+ for any f in it. Therefore (in view of Remark 3) it is sufficient to prove the calculability of the function $\lambda xy.\,(\text{if } x = y \text{ then } 1 \text{ else } 0)$ in the concrete $\mathcal{A}$ considered now and the elementary definability of the functions $\lambda x.\, 0$, $\lambda x.\, 1$ and δ_L. This can be done by means of the equalities

$$\begin{aligned}
&(\text{if } x = y \text{ then } 1 \text{ else } 0) \ = \ \delta_L(x, y, \delta_L(y, x, 1, 0), 0)\,,\\
&1 \ = \ (\text{if } x = x \text{ then } 1 \text{ else } 0)\,, \quad 0 \ = \ (\text{if } 1 = 1 + 1 \text{ then } 1 \text{ else } 0),\\
&\delta_L(x, y, s, t) \ = \ g_0^+(x, y, y) \cdot s \ + \ g_0^+(y + 1, x, x) \cdot t\,,
\end{aligned}$$

where $g = \lambda t_1 t_2 z.(\text{if } t_1 + z = t_2 \text{ then } 1 \text{ else } 0)$.

Example 2. Let $L = \mathbb{N}$, $\mathcal{A} = \{\lambda x.\, 0\,,\ \lambda xy.\, x{+}y\,,\ \lambda xy.\, x{\cdot}y\,,\ \lambda x.\, |x|\,,\ \lambda xy.\, x\#y\}$, where $|x|$ is the length of the binary representation of x if $x > 0$ and 0 otherwise, $x\#y$ is $2^{|x|\cdot|y|}$. In [1] some theories of arithmetic are studied that have the functions from $\mathcal{A}$ and the functions $\lambda x.\, x + 1$, $\lambda x.\, \lfloor x/2 \rfloor$ as their primitive functions and $\leq$ as their primitive relation (the study of these theories, and especially of formulas in them with bounded quantifiers, is related to problems of feasible computability). The functions $\lambda x.\, x + 1$, $\lambda x.\, \lfloor x/2 \rfloor$ are $\mathcal{A}$-calculable thanks to the equalities

$$1 \ = \ 0\#0\,, \quad \lfloor x/2 \rfloor \ = \ \bigvee_{y \in [0..x]} \delta_L(y + y, x, y, 0)\,,$$

hence adding them to $\mathcal{A}$ would not enlarge the class of the $\mathcal{A}$-calculable functions. A comparison with Example 1 shows that all elementary definable functions belong to this class. The converse is not true, because the elementary definable

[1] The class in question is a subclass (most likely a proper one) of the class of the lower elementary functions introduced in [3].

functions are dominated by polynomials, whereas the function $\lambda xy.\, x\#y$ is not. By a certain fairly general statement concerning calculability (to be proved in the last section), the class considered now can be characterized in a simple way through definability by formulas with bounded quantifiers in the language of the above-mentioned theories (the elementary definable functions can be characterized in a similar way by formulas with bounded quantifiers in a more restricted language).

Example 3. Let $L = \mathbb{N}$ and the ternary functions f_1 and f_2 be defined as follows:

$$f_1(x,y,z) = (\text{if } x+y=z \text{ then } 0 \text{ else } 1)\,,$$
$$f_2(x,y,z) = (\text{if } x\cdot y=z \text{ then } 0 \text{ else } 1)\,.$$

Since f_1 and f_2 are piecewise termal in the set $\mathcal{A}$ from Example 1, all functions calculable in $\{f_1,f_2\}$ are calculable in that $\mathcal{A}$, hence they are elementary definable. However, the converse is not true. For example, the constant 1 (regarded say as a one-argument function) is not calculable in the set $\{f_1,f_2\}$, since, whenever an n-ary function f is calculable in this set, the inequality $f(x_1,\ldots,x_n) \le x_1 \vee \ldots \vee x_n$ holds for all $x_1,\ldots,x_n$ in $\mathbb{N}$. We note that the constant 0 is still calculable (and even termal) in $\{f_1,f_2\}$ thanks to the equality $0 = f_2(f_2(x,x,x), f_2(x,x,x), f_2(x,x,x))$. By Remark 2, the functions of the form $\lambda x_1 \ldots x_n.\, x_1 \vee \ldots \vee x_n$ are also calculable in $\{f_1,f_2\}$. As an example of $\{f_1,f_2\}$-calculable function, that is not piecewise termal in $\{f_1,f_2\}$, we shall mention the predecessor function (defined as 0 at 0) – its value at x can be represented as

$$\bigvee_{y\in[0..x]} \delta_L(f_1(y, f_1(x,x,x), x), 0, y, 0)\,.$$

Example 4. Let $L = \mathbb{Z}$, $\mathcal{A} = \{\,\lambda x.\, 1,\ \lambda xy.\, x-y,\ \lambda xy.\, x\cdot y\,\}$. Then the functions $\lambda x.\, 0$, $\lambda x.\, -x$, $\lambda xy.\, x+y$ are termal in $\mathcal{A}$ thanks to the equalities $0 = x-x$, $-x = 0-x$, $x+y = x-(-y)$. The class of the $\mathcal{A}$-calculable functions is the smallest class of functions containing $\mathcal{A}\cup\{\delta_L\}\cup\mathcal{P}$ that is closed under compositions and contains f_0^+ for any f in it – this can be shown by using the equalities

$$f^+(\bar{x},u,v) = \bigvee_{z\in[0..v-u]} f(\bar{x}, v-z)\,,\quad f^-(\bar{x},u,v) = -\bigvee_{z\in[0..v-u)]} -f(\bar{x}, v-z)\,.$$

The $\mathcal{A}$-calculable functions can be characterized also in the following more explicit way: an n-argument function f is $\mathcal{A}$-calculable if and only if there exist two elementary definable $2n$-argument functions ψ and θ in $\mathbb{N}$ such that

$$f(i_1-j_1,\ldots,i_n-j_n) = \psi(i_1,j_1,\ldots,i_n,j_n) - \theta(i_1,j_1,\ldots,i_n,j_n)$$

for all $i_1,j_1,\ldots,i_n,j_n$ in $\mathbb{N}$.

Example 5. Let $L = \{0,1\}$, and let $\mathcal{A}$ consist of the constants 0 and 1 regarded as one-argument functions. Since $\lambda xy.\, \delta_L(x,y,1,0)$ is the Boolean implication, the Post Theorem shows that all Boolean functions are $\mathcal{A}$-calculable. Another proof of this will be given in the next section.

Example 6. Let $L = \{0,1\}$, $\mathcal{A} = \emptyset$. Then any $\mathcal{A}$-calculable function f belongs to both Post classes T_0 and T_1, i.e. $f(0,\dots,0) = 0$ and $f(1,\dots,1) = 1$. A proof of the converse statement will be given in the next section.

Example 7. Let $L = \{0,1\}$, $\mathcal{A} = \{\lambda x.\, 0\}$. Then all $\mathcal{A}$-calculable functions belong to T_0, and the converse statement will be proved in the next section.

Example 8. Let L be an infinite subset of $\mathbb{N}$, f be a unary function that transforms any element of L into some greater one, and g be the unary function that transforms any x from L into the least element of L greater than x. Then g is $\{f\}$-calculable thanks to the equality

$$g(x) = \bigwedge_{y\in[x..f(x)]_L} \delta_L(y,x,f(x),y)\,.$$

(However, a choice of L and f is possible such that g cannot be obtained by means of a recursive operator from f and the functions $\lambda xy.\ x \vee y$, $\lambda xy.\, x \wedge y$, $\lambda xz.\, \mathrm{card}\{y \in L \mid x \le y \le z\}$ mentioned in Remark 1.)

Example 9. Let L consist of all finite subsets of some infinite set, and $\mathcal{A}$ be any set of functions in L. The equalities

$$x_1 \setminus x_2 = \bigwedge_{y\in[\emptyset..x_1]_L} (\text{if } x_1 \le x_2 \vee y \text{ then } y \text{ else } x_1)\,, \quad \emptyset = x \setminus x$$

show that $\lambda x_1 x_2.\, x_1 \setminus x_2$ is $\mathcal{A}$-calculable if and only if $\lambda x.\, \emptyset$ is $\mathcal{A}$-calculable.

3 $\mathcal{A}$-calculable predicates

We continue adhering to the assumptions and conventions from the beginning of the previous section. Thus a non-degenerate locally finite lattice L and a set $\mathcal{A}$ of functions in it will be again supposed to be given. For any positive integer n and any n-argument predicate p on L the $(n+2)$-argument function

$$\lambda x_1 \dots x_n st.\, (\text{if } p(x_1,\dots,x_n) \text{ then } s \text{ else } t)$$

will be denoted by p^*, and it will be called *the representing function of* p (obviously there is a one-to-one correspondence between the predicates and their representing functions). A predicate will be called *calculable in* $\mathcal{A}$ or $\mathcal{A}$*-calculable*, for short, if the representing function of this predicate is $\mathcal{A}$-calculable.

Remark 7. If there are two distinct elements a and b of L such that the corresponding one-argument constant functions are $\mathcal{A}$-calculable then one can characterize the $\mathcal{A}$-calculability of predicates in a more usual way. In such a situation, an n-argument predicate p on L is $\mathcal{A}$-calculable if and only if the n-argument function

$$p^*_{a,b} = \lambda x_1 \dots x_n.\, (\text{if } p(x_1,\dots,x_n) \text{ then } a \text{ else } b)$$

is $\mathcal{A}$-calculable. This is clear from the equalities

$$\begin{aligned} p^*_{a,b}(\bar{x}) &= p^*(\bar{x}, a, b)\,, \\ p^*(\bar{x}, s, t) &= \delta_L(p^*_{a,b}(\bar{x}), a, \delta_L(a, p^*_{a,b}(\bar{x}), s, t), t)\,. \end{aligned}$$

Next example can be used together with Example 7 to show that the assumption about $\mathcal{A}$-calculability of a and b in the previous remark cannot be omitted (because the function $\lambda x.$ (if $x = 1$ then 0 else 1) is not $\mathcal{A}$-calculable in the situation from Example 7).

Example 10. Let $L = \{a, b\}$, where $a < b$. Then the predicates $p = \lambda x.\,(x = a)$ and $q = \lambda x.\,(x = b)$ are $\mathcal{A}$-calculable thanks to the equalities

$$\begin{aligned} p^*(x, s, t) &= \delta_L(s, t, \delta_L(x, s, s, t), \delta_L(x, t, s, t))\,, \\ q^*(x, s, t) &= \delta_L(s, t, \delta_L(t, x, s, t), \delta_L(s, x, s, t))\,. \end{aligned}$$

Immediate examples of $\mathcal{A}$-calculable predicates (for any $\mathcal{A}$) are the identically true and the identically false ones, as well as the predicate $\lambda x_1 x_2.\,(x_1 \le x_2)$. The class of the $\mathcal{A}$-calculable n-argument predicates is closed under negation, conjunction and disjunction, as seen from the equalities

$$\begin{aligned} (\text{not } p)^*(\bar{x}, s, t) &= p^*(\bar{x}, t, s), \\ (p \text{ and } q)^*(\bar{x}, s, t) &= p^*(\bar{x}, q^*(\bar{x}, s, t), t), \\ (p \text{ or } q)^*(\bar{x}, s, t) &= p^*(\bar{x}, s, q^*(\bar{x}, s, t)). \end{aligned}$$

The class of the $\mathcal{A}$-calculable predicates is closed also under substitution of $\mathcal{A}$-calculable funcrions, i.e. whenever p is an $\mathcal{A}$-calculable m-argument predicate, and $f_1, \ldots, f_n$ are $\mathcal{A}$-calculable n-argument functions, the predicate

$$q = \lambda x_1 \ldots x_n.\, p(f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$$

is also $\mathcal{A}$-calculable. This is clear from the equality

$$q^*(\bar{x}, s, t) = p^*(f_1(\bar{x}), \ldots, f_m(\bar{x}), s, t).$$

By using the above properties one can easily see the $\mathcal{A}$-calculability of a lot of other predicates. For example $\lambda x_1 x_2.\,(x_1 = x_2)$ is $\mathcal{A}$-calculable as a conjunction of $\lambda x_1 x_2.\,(x_1 \le x_2)$ and $\lambda x_1 x_2.\,(x_2 \le x_1)$. In the conditions of Example 10 each n-argument predicate on L turns out to be $\mathcal{A}$-calculable, since it can be obtained from predicates of the form $\lambda x_1 \ldots x_n.\,(x_i = a)$ by means of negation, conjunction and disjunction.

The definition of calculability ensures one more kind of closedness of the class of the $\mathcal{A}$-calculable predicates. Namely, if p is an $\mathcal{A}$-calculable $(n+1)$-argument predicate on L for some non-negative integer n then the following two predicates are also $\mathcal{A}$-calculable:

$$\begin{aligned} q &= \lambda x_1 \ldots x_n uv.\,(\, p(x_1, \ldots, x_n, y) \text{ for all } y \in [u..v]\,)\,, \\ r &= \lambda x_1 \ldots x_n uv.\,(\, p(x_1, \ldots, x_n, y) \text{ for some } y \in [u..v]\,)\,. \end{aligned}$$

This follows from the equalities

$$q^*(\bar{x}, u, v, s, t) \;=\; \delta_L\bigl(\bigvee_{y\in[u..v]} p^*(\bar{x}, y, s\wedge t, s\vee t), s\wedge t, s, t\bigr)\,,$$

$$r^*(\bar{x}, u, v, s, t) \;=\; \delta_L\bigl(\bigwedge_{y\in[u..v]} p^*(\bar{x}, y, s\wedge t, s\vee t), s\wedge t, s, t\bigr)\,.$$

For any n-argument function f the predicate $\lambda x_1 \dots x_n y.\,(y = f(x_1, \dots x_n))$ will be said to *represent* f. If f is $\mathcal{A}$-calculable then this predicate is also $\mathcal{A}$-calculable, as it follows from some of the above-mentioned properties. Next lemma gives a way for reasoning in the opposite direction, namely from the $\mathcal{A}$-calculability of the predicate representing a function to the $\mathcal{A}$-calculability of the function itself.

Lemma 1. *Let f, g, h be n-argument functions such that $g(\bar{x}) \le f(\bar{x}) \le h(\bar{x})$ for any $\bar{x}$ in L^n, and the functions g and h, as well as the predicate representing f, are $\mathcal{A}$-calculable. Then the function f is also $\mathcal{A}$-calculable.*

Proof. If p is the predicate that represents f then

$$f(\bar{x}) \;=\; \bigvee_{y\in[g(\bar{x})..h(\bar{x})]} p^*(\bar{x}, y, y, g(\bar{x}))$$

for any $\bar{x}$ in L^n. $\square$

An n-argument function f will be called $\mathcal{A}$-*calculable on* a subset X of L^n if f coincides on X with some $\mathcal{A}$-calculable n-argument function. Clearly an n-argument function is $\mathcal{A}$-calculable if and only if it is $\mathcal{A}$-calculable on the whole L^n. A combined application of this fact and the following lemma can be useful for proving the $\mathcal{A}$-calculability of some functions.

Lemma 2. *Let $X_1, \dots, X_k$, where $k \ge 2$, be subsets of L^n, and the predicates $\lambda x_1 \dots x_n.\,(\,(x_1, \dots, x_n) \in X_i\,)$, $i = 2, \dots, k$, be $\mathcal{A}$-calculable. If an n-argument function is $\mathcal{A}$-calculable on each of the sets $X_1, \dots, X_k$, then it is $\mathcal{A}$-calculable also on their union.*

Proof. It is sufficient to prove the statement of the lemma for the case of $k = 2$, since the statement for the general case can be obtained from there by induction. The reasoning for the case of $k = 2$ looks as follows: if an n-argument function f coincides on X_i with the $\mathcal{A}$-calculable function g_i for $i = 1, 2$, then f coincides on $X_1 \cup X_2$ with the function $\lambda x_1 \dots x_n.\,p^*(\bar{x}, g_2(\bar{x}), g_1(\bar{x}))$, where $p = \lambda x_1 \dots x_n.\,(\,(x_1, \dots, x_n) \in X_2\,)$. $\square$

By application of Lemma 2, we shall fulfil now the promises made in the three examples with $L = \{0, 1\}$ of the previous section. In the case of Example 5, given an arbitrary n-ary function f, we take $k = 2$, $X_1 = \{\bar{x} \,|\, f(\bar{x}) = 0\}$ and $X_2 = \{\bar{x} \,|\, f(\bar{x}) = 1\}$. In the case of Example 6, given an n-ary function f belonging to $T_0 \cap T_1$, we take $k = n$ and $X_i = \{\bar{x} \,|\, f(\bar{x}) = x_i\}$ for $i = 1, \dots, n$.

Finally, in the case of Example 7, given an n-ary function f belonging to T_0, we take $k = n + 1$ and $X_{n+1} = \{\bar{x} \mid f(\bar{x}) = 0\}$, the sets $X_1, \ldots, X_n$ being defined in the same way as in the case of Example 6. In each of these cases, the given function f is $\mathcal{A}$-calculable on any of the corresponging sets $X_1, \ldots, X_k$, hence it is $\mathcal{A}$-calculable on their union (because all predicates are $\mathcal{A}$-calculable in the case of a two-element L). On the other hand, it is easy to show that the union in question is equal to $\{0, 1\}^n$ in each of the three cases.

4 Definability by formulas with bounded quantifiers

Let a signature Σ be given with some function symbols and only one predicate symbol, and let this predicate symbol be a binary one. Let S be a structure for Σ such that the domain of S is $\mathbb{N}$, and let the predicate symbol of Σ be interpreted in S as the relation $\leq$ between natural numbers. In addition to the interpretation in S of any functional symbol of Σ we shall consider also its *adapted interpretation* in S - meaning the interpretation of the symbol in S if the symbol is not a constant of Σ and the one-argument constant function whose value is the interpretation of the symbol otherwise. Taking L to be the set $\mathbb{N}$ with the relation $\leq$, we shall consider any set $\mathcal{A}$ of functions such that each function in $\mathcal{A}$ is the adapted interpretation in S of some function symbol of Σ, and the adapted interpretation of each function symbol of Σ is $\mathcal{A}$-calculable.[2] Under some additional assumptions we shall study the connection between $\mathcal{A}$-computability and definability by bounded formulas of the first-order language $\mathbf{L}_\Sigma$ corresponding to Σ.

Having in mind the above-mentioned interpretation of the predicate symbol of Σ, we shall write the atomic formulas of $\mathbf{L}_\Sigma$ as inequalities between terms. By definition, a formula of $\mathbf{L}_\Sigma$ is *bounded* if it is constructed from atomic formulas by means of negation, conjunction, disjunction and bounded quantifiers of the forms $(\forall \xi \leq \tau)$, $(\exists \xi \leq \tau)$ with term τ not containing the variable ξ (assuming that $(\forall \xi \leq \tau)F$ and $(\exists \xi \leq \tau)F$ are abbreviations for $\forall \xi((\xi \leq \tau) \rightarrow F)$ and $\exists \xi((\xi \leq \tau) \ \& \ F)$, respectively).

The additional assumptions we make are the following ones: (i) the constant 0 regarded as a one-argument function is piecewise termal in $\mathcal{A}$, (ii) a two-argument function exists that is termal in $\mathcal{A}$ and dominates its arguments, and (iii) each function belonging to $\mathcal{A}$ is dominated by some function termal in $\mathcal{A}$ and monotonically increasing with respect to any of its arguments.

The assumption (i) allows to apply Remark 3, and the other two assumptions imply that each $\mathcal{A}$-calculable function is dominated by some function termal in $\mathcal{A}$. All three of them are satisfied if $\mathcal{A}$ is as in Example 1 or Example 2 (even

[2] The simplest choice is to take as $\mathcal{A}$ the set consisting of the adapted interpretations in S of all function symbols of Σ, but sometimes certain of these interpretations can be omitted. For instance we must have in Σ function symbols for the functions $\lambda x.\, x + 1$ and $\lambda x.\, \lfloor x/2 \rfloor$ in order to get the language of the theories studied in [1], but it is not necessary to include the functions themselves in the set $\mathcal{A}$ – this set can be as in Example 2.

if one strengthens the assumptions by replacing "is piecewise termal in" and "is termal in" with "belongs to" and by skipping the phrase "dominated by some function termal in $\mathcal{A}$ and").

Lemma 3. *For any $\mathcal{A}$-calculable function the predicate representing it is definable by means of some bounded formula of $\mathbf{L}_\Sigma$.*

Proof. By using the characterization of $\mathcal{A}$-calculability indicated in Remark 3. The domination property discussed above is used in the reasoning about composition for turning some quantifiers into bounded ones. □

Lemma 4. *All predicates definable by means of bounded formulas of $\mathbf{L}_\Sigma$ are $\mathcal{A}$-calculable.*

Proof. By induction using properties indicated in the previous section. □

Theorem 1. *A function f is $\mathcal{A}$-calculable if and only if the predicate representing f is definable by means of some bounded formula of $\mathbf{L}_\Sigma$ and f is dominated by some function termal in $\mathcal{A}$.*

Proof. By Lemmas 1, 3 and 4. □

Theorem 2. *A predicate in L is $\mathcal{A}$-calculable if and only if it is definable by means of some bounded formula of $\mathbf{L}_\Sigma$.*

Proof. The "if"-direction is given by Lemma 4. Suppose now that p is an $\mathcal{A}$-calculable n-argument predicate in L. Let q be the n-argument predicate that is false at $(0,\dots,0)$ and coincides with p at all other elements of L^n. The predicate q is definable by means of a bounded formula of $\mathbf{L}_\Sigma$, since $q(x_1,\dots,x_n)$ is true if and only if $x_i \neq 0$ and $x_i = p^*(x_1,\dots,x_n,x_i,0)$ for some i in $\{1,\dots,n\}$. Then it remains to note that either $p = q$ or p is equal to the disjunction of q and the predicate $\lambda x_1 \dots x_n.\,(\,(x_1,\dots,x_n) = (0,\dots,0)\,)$. □

Remark 8. The assumptions made in this section are still not sufficient for the existence of $\mathcal{A}$-calculable non-zero constant functions (consider for instance the case of $\mathcal{A} = \{\lambda x.\,0,\ \lambda xy.\,x+y\}$). However, if there exists an $\mathcal{A}$-calculable non-zero constant function (for instance if $\mathcal{A}$ is as in Example 1 or Example 2) then the above proof can be replaced by an essentially simpler one. Indeed, if b is the value of such a constant function then the definability of the $\mathcal{A}$-calculable n-argument predicate p can be seen from the equality

$$p \,=\, \lambda x_1 \dots x_n.\,(0 = p^*(x_1,\dots,x_n,0,b)\,)\,.$$

References

1. Buss, S. R.: Bounded Arithmetic. Bibliopolis, Naples, 1986
2. Moschovakis, Y. N.: Abstract first order computability. I. Trans. Amer. Math. Soc. **138** (1969) 427–464
3. Skolem, Th.: A theorem on recursively enumerable sets. Abstr. of Short Comm., Int. Congress Math., Stockholm, 1962, p. 11
4. Skordev, D.: On a class of primitive recursive functions. Ann. de l'Univ. de Sofia, Fac. de Math. **60** (1967) 105–111 (in Russian, with English summary)

Non-total Enumeration Degrees

Boris Solon

Ivanovo State University of Chemistry and Technology
Ivanovo, Russia
solon@icti.ivanovo.su

Abstract. An enumeration degree in which there are no total functions is called a *non-total* enumeration degree. It appears that the non-totality of enumeration degrees is the quasi-minimality according to Slaman and Sorbi.

1 Enumerations of sets and enumeration degrees

Terminology and notations similar to those of monograph [8] are used. Let ω denote the set of positive integers, $A, B, C, \ldots, X$ (with or without indices) will be used to denote subsets of ω; D (with or without indices) will be used to denote finite subsets of ω. For given partial function $\alpha : \omega \to \omega$ let $\delta\alpha$, $\rho\alpha$ and $\tau\alpha = \{\langle x, \alpha(x)\rangle : x \in \delta\alpha\}$ be the domain, the range and the graph of α respectively. The use of f, g only to denote *total* functions, i.e. $\delta f = \delta g = \omega$ is restricted. Let us denote the characteristic function of A by $c_A(x) = \{(x, 0) : x \in A\} \cup \{(x, 1) : x \notin A\}$. A set A is said to be *single-valued*, if $A = \tau\alpha$ for some partial function α.

Let $A \leq_e B$ (A *is enumeration reducible to* B or A *is e-reducible to* B), if there is a uniform algorithm for enumerating A given any enumeration of B (see [2]). Formally

$$A \leq_e B \iff (\exists n)(\forall x)[x \in A \iff (\exists u)[\langle x, u\rangle \in W_n \ \&\ D_u \subseteq B]].$$

Let $\varPhi_n : 2^\omega \to 2^\omega$ be such mapping that for any X

$$\varPhi_n(X) = \{x : (\exists u)[\langle x, u\rangle \in W_n \ \&\ D_u \subseteq X]\}.$$

Then $A \leq_e B \iff (\exists n)[A = \varPhi_n(B)]$. $\varPhi_n$ is called *the enumeration operator* or *e-operator with c.e. index* n.

Let as usual $A \equiv_e B \iff A \leq_e B \ \&\ B \leq_e A$, let $d_e(A) = \{B : B \equiv_e A\}$ be *the e-degree of* A and finally let $d_e(A) \leq d_e(B) \iff A \leq_e B$. It is easy to see that this defines a partial ordering relation on e-degrees. For partial function α, β $\alpha \leq_e A$ or $\alpha \leq_e \beta$ if $\tau\alpha \leq_e A$ or $\tau\alpha \leq_e \tau\beta$, respectively. The set of e-degrees partially ordered by $\leq$ is denoted as $\mathbf{D}_e$. It is well known that $\mathbf{D}_e$ forms an upper semilattice with least element $\mathbf{0}_e = \{W_n : n \in \omega\}$ in which the least upper bound of the e-degrees $\mathbf{a}$ and $\mathbf{b}$ is $\mathbf{a} \vee \mathbf{b} = d_e(A \oplus B)$. It is known [1] that there are e-degrees $\mathbf{a}$ and $\mathbf{b}$ which do not have the greatest lower bound $\mathbf{a} \wedge \mathbf{b}$ in $\mathbf{D}_e$. So the binary operation $\wedge$ is partial.

We denote the upper semilattice of Turing degrees as $\mathbf{D}_T$. As for all A, B

$$A \leq_T B \iff c_A \leq_e c_B$$

there is an isomorphic embedding $\epsilon : \mathbf{D}_T \to \mathbf{D}_e : \epsilon(d_T(A)) = d_e(c_A)$. An e-degree $\mathbf{a}$ is said to be *total*, if it contains a total function. We denote the set of all total e-degrees by $\mathbf{T}$. It is rather obvious that $\mathbf{T}$ is an upper subsemilattice of $\mathbf{D}_e$ and $\rho\epsilon = \mathbf{T}$.

We say that any *enumeration* of $A \neq \emptyset$ is a total function $p : \omega \to A$ so that $\rho p = A$. We denote the set of all enumerations of A by $\mathbf{P}(A)$. Let $\mathbf{D}_e(A) = \{d_e(\tau p) : p \in \mathbf{P}(A)\}$ be partially ordered by $\leq$. We note that if $|A| = 1$ then $\mathbf{D}_e(A) = \{\mathbf{0}_e\}$.

In general case we have the following

Proposition 1. *If* $|A| \geq 2$ *and* $\mathbf{a} = d_e(A)$ *then* $\mathbf{D}_e(A) = \mathbf{T}(\geq \mathbf{a}) = \{\mathbf{x} \in \mathbf{T} : \mathbf{x} \geq \mathbf{a}\}$.

The following theorem which is a known result of [6] shows that the e-reducibility can be defined via enumerations.

Theorem 1. *For any sets* A, $|A| \geq 2$ *and* B

$$A \leq_e B \iff \mathbf{D}_e(B) \subseteq \mathbf{D}_e(A).$$

The following theorem 2 below has intuitive underlying reason. It appears that e-reducibility of sets can be defined through enumerations (as total functions), and in this definition it is possible to change the places of the quantifiers (see to compare our intuitive definition of e-reducibility): A is e-reducible to B if for any enumeration of B there is an algorithm allowing to get some enumeration of A. Formally,

Theorem 2. $A \leq_e B \iff (\forall p \in \mathbf{P}(B))(\exists n)[A = \Phi_n(\tau p)]$.

The proof of this theorem uses the idea from Selman's proof.

The properties of $\mathbf{D}_e(A)$ depend essentially on A. For example now as it is shown corollary 1 of theorem 1 some algebraic properties of $\mathbf{D}_e(A)$ depend on whether $d_e(A)$ is total or not total e-degree.

Corollary 1. *For any set* A

$$d_e(A) \in \mathbf{T} \iff \mathbf{D}_e(A) \textit{ has the least element.}$$

From this corollary it follows that if $\mathbf{a} = d_e(A)$ is a non-total degree then firstly $\mathbf{a} \notin \mathbf{D}_e(A)$ and secondly $\mathbf{D}_e(A)$ has no the least elements. What is it possible to tell about non-total e-degrees except that they do not contain total functions?

2 Quasi-minimality as a sufficient condition of non-totality

The first important question about e-reducibility was solved by Yu. Medvedev [3]: $\mathbf{D}_e \setminus \mathbf{T} \neq \emptyset$. Yu. Medvedev announced the existence of a non-computably enumerable set A so that

$$\forall f[f \leq_e A \Rightarrow f \textit{ is computable}].$$

In Rogers's monograph [4] (see p.280) this result was proved in the following way:

$$\exists\alpha[\alpha \textit{ is not partial computable } \& \ (\forall f)[f \leq_e \alpha \Rightarrow f \textit{ is computable}]].$$

It is clear that $d_e(\tau\alpha)$ is not total, i.e. it is *non-total*. J. Case [1] called Medvedev's sets *quasi-minimal* and their degrees *quasi-minimal e-degrees*. We note that there are non-total e-degrees which are not quasi-minimal. For this we recall what is known as $\mathbf{c}$ -quasi-minimal e-degrees. A set A is called *C-quasi-minimal* and the e-degree $\mathbf{a} = d_e(A)$ is called $\mathbf{c}$ *-quasi-minimal* where $\mathbf{c} = d_e(C)$ if $C < A$ and

$$\forall f[f \leq_e A \Rightarrow f \leq_e C].$$

The existence of $\mathbf{c}$ -quasi-minimal e-degrees for any $\mathbf{c} \in \mathbf{D}_e$ was proved by L.P. Sasso [5]. Let $\mathbf{Q_c}$ be the set of all $\mathbf{c}$-quasi-minimal e-degrees.

The family $\{\mathbf{Q_c} : \mathbf{c} \in \mathbf{D_e}\}$ concerning the set-theoretic inclusion $\subseteq$ is arranged rather difficultly. Not completing all possible relation we shall note some of them in the following

Proposition 2. *For any* $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c} \in \mathbf{D}_e$

(i) $\mathbf{a} \in \mathbf{Q_c} \iff \{\mathbf{a}\} \cup \mathbf{Q_a} \subseteq \mathbf{Q_c}$;

(ii) $\mathbf{c} < \mathbf{a} \Rightarrow [\mathbf{a} \notin \mathbf{Q_c} \iff \mathbf{Q_a} \cap \mathbf{Q_c} = \emptyset]$;

(iii) $\mathbf{c} > \mathbf{0}_e \ \& \ \mathbf{c} \notin \mathbf{Q} \Rightarrow \mathbf{Q} \cap \mathbf{Q_c} = \emptyset$;

(iv) $\mathbf{a}|\mathbf{b} \ \& \ [\mathbf{a} \in \mathbf{T} \vee \mathbf{b} \in \mathbf{T}] \Rightarrow \mathbf{Q_a} \cap \mathbf{Q_b} = \emptyset$;

(v) $\mathbf{a} \in \mathbf{Q_c} \ \& \ \mathbf{b} \in \mathbf{Q_c} \Rightarrow [\mathbf{a} \vee \mathbf{b} \in \mathbf{Q_c} \iff \mathbf{Q_a} \cap \mathbf{Q_b} \neq \emptyset]$.

In fact for any $\mathbf{a}, \mathbf{b} \in \mathbf{Q_c}$ both situations are possible: $\mathbf{a} \vee \mathbf{b} \in \mathbf{Q_c}$ and $\mathbf{a} \vee \mathbf{b} \notin \mathbf{Q_c}$.

In the articles [9] and [10] the special classes of sets of which e-degrees are non-total are determined. Let $a_0, a_1, \ldots$ be the elements of the infinite set A in ascending order (or be *the direct enumeration of* A). A set A is called *e-hyperimmune* if there is no total function g so that $g \leq_e A$ and $a_n \leq g(n)$ for all $n \in \omega$. As shown in the following theorem there are e-hyperimmune (and therefore, non-total) e-degrees which are not $\mathbf{f}$-quasi-minimal for any $\mathbf{f} \in \mathbf{T}$.

Theorem 3. *There is e-hyperimmune set A so that e-degree* $\mathbf{a} = d_e(A)$ *is not* $\mathbf{f}$*-quasi-minimal for any* $\mathbf{f} \in \mathbf{T}$.

Proof. We construct step by step a set A in order to satisfy the following requirements:

eHI: $\forall g[g \leq_e A \Rightarrow \exists m[a_m > g(m)]]$,
where a_m is m-th member of the direct enumeration of A;

NQ: $\forall f[f \leq_e A \Rightarrow \exists g[g \leq_e A \ \& \ g \not\leq_e f]]$.
(Here and below the symbols f and g are used as variables which range over the set of all total functions.) Let $g_0 <_e g_1 <_e \dots$ be a sequence of the total functions, $G_s = \{\langle s, x, g_s(x)\rangle : x \in \omega\}$ and $H_s = \{\langle m, x, y\rangle : m > s \ \& \ m, x, y \in \omega\}$ for all $s \in \omega$. The part of A which has been constructed by the ending of step s is denoted by A_s. The construction provides $A_0 \subseteq A_1 \subseteq \dots \subseteq A_s \subseteq \dots$. Let $A = \cup_{s\in\omega} A_s$. While constructing we choose the sequence $\{z_s\}_{s\in\omega}$ from A which will be of the auxiliary character. Below the symbol D denotes a variable which ranges over the family of all finite sets. We recall just one more notation: $B\lceil z = \{x : x < z \ \& \ x \in B\}$.

Step 0. Set $A_0 = G_0$ and $z_0 = \min A_0$.

Step s+1. Let $s = 2n$. We denote by $\tilde{H}_s = H_s \setminus \{0, 1, \dots, z_s^*\}$ where $z_s^* = max\{z_i : i \leq s\}$. See whether

$$\exists D[D \subseteq \tilde{H}_s \ \& \ \Phi_n(A_s \cup D) \ is\ not\ single-valued]. \tag{1}$$

If (1) holds then let D^* have the least canonical index. Set $A_{s+1} = A_s \cup G_{s+1}^* \cup D^*$ where $G_{s+1}^* = G_{s+1} \setminus \{0, 1, \dots, z_s^*\}$. Otherwise

$$\forall D[D \subseteq \tilde{H}_s \Rightarrow \Phi_n(A_s \cup D) \ is\ single-valued]. \tag{2}$$

In this case we set $A_{s+1} = A_s \cup G_{s+1}^*$. In any cases we set $z_{s+1} = z_s$.

Let $s = 2n + 1$. See whether

$$(\exists D)(\exists m)[D \subseteq \tilde{H}_s \ \& \ \Phi_n(A_s\lceil z_s^* \cup D)(m)\downarrow \ \& \ \Phi_n(A_s\lceil z_s^* \cup D)(m) < a_m], \tag{3}$$

where a_m is the m-th member of the direct enumeration of $A_s\lceil z_s^* \cup D$. We define here

$$\Phi_n(A_s\lceil z_s^* \cup D)(m)\downarrow \iff \exists\alpha[\Phi_n(A_s\lceil z_s^* \cup D) = \tau\alpha \ \& \ m \in \delta\alpha]$$

where α is a variable which ranges over the family of all one-place partial functions. If (3) holds then let D^* have the least canonical index and let m^* be the least number. Set $A_{s+1} = A_s \cup D^*$ and $z_{s+1} = a_{m*}$. Otherwise set $A_{s+1} = A_s$. The description of our construction is completed.

That the requirement eHI is satisfied will be proved. Let $g \leq_e A$ then $\tau g = \Phi_n(A)$ for some $n \in \omega$. We consider the step $s + 1$ where $s = 2n + 1$. If the condition (3) holds then

$$g(m^*) = \Phi_n(A_s\lceil z_s^* \cup D^*)(m^*) = \Phi_n(A_{s+1})(m^*) = \Phi_n(A)(m^*) < a_{m*} = z_{s+1},$$

as the direct enumeration of $A_{s+1}\lceil z_{s+1}^*$ does not change at next steps and it coincide with the direct enumeration of $A\lceil z_{s+1}$.

We suppose that the condition (3) does not hold then

$$(\forall D)(\forall m)[D \subseteq \tilde{H}_s \ \& \ \Phi_n(A_s\lceil z_s^* \cup D)(m) \downarrow \Rightarrow \Phi_n(A_s\lceil z_s^* \cup D)(m) \geq a_m],$$

where a_m is m-th member of the direct enumeration of $A_s\lceil z_s^* \cup D$. As $A_s\lceil z_s^* \cup \{x\} \subseteq A$ for all $x \in A$ then $\Phi_n(A_s\lceil z_s^* \cup \{x\}) \subseteq \Phi_n(A) = \tau g$. Therefore

$$(\forall x \in A)(\forall m)[x > z_s^* \Rightarrow \Phi_n(A_s\lceil z_s^* \cup D)(m) \downarrow \ \& \ \Phi_n(A_s\lceil z_s^* \cup D)(m) \geq a_m].$$

Let $A_s\lceil z_s^* \cup \{x\}$ contains $j+1$ elements for $x > z_s^*$. Then for $m = j$ we have

$$(\forall x \in A)[x > z_s^* \Rightarrow \Phi_n(A_s\lceil z_s^* \cup D)(j) \geq x].$$

As A be the infinite set then the value of $g(j)$ cannot be defined and this contradicts the totality of the function g. Hence if $\tau g = \Phi_n(A)$ for some $n \in \omega$ then the condition (3) must be fulfilled and then $\exists m[a_m > g(m)]$ where a_m is m-th member of the direct enumeration of A. This means that A is e-hyperimmune set.

Now one can prove that requirement NQ is satisfied as well. For this it is sufficient to verify the validity of two statements:

(i) $g_s \leq_e A$ for all $s \in \omega$ and

(ii) $A_s \leq_e g_s$ for all $s \in \omega$.

We complete the proof of the theorem. Let $f \leq_e A$, i.e. $\tau f = \Phi_n(A)$ for some $n \in \omega$. We consider step $s+1$ where $s = 2n$. It is clear that the condition (1) does not hold. Hence $\Phi_n(A_s \cup \tilde{H}_s)$ is the single-valued set. We have $\tau f = \Phi_n(A) \subseteq \Phi_n(A_s \cup \tilde{H}_s)$. By the totality of the function f and the single-valuedness of $\Phi_n(A_s \cup \tilde{H}_s)$ one obtains $\tau f = \Phi_n(A_s \cup \tilde{H}_s)$ i.e. $f \leq_e A_s \cup \tilde{H}_s \leq_e A_s$.

We suppose that $g_{s+1} \leq_e f$ then $g_{s+1} \leq_e f \leq_e A_s \leq_e g_s$ what contradicts the hypothesis $g_{s+1} \not\leq_e g_s$. Thus

$$\forall f[f \leq_e A \Rightarrow \exists s[g_s \leq_e A \ \& \ g_s \not\leq_e f]]$$

and requirement NQ is satisfied. This implies that A is not τf-quasi-minimal set for any total function f. The theorem is proved completely.

From this theorem it follows that **f**-quasi-minimality is not a necessary condition of non-totality of e-degrees.

3 Quasi-minimality as a necessary condition of non-totality

In the article [7] a relativized variant of the quasi-minimality generalizing the concept of the **c**-quasi-minimal e-degree is offered. We recall this

Definition 1 (Slaman, Sorbi). *Let* $\mathbf{I} \neq \emptyset$ *be a set of e-degrees. E-degree* $\mathbf{a}$ *is called* $\mathbf{I}$*-quasi-minimal if*

(i) $\mathbf{c} < \mathbf{a}$ *for any* $\mathbf{c} \in \mathbf{I}$,

(ii) $\mathbf{f} \leq \mathbf{a} \iff \exists \mathbf{c}[\mathbf{c} \in \mathbf{I} \ \& \ \mathbf{f} \leq \mathbf{c}]$ *for any* $\mathbf{f} \in \mathbf{T}$.

In particular if $\mathbf{I} = \{\mathbf{c}\}$ then $\mathbf{I}$-quasi-minimal e-degree is $\mathbf{c}$-quasi-minimal e-degree. It is clear that any $\mathbf{I}$-quasi-minimal e-degree is non-total. From [7] it follows that there is $\mathbf{G}$-quasi-minimal e-degree for any growing sequence $\mathbf{G} = \{\mathbf{g}_s\}_{s\in\omega}$ of total e-degrees $\mathbf{g}_s = d_e(\tau g_s)$ for all $s \in \omega$.

Now we are able to suggest the following characterization of non-total e-degrees.

Theorem 4. *Any non-total e-degree* $\mathbf{a}$ *is* $\mathbf{f}$- *quasi-minimal for some total e-degree* $\mathbf{f} < \mathbf{a}$ *or* $\mathbf{G}$*-quasi-minimal for some family of total e-degrees* $\mathbf{G} = \{\mathbf{g}_s\}_{s\in\omega}$ *forming the growing sequence* $\mathbf{g}_0 < \mathbf{g}_1 < \cdots < \mathbf{g}_s < \cdots < \mathbf{a}$.

References

1. Case J.: Enumeration reducibility and partial degreees. Annals Math. Logic **2**(1971) 419–439
2. Fridberg R., Rogers H.: Reducibility and completness for sets of integers. Z. math. Logic Grundl. Math. **5**(1959) 117–125
3. Medvedev Yu.T.: Degrees of difficulty of the mass problem. Dokl.Acad.Nauk SSSR **104**(1955) 501–504
4. Rogers H.,Jr.: Theory of Recursive Functions and Effective Computability. McGraw-Hill. New York. 1967
5. Sasso L. P.: A survey of partial degrees. J. Symb. Logic **40** (1975) 130–140
6. Selman: Arithmetical reducibilities I. Z. Math. Logik Grundlag. Math. **17** (1971) 335–350
7. Slaman T.A., Sorbi A.: Quasi-minimal enumeration degrees and minimal Turing degrees. Preprint (1996), 1–26
8. Soar Robert I.: Recursively Enumerable Sets and Degrees. Springer-Verlag. Berlin, Heidelberg, New York, London. 1987
9. Solon B.J.: e-hyperimmune sets. Sybir.Math.J. **33**(1992) 211-214
10. Solon B.J.: e-immune sets. Sybir.Math.J. **41**(2000) 676-691

A lambda calculus for real analysis

Paul Taylor

University of Manchester
Manchester, UK
`http://www.cs.man.ac.uk/~pt`
`pt@cs.man.ac.uk`

Abstract Stone Duality is a revolutionary theory that axiomatises general topology directly, without using either set theory or infinitary lattice theory. Every expression in the calculus denotes both a continuous function and a program. The closed interval is compact, whereas in the Russian school of computable analysis it is not. Nevertheless, the reasoning looks remarkably like a sanitised form of that in classical topology.

This paper applies ASD to elementary real analysis, culminating in the Intermediate Value Theorem, ie the solution of equations $f(x) = 0$ for continuous $f\colon \mathbb{R} \to \mathbb{R}$. As is well known from both numerical and constructive considerations, the equation cannot be solved if f "hovers" near 0, whilst tangential solutions will never be found.

In ASD, both these failures and the general method of finding solutions of the equation when they exist are explained by the new concept of "overtness".

Overtness, like compactness, is a purely topological concept. It replaces metrical properties such as total boundedness, and cardinality conditions such as having a countable dense subset. It is also related to locatedness in constructive topology and recursive enumerability in recursion theory.

Overtness and compactness are expressed by the two modal operators, which may be interpreted as higher-order, parallel, non-deterministic logic programs, which find solutions of equations in a uniform way.

Computability in Specification

Raymond Turner

University of Essex
Colchester, UK
http://cswww.essex.ac.uk/staff/turner.htm

Abstract. We develop a *Core Specification Theory* (**CST**) and define a notion of *computability* for this theory. We then introduce relational and functional specifications and use the framework to explore the connections between specification and computability.

1 Specification and Implementation

Generally, specification languages ([2], [7], [8], [4], [8], [9], [10]) do not insist that specifications should be implementable. For example, both Z and **VDM** allow the specification of non-computable relations and functions. But should they[1]? Given that the aim of specification is to produce specifications of implementable systems, there seems little point in specifying ones that are not. The same issue would never arise with programming languages. However, with specification languages, there are considerations having to do with expressive power that muddy the waters. In particular, we might be prepared to allow non-implementable specifications if they facilitate more succinct and transparent ways of expressing the implementable ones. The objective of this paper is to perform a preliminary investigation of these issues. To this end, we introduce and study a notion of *computability* for specification languages. Obviously, we cannot do so for every language; instead, we turn to the following idea.

2 A Core Specification Theory

By a *Specification Theory* (**ST**) we shall mean a package that consists of a specification language, a logic and a collection of axioms that govern its associated constructs. To support our investigation, we put in place a *core specification theory* (**CST**) [12]. It is a theory of *numbers*, *sets* and *products*. These constructors are present in all the major specification languages in one form or another. So this theory will provide us with a core framework with which to study the interaction between *computability* and *specification*.

[1] [11] have carried out a very thorough study of computability in abstract data types where notions of soundness and completeness for a specification method are presented. Despite this, actual specification languages do not seem to restrict matters to computable specifications -under any definition. For example, in Z there is no restriction on the wff that may be used in a specification.

The language has three syntactic categories: *wff*, *types* and *terms*. Generally, we shall employ upper case Roman letters A, B, C, D,...for type terms. These are generated from N, the *natural number* type, by the construction of *Cartesian products* ($\otimes$) and *sets* (*Set)*. For the terms, apart from the individual variables, we admit the constant zero (0) and the numerical successor function ($^+$). For Cartesian products, we include the pairing function () and the selection functions (π_i). Finally, for sets we require a constants ($\emptyset_T$) for the empty sets of each type, and a binary *insertion* function ($\circledast$) that adds an element to a set. Generally, we employ lower case Roman letters $a, b, c, ...$ for individual terms with $u, u_1, u_2, ...v, v_1, v_2, ..w, ..x, ..y, ..z$ reserved for term variables. Finally, we introduce the wff, for which we employ lower case Greek letters. The atomic wff include absurdity (Ω), equality, set membership ($\in$) and the ordering relation on the natural numbers ($<$). General wff are generated from these by the propositional connectives and the type quantifiers.

$$\begin{aligned}
&T ::= N \mid T \otimes T \mid Set(T)\\
&t ::= x \mid 0 \mid t^{+}\\
&\quad (t,t) \mid \pi_1(t) \mid \pi_2(t)\\
&\quad \emptyset_T \mid t \circledast t\\
&\phi ::= \Omega \mid t = t \mid t \in t \mid t < t\\
&\quad \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi\\
&\quad \forall x : T \cdot \phi \mid \exists x : T \cdot \phi
\end{aligned}$$

Where e is a term or wff, we write $e[t_1, .., t_n/x_1, .., x_n]$ for the substitution of the terms t_i for the variables x_i. Type membership is defined as follows: $t : T \triangleq \exists x : T \cdot x = t$.

We present the logic in a sequent style, where sequents take the following form $\Gamma \vdash \phi$, where Γ is a finite set of wff. The system is determined by the following axioms and rules. The structural rules and the equality axiom schemes take the obvious shape.

$$\begin{array}{llll}
\mathbf{A} & \phi \vdash \phi & \mathbf{W} & \dfrac{\Gamma \vdash \psi}{\Gamma, \phi \vdash \psi}\\
\mathbf{E_1} & x : T \vdash x = x & & \\
\mathbf{E_2} & x : T, y : T \vdash x = y \rightarrow (\phi[x] \rightarrow \phi[y]) & &
\end{array}$$

The rules for the propositional connectives are the standard classical ones, and the following quantifier rules are subject to the normal side conditions about dependency.

$$\begin{array}{llll}
\forall\mathbf{i} & \dfrac{\Gamma, x : A \vdash \phi}{\Gamma \vdash \forall x : A \cdot \phi} & \forall\mathbf{e} & \dfrac{\Gamma \vdash \forall x : A \cdot \phi \qquad \Gamma \vdash t : A}{\Gamma \vdash \phi[t/x]}\\
\exists\mathbf{i} & \dfrac{\Gamma \vdash \phi[t/x] \qquad \Gamma \vdash t : A}{\Gamma \vdash \exists x : A \cdot \phi} & \exists\mathbf{e} & \dfrac{\Gamma \vdash \exists x : A \cdot \phi \qquad \Gamma, x : A, \phi \vdash \eta}{\Gamma \vdash \eta}
\end{array}$$

The axioms for the numbers are given as follows.

$$\begin{array}{ll}
\mathbf{N_1} & 0 : N \\
\mathbf{N_2} & x : N \vdash x^+ : N \\
\mathbf{N_3} & x : N \vdash x^+ \neq 0 \\
\mathbf{N_4} & x : N, y : N \vdash x^+ = y^+ \to x = y \\
\mathbf{N_5} & \dfrac{\phi[0] \qquad\qquad x : N \vdash \phi[x] \to \phi[x^+]}{x : N \vdash \phi[x]} \\
\mathbf{N_6} & x : N \vdash \neg(x < 0) \\
\mathbf{N_7} & x : N, y : N \vdash x < y^+ \leftrightarrow (x < y \vee x = y)
\end{array}$$

Cartesian products are governed by the following axioms.

$$\begin{array}{ll}
\mathbf{P_1} & x : A, y : B \vdash (x, y) : A \otimes B \\
\mathbf{P_2} & z : A \otimes B \vdash \pi_1(z) : A \\
\mathbf{P}_3 & z : A \otimes B \vdash \pi_2(z) : B \\
\mathbf{P_4} & x : A, y : B \vdash \pi_1(x, y) = x \wedge \pi_2(x, y) = y \\
\mathbf{P_5} & x : A \otimes B \vdash x = (\pi_1(x), \pi_2(x))
\end{array}$$

Finally, the axioms for sets parallel those for numbers.

$$\begin{array}{ll}
\mathbf{S_1} & \emptyset_A : Set(A) \\
\mathbf{S_2} & x : A, y : Set(A) \vdash x \circledast y : Set(A) \\
\mathbf{S_3} & x : A, y : Set(A) \vdash x \circledast (x \circledast y) = x \circledast y \\
\mathbf{S}_4 & x : A, y : A, z : Set(A) \vdash x \circledast (y \circledast z) = y \circledast (x \circledast z) \\
\mathbf{S_5} & \dfrac{\phi[\emptyset_A] \qquad\qquad x : A, y : Set(A) \vdash \phi[y] \to \phi[x \circledast y]}{y : Set(A) \vdash \phi[y]} \\
\mathbf{S_6} & x : A \vdash x \notin \emptyset_A \\
\mathbf{S_7} & y : Set(A), x : A, z \in A \vdash z \in x \circledast y \leftrightarrow (z = x \vee z \in y)
\end{array}$$

This completes the statement of the **CST**. Note that the following is provable, i.e. sets are extensional.

Proposition 1. *(Extensionality)*

$$\forall x : Set(A) \cdot \forall y : Set(A) \cdot (\forall z : A \cdot z \in x \leftrightarrow z \in y) \to x = y$$

We shall now enrich the language slightly by adding *type variables*. They will only play a schematic role as place holders in our treatment of *polymorphic specifications*. While we still employ upper case Roman letters for type terms, we now reserve $X, X_1, X_2, , ..., Y, Y_1, Y_2, .., Z, ...$ for type variables. The type terms are now allowed to include type variables as atomic type terms, and all the rules and axioms are extended to this extended language. Call this extended theory $\mathbf{CST}^+$. Note that the following is derivable in this theory - where the type variables $X_1, .., X_n$ occur in Γ, ϕ and $\Gamma[T_1, .., T_n]$ and $\phi[T_1, .., T_n]$ are the result of replacing X_i by T_i, $1 \leq i \leq n$.

$$\frac{\Gamma \vdash \phi}{\Gamma[T_1, .., T_n] \vdash \phi[T_1, .., T_n]} \qquad \textbf{(SUB)}$$

To see this, one has only to note that each axiom and rule is stated for arbitrary type terms, and so, if it holds for the case where the type variables occur, it holds when $T_1, .., T_n$ replaces them. Similar considerations yield that $\mathbf{CST}^+$ is a conservative extension of **CST**. So we shall use **CST** for both.

3 Computability

Our notion of computability for **CST** owes much to [6], [1], [3] and unpublished work by Wilfred Hodges. In particular, they argue that computability can be regarded as Σ-definability. However, unike the above, **CST** has types. This brings some twists and turns, including the need to include products and numbers.

Definition 1. *The class of* $\mathbf{\Sigma}-$***Wff*** *of* **CST** *are given by the following syntax.*

$$\begin{aligned}&\sigma ::= \alpha \mid \neg\alpha \mid \sigma \wedge \sigma \mid \sigma \vee \sigma \mid \exists x : T \cdot \sigma\\ &\textit{where}\\ &\alpha ::= \ \Omega \mid t = t \mid t \in t \mid t < t\end{aligned}$$

The class of $\Pi-$***Wff*** *are obtained by replacing, in the above syntax,* $\exists x : T \cdot \sigma$ *by* $\forall x : T \cdot \sigma$. *Let* σ *be a* $\mathbf{\Sigma}-$*wff and* π *a* $\Pi-$*wff. If*

$$\mathbf{CST} \vdash \sigma \leftrightarrow \pi$$

Then we shall say that σ *is* Δ *or* ***Computable****.*

The proof-theoretic strength of a theory is largely determined by the strength of its induction principles. So we consider a sub-theory in which these are restricted to Σ-wff. More precisely, in the rules $\mathbf{N}_5$ and $\mathbf{S}_5$, ϕ must be a Σ-wff. We shall name the theory **CST** with these restricted induction principles, $\mathbf{CST} \restriction \mathbf{\Sigma}$**.**

4 Specifications

In system specification, new relation/function symbols are introduced/specified. We first deal with relations. We specify new relations via the following definitional style - a style reminiscent of Z.

Definition 2. *Let* ϕ *be any* $\Sigma-$*wff. Let* $T_1, ..., T_k$ *be distinct type terms and let* $x_1, ..., x_k$ $(k \geq 0)$ *be all the free variables of* ϕ. *Furthermore, let* $X_1, .., X_n$ *be exactly the type variables that occur in at least one of the types* $T_1, ..., T_k$, *and include all the type variables of* ϕ. *Then a* ***Schema Specification*** *has the form*

$$R \triangleq [x_1 : T_1, .., x_k : T_k \mid \phi[R, x_1, .., x_k]] \qquad \textbf{(S)}$$

We shall call the type information the ***Declaration*** *and the wff the* ***Predicate****. We extend the definition of* **Σ-*wff*** *to include all atomic wff of the form* $R(t_1, .., t_k)$.

This introduces a new k-place relation symbol R into the language. Notice that we permit R to occur recursively in the predicate. To facilitate matters, we assume that ϕ contains some k-place predicate variable introduced for substitution purposes i.e. $\phi[R, x_1, .., x_k]$ is obtained by substituting R for it.

These specifications are to be understood as *contextual definitions*. More formally, **S** introduces a new relation R that satisfies all *type substitution instances* of the following.

$$x_1 : T_1, .., x_k : T_k \vdash R(x_1, .., x_k) \leftrightarrow \phi[R, x_1, .., x_k] \qquad \textbf{(Rel)}$$

i.e. every instance that may be obtained by replacing, in the declaration and the predicate, X_i by some A_i, $1 \leq i \leq n$. This ensures that the relation is *polymorphic* and that the **SUB** rule still applies to the extended theory. Following Z, we shall also write schema in their more graphic form.

Example 1. The following is recursive specification of addition.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Add}} \\ \hline
x : N, y : N, z : N \\ \hline
(y = 0 \wedge x = z) \vee \\
y \neq 0 \wedge \exists u : N \cdot y = u^+ \wedge \exists w : N \cdot z = w^+ \wedge \mathit{Add}(x, u, w) \\ \hline
\end{array}$$

Example 2. Provided that ϕ is a Σ-wff, we may introduce bounded quantifiers as follows.

$$\begin{array}{|l}
\multicolumn{1}{l}{\forall_\phi} \\ \hline
y : \mathit{Set}(X) \\ \hline
(y = \emptyset_X) \vee \\
\exists u : X \cdot \exists v : \mathit{Set}(X) \cdot y = u \circledast v \wedge \phi[u] \wedge \forall_\phi(v) \\ \hline
\end{array}$$

Written in their familiar guise, these satisfy all type substitution instances of the following.

$$y : \mathit{Set}(X) \vdash \forall x \in y \cdot \phi \leftrightarrow \forall x : X \cdot x \in y \rightarrow \phi$$
$$y : \mathit{Set}(X) \vdash \exists x \in y \cdot \phi \leftrightarrow \exists x : X \cdot x \in y \wedge \phi$$

This enables us to specify the following set constructors of standard set theory.

Example 3. (**Power Set**)

$$\begin{array}{|l}\hline \textit{Power} \\ u : Set(X), v : Set(Set(X)) \\ \hline (\forall x \in v \cdot x \subseteq u) \wedge \emptyset \in v \wedge \forall z \in u \cdot \forall w \in v \cdot z \circledast w \in v \\ \hline \end{array}$$

Example 4. (**Generalised Union**)

$$\begin{array}{|l}\hline \cup \\ u : Set(Set(X), v : Set(X) \\ \hline (\forall x \in v \cdot \exists w \in u \cdot x \in w) \wedge (\forall w \in u \cdot x \in w \rightarrow x \in v) \\ \hline \end{array}$$

The next example operation is also schematic with respect to the included wff: it is a specification of *separation* for sets.

Example 5. (**Separation**) Let ψ be Σ-wff.

$$\begin{array}{|l}\hline Sep^{\psi} \\ u : Set(X), v : Set(X) \\ \hline (\forall z \in v \cdot z \in u \wedge \psi[z]) \wedge \forall z \in u \cdot \psi[z] \rightarrow z \in v \\ \hline \end{array}$$

There are some important special cases of schema that are theoretically and practically significant.

Definition 3. *Let $R \triangleq [x : I, y : O \mid \psi]$. Define the **Domain** of R as follows.*

$$DomR = [x : I \mid \exists y : O \cdot \psi]$$

*We shall say that R defines a **Total** relation if $\forall x : I \cdot DomR(x)$ and that R is **Functional** iff $\forall x : I \cdot \exists^{\leq 1} y : O \cdot \psi[x, y]$. By **SUB**, if these hold. they do so for all instances.*

Proposition 2. *(* $\mathbf{CST} \restriction \mathbf{\Sigma}$ *) Generalised Union, Power and Separation are total and functional*

Proof. By restricted induction and extensionality.■

However, relation specifications, even when functional, have not been introduced as *genuine* function symbols which can be applied to arguments in the standard way. This is the content of the following.

Definition 4. *Let* $R \triangleq [x : I, y : O \mid \psi]$ *be functional. We may then introduce a new function symbol into the language via the following* ***Recursive Function Schema***

$$F \triangleq_{Pun} [x : I, y : O \mid \psi[F, x, y]] \qquad \textbf{(FS)}$$

*We shall call this a Σ-**Function**. The class of $\mathbf{\Sigma}$-wff is extended to include new terms of the form $F(t)$ where F is a Σ-function i.e. the atomic wff can now include these terms. In the special case where R is also total, we shall write Fun for Pun.*

The specification **FS** is intended to introduce a new function symbol into the language and, in particular, $F(t)$ is a new term. **FS** is intended to go beyond **S**. More specifically, given **PF**, **FS** is to be logically interpreted as the introduction of a new function symbol that satisfies the following.

$$x : I \vdash F(x) : O \qquad (\mathbf{F}_1)$$

$$x : I \vdash (\exists y : O \cdot \psi[x, y]) \rightarrow \psi[x, F(x)] \qquad (\mathbf{F}_2)$$

Every time we introduce a new relational or function symbol, we enrich the language and the theory. However, we require that specifications should generate *conservative extensions*: otherwise, we would have no guarantee that the properties of the old theory, which we may rely on during specification, are maintained. Suppose that we have extended the language of **CST** with a new *well-typed*[2] relation R. Let $\mathbf{CST^R}$ be the theory in which all the rules of **CST** are extended to this new language, together with the axiom *Rel*. Similarly, let $\mathbf{CST^F}$ be the theory with a new *well-typed* function symbol F and where all the rules of **CST** are extended to this new language together with the axioms $\mathbf{F}_1$, $\mathbf{F}_2$. The details of the following can be found in [12]. The proof is in three stages that we shall now sketch.

– Use a translation to show that non-recursive specifications are conservative. For example, for relations, we translate them away as follows, where we illustrate with:

$$R \triangleq [x : T[X] \mid \phi[X, x]]$$

We translate relative to a type assignment context c that assigns types to variables

$$R(t)^c = \phi[A, t] \textit{ where relative to } c,\ t \textit{ has the unique type } T[A]$$

[2] An atomic wff of the form $a \in b$ is **well-typed** if, in the context, a has type T and b has type $Set(T)$. One of the form $a = b$ is **well-typed** if, in the context, a has type T and b has type T. Roughly, in a given declaration context, a wff is *well-typed* if all its atomic wff are. We say roughly, since $\forall x : T \cdot \phi$ will be *well-typed* in a context c, if ϕ is in c augmented by the declaration $x : T$.

The other wff translate compositionally -except the quantifiers that translate as

$$(\forall x : T \cdot \phi)^c = \forall x : T \cdot \phi^{c,x:T}$$

- Show that specifications that are *monotone* and *compact* can be unpacked in terms of non-recursive ones. For example, if the following recursive specification is monotone and compact

$$R \triangleq [x : A \mid \phi[R, x]]$$

It can be unpacked from *below* as:

$$R \triangleq \begin{bmatrix} x : A \mid \exists w : N \cdot w > 0 \wedge \exists z : Set(N \otimes Set(A)) \cdot Map(z) \wedge x \in z(w) \wedge \\ Dom(z) = [k] \wedge \forall y < w \cdot \forall u \in z(y) \cdot \phi[u, z(y)] \end{bmatrix}$$

- The argument is completed by showing that Σ-wff are *monotone* and *compact*. This is by induction on the structure of Σ-wff, and is standard.

Theorem 1. ***CST*$^{\mathbf{R}}$** *and* ***CST*$^{\mathbf{F}}$** *are conservative extensions of* ***CST***

5 Refinement

We have offered, by way of our examples, some evidence that computable specifications are enough for practice. Extensionally, they surely ought to be. However, they are not always as elegant and transparent as one might want. The following idea addresses this worry.

Definition 5. *Let*

$$R \triangleq [x : I, y : O \mid \psi]$$
$$S \triangleq [x : I, y : O \mid \eta]$$

We shall say that S *is a* ***Refinement*** *of* R, *written as* $R \sqsubseteq S$, *iff*

$$\forall x : I \cdot DomR(x) \rightarrow DomS(x) \tag{1}$$
$$\forall x : I \cdot DomR(x) \rightarrow \forall y : O \cdot \psi[x, y] \leftrightarrow \eta[x, y] \tag{2}$$

This gives us more expressive power. Our suggestion is that we should only employ non-sigma specifications, if they can be refined to Σ-ones. For example, our original power set operation is a refinement of the following more obvious and transparent one.

$$\begin{array}{|l} \multicolumn{1}{l}{Power'} \\ \hline u : Set(X), v : Set(Set(X)) \\ \hline \forall x : Set(X) \cdot x \in v \leftrightarrow x \subseteq u \\ \hline \end{array}$$

There are more demanding examples, but we require more infrastructure to set them up. In case the reader thinks that all specifications can be refined to Σ ones, any numerical relation that is not Recursively enumerable, supplies a counter-example.

6 Models in PA

Since the following constructions are well documented in [5], we shall only sketch matters. Let $\mathcal{N} =< N, 0, ^+ >$ be any model of **PA**. We represent pairing in the model in the standard way. It is primitive recursive function and so can be represented in **PA**. Moreover, it is a matter of arithmetic, that this is injective and surjective and that there are **PA** representable functions π_1, π_2 that satisfy the standard axioms. In the model we define: $A \otimes B \triangleq \{(x, y) \cdot x \in A \wedge y \in B\}$. For the representation of finite sets, we require the following result from [5].

- For each x, y, there are unique $u \leq y, v \leq 1, w \leq 2^x$ such that $y = u \times 2^{x+1} + v \times 2^x + w$. Then define: $x \in y \triangleq Bit(x, y) = 1$ where $Bit(x, y)$ is the unique $v \leq 1$ such that: $\exists u \leq y \cdot \exists w < 2^x \cdot y = u \times 2^{x+1} + v \times 2^x + w$. The proof of the following may also be found in [5].
- In **PA** we have: $\forall x \cdot \forall y \cdot \exists! z \cdot \forall u \cdot u \in z \leftrightarrow u = x \vee u \in y$. This provides a representation of union, and hence insertion. This leaves us to represent the *Set* constructor. Here we appeal to the representation of recursive predicates in (**PA**). First, we add to **PA** a unary numerical predicate symbol X. The following is a standard result from formal number theory.
- For each Σ-wff ϕ, there is a Σ-definable θ such that

  1. $\forall x \cdot \phi[\theta, x] \rightarrow \theta[x]$
  2. $[\forall x \cdot \phi[\psi, x] \rightarrow \psi[x]] \rightarrow \forall x \cdot \theta[x] \rightarrow \psi[x]$

  where $\phi[\theta]$ is to be interpreted as ϕ with every occurrence of any wff of the form $\theta[u]$ substituted for $X(u)$.

To construct a representation of the *Set* constructor, we first apply this result to the Σ-wff: $\phi[x] = x = 0 \vee \exists u : A \cdot \exists v \cdot X \cdot x = u \circledast v$. Given the θ for this ϕ, we represents the set constructor in the model as: $Set[A] = \{x \cdot \theta[x]\}$. This leads to the following.

Lemma 1. *Every model of* ***PA*** *may be expanded to a model of* ***CST****.*

Standard argumentation then yields:

Theorem 2. ***CST*** *is a conservative extension of* ***PA****. Indeed* **CST** $\restriction \mathbf{\Sigma}$ *is conservative over primitive recursive arithmetic.*

This whole model construction can be turned into a translation where the Σ-wff are translated to those of **PA**. We conclude that given these computability considerations, specification theories need not have the proof theoretic power of Higher order logic, let alone **ZF** set theory.

References

1. Barwise, Jon. Admissible Sets and Structures. Springer Verlag.
2. Brien, S.M. and Nicholls, J.E. Z *Based Standard Version 1.0*, Oxford University Computing Laboratory. PRG 107, 1992.
3. Ershov, Y.L. Definability and Computability, Nauka, Novosibirsk, 1996.
4. Gibbins, P.F. **VDM**: *Axiomatising its propositional logic.* BCS Computer Journal, Vol. 31, pp. 510-516, 1988.
5. Hajek, Petr. and Pudlak, Pavel. *Metamathematics of First Order Arithmetic.* Springer-Verlag,1991.
6. Montague, R. Recursion Theory as a branch of model theory. In Logic, Methodogology and Philosophy of Science III. Proc. of the 1967 Congress., B. van Roostelaar et al. editors, North Holland publishing company.,Amsterdam, pp63-86, 1968
7. Middelburg, K. *The Logical Semantics of Flat VDMSL.* Technical Report, Neher Laboratories, Number 954 RNL/89, December 1989.
8. Moore, R. and Ritchie, J. *Proof in VDM-A practitioners Guide*, FACIT, Springer Verlag, ISBN 3-540-19813-X, 1994.
9. Nicholls, J.E. (ed). *Z-Notation version 1.2,* 1995.
10. Owre, S. and Shankar, N. *The Formal Semantics of PVS.* NASA/CR-1999-209321, May 1999.
11. Bergstra, J.A. and Tucker, J.V. Algebraic specifications of Computable and semicomputable data types, Theoretical Computer Science, 50 (1987). pp 137-181.
12. Turner, R. *Foundations of Specification I, II.* Essex University Technical Reports. CSM-396,397. To appear in the Journal of Logic and Computation.

Kleene-Kreisel Functionals and Computational Complexity

Paul J. Voda[1] and Lars Kristiansen[2]

[1] Department of Applied Informatics
Faculty of Mathematics, Physics and Informatics
Comenius University Bratislava
Mlynská dolina
842 48 Bratislava, Slovakia
`http://www.fmph.uniba.sk/~voda`
`voda@fmph.uniba.sk`

[2] Faculty of Engineering, Høgskolen i Oslo
Cort Adelers gate 30
N-0254 Oslo, Norway
`larskri@iu.hio.no`
`http://www.iu.hio.no/~larskri`

In [2] we have asked a question *What do we get if we replace the constant* 0 *of Gödel's primitive recursive functionals (system T) by* 1 *and omit the successor function S?* The resulting system T^- is seemingly weak because no function can grow beyond the maximum of its arguments and 1, but rather surprisingly, the predicates of T^- are exactly the Kalmár elementary predicates.

Let T_k^- denote the class of predicates we get when the recursors in the system T^- are restricted to yield values of at most type level k. Here the type 0 has level 0 and $\sigma \to \tau$ has level $\max(n+1, m)$ where n and m are the levels of σ and τ respectively. We have proved in [3] that the predicates in the hierarchy $T_0^- \subseteq T_1^- \subseteq T_2^- \subseteq T_3^- \subseteq T_4^- \ldots$ match level by level the complexity-theoretic alternating space-time hierarchy:

$$\mathrm{SPACE}(\mathrm{LIN}) \subseteq \mathrm{TIME}(2^{\mathrm{LIN}}) \subseteq \mathrm{SPACE}(2^{\mathrm{LIN}}) \subseteq \mathrm{TIME}(2^{2^{\mathrm{LIN}}}) \subseteq \mathrm{SPACE}(2^{2^{\mathrm{LIN}}}) \ldots$$

Suppose that we encode the words of finite alphabets as functions of type $0 \to 0$ and consider the $0,1$-valued functionals $F : (0 \to 0) \to 0 \to 0$ which are applied as $F(a, n)$ with n the length of the input word a. Such predicates F in the above hierarchy T_k^- match now the correspoding complexity classes:

$$\mathrm{LOGSPACE} \subseteq \mathrm{PTIME} \subseteq \mathrm{PSPACE} \subseteq \mathrm{POLYEXPTIME} \subseteq \mathrm{POLYEXPSPACE} \ldots$$

A characterization similar to the last one was obtained in [1]. There is a crucial difference, however. The authors of [1] work with finite functionals where the type 0 is restricted to the numbers less than n. Our functionals are infinite because the type 0 is interpreted as the domain of natural numbers.

This parallel suggested to us that the infinite functionals can be viewed as limits of finite ones where the cardinality n of the type 0 grows towards infinity. The construction can be compared to the definition of reals by the limits of Cauchy sequences of rationals. A couple of years ago we were able to define a class of, what we called, *convergent functionals*, as limits of finite ones. We have almost characterized (we did not have a proof of a single lemma) the convergent functionals as the Kleene-Kreisel functionals which are the total functionals from the class of Scott-Ershov partial continuous functionals. The first author presented in the June of 2003 our work on convergent functionals at a logic seminar in Oslo where Dag Norman, who was sitting in the audience, immediately recognized the construction as equivalent to a model-theoretical one by reduced products which he had co-authored in [4].

This characterization remains relatively unknown although it gives a strikingly simple semantics to the Kleene-Kreisel functionals. Moreover, it has a direct connection to the functionals over finite domains which are so important from the point of view of complexity. To remedy the situation we will present our limit construction where, as an improvement over [4], we have been able to discover for every Kleene-Kreisel functional the existence of a canonical sequence of finite functionals whose limit is the functional. The canonical sequence converges to the limit without any detours, i.e. as soon as possible. The natural definitions of functionals from T^- happen to be by their canonical sequences.

We will present as a further result a theorem similar to the famous *trade-off* theorem of Schwichtenberg [5]. We think that our theorem nicely illuminates the surprising alternation of space and time in the above hierarchies. Schwichtenberg's theorem permits to trade in the definition of (infinite) functionals the detours through type levels $k+1$ yielded by α-ordinal recursion for exponentially longer recursion of length w^α yielding values of type level k only. Our theorem permits to trade the detours through the types of level $2k+i$, $i=0,1$ (standing for space) with recursion of length n (standing for time) for recursion of length 2^n_{k+i} (the tower of $k+i$ exponentials) which yields values of type levels k. Finite functionals of type levels k can be coded in space 2^n_k. With $i=0$ we obtain a space class and with $i=1$ a time class.

References

1. A. Goerdt, H. Seidl. *Characterizing complexity classes by higher type primitive recursive definitions*, Lecture Notes in Computer Science LNCS vol. 464, Springer 1990.
2. L. Kristiansen, P. J. Voda. *The surprising power of restricted programs and Goedel's functionals*, Computer Science Logic Conf. CSL'03, LNCS vol. 2803, Springer Verlag, 2003
3. L. Kristiansen, P. J. Voda. *Programming languages capturing complexity classes* Nordic Workshop on Programming Theory NWPT-04 Uppsala, 2004.
4. D. Normann, E. Palmgren, V. Stoltenberg-Hansen. *Hyperfinite type structures*, Journal of Symbolic Logic, vol. 64 1216-1242, 1999.
5. H. Schwichtenberg. *Elimination of higher type levels in definitions of primitive recursive functionals*, in Logic Colloquium' 73, eds. Rose, Shepherdson. North Holland 1974.

A Very Slow Growing Hierarchy for the Howard Bachmann Ordinal

Andreas Weiermann[⋆]

Mathematical Institute of Utrecht University
Postbox 80010, 3508 Utrecht, The Netherlands,
`weierman@math.uu.nl`

Abstract. Continuing [4] we investigate natural systems of fundamental sequences for the limits less than the Howard Bachmann ordinal η_0 and show that there is a (natural) Bachmann system of fundamental sequences such that the resulting slow growing hierarchy $(G_\alpha)_{\alpha<\eta_0}$ does not majorize the elementary recursive functions. This result is surprising since usually the slow growing hierarchy along η_0 classifies the provably recursive functions of PA. The assignment we use is natural in so far as the resulting fast growing Hardy hierarchy $(H_\alpha)_{\alpha<\eta_0}$ (cf. [2]) classifies the provably recursive functions of ID_1.
The paper is based on an involved technical verification. From a technical point of view it only requires basic familiarity with [1].

This article is motivated by the classical classification problem for the recursive functions and the resulting problem of comparing the slow and fast growing hierarchies. It has been claimed, for example in [3] p. 439 l.-5, that for sufficiently big prooftheoretic ordinals the slow and fast growing hierarchies will match up. The results of this paper indicate that this claim might not be true in general.

For an ordinal α less than the Howard Bachmann ordinal let $N\alpha$ be the number of symbols in α which are different from 0 and $+$. For a limit ordinal λ let $\lambda[x] := \max\{\beta < \lambda : N\beta \leq N\lambda + x\}$. This assignment of fundamental sequences is natural and does not change the growth rate of the induced fast growing hierarchy but the induced slow growing hierarchy (along the Howard Bachmann ordinal) consists of elementary functions only.

It is further somewhat surprising that we encounter the following sharp threshold phenomenon for the slow growing hierarchy. For a given real number $\varepsilon > 0$ let $\lambda_\varepsilon[x] := \max\{\beta < \lambda : N\beta \leq (1+\varepsilon)\cdot N\lambda + x\}$. Then for $\varepsilon = 0$ the resulting slow growing hierarchy is very slow growing whereas for any $\varepsilon > 0$ the resulting slow growing hierarchy becomes fast growing and matches up with the fast growing hierarchy at the ordinal of PA. We conjecture that within the phase transition, i.e. when in the definition of $\lambda_\varepsilon[x]$ the number ε is a function of λ and x, we may arrange other behaviours of the resulting slow growing hierarchy.

[⋆] The author is a Heisenberg fellow of the DFG and has been supported in part by DFG grant We 2178 6/1

Definition 1. *Inductive definition of a set of terms* T *and a set* $\mathrm{P} \subseteq T$.
$0 \in \mathrm{T}$,
$a \in \mathrm{T} \,\&\, i \in \{0,1\} \;\Rightarrow\; D_i a \in \mathrm{P}$,
$a_0, \ldots, a_n \in P$ *and* $n \geq 1 \Rightarrow\; (a_0, \ldots, a_n) \in \mathrm{T}$.

Notations:
1. a, b, c, d, e range over T.
2. If $a \in P$, then we identify the one element sequence (a) with the term a.
3. The empty sequence $(\,)$ is identified with the term 0.
4. x, y, z, i, l, m, n range over non negative integers.

Definition 2. *Recursive definition of* $a \prec b$ *for* $a, b \in \mathrm{T}$.
$a \prec b$ holds, iff one of the following cases holds:
1. $a = 0$ *and* $b \neq 0$,
2. $(a = D_0 a_0 \,\&\, b = D_1 b_0)$ *or*
3. $(a = D_i a_0 \,\&\, b = D_i b_0 \,\&\, a_0 \prec b_0)$,
4. $a = (a_0, \ldots, a_m) \,\&\, b = (b_0, \ldots, b_n) \,\&\, 1 \leq m + n$ *and*
$[(m < n \,\&\, \forall i \leq n(a_i = b_i))$ *or* $(\exists k \leq \min\{m, n\}(\forall i < k(a_i = b_i) \,\&\, a_k \prec b_k))]$

Lemma 1. $(\mathrm{T}, \prec)$ *is a linear order.*

Definition 3. *Assume that* $M, N \subseteq \mathrm{T}$.
$M \preceq N \;:\Longleftrightarrow\; \forall x \in M \exists y \in N(x \preceq y)$.
$a \preceq N \;:\Longleftrightarrow\; \{a\} \preceq N$.
$M \prec a \;:\Longleftrightarrow\; \forall x \in M(x \prec a)$.
$(a_1, \ldots, a_m) + (b_1, \ldots, b_n) := (a_1, \ldots, a_i, b_1, \ldots, b_n)$
where $i \in \{1, \ldots, m\}$ *is maximal with* $b_1 \preceq a_i$.

Definition 4. *Recursive definition of* $\mathrm{K}^* a$ *and* $\mathrm{K} a$ *for* $a \in \mathrm{T}$.
$\mathrm{K}^* 0 := \emptyset, \quad \mathrm{K} 0 := \emptyset$,
$\mathrm{K}^*(a_0, \ldots, a_n) := \bigcup_{i \leq n} \mathrm{K}^* a_i, \quad \mathrm{K}(a_0, \ldots, a_n) := \bigcup_{i \leq n} \mathrm{K} a_i$,
$\mathrm{K}^* D_1 a := \mathrm{K}^* a, \quad \mathrm{K} D_1 a := \mathrm{K} a$,
$\mathrm{K}^* D_0 a := \{a\} \cup \mathrm{K}^* a, \quad \mathrm{K} D_0 a := \{D_0 a\}$.

Lemma 2. $\mathrm{K}^* c \prec a \;\Longrightarrow\; \mathrm{K} c \prec D_0 a$.

Definition 5. *Inductive definition of a set of terms* $\mathrm{OT} \subseteq \mathrm{T}$.
$0 \in \mathrm{OT}$,
$a_0, \ldots, a_n \in \mathrm{OT} \cap P$, $n \geq 1$ *and* $a_n \preceq \ldots \preceq a_0 \;\Longrightarrow\; (a_0, \ldots, a_n) \in \mathrm{OT}$,
$a \in \mathrm{OT} \;\Longrightarrow\; D_1 a \in \mathrm{OT}$,
$a \in \mathrm{OT} \,\&\, \mathrm{K}^* a \prec a \;\Longrightarrow\; D_0 a \in \mathrm{OT}$.

Notations.
$1 := D_0 0$, $\omega := D_0 1$, $\Omega := D_1 0$.
$\mathrm{T}_0 := \{x \in \mathrm{T} : x \prec \Omega\}, \quad \mathrm{OT}_0 := \mathrm{OT} \cap \mathrm{T}_0$.

Theorem 1. $(\mathrm{OT}, \prec)$ *is a well-order. The order type of* $(\mathrm{OT}_0, \prec)$ *is equal to* η_0, *the Howard Bachmann ordinal.*

Definition 6. $b \lhd_c a \; :\Longleftrightarrow \quad b \prec a \;\&\; \forall d \in \mathrm{OT}\; (b \preceq d \preceq a \Rightarrow \mathrm{K}^* b \preceq \mathrm{K}^* d \cup \mathrm{K}^* c)$ *[*$a, b, c \in \mathrm{T}$*]..*

Lemma 3. *1.* $a \prec D_1 a$.
2. $a, b \in \mathrm{OT}_0 \;\&\; a \prec b \implies \mathrm{K}^* a \preceq \mathrm{K}^* b$.
3. $b \lhd_c a \;\&\; \mathrm{K}^* a \prec a \;\&\; \mathrm{K}^* c \prec b$
$\&\; a, b \in \mathrm{OT} \implies \mathrm{K}^* b \prec b$.
4. $b \lhd_c a \implies d + b \lhd_c d + a \;\&$
$D_i b \lhd_c D_i a \;(i \in \{0, 1\})$.

Proof. We prove assertion 3. Assume that $b \preceq \mathrm{K}^* b$. Choose a subterm d of b with $b \preceq \mathrm{K}^* d$ such that the length of d is minimal possible. Then $d = D_0 e$ with $\mathrm{K}^* e \prec b \preceq e \preceq a$, since $\mathrm{K}^* b \preceq \mathrm{K}^* c \cup \mathrm{K}^* a \prec a$. Then we obtain $\mathrm{K}^* b \preceq \mathrm{K}^* c \cup \mathrm{K}^* e \prec b$. Contradiction.

Definition 7. *Definition of* $\mathsf{tp}(a) \in \{0, 1, \omega, \Omega\}$ *for* $a \in \mathrm{T}$.
$\mathsf{tp}(0) := 0$.
$\mathsf{tp}(1) := 1$.
$\mathsf{tp}(\Omega) := \Omega$.
$\mathsf{tp}(a) = 1 \;\Rightarrow\; \mathsf{tp}(D_i a) := \omega$.
$\mathsf{tp}(a) = \omega \;\Rightarrow\; \mathsf{tp}(D_i a) := \omega$
$\mathsf{tp}(a) = \Omega \;\Rightarrow\; \mathsf{tp}(D_0 a) := \omega$.
$\mathsf{tp}(a) = \Omega \;\Rightarrow\; \mathsf{tp}(D_1 a) := \Omega$.
$\mathsf{tp}((a_0, \ldots, a_n)) := \mathsf{tp}(a_n)$.

Definition 8. *Recursive definition of* $a\{c\} \in \mathrm{T}$ *for* $c \in T_0$ *and* $a \in \mathrm{T}$ *with* $tp(a) = \Omega$.
$\Omega\{c\} := c$.
$(D_1 a)\{c\} := D_1 a\{c\}$.
$a = (a_0, \ldots, a_n) \;\Rightarrow$
$a\{c\} := (a_0, \ldots, a_{n-1}) + a_n\{c\}$.

Lemma 4. *1.* $\mathsf{tp}(a) = \Omega \;\&\; c \in T_0 \;\Rightarrow\; a\{c\} \lhd_c a$.
2. $\mathsf{tp}(a) = \Omega \;\&\; c, d \in T_0 \;\&\; c \prec d \;\Rightarrow\; a\{c\} \prec a\{d\}$.
3. $\mathsf{tp}(a) = \Omega \;\&\; c \in T_0 \;\Rightarrow\; \mathrm{K}^* a\{c\} \preceq \mathrm{K}^* a\{0\} \cup \mathrm{K}^* c$.
4. $a \in \mathrm{OT}, \mathsf{tp}(a) = \Omega \;\&\; c \in OT_0 \;\Rightarrow\; a\{c\} \in \mathrm{OT}$.

Definition 9. *Recursive definition of* Na *for* $a \in T$.
$N0 := 0$.
$N(a_0, \ldots, a_n) := Na_0 + \ldots + Na_n$.
$ND_i a := 1 + Na$.

Definition 10. *Definition of* $a[x]$ *and* $a[\![x]\!]$ *for* $a \in \mathrm{OT}$.
$a[x] := \max\{b \in \mathrm{OT} : b \prec a \;\&\; Nb \le Na + x\}$.
$a[\![x]\!] := \max\{b \in \mathrm{OT} : b \prec a \;\&\; Nb \le x\}$.

Lemma 5. *1.* $(a_0, \ldots, a_{n-1}, a_n)[x] = (a_0, \ldots, a_{n-1}) + a_n[x]$.
2. $a = (a_0, \ldots, a_n)$, $b = (a_1, \ldots, a_n)$, $x \ge Na_0 \implies a[\![x]\!] = a_0 + b[\![x - Na_0]\!]$.

3. $a = (a_0, \dots, a_n)$, $x < Na_0 \implies a[\![x]\!] = a_0[\![x]\!]$.
4. $a[\![0]\!] = 0$.
5. $(D_1a)[x] = D_1a[x]$.
6. $x > 0 \implies (D_1a)[\![x]\!] = D_1a[\![x-1]\!]$.

Definition 11. *Recursive definition of $G_a(x)$ for $a \in \mathrm{OT}_0$.*
$G_0(x) := 0$.
$G_{a+1}(x) := G_a(x) + 1$.
$G_a(x) := G_{a[x]}(x)$ *if* $\mathsf{tp}(a) = \omega$.

Lemma 6. *Let $a, b \in OT_0$.*
1. $a = (a_0, \dots, a_n) \implies G_a(x) = G_{a_0}(x) + \cdots + G_{a_n}(x)$.
2. $a = D_0(b+1) \implies G_a(x) = G_{D_0b}(x) + G_{a[\![x+1]\!]}(x)$.
3. $a \preceq b$ & $Na \le x+1 \implies G_a(x) \le G_b(x)$.

Definition 12. *Definition of $T_a(x)$ for $a \in T$.*
$T_0(x) := 1$.
$T_a(x) := T_{a[\![x]\!]}(x) + 2$.

Remark. $T_a(x)$ is defined by recursion on the cardinality of the set $\{b \prec a : Nb \le x\}$.

Lemma 7. *1.* $a \preceq b$ & $Na \le x \Rightarrow T_a(x) \le T_b(x)$.
2. $a \preceq b \Rightarrow T_a(x) \le T_b(x)$.

Proof. 1. By induction on the cardinality of the set $\{c \prec b : Nc \le x\}$. Assume that $T_b(x) = T_{b[\![x]\!]}(x) + 2$. Then $a \preceq b[\![x]\!]$ and the induction hypothesis yields $T_a(x) \le T_{b[\![x]\!]}(x)$ and the assertion follows.
2. If $a = 0$ then the assertion is clear. Assume that $T_a(x) = T_{a[\![x]\!]}(x) + 2$ and $T_b(x) = T_{b[\![x]\!]}(x) + 2$. Then $a[\![x]\!] \preceq b[\![x]\!]$ and the assertion follows from 1.

Definition 13. *Recursive definition of $C_x(a,g)$ for $a \in OT$ and $g, x < \omega$.*
1. $C_x(0,g) := 0$.
2. $C_x((a_0, \dots, a_n), g) := C_x(a_0,g) + \cdots + C_x(a_n,g)$.
3. $C_x(D_0a, g) := g \cdot G_{D_0a}(x)$.
4. $C_x(D_1a, g) := g^{2^{T_{D_1a}(x+1)}} \cdot (C_x(a,g) + 1)$.

Lemma 8. *If $a \in OT_0$ then $C_x(a,g) = g \cdot G_a(x)$.*

Proof by induction on Na.
1. $a = 0$. Then the assertion is obvious.
2. $a = (a_0, \dots, a_n)$. Then the induction hypothesis yields $C_x(a,g) = C_x(a_0,g) + \cdots + C_x(a_n,g) = g \cdot G_{a_0}(x) + \cdots + g \cdot G_{a_n}(x) = g \cdot G_{(a_0,\dots,a_n)}(x)$.
3. $a = D_0b$. Then $C_x(a,g) = g \cdot G_a(x)$.

Lemma 9. *If $a, b \in \mathrm{OT}$, $\mathrm{K}^*b \prec a$, $Nb \le x+1$ and $g = G_{(D_0a)[\![x+1]\!]}(x)$ then $C_x(b,g) \le g^{2^{T_b(x+1)+1}}$.*

Proof by induction on Nb.
1. $b = 0$. Then the assertion is obvious.
2. $b = (b_0, \dots, b_n)$. Then $n + 1 \leq x + 1$, $Nb_i \leq x + 1$ and $\mathrm{K}^*b_i \subseteq \mathrm{K}^*b \prec a$ for $i = 0, \dots, n$. The induction hypothesis yields $C_x(b_i, g) \leq g^{2^{T_{b_i}(x+1)+1}}$ for $i = 0, \dots, n$. Thus $C_x(b, g) = C_x(b_0, g) + \cdots + C_x(b_n, g) \leq g^{2^{T_{b_0}(x+1)+1}} + \cdots + g^{2^{T_{b_n}(x+1)+1}} \leq (x + 1) \cdot g^{2^{T_b(x+1)-1}} \leq g^{2^{T_b(x+1)+1}}$,
since $x + 1 \leq g$ and $T_{b_i}(x + 1) \leq T_b(x + 1)$ by Lemma 7.
3. $b = D_0c$. Then $C_x(b, g) = g \cdot G_{D_0c}(x)$. $\mathrm{K}^*b \prec a$ yields $b \prec D_0a$. Thus $Nb \leq x+1$ yields $b \preceq (D_0a)[\![x + 1]\!]$ and $G_b(x) \leq G_{(D_0a)[\![x+1]\!]}(x)$ hence $C_x(b, g) \leq g^2$.
4. $b = D_1c$. Then the induction hypothesis yields $C_x(c, g) \leq g^{2^{T_c(x+1)+1}}$ hence $C_x(b, g) = g^{2^{T_{D_1c}(x+1)}} \cdot (C_x(c, g)+1) \leq g^{2^{T_{D_1c}(x+1)}} \cdot (g^{2^{T_c(x+1)+1}}+1) \leq g^{2^{T_{D_1c}(x+1)+1}} = g^{2^{T_b(x+1)+1}}$ since $T_c(x + 1) + 1 < T_{D_1c}(x + 1)$ because $c \prec D_1c$ and $Nc \leq x$.

Lemma 10. *If* $x < Na$ *then* $\mathrm{K}^*a[\![x]\!] \preceq \mathrm{K}^*a$.

Proof by induction on Na using Lemma 5.
1. $a = 0$. Then the assertion is obvious.
2. $a = (a_0, \dots, a_n)$. Let $b := (a_1, \dots, a_n)$.
2.1. $x < Na_0$. Then $a[\![x]\!] = a_0[\![x]\!]$ and the induction hypothesis yields $\mathrm{K}^*a[\![x]\!] = \mathrm{K}^*a_0[\![x]\!] \preceq \mathrm{K}^*a_0 \subseteq \mathrm{K}^*a$.
2.2. $x = Na_0$. Then $a[\![x]\!] = a_0$ and $\mathrm{K}^*a_0 \subseteq \mathrm{K}^*a$.
2.3. $x > Na_0$. Then $a[\![x]\!] = a_0 + b[\![x - Na_0]\!]$. $x < Na$ yields $x - Na_0 < Nb$ and the induction hypothesis yields $\mathrm{K}^*b[\![x - Na_0]\!] \preceq \mathrm{K}^*b$. Thus $\mathrm{K}^*a[\![x]\!] = \mathrm{K}^*a_0 \cup \mathrm{K}^*b[\![x - Na_0]\!] \preceq \mathrm{K}^*a_0 \cup \mathrm{K}^*b = \mathrm{K}^*a$.
3. $a = D_0b$. Then $a[\![x]\!] \prec a \prec \varOmega$, thus $\mathrm{K}^*a[\![x]\!] \preceq \mathrm{K}^*a$ by assertion 2 of Lemma 3.
4. $a = D_1b$. Then $Na = 1 + Nb$.
4.1. $b = 0$. Then $Na = 1$ hence $x = 0$ and the assertion is obvious.
4.2. $\mathsf{tp}(b) \in \{\omega, \varOmega\}$. We may assume that $x > 0$. Then $(D_1b)[\![x]\!] = D_1b[\![x - 1]\!]$. $x < Na$ yields $x - 1 < Nb$. The induction hypothesis yields $\mathrm{K}^*b[\![x - 1]\!] \preceq \mathrm{K}^*b$, hence $\mathrm{K}^*a[\![x]\!] \preceq \mathrm{K}^*a$.
4.3. $b = c + 1$. Then $a[\![x]\!] = D_1c \cdot y + (D_1c)[\![z]\!]$ where $ND_1c > z$. The induction hypothesis yields $\mathrm{K}^*(D_1c)[\![z]\!] \preceq \mathrm{K}^*D_1c$ hence $\mathrm{K}^*a[\![x]\!] \preceq \mathrm{K}^*a$.

Lemma 11. *If* $a, b \in OT$, $\mathsf{tp}(b) = \omega$ *and* $b[x] \preceq a \preceq b$ *then* $\mathrm{K}^*b[x] \leq \mathrm{K}^*a$.

Proof by induction on Nb.
1. $b = (b_0, \dots, b_n)$. Then $b[x] = (b_0, \dots, b_{n-1}) + b_n[x]$ by assertion 1 of Lemma 5. $b[x] \preceq a \preceq b$ yields $a = (b_0, \dots, b_{n-1}) + c$ for some c with $b_n[x] \preceq c \preceq b_n$. The induction hypothesis yields $\mathrm{K}^*b_n[x] \preceq \mathrm{K}^*c$ hence $\mathrm{K}^*b[x] \preceq \mathrm{K}^*a$.
2. $b = D_0c$. Then $b[x] \preceq a \preceq b \prec \varOmega$ and $\mathrm{K}^*b[x] \preceq \mathrm{K}^*a$.
3. $b = D_1c$.
3.1. $\mathsf{tp}(c) = \omega$. Then $b[x] = D_1c[x] \preceq a \preceq D_1c$. Thus $a = D_1d + e$ for some d with $c[x] \preceq d \preceq c$. The induction hypothesis yields $\mathrm{K}^*c[x] \preceq \mathrm{K}^*d$ hence $\mathrm{K}^*b[x] \preceq \mathrm{K}^*a$.
3.2. $c = d + 1$. Then $(D_1c)[x] = D_1d \cdot y + (D_1d)[\![z]\!]$ with $z < ND_1d$ and $y > 0$.

Lemma 10 yields $\mathrm{K}^*(D_1d)[\![z]\!] \leq \mathrm{K}^*D_1d$. $D_1d \cdot y + (D_1d)[\![z]\!] \preceq a \preceq D_1c$ yields $a = D_1d + e$ for some $e \prec D_1c$ hence $\mathrm{K}^*b[x] \preceq \mathrm{K}^*a$.

Lemma 12. *Assume that $b \in OT$.*
1. $b[x] \lhd_x b$.
2. $\mathrm{K}^*b[x] \prec b[x]$.
3. $D_0b[x] \in OT$.

Lemma 13. *Assume $D_0b \in \mathrm{OT}$. If $x < \omega$ and $\mathsf{tp}(b) = \omega$ then $D_0b[x] \in \mathrm{OT}$ and $(D_0b)[x] = D_0b[x]$.*

Proof. Lemma 12 yields $\mathrm{K}^*b[x] \prec b[x]$, hence $D_0b[x] \in OT$. By definition we have $D_0b[x] \preceq (D_0b)[x]$. Assume now that $(D_0b)[x] = D_0c + d$ for some $d \prec D_0(c+1)$. If $d \neq 0$ then $D_0(c+1)$ would be a better choice for $(D_0b)[x]$ than $D_0c + d$. Hence $d = 0$. We have $N(D_0b)[x] = 1 + Nb + x = 1 + Nc$, thus $Nc = Nb + x$. $c \prec b$ yields $c \preceq b[x]$ hence $D_0c \leq D_0b[x]$. Therefore $D_0b[x] = (D_0b)[x]$.

Lemma 14. *Assume that $a, b \in OT$, $\mathsf{tp}(b) = \omega$, $\mathrm{K}^*b \prec a$ and $g = G_{(D_0a)[\![x+1]\!]}(x)$. Then $C_x(b[x], g) \leq C_x(b, g)$.*

Proof by induction on b.
1. $b = (b_0, \ldots, b_n)$ with $\mathsf{tp}(b_n) = \omega$.
Then $\mathrm{K}^*b_n \prec a$ and the induction hypothesis yields $C_x(b_n[x], g) \leq C_x(b_n, g)$. Then $C_x(b[x], g) = C_x(b_0, g) + \cdots + C_x(b_n[x], g) \leq C_x(b_0, g) + \cdots + C_x(b_n, g) = C_x(b, g)$. 2. $b = D_0c$.
Then $b[x] \prec \Omega$ and Lemma 8 yields $C_x(b[x], g) = g \cdot G_{b[x]}(x) = g \cdot G_b(x) = C_x(b, g)$.
3. $b = D_1c$ where $\mathsf{tp}(c) = \omega$.
We have $\mathrm{K}^*c \subseteq \mathrm{K}^*b \prec a$ and the induction hypothesis yields $C_x(c[x], g) \leq C_x(c, g)$. Therefore Lemma 7 yields $C_x(b[x], g) = C_x(D_1c[x], g)$
$= g^{2^{T_{D_1c[x]}(x+1)}} \cdot (C_x(c[x], g) + 1) \leq g^{2^{T_{D_1c}(x+1)}} \cdot (C_x(c, g) + 1) = C_x(b, g)$. 4. $b = D_1c$ where $c = d + 1$.
In this critical case we have $b[x] = D_1d + (D_1c)[\![x+1]\!] = D_1d \cdot y + (D_1c)[\![z]\!]$ where $z < ND_1c$ and $y > 0$. Lemma 10 yields $\mathrm{K}^*(D_1c)[\![z]\!] \preceq \mathrm{K}^*D_1c \preceq \mathrm{K}^*b \prec a$. Thus $\mathrm{K}^*(D_1c)[\![x+1]\!] \prec a$. Hence Lemma 9 yields $C_x(b[x], g) = C_x(D_1d, g) + C_x((D_1c)[\![x+1]\!], g) \leq g^{2^{T_{D_1d}(x+1)}} \cdot (C_x(d, g)+1) + g^{2^{T_{(D_1c)[\![x+1]\!]}(x+1)+1}} \leq g^{2^{T_{D_1c}(x+1)}} \cdot (C_x(c, g)+1) = C_x(b, g)$ since $T_{D_1c}(x+1) > T_{(D_1c)[\![x+1]\!]}(x+1)+1$ and $T_{D_1c}(x+1) \geq T_{D_1d}(x+1)$.

Lemma 15. $\mathsf{tp}(a) = \Omega$ & $a\{0\} \prec b \prec a \;\Rightarrow\; Na\{0\} < Nb$.

Proof by induction on Na.
1. $a = \Omega$. Then $a\{0\} = 0$ and the assertion is obvious.
2. $a = (a_0, \ldots, a_n)$. $a\{0\} \prec b \prec a$ yields $b = (a_0, \ldots, a_{n-1}) + c$ for some c with $a_n\{0\} \prec c \prec a_n$. The induction hypothesis yields $Na_n\{0\} < Nc$ hence $Na\{0\} < Nb$.
3. $a = D_1c$. $a\{0\} = D_1c\{0\} \prec b \prec D_1c$ yields $b = D_1d + e$ for some $e \prec D_1(d+1)$

and $c\{0\} \preceq d \prec c$. The induction hypothesis yields $Nc\{0\} \leq Nd$. If $e \neq 0$ then $Nb \geq Nd + Ne + 1 > Na$. If $e = 0$ then $c\{0\} \prec d \prec c$. The induction hypothesis yields $Nc\{0\} < Nd$ hence the assertion.

Lemma 16. *Assume that $D_0 a, b, c \in OT$, $\mathsf{tp}(a) = \mathsf{tp}(c) = \Omega$, $b \preceq c \preceq a$, $\mathrm{K}^* b \prec a$ and $Nb \leq Nc + x$. Then $b \preceq c\{(D_0 a)[\![x+1]\!]\}$.*

Proof by induction on Nb.
1. $b = 0$. Then the assertion is obvious.
2. $b = (b_0, \ldots, b_m)$. Assume that $c = (c_0, \ldots, c_n)$.
2.1. $b_0 = c_0, \ldots, b_m = c_m$ and $m < n$. Then $b \preceq (c_0, \ldots, c_m) \preceq c\{(D_0 a)[\![x+1]\!]\}$.
2.2. $\exists i \leq \min\{m, n\}[b_0 = c_0, \ldots, b_{i-1} = c_{i-1}, b_i \prec c_i]$.
If $i < n$ then $b \prec (c_0, \ldots, c_i) \preceq c\{(D_0 a)[\![x+1]\!]\}$.
Assume $i = n$ and $b_n, \ldots, b_m \prec c_n]$.
2.2.1. $m = n$. $Nb \leq Nc + x$ yields $Nb_n \leq Nc_n + x$. The induction hypothesis yields $b_n \preceq c_n\{(D_0 a)[\![x+1]\!]\}$ hence $b \preceq c\{(D_0 a)[\![x+1]\!]\}$.
2.2.2. $m > n$. $Nb \leq Nc + x$ yields $Nb_n, \ldots, Nb_m \leq Nc_n + x - 1$ The induction hypothesis yields for $x > 0$ that $b_n, \ldots, b_m \preceq c_n\{(D_0 a)[\![x]\!]\} \prec c_n\{(D_0 a)[\![x+1]\!]\}$. For $x = 0$ we obtain $b_n, \ldots, b_m \preceq c_n\{0\} = c_n\{(D_0 a)[\![x]\!]\}$ by Lemma 15. Since $c_n\{(D_0 a)[\![x+1]\!]\} \in P$ we obtain $(b_l, \ldots, b_n) \prec c_n\{(D_0 a)[\![x+1]\!]\}$ hence $b \prec c\{(D_0 a)[\![x+1]\!]\}$.
3. $b = D_0 c$. $\mathrm{K}^* b \prec a$ yields $\mathrm{K}^* c \cup \{c\} \prec a$ hence $b \prec D_0 a$, thus $b \preceq (D_0 a)[\![Nb]\!]$.
3.1. $c = \Omega$. Then $Nc = 1$ and $b \preceq (D_0 a)[\![x+1]\!] = c\{(D_0 a)[\![x+1]\!]\}$.
3.2. $\Omega \prec c$. Then $b \preceq \Omega \preceq c[0] \preceq c\{(D_0 a)[\![x+1]\!]\}$.
4. $b = D_1 d$.
4.1. $c = (c_0, \ldots, c_n)$ with $n \geq 1$.
Then $b = D_1 d \preceq c_0 \preceq c\{0\} \leq c\{(D_0 a)[\![x+1]\!]\}$.
4.2. $c = D_1 e$. $Nb \leq Nc + x$ yields $Nd \leq Ne + x$. The induction hypothesis yields $d \preceq e\{(D_0 a)[\![x+1]\!]\}$ since $e \prec D_1 e \preceq a$. Thus $b = D_1 d \preceq D_1 e\{(D_0 a)[\![x+1]\!]\} = c\{(D_0 a)[\![x+1]\!]\}$.

Corollary 1. *Assume that $\mathsf{tp}(a) = \Omega$, $b \preceq a$, $\mathrm{K}^* b \prec b$ and $Nb \leq Na + x$. Then $b \leq a\{(D_0 a)[\![x+1]\!]\}$.*

Proof. Put $c = a$ in Lemma 16.

Lemma 17. *Assume that $D_0 a \in$ OT and $\mathsf{tp}(a) = \Omega$. Let $z < ND_0 a$. Then $(D_0 a)[\![z]\!] = D_0 a\{0\}$ if $z = ND_0 a[0]$ and $(D_0 a)[\![z]\!] = (D_0 a\{0\})[\![z]\!]$ else.*

Proof. It suffices to show $(D_0 a)[\![z]\!] \preceq D_0 a\{0\}$. Assume for a contradiction that $D_0 a\{0\} \prec (D_0 a)[\![z]\!] \prec D_0 a$. Then $(D_0 a)[\![z]\!] = D_0 b + d$ for some b with $a\{0\} \preceq b \prec a$. Lemma 15 yields $Nb \geq Na\{0\}$. If $d \neq 0$ then $N(D_0 b + d) \geq ND_0 a\{0\} + 1 = ND_0 a$. This contradicts $z < ND_0 a$. Hence $d = 0$ and $a\{0\} \prec b \prec a$. Lemma 15 yields $Nb > Na\{0\}$ hence $ND_0 b \geq ND_0 a$. This contradicts $z < ND_0 a$.

Lemma 18. *Assume that $D_0 a \in$ OT and $\mathsf{tp}(a) = \Omega$. Let $z < N(D_0 a)$. Let $d_0 a(0, z) := (D_0 a\{0\})[\![z]\!]$ and $d_0 a(y+1, z) := D_0 a\{d_0 a(y, z)\}$. Then $d_0 a(y, z) \prec d_0 a(y+1, z)$ and $d_0 a(y, z) \in OT$. Moreover $(D_0 a)[\![x]\!] = d_0 a(y, z')$ where y and z' are chosen such that $(Na + 1) \cdot y + z' = x$ and $z' < Na + 1$.*

Proof. By induction on y we show $d_0a(y,z) \prec d_0a(y+1,z)$.
Assume first that $y = 0$.
Then $d_0a(0,z) = (D_0a\{0\})[\![z]\!]$ and $d_0a(1,z) = D_0a\{(D_0a\{0\})[\![z]\!]\}$. If $z = 0$ then $d_0a(0,z) < d_0a(1,z)$ is obvious. Assume that $z \neq 0$. Lemma 10 yields $\mathrm{K}^*(D_0a\{0\})[\![z]\!] \preceq \mathrm{K}^*(D_0a\{0\}) \preceq \mathrm{K}^*a\{0\} \prec a\{(D_0a\{0\})[\![z]\!]\}$. Lemma 2 yields $\mathrm{K}(D_0a\{0\})[\![z]\!] \prec D_0a\{(D_0a\{0\})[\![z]\!]\}$ hence $(D_0a\{0\})[\![z]\!] \prec D_0a\{(D_0a\{0\})[\![z]\!]\}$.
Now assume that $y = y'+1$.
The induction hypothesis yields $d_0a(y',z) \prec d_0a(y'+1,z)$ hence $a\{d_0a(y',z)\} \prec a\{d_0a(y'+1,z)\}$ thus $d_0a(y,z) \prec d_0a(y+1,z)$.

By induction on y we show $d_0a(y,z) \in \mathrm{OT}$.
Assume $y = 0$.
Then $d_0a(y,z) = (D_0a\{0\})[\![z]\!] \in \mathrm{OT}$.
Assume $y = y'+1$.
Then $d_0a(y,z) := D_0a\{d_0a(y',z)\}$. The induction hypothesis yields $d_0a(y',z) \in \mathrm{OT}$ hence $a\{d_0a(y',z)\} \in \mathrm{OT}$.
We have to show $\mathrm{K}^*a\{d_0a(y',z)\} \prec a\{d_0a(y',z)\}$ and compute $\mathrm{K}^*a\{d_0a(y',z)\} \preceq \mathrm{K}^*a\{0\} \cup \mathrm{K}^*d_0a(y',z)$. Lemma 4 yields $a\{0\} \lhd_0 a$ hence $\mathrm{K}^*a\{0\} \prec a\{0\}$ since $\mathrm{K}^*a \prec a$. We first consider the case $y' = 0$. If $z = 0$ then $d_0a(y',z) = 0$ hence $\mathrm{K}^*a\{d_0a(y',z)\} = \mathrm{K}^*a\{0\} \prec a\{0\} = a\{d_0a(y',z)\}$. If $z > 0$ then $(D_0a\{0\})[\![z]\!] \neq 0$ and Lemma 10 yields $\mathrm{K}^*d_0a(y',z) \preceq \mathrm{K}^*D_0a\{0\} \preceq \mathrm{K}^*a\{0\} \cup a\{0\} \preceq a\{0\} \prec a\{(D_0a\{0\})[\![z]\!]\}$.
Now assume that $y' > 0$. We already have shown that $\{d_0a(y'-1,z)\} \prec d_0a(y',z)$. The induction hypothesis yields $d_0a(y',z) \in \mathrm{OT}$ hence $\mathrm{K}^*a\{d_0a(y'-1,z)\} \prec a\{d_0a(y'-1,z)\}$ hence $\mathrm{K}^*a\{d_0a(y',z)\} = \mathrm{K}^*D_0a\{d_0a(y'-1,z)\} \preceq \mathrm{K}^*a\{0\} \cup \mathrm{K}^*\{d_0a(y'-1,z)\} \cup \{d_0a(y'-1,z)\} \preceq \{d_0a(y'-1,z)\} \prec d_0a(y',z)$.

Now we prove $(D_0a)[\![x]\!] = d_0a(y,z')$ by induction on x where $(Na+1)\cdot y+z' = x$ and $z' < Na+1$. If $x < N(D_0a)$ then the assertion follows from Lemma 17. Now assume that $x \geq N(D_0a)$. The choice of y and z' and the definition of $(D_0a)[\![x]\!]$ yield $(D_0a)[\![x]\!] \geq d_0a(y,z')$ since $Nd_0a(y,z') = x$.
Now assume that $(D_0a)[\![x]\!] = D_0b+c$ with $b \prec a$ and $c \prec D_0(b+1)$. Then $c = 0$ since otherwise $D_0(b+1)$ would be a better choice than D_0b+c for $(D_0a)[x]$. We have $b \preceq a$, $\mathsf{tp}(a) = \Omega$, $\mathrm{K}^*b \prec b$ and $Nb \leq x = Na+1+x-Na-1$. The induction hypothesis yields $(D_0a)[\![x-Na-1]\!] = d_0a(y-1,z)$. Lemma 1 yields $b \preceq a\{(D_0a)[\![x-Na]\!]\}$ hence $D_0b \preceq D_0a\{(D_0a)[\![x-Na-1]\!]\} = d_0a(y,z) \in \mathrm{OT}$.

Corollary 2. *Assume that* $\mathsf{tp}(a) = \Omega$. *Then* $D_0a\{(D_0a)[\![x+1]\!]\} \in \mathrm{OT}$ *and* $(D_0a)[x] = D_0a\{(D_0a)[\![x+1]\!]\}$.

Proof. Lemma 18 yields $(D_0a)[x] = (D_0a)[\![Na+1+x]\!] = D_0a\{(D_0a)[\![x+1]\!]\} \in \mathrm{OT}$.

Definition 14. *Recursive definition of a nominal form* $\mathcal{C}_x(a,g)$ *for* $a \in OT$ *with* $\mathsf{tp}(a) = \Omega$ *and* $g < \omega$.
1. $\mathcal{C}_x(\Omega,g) := \star$.

2. $\mathcal{C}_x((a_0,\dots,a_{n-1},a_n),g) := C_x(a_0,g)+\cdots+C_x(a_{n-1},g)+\mathcal{C}_x(a_n,g)$.
3. $\mathcal{C}_x(D_1 a,g) := g^{2^{T_{D_1 a}(x+1)}}\cdot(\mathcal{C}_x(a,g)+1)$.

If $\mathcal{C}$ is a nominal form then $\mathcal{C}[\star := c]$ denotes the result of replacing every occurrence of $\star$ in $\mathcal{C}$ by c.

Lemma 19. *If $a \in \mathrm{OT}$, $\mathtt{tp}(a) = \Omega$ and $c \in OT_0$ then $C_x(a[c],g) \le \mathcal{C}_x(a,g)[\star := g\cdot G_c(x)]$.*

Proof by induction on Na.
1. $a = \Omega$. Then Lemma 8 yields $C_x(a[c],g) = C_x(c,g) = g\cdot G_c(x) = \star[\star := g\cdot G_c(x)]$.
2. $a = (a_0,\dots,a_n)$. Then the induction hypothesis yields $C_x(a[c],g) = C_x(a_0,g)+\cdots+C_x(a_{n-1},g)+C_x(a_n[c],g) \le C_x(a_0,g)+\cdots+C_x(a_{n-1},g)+\mathcal{C}_x(a_n,g)[\star := g\cdot G_c(x)] = \mathcal{C}_x(a,g)[\star := g\cdot G_c(x)]$ 3. $a = D_1 b$.
Then assertion 2 of Lemma 7 and the induction hypothesis yields $C_x(a[c],g) = C_x(D_1(b[c]),g) = g^{2^{T_{D_1 b[c]}(x+1)}}\cdot(C_x(b[c],g)+1) \le g^{2^{T_{D_1 b}(x+1)}}\cdot(\mathcal{C}_x(b,g)[\star := g\cdot G_c(x)]+1) = \mathcal{C}_x(a,g)[\star := g\cdot G_c(x)]$.

Lemma 20. *If $a \in \mathrm{OT}$ and $\mathtt{tp}(a) = \Omega$ then $\mathcal{C}_x(a,g)[\star := g^2] \le C_x(a,g)$.*

Proof by induction on Na.
1. $a = \Omega$. Then $\mathcal{C}_x(a,g)[\star := g^2] = g^2$ and $C_x(D_1 0,g) = g^{2^{T_{D_1 0(x+1)}}}\cdot(0+1) \ge g^2$.
2. $a = (a_0,\dots,a_n)$
Then the induction hypothesis yields $\mathcal{C}_x(a,g)[\star := g^2] = C_x(a_0,g)+\cdots+C_x(a_{n-1},g)+\mathcal{C}_x(a_n,g)[\star := g^2] \le C_x(a_0,g)+\cdots+C_x(a_{n-1},g)+C_x(a_n,g) = C_x(a,g)$. 3. $a = D_1 b$. Then the induction hypothesis yields $\mathcal{C}_x(a,g)[\star := g^2] =$
$g^{2^{T_{D_1 b}(x+1)}}\cdot(\mathcal{C}_x(b,g)[\star := g^2]+1)$
$\le g^{2^{T_{D_1 b}(x+1)}}\cdot(C_x(b,g)+1)$
$= C_x(a,g)$.

Theorem 2. *Let $D_0 a \in OT_0$ and $g := G_{(D_0 a)[\![x+1]\!]}(x)$. Let $D_0 b \in OT$ and assume that $\mathrm{K}^* D_0 b \prec a$. Then*

$$G_{D_0 b}(x) \le 1 + C_x(b,g).$$

Proof by induction on $D_0 b$.
1. $G_{D_0 0}(x) = 1 \le 1 + C_x(0,g)$.
2. $b = c+1$. Then $G_{D_0 b}(x) = G_{D_0 c+(D_0 b)[\![x+1]\!]}(x) = G_{D_0 c}(x) + G_{(D_0 b)[\![x+1]\!]}(x) \le 1 + C_x(c,g) + g = 1 + C_x(c+1,g)$ since $C_x(1,g) = g$ and $G_{(D_0 b)[\![x+1]\!]}(x) \le G_{(D_0 a)[\![x+1]\!]}(x)$ by assertion 3 of Lemma 6.
3. $tp(b) = \omega$.
Then the induction hypothesis, Lemma 13 and Lemma 14 yield
$G_{D_0 b}(x) = G_{(D_0 b)[x]}(x) = G_{D_0(b[x])}(x) \le 1 + C_x(b[x],g) \le 1 + C_x(b,g)$ 4. $tp(b) = \Omega$.
Then the induction hypothesis, Lemma 2, Lemma 19 and Lemma 20 yield $G_{D_0 b}(x) = G_{D_0 b[(D_0 b)[\![x+1]\!]]}(x) \le 1 + C_x(b\{(D_0 b)[\![x+1]\!]\},g) \le 1 + \mathcal{C}_x(b,g)[\star := g\cdot G_{(D_0 b)[\![x+1]\!]}(x)](x) \le 1 + \mathcal{C}_x(b,g)[\star := g^2] \le 1 + C_x(b,g)$

Lemma 21. *Let* $U_x := \{a \in T : Na \leq x\}$ *and* $\#U_x$ *be the cardinality of* U_x*. Then* $\#U_x \leq 4^{4^{4^x}}$.

Proof. By induction on x. Obviously $\#U_0 = 1$ and $\#U_{x+1}$ is less than or equal to one plus the cardinality of the Cartesian product $\{0,1,2\} \times U_x \times \cdots U_x \times U_x$ with $x+2$ factors. For, if $a \in T$ then a is either of the form $(a_0, \dots, a_n)$ with $n \leq x$ and $a_i \in T_x$ or a is of the form $D_0 b$ or $D_1 b$ for $b \in T_x$. Hence, arguing by induction, $\#U_{x+1} \leq 3 \cdot (\#U_x)^x \leq 3 \cdot (4^{4^{4^x}})^x \leq 4^{4^{4^{x+1}}}$.

Theorem 3. *Let* $p(x) := 4^{4^{4^x}}$. *If* $a \in \mathrm{OT}_0$ *and* $Na \leq x$ *then* $G_a(x) \leq p(p(p(T_a(x+1))))$.

Proof. By induction on Na.
1. $a = 0$. Then the assertion is obvious.
2. $a = (a_0, \dots, a_n)$
Then the induction hypothesis yields $G_a(x) = G_{a_0}(x) + \cdots + G_{a_n}(x) \leq p^3(T_{a_0}(x+1)) + \cdots + p^3(T_{a_n}(x+1)) \leq p^3(T_a(x+1))$.
2. $a = D_0 b$. Let $g := (D_0 a)[\![x+1]\!]$. Then the induction hypothesis yields $G_a(x) \leq C_x(a,g) \leq g^{2^{T_a(x+1)}} \leq (p^3(T_{a[\![x+1]\!]}(x+1)))^{2^{T_a(x+1)}} \leq p^3(T_a(x+1))$.

Theorem 4. *Let* $4_0(x) := x$ *and* $4_{n+1}(x) := 4^{4_n(x)}$. *If* $a \in \mathrm{OT}_0$ *and* $Na \leq x$ *then* $G_a(x) \leq 4_{12}(x)$.

References:

[1] W. Buchholz: A new system of proof-theoretic ordinal functions. Ann. Pure Appl. Logic 32 (1986), no. 3, 195–207.
[2] W. Buchholz, E.A. Cichon, Adam, A. Weiermann: A uniform approach to fundamental sequences and hierarchies. Math. Logic Quart. 40 (1994), no. 2, 273–286.
[3] J.Y. Girard: Proof Theory and Logical Complexity. Bipliopolis 1987.
[4] A. Weiermann: A very slow growing hierarchy for $\mathit{\Gamma}_0$. Logic Colloquium '99, 182–199, Lect. Notes Log., 17, Assoc. Symbol. Logic, Urbana, IL, 2004.

Phase transition results for subrecursive hierarchies and rapidly growing Ramsey functions

Andreas Weiermann

Mathematical Institute of Utrecht University
Postbox 80010, 3508 Utrecht, The Netherlands,
weierman@math.uu.nl

We consider hierarchies of recursive functions which are defined relative to a given parameter which is either a real number or a function. For small or slowly growing values of the parameter the induced hierarchy will be rather slow growing. When the parameter crosses through a certain threshold the resulting hierarchy will consist of rapidly growing functions.

In the presentation we will describe this setting in general terms. We give detailed phase transition results for the Paris Harrington function of dimension 3 with three colours. These Ramsey functions vary in growth from being double exponential to at least Ackermannian. We conjecture that a full classification (which has not been achieved yet) of the phase transition in this case will provide progress on a classic USD 500 problem of Erdoes.

Moreover we consider the phase transition problem for the slow growing hierarchy where for a limit ordinal λ the fundamental sequence at argument x is given by the max over all $\beta < \lambda$ such that the norm of β is less than or equal to x plus $(1+\epsilon)$ times the norm of λ. There will be a sharp phase transition for ϵ close to 0.

Computability at the Planck Scale[⋆]

Paola Zizzi

Dipartimento di Matematica Pura ed Applicata,
Università di Padova
via G. Belzoni n.7
I–35131 Padova, Italy

Abstract. We consider the issue of computability at the most fundamental level of physical reality: the Planck scale. To this aim, we consider the theoretical model of a quantum computer on a non commutative space background, which is a computational model for quantum gravity. In this domain, all computable functions are the laws of physics in their most primordial form, and non computable mathematics finds no room in the physical world.
Moreover, we show that a theorem that classically was considered true but non computable, at the Planck scale becomes computable but non decidable. This fact is due to the change of logic for observers in a quantum-computing universe: from standard quantum logic and classical logic, to paraconsistent logic.

A quantum computer [24] can simulate [13] perfectly and efficiently a quantum mechanical system. Precision is due to discreteness from both sides: qubits in the quantum register, and discrete spectra in the quantum system. Efficiency is due to quantum parallelism in the quantum computer. However, there is not an isomorphism between the quantum computer and the quantum system, because the latter lies on a classical background (classical space-time, which is a smooth manifold).

The classical background is not taken into account during the simulation, but only at the end, when a classical output is obtained by measurement. Because of that, a great deal of quantum information remains hidden. The hidden quantum information is due to the irreversibility of a standard quantum measurement, but this is related to the situation of the external observer in a classical background, which cannot be put in one-to-one correspondence with the machine computational state.

One might argue, then, that in a discrete space-time like a lattice, one could overcome this problem. However, we wish to recover the hidden information in a physical discrete space-time, that is, both discrete and Lorentz invariant. And, obviously, a lattice breaks Lorentz invariance. The simplest way out is to consider a quantum space background like a fuzzy sphere [23], which is both discrete and

⋆ I wish to thank G. Battilotti and G. Sambin for useful discussions. Work supported by the research project "Logical Tools for Quantum Information Theory", Department of Pure and Applied Mathematics, University of Padova.

Lorentz invariant. This choice arises quite naturally. In fact, in a recent paper [36], we showed that the non commutative algebra of quantum logic gates of a quantum computer of N qubits, is associated, by the non-commutative version of the Gelfand-Naimark theorem [21], with a quantum space which is a fuzzy sphere with $n = 2^N$ elementary cells.

The background space and the quantum computer are then in a one-to-one correspondence. An "external" observer on the quantum space background automatically becomes an "insider" observer with respect to the quantum computer. Thus, in this model, one can conceive a reversible quantum measurement [36], performed by an "insider observer", a hypothetical being living on the fuzzy sphere. There is no hidden information anymore: all the quantum information remains available.

Recently, we investigated about the logic [2] of the internal observer, which turns to be a paraconsistent [9], [27] and symmetric logic, like basic logic [30], where both the non-contradiction and the excluded middle principles are invalidated. The isomorphism between the quantum computer and the quantum space background turns to be quite useful, as it leads to a quantum-computational basic formulation [37] of quantum gravity (space-time at the Plank scale).

It is usually assumed that, at the Planck scale the very concept of space-time becomes meaningless, and should be replaced by a discrete geometrical structure. (A major example is Loop Quantum Gravity (LQG) [28], [32], [26], where the discrete structure is realized by spin networks [25], [29]).

We believe that the modification of the concept of space-time at the Planck scale leads to a modification of the concept of computability. (This does not mean, however, that at the Planck scale there is the so-called hyper computability [17], [20], [6], [5], by which it would be possible to construct some hypermachines, or utilize some quantum systems that can calculate non-recursive functions).

It might be that the quantum hidden information, due to the inconsistency between Quantum Mechanics and its classical background, is responsible of the appearance of non computable problems, in relation with the standard quantum logic [3] of the external observer. In fact the proof of a theorem can be considered as an algorithm, and when this algorithm is run by a quantum computer, you can only know if it is true or false but the proof will remain unknown to you. Then, one is faced with the same kind of problems encountered in the simulation of a quantum system.

Instead, if one imagines a quantum computer embedded in a quantum space or, in other words, a "quantum computing space" (the classical world having been "erased"), the very definition of computability is modified. This is related to the new logic of the "internal" observer. In fact, if an "observer" could enter a quantum space and put itself in a one-to-one correspondence with the quantum states of the computer, then a complete meta-system would arise.

In other words, in a quantum space-time isomorphic to a quantum computer, all problems would appear to be computable. However, the complete meta-system would appear "inconsistent" with respect to a classical world. Of course the Planck scale is (at least at present) unreachable experimentally, nevertheless, studying quantum computability at the fundamental scale, can lead to a deeper insight into the foundations of Quantum Mechanics.

Deutsch [10], [11] claims that the laws of Physics determine which functions can be computed by a universal computer. Also, he says that the only thing that privileges the set of computable functions (or the set of quantum-computable functions) is that it is instantiated in the laws of physics. But one can go a bit further: in the quantum computer view of space-time at the Planck scale [38], quantum space-time is a universal quantum computer that quantum-evaluates recursive functions which are the laws of Physics in their most primordial and symbolic form. In other words, at the Planck scale because of the isomorphism between a quantum computer and quantum space-time (quantum gravity), the laws of physics are identified with quantum functions. This is the physical source of computability, and leads to the conclusion that at the Planck scale, only computable mathematics exists.

We would like to make a remark: Deutsch says that all computer programs may be regarded as symbolic representations of some of the laws of physics, but it is not possible to interpret the whole universe as a simulation on a giant quantum computer because of computational universality. We fully agree with that, and we wish to make it clear that, in our view, quantum space-time is not a simulation but is itself a quantum computer, and, by quantum evaluating the laws of Physics, it just computes its own evolution.

One might then ask what would happen to the observer logical dichotomy mentioned above, in the case the whole universe were a giant quantum computing-fuzzy sphere. Actually, by using the holographic principle [18], and the isomorphism between the quantum computer and the fuzzy sphere, one is able to settle a minimal model [37] for Loop Quantum Gravity, which has been called Computational Loop Quantum Gravity (CLQG). This leads to a discrete area spectrum for the cosmological horizon, which can be viewed as the surface of a giant fuzzy sphere, whose elementary cells encode the whole quantum information of the universe.

The amount of quantum information stored by the cosmological horizon since the Big Bang was computed in [39], [38] and [22], and it turned to be $N \approx 10^{120}$ qubits. The area of an elementary cell [9] of the event horizon of a fuzzy black hole-quantum computer is:

$$A_{EC} = \frac{N}{2^N} A_0,$$

where A_0 is the elementary area in LQG, of the order of a Planck area: $A_0 \approx l_P^2$, where $l_p \approx 10^{-33} cm$ is the Planck length. In the case the whole universe were a fuzzy quantum computer, an elementary cell of the cosmological horizon would have an area:

$$A_{EC} = \frac{10^{120}}{2^{10^{120}}} A_0,$$

which is extremely small, but finite.

If the internal observer, inside the fuzzy quantum universe, is still able to intuitively recognize discreteness of space-time, he will also be able to use paraconsistent logic in his judgments. If instead the observer focuses on his perceptions, he will make, in his mind, automatically the two limits: $N \longrightarrow \infty$ and $A_{EC} \longrightarrow 0$. In this way he would pretend he lives in a classical space-time, and will automatically become an Aristotelian logician, with respect to the universe as a whole. It should be noticed that those limits are full of implications. In fact, they mean that an infinite string of bits (infinite classical information) is encoded in a point of space. We would like to call this situation: "Information singularity" as it reminds the Big Bang, where an infinite amount of energy (instead of information) is concentrated in a point.

However, there is effectively an increase of information entropy, in this model, as far as N increases, as well as a tendency to the continuum. We are faced with signals of higher complexity and quantum information loss. It becomes harder and harder, for the internal observer, in this situation, to maintain a coherent superposition of truth values in his logical judgements. In fact, he starts to lose his capacity to distinguish the discrete space-time structure from a smooth manifold, and perceives an increase of complexity. We argue then, that non computable mathematics is an emergent feature of computable mathematics at higher levels of complexity, as perceived by the observer/logician. In digital physics [14], [34], which relies on the strong Church-Turing thesis (the universe is equivalent to a Turing machine) it is assumed that it exists a program for a universal computer which computes the dynamical evolution of the universe.

Of course, this is an algorithmic view of physical reality, and it can be easily adopted if one assumes that real numbers and continuous physical quantities comprised continuous space-time, are absent. Because of that, when the limits discussed above are taken into account, the digital paradigm starts to fade, and needs some extra insights. We believe that the relation between non computability and emergent information complexity might be better investigated in the context of Chaitin's algorithmic information theory (AIT) [7].

In our model, as we have seen, the universe is the result of the identification of a huge quantum computer with a quantum (non commutative) space. The software version of the cosmic computer represents the universe as a simulation. But we, like Deutsch, strongly disagree with this approach. Instead, in the hardware version, the universe computes its own dynamical evolution. The cosmological models described in [38], [39] and particularly in [37], where LQG is itself the cosmic computer, obviously belong to the hardware version. As Deutsch illustrated in [12], a universal quantum Turing machine (or quantum computer) has the same computability power of a universal classical Turing machine.

It follows that, as a quantum computer, the universe can compute only Turing-computable functions. As a physical universe, it computes the laws of physics in their most primordial form. Then, the laws of physics are all Turing-computable. Consequently, non computable mathematics is outside the realm of the physical world.

The observer is internal with respect to the fuzzy quantum-computer universe, and can only use paraconsistent logic, with respect to the universe as a whole. In the mind of the observer, there are two strong axioms, which are the converse of the non contradiction and excluded middle principles, respectively. The first axiom, namely the converse of the non contradiction principle: $\vdash A \wedge \neg A$, does not allow the observer to use classical truth values in his judgements (measurements) and, for him, classical decidability is now meaningless.

However, the internal observer is also external with respect to a quantum subsystem, like a quantum computer inside the quantum computing universe. In this case, the observer utilizes standard quantum logic. The Boolean outputs of his standard (irreversible) quantum measurement are the truth values of classical logic. The double logic of the observer, leads to a change in the interpretation of the first Gödel's incompleteness theorem [16] at the Planck scale, as we will see in the following. Simply, there is a change in the concept of truth: what classically was true but non-computable appears now to be computable but non decidable.

Let us consider the famous "couple" of quantum computing: Alice (A) and Bob (B). Alice (A) lives in a classical space-time, which is also the classical background of the quantum computer (QC). A is an external observer with respect to the QC, and is equipped with standard quantum logic when she performs a standard quantum measurement; otherwise, she is endowed with classical logic. Instead, B lives in a quantum space isomorphic to the QC, he is an internal observer, and is equipped with paraconsistent logic.

Let us suppose that a theorem (T) is given to A. T is computable, but the computation of T is too long (and hard) for A and for her classical computational tools (classical computer or classical Turing machine). Then, A gives T as an algorithm to the QC. By using quantum parallelism (and entanglement), the QC computes T efficiently. T is true, and the output is 1. However, A will never have the demonstration at her disposal, because, to get the result, she had to destroy the superposition, and lose quantum information. Thus, T appears to A as Gödel's formula G would have appeared to a mathematician: true, but not (for A) computable. Of course, T "is" computable, as the QC has just computed it. But, with respect to the external observer (A), the QC has behaved like another human (B), the internal observer. In fact B can verify that T is computable, but will tell A that T is not computable, as he will never provide her the demonstration.

Then, A will enter in a quantum space whose elementary cells are in a one-to-one correspondence with the computational states (B's states). Now, A can verify herself that T is computable, as B is computing it. But now, neither A nor B will ever know whether T is true or false, because 0 and 1 are classical truth values, which are not obtainable anymore, now that the classical world has been "erased" for the couple A-B.

Let us suppose that the QC is the whole universe. Now A and B are both internal observers, and both use paraconsistent logic with respect to the universe as a whole. The question: "Is T is true or false?" has no answer. In the passage from the classical to the quantum stage, there is a change in the meaning of definitions.

In the classical setting, T was true, but the quantum demonstration was not provided, so T appeared as Gödel's sentence G, true but non computable. In the quantum setting, the quantum demonstration of T is available: T clearly appears to be computable. But the classical truth values are now meaningless: T is (classically) undecidable.

References

1. Battilotti, G. : Basic logic and quantum computing: logical judgements by an insider observer, Proc. FQI04, 16-19 april 2004, Camerino, Italy, International Journal of Quantum Information **3**,1 (2005) 105-109. arXiv: quant-ph/0407057.
2. Battilotti, G., Zizzi, P. A.: Logical Interpretation of a Reversible Measurement in Quantum Computing, arXiv: quant-ph/0408068.
3. Birkhoff, G. ,von Neumann,J.: The Logic of Quantum Mechanics, Annals of Mathematics **37**(1936) 823-843.
4. Bombelli, L., Lee, J., Meyer, D. and Sorkin, R.: Space-time as a causal set, Phys. Rev. Lett. **59** (1987) 521-524.
5. Calude C. S., Pavlov, B.: Coins, Quantum measurements, and Turing's barrier, Quantum Information Processing **1** (2002) 107-127.
6. Capeland, B. J.: Super Turing-Machines, Complexity, **4** (1998) 30-37.
7. Chaitin, G. J.: From philosophy to program size, Lecture Notes on Algorithmic Information Theory from the 8th Estonian Winter School in Computer Science, EWSCS'03.
8. Dalla Chiara, M. L., Giuntini, R.: Paraconsistent Quantum Logic, Foundations of Physics **19**(1989) 891-904.
9. Dalla Chiara, M. L., Giuntini, R., Leporini, R.: Quantum Computational Logics. A Survey, arXiv: quant-ph/0305029.
10. Deutsch, D: Physics, Philosophy and Quantum Technology, Proc. Sixth Int. Conf. on Quantum Communication, Measurement and Computing, Rinton Press, Princeton, NJ 2003.
11. Deutsch, D.: Machines, Logic and quantum Physics. math.HO/9911150.
12. Deutsch, D.: Quantum theory, the Church-Turing principle and the universal quantum computer, Proc. Royal Society of London A **400** (1985) 97-117.
13. Feynman, R. P.: Simulating physics with computers, IJTP **21** (1982) 467.
14. Fredkin, E.: An Introduction to Digital Philosophy, IJTP **42** 2 (2003) 189-247.
15. Girard, J. Y.: Linear Logic, Theoretical Computer Science **50**(1987) 1-102.
16. Gödel, K.: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. Monatshefte für Mathematik und Physik, **38** (1931) 173-198. Translated in van Heijenoort: From Frege to Gödel. Harvard University Press, 1971.
17. Hogarth, M. L.: Non-Turing Computers and Non-Turing Computability, PSA, 1 (1994) 126-138.
18. ' t Hooft, G.: Dimensional reduction in quantum gravity. arXiv: gr-qc/9310026;
19. ' t Hooft, G.: The Holographic Principle. arXiv: hep-th/0003004.
20. Kieu, T.: Computing the noncomputable. arXiv: quant-ph/0203034
21. Kruszynski, P., Woronowicz, S. L.: A non-commutative Gelfand-Naimark theorem, J. Operator Theory **8** (1982) 361.
22. Lloyd, S.: Computational capacity of the universe, Phys. Rev. Lett. **88** (2002) 237901. arXiv: quant-ph/0110141.

23. Madore, J.: The fuzzy sphere, Class. Quant. Grav. **9** (1992) 69.
24. Nielsen, N. A., Chuang, I. L.: *Quantum Computation and Quantum Information*, Cambridge University Press (2000).
25. Penrose, R.: Theory of quantised directions, in: Quantum Theory and Beyond, ed. T. Bastin, Cambridge University Press (1971) 875.
26. Perez, A.: Introduction to Loop Quantum Gravity and Spin Foams. arXiv: gr-qc/0409061.
27. Priest, G.: Paraconsistent logic, in Handbook of Philosophical logic, second edition, D. Gabbay and F. Guenthner Eds., Kluwer Academic Publishers 2002.
28. Rovelli, C.: *Quantum Gravity*, Cambridge, U.K. Univ. Press (2004).
29. Rovelli, C., Smolin, L.: Spin Networks in Quantum Gravity, Phys. Rev. D52 (1995) 5743.
30. Sambin, G., Battilotti, G. and Faggian, C.: Basic Logic: Reflection, Symmetry, Visibility, The Journal of Symbolic Logic, **65** (2000) 979-1013.
31. Susskind, L.: The world as an hologram. arXiv: hep-th/9409089.
32. Thiemann, T.: Introduction to modern canonical quantum general relativity. arXiv: gr-qc/0110034.
33. Wheeler, J. A.: *Geometrodynamics*, Academic Press, New York (1962).
34. Wolfram, S.: A New Kind of Science, Wolfram media, Inc. Champaign IL (2002).
35. Wootters, W. K., Zureck, W. H.: A single quantum cannot be cloned, Nature **299** (1982) 802-803.
36. Zizzi, P. A.: Qubits and Quantum Spaces, Proc. FQI04, 16-19 april 2004, Camerino, Italy. International Journal of Quantum Information, **3**,1 (2005) 287-291. arXiv: quant-ph/0406154.
37. Zizzi, P. A.: A minimal model for quantum gravity, Physics Letters A, to appear. arXiv: gr-qc/0409069.
38. Zizzi, P.: Spacetime at the Planck Scale: The quantum Computer View. arXiv: gr-qc/0304032.
39. Zizzi, P.: The Early Universe as a Quantum Growing Network. arXiv: gr-qc/0103002.