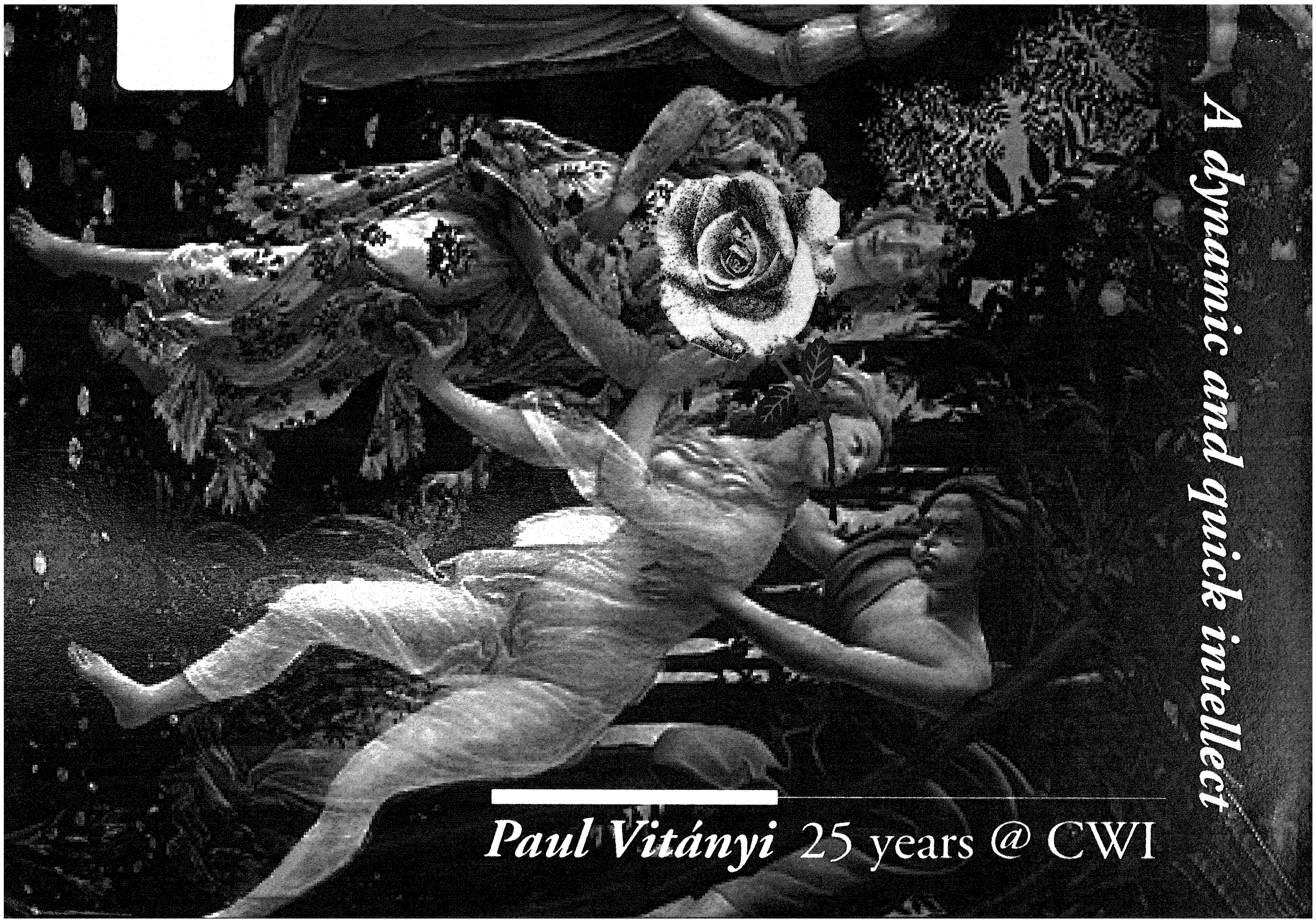
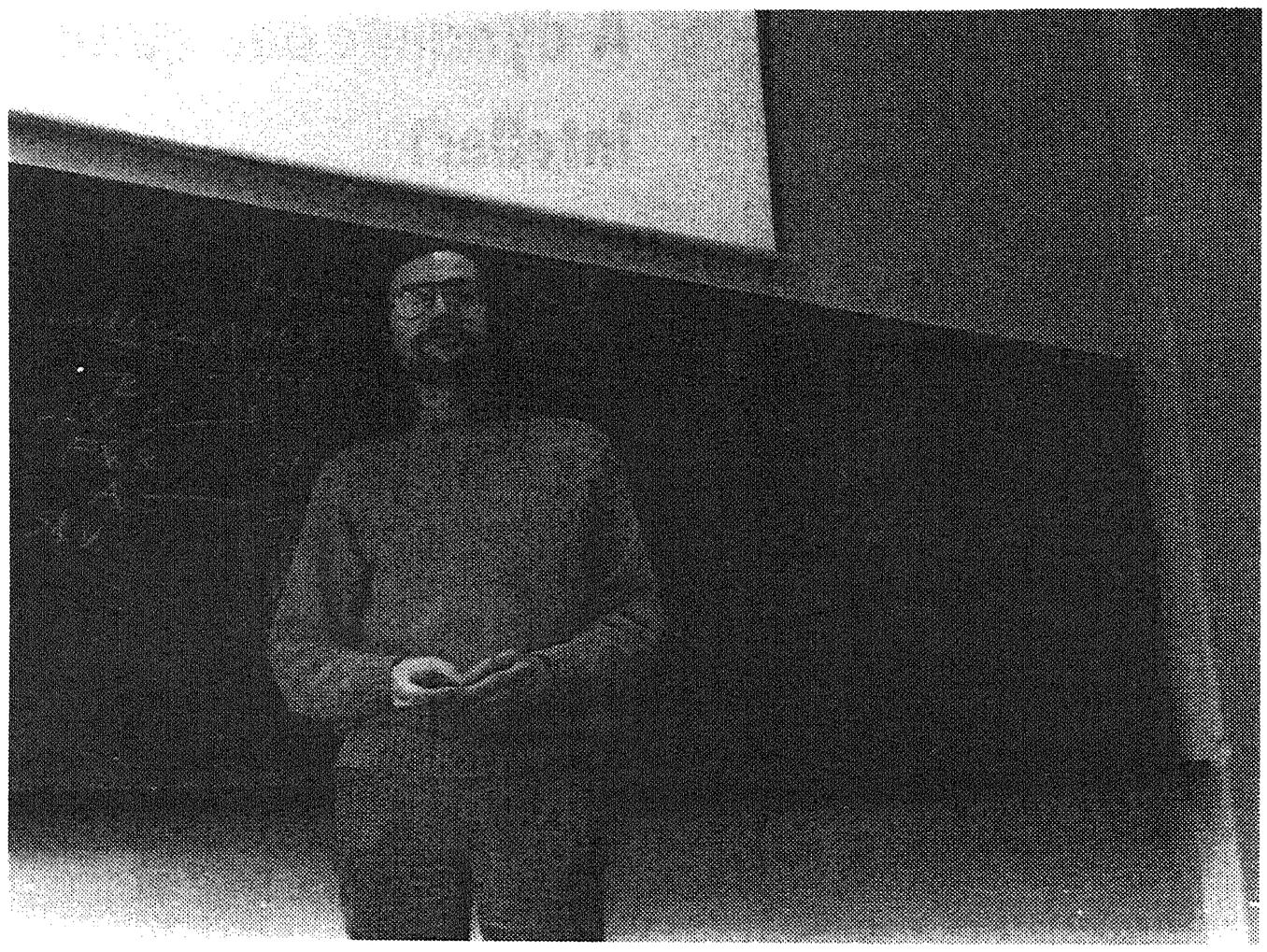

A dynamic and quick intellect
Paul Vitányi 25 years @ CWI

A dynamic and quick intellect



Paul Vitányi **A dynamic and quick**

25 years @ CWI **intellect**

CWI
Amsterdam, 1996

Editor: John Tromp
Cover: Tobias Baanders
Layout: Jaap-Henk Hoepman
Printing & binding: CWI, Amsterdam

Acknowledgements

Little did I suspect, when volunteering to take on the role of editor of this Liber Amicorum celebrating Paul Vitányi's $.01_2$ century (that's 25 years for the binary challenged) employment at the CWI, how refreshing it is to have 26 co-authors contribute to the compilation of no less than 144 pages in a matter of weeks. My prior 'editorial' experience, which is limited to the writing of my thesis, supervised by none other than Paul, brings back memories of quite a bit more perspiration.

Paul makes a grateful subject of such a volume, being an prolific, inspiring scientist as well as a colorful character in everyday life.

The title is stolen literally from a psychological report aimed at helping Paul choose among the different types of high school available in Holland. It mentions " ... een dynamisch en snelwerkend intellect" in addition to some less flattery remarks best omited here (I have it on good authority that poking fun at the subject is OK:-)

The photos in this Liber show the many faces of Paul through the years, while the last page features a photo of his parents.

Many people have helped to make this Liber see the light of day. I would like to extend my gratitude to all contributing authors, to Jaap-Henk Hoepman

for the slick layout, to Peter van Emde Boas for generously contributing a Foreword as well as access to his large photo collection, to Tobias Baanders for the cover design, to Jan Schipper for the timely printing, and to Paul's loving wife, Pauline van de Ven, for the colorful aquarelle much of whose beauty unfortunately went lost in the transformation to a greyscale image. Also a big thanks to the CWI for making this all possible.

Finally, no thanks to fileserver 'zeus' which insisted on a few dozen annoyingly slow reboots during the course of editing.

John Tromp, October 1996

Peter van Emde Boas **Foreword**

Paul Vitányi at the CWI for 25 years. The most surprising part of this message is obtained by extending its linguistic form: After 25 years, Paul Vitányi is *still* working at the CWI. Given the fact that CWI is the well known breeding place for Dutch Mathematicians and Scientists and that the larger part of his colleagues in the mean time have moved somewhere else, the question is: why is he still there?

Oh yes; Paul did go somewhere else in the mean time. He obtained a professorate at the University of Amsterdam when that institute decided that that was the appropriate way of both providing Paul with the Ius Promovendi he would need eventually, and the University with the enhanced prestige by having this internationally renowned Computer Scientist on its Faculty. Had he really wanted, he could easily have become full professor, either abroad or in the Netherlands, but why bother and subject yourself to the inconvenient aspects of this job like teaching basic material and administrative junk? He wants to be a researcher in the first place.

With Jan van Leeuwen and Peter van Emde Boas, Paul is one of the founders of Algorithms and Complexity in the Netherlands. From these three founders, Paul in some sense kept closest to this topic. He was also given the right

opportunities to do so. He had the competence and time to settle a number of notoriously hard open problems in machine model based complexity, obtaining some results which are optimal in the sense that one can show that they never will be improved upon. Faced with a rather chaotic state of the literature on descriptive complexity, he and Ming Li decided to inventory and summarise the existing literature, a project which eventually resulted in a series of surveys, a large number of new results and applications, and a bestselling monograph which, to a large extent, has enhanced his visibility. Finally, his interest in distributed computations also led him to some hard results in one of the most error ridden areas in this field: the theory of atomic registers.

His interest, however, was never constrained by these specific theories. Whenever new computational fields appeared he would get involved sooner or later. VLSI theory with the TU Eindhoven group, neural networks, learning theory (for which he took the initiative of establishing a European Network which didn't make it), and most recently Quantum Computation, where he applied for and obtained an interdisciplinary PhD student position at the University of Amsterdam one year before the establishing of such positions was regulated, simply by asking for it to the board.

In many regards, however, Paul should be considered a relic of a past which no longer exists. He applies high standards to both students and PhD researchers, and won't hesitate to fail them if those standards are not met. Still, this rarely happens, presumably because he gives the students time and experience to select themselves. His PhD students typically have over five accepted journal and conference papers by the time they graduate. He accepts a severe workload in Program Committees and editorial jobs. He stays out of politics and administration, disregarding the negative impacts this may have on his career. And regardless the ever increasing difficulties of finding the money, he spends long periods abroad to do research, give prestigious talks and write papers and books.

But in some ways Paul is lazy. His PhD was late—on a subject he was hardly working in at the time he graduated (the topic was Lindenmayer systems at a time he had fully moved towards machine model based complexity). He remained at his research position at the CWI while most of his colleagues took University positions. He didn't seriously consider a professorate until he could obtain the one which didn't require him to move. He first started to

submit applications to SION when he was threatened that bad things would happen if he didn't generate something other than theorems (needless to state that the project proposals written since were evaluated at top position in the annual rounds, until NWO decided to dissolve the existing peer evaluation procedure).

Also in some other regards Paul represents the professor from a past era. He is literate and shows it, knowing more about Dutch history and culture than many natives. He drives a car through Amsterdam at a time everybody considers that either "just not done" or crazy. He is notorious for his interaction with lower University administrators, particularly to those which either fail to provide him the required services (like an entry pass to the parking garage) or have the bad taste of bringing him the bad news that what he wants can't be done under the administrative rules. His office is a mess, with huge piles of papers and manuscripts. His bookshelves hardly show more order; below these papers there must be a treasure of unpublished results and manuscripts like the book on counting. For years he has lived right in the Center of Amsterdam on one of the canals, accepting the risk that his visitors (and occasionally he himself) get mugged by the omnipresent junks and thieves.

Altogether, Paul represents stability and quality in a world which is moving too fast and seems to have become crazy. Both qualities seem to indicate that he may survive even when everything else (including those who have been so good to contribute to this volume) have gone. So let us therefore forget for the moment that age regulations won't give him the possibility, and wish him another 25 years of health, interaction with fine colleagues and marvelous results.

Contents

Krzysztof R. Apt

Puzzling Paul

1 Past

I think I met Paul for the first time on Friday, September 8, 1975. Namely, I joined the Mathematical Centre on September 1, 1975 and got a desk in a room on the second floor. Another person in the room was Jophien van Waalen. The third desk belonged to Paul who was at that time on vacation. Paul returned to work one week later and after brief introductions we started exchanging puzzles. As we were brought up in different parts of Europe, we happened to know mostly different ones. So not surprisingly, the day quickly passed before we concluded the exchange.

One of the puzzles Paul told me then was: divide a square into eight acute triangles. I recall spending part of the next day, Saturday, trying to solve the puzzle. On Monday I proudly presented the solution to Paul.

Two years later, in 1977, I met Michael Rabin at a conference in Aalborg in Denmark. During the concert we were sitting next to each other and I told him this puzzle. A year later, or so, I met Rabin at another conference. As soon as he noticed me in a crowd, he rushed towards me and after a brief "Hello" told me that not only he quickly solved this puzzle, but also his wife,

who is a lawyer.

Some ten years later, while working in Austin, Texas, I suggested to solve the above puzzle to Edsger W. Dijkstra. I guess it must have been on Monday, because a day later Edsger emerged in my office informing me in his usual solemn voice that "we solved this puzzle during our afternoon club". I am not sure, but think that at the same time an invitation to join the "Tuesday Afternoon Club" was extended to me.

During the 1989-1990 academic year Paul visited me in Austin (for the second time in three years ...) and gave a survey talk on Kolmogorov Complexity. During the talk he mentioned the following problem due to John von Neumann: implement a fair coin using a biased coin. Among those present in the audience was Dijkstra. Soon after Paul's talk Dijkstra presented to him a solution. This problem intrigued Dijkstra so much that shortly after Paul's visit he wrote a note entitled "Making a fair roulette from a possibly biased coin", eventually published in *Information Processing Letters*, 36(4), page 193, 1990.

The association with Austin brings me to the following puzzle which I believe I have invented myself but for some reason forgot to pose to Paul. To test the knowledge of first-order logic I used to ask the students in Austin the following problem during an exam: "Is infertility hereditary"? Somehow this problem always led to heated discussions, so I started to pose it also to my colleagues. The most brilliant answer was given to me by the logician Andreas Blass from Ann Arbor University in Michigan. He immediately replied: "It depends. If heredity means that *some* child inherits the property, then the answer is "No", but if it means that *all* children inherit the property, then the answer is "Yes"."

Sometime during 1976 Professor Igarashi from Japan visited the Mathematical Centre and Paul and I went out for a drink with him. During the evening, after yet another round of beers, we started discussing the puzzles about the muddy children and about forty unfaithful wives. Somehow, the discussion turned into the topic of how to reason formally about such puzzles. None of us pursued the topic but we all felt that it should be possible.

Ten years later both Paul and myself witnessed the meteoric rise of the theory of knowledge, initiated by Daniel Lehman, Joe Halpern and others. The theory was developed precisely in order to reason formally about such puzzles and quickly turned out to be useful for reasoning about distributed

processes and network protocols, as well. Since then hundreds of papers on this subject were written and recently a book has appeared (R. Fagin et al., *Reasoning about Knowledge*, The MIT Press, 1995). A couple of years ago Paul and myself wondered how shortsighted we were that we did not pursue this topic in 1976.

2 Present

Paul is often puzzled why I don't work on more interesting topics like, say, Kolmogorov Complexity. So let me explain the flavour of the problems I study by means of a ... puzzle. Our goal is to analyze in detail the following well-known crypt-arithmetic problem.

Replace different letters by different digits, so that

```
  SEND
+ MORE
------
 MONEY
```

forms a correct sum.

As immediately noticed by Olga Troelstra, `M` has to be 1, so `S` has to be 8 or 9. Continuing along these lines we can actually conclude that `S` has to be in fact 9 etc.

But does there exist some more systematic way of analyzing such puzzles? The answer is yes, by means of constraint propagation.

3 Preliminaries

By a *linear expression* we mean a term of the form

$$a_1x_1 + \ldots + a_nx_n + a_{n+1}$$

where $n \geq 0$, $a_1, \ldots, a_n$ are non-zero integers, $x_1, \ldots, x_n$ are different variables and a_{n+1} is an integer.

By an *inequality constraint* we mean a formula of the form

$$s \leq t$$

where s and t are linear expressions and by an *equality constraint* we mean a formula of the form

$$s = t.$$

Further, we call $x \neq y$, for variables x, y, an *inequation constraint*. By an *interval* we mean an expression of the form

$$[a..b]$$

where a and b are integers. $[a..b]$ denotes the set of all integers between a and b. Finally, by a *range* we mean an expression of the form

$$x \in I$$

where x is a variable and I is an interval. Given the interval $[a..b]$, we abbreviate $x \in [a..b]$ to $x = a$ if $a = b$ and to **false** if $a > b$.

We introduce here propagation rules of the form

$$\frac{constraint, ranges_1}{ranges_2}$$

where *constraint* is a constraint which involves n variables, $ranges_1$ is a sequence of n ranges, each involving a different variable of *constraint* and similarly for $ranges_2$.

Such a rule is called *sound* if for all integer values of the involved variables, the truth of the constraint and the ranges in its premise implies the truth of the ranges in its conclusion. So a rule is sound if each solution of the constraint in the premise that satisfies the ranges in the premise also satisfies the ranges in the conclusion. Informally, a rule is sound if the replacement of the old ranges by the new ones does not lead to the loss of any solution. For example, the rule

$$\frac{x \leq 3, x \in [0..5]}{x \in [0..3]}$$

is sound. Somewhat less obvious is that the rule

$$\frac{3x - 5y = 4,\ x \in [0..9], y \in [1..8]}{x \in [3..8], y \in [1..4]}$$

is also sound (see further in the text).

4 Propagation Rules for Inequality Constraints

Each inequality constraint can be equivalently written in the form

$$\sum_{i \in POS} a_i x_i - \sum_{i \in NEG} a_i x_i \leq b \tag{3.1}$$

where a_i is a positive integer for $i \in POS \cup NEG$ and b is an integer. Assume now the ranges

$$x_i \in [l_i..h_i]$$

for $i \in POS \cup NEG$.

Choose now some $j \in POS$. Rewriting (3.1) as

$$a_j x_j \leq b - \sum_{i \in POS - \{j\}} a_i x_i + \sum_{i \in NEG} a_i x_i$$

and by subsequently computing the maximum of the expression on the right-hand side w.r.t. the ranges of the involved variables we get

$$x_j \leq \alpha_j$$

where

$$\alpha_j = \frac{b - \sum_{i \in POS - \{j\}} a_i l_i + \sum_{i \in NEG} a_i h_i}{a_j}$$

so, since the variables assume integer values,

$$x_j \leq \lfloor \alpha_j \rfloor.$$

We conclude that

$$x_j \in [l_j..min(h_j, \lfloor \alpha_j \rfloor)].$$

By symmetry we get analogous conditions for x_j with $j \in NEG$. This brings us to the following propagation rule for inequality constraints

$$\frac{\sum_{i \in POS} a_i x_i - \sum_{i \in NEG} a_i x_i \leq b,\ x_1 \in [l_1..h_1], \ldots, x_n \in [l_n..h_n]}{x_1 \in [l'_1..h'_1], \ldots, x_n \in [l'_n..h'_n]} \tag{3.2}$$

where for $j \in POS$

$$l'_j = l_j,\ h'_j = min(h_j, \lfloor \alpha_j \rfloor)$$

and for $j \in NEG$

$$l'_j = max(l_j, \lceil \beta_j \rceil), h'_j = h_j.$$

with

$$\beta_j = \frac{-b + \sum_{i \in POS} a_i l_i - \sum_{i \in NEG - \{j\}} a_i h_i}{a_j}$$

(see E. Davis, "Constraint Propagation with Interval Labels", *Artificial Intelligence* 32(2), pp. 281–331, 1987.

5 Propagation Rules for Equality Constraints

By combining the corresponding propagation rules for these two inequality constraints we obtain the following propagation rule for an equality constraint.

$$\frac{\sum_{i \in POS} a_i x_i - \sum_{i \in NEG} a_i x_i = b,\ x_1 \in [l_1..h_1], \ldots, x_n \in [l_n..h_n]}{x_1 \in [l'_1..h'_1], \ldots, x_n \in [l'_n..h'_n]} \tag{3.3}$$

where for $j \in POS$

$$l'_j = max(l_j, \lceil \gamma_j \rceil),\ h'_j = min(h_j, \lfloor \alpha_j \rfloor)$$

with

$$y_j = \frac{b - \sum_{i \in POS - \{j\}} a_i h_i + \sum_{i \in NEG} a_i l_i}{a_j}$$

and for $j \in NEG$

$$l'_j = max(l_j, \lceil \beta_j \rceil),\ h'_j = min(h_j, \lfloor \delta_j \rfloor)$$

with

$$\delta_j = \frac{-b + \sum_{i \in POS} a_i h_i - \sum_{i \in NEG - \{j\}} a_i l_i}{a_j}.$$

As an example of the use of the above constraint propagation rule reconsider the constraint $3x - 5y = 4$ with the ranges $x \in [0..9]$ and $y \in [1..8]$. A straightforward calculation shows that $x \in [3..9]$, $y \in [1..4]$ is the conclusion of rule (3.3). Another application of the rule yields $x \in [3..8]$ and $y \in [1..4]$ upon which the process stabilizes. In fact, both $x = 3, y = 1$ and $x = 8, y = 4$ are solutions of the equation $3x - 5y = 4$, so the ranges cannot be reduced anymore.

6 Propagation Rules for Inequation Constraints

The propagation rules for inequation constraints are of a particularly simple form.

$$\frac{x \neq y,\ x \in [a..b], y = a}{x \in [a + 1..b]}$$

$$\frac{x \neq y,\ x \in [a..b], y = b}{x \in [a..b - 1]}$$

and similarly with $x \neq y$ replaced by $y \neq x$.

Recall that $y = a$ is a shorthand for $y \in [a..a]$. In the above rules we dropped $y = a$ (respectively $y = b$) from their conclusions. It is clear that these rules are sound.

7 Back to the SEND + MORE = MONEY puzzle

Formally, this puzzle calls for a solution of the equality constraint

$$\begin{aligned} & 1000S + 100E + 10N + D + 1000M + 100O + 10R + E \\ = \quad & 10000M + 1000O + 100N + 10E + Y \end{aligned}$$

and 28 inequation constraints $x \neq y$ for $x, y \in \{S, E, N, D, M, O, R, Y\}$ with, say, x preceding y in the alphabetic ordering, with the range [1..9] for S and M and the range [0..9] for the other variables.

By rewriting the above equality as

$$9000\mathrm{M} + 900\mathrm{O} + 90\mathrm{N} + \mathrm{Y} - (91\mathrm{E} + \mathrm{D} + 1000\mathrm{S} + 10\mathrm{R}) = 0$$

and applying the propagation rule (3.3) with the ranges as given above we obtain the following sequence of new ranges:

$$S = 9, E \in [0..9], N \in [0..9], D \in [0..9], M = 1, O \in [0..1], R \in [0..9], Y \in [0..9].$$

At this stage a repeated use of the same rule yields no new outcome. However, by virtue of the fact that $M = 1$ we can now apply the propagation rule for the inequation constraint $M \neq O$ to conclude that $O = 0$. Using now the facts that $M = 1, O = 0, S = 9$, the propagation rules for disequalities can be repeatedly applied to shrink the ranges of the other variables. This yields the following new sequence of ranges:

$$S = 9, E \in [2..8], N \in [2..8], D \in [2..8], M = 1, O = 0, R \in [2..8], Y \in [2..8].$$

Now five successive iterations of rule (3.3) yield the following sequences of shrinking ranges of E and N with other ranges unchanged:

$$E \in [2..7], N \in [3..8],$$

$$E \in [3..7], N \in [3..8],$$

$$E \in [3..7], N \in [4..8],$$

$$E \in [4..7], N \in [4..8],$$

$$E \in [4..7], N \in [5..8]$$

upon which the constraint propagation process stabilizes. In particular, the inequation propagation rules are not applicable either.

So using the constraint propagation rules we reduced the original ranges to

$$S = 9, E \in [4..7], N \in [5..8], D \in [2..8], M = 1, O = 0, R \in [2..8], Y \in [2..8].$$

From now backtracking, again coupled with a constraint propagation, yields effortlessly the following unique solution:

$$S = 9, E = 5, N = 6, D = 7, M = 1, O = 0, R = 8, Y = 2.$$

All this looks like an elaborate procedure, but in constraint logic programming languages, like CHIP or ECLIPSE this algorithmic behaviour can be achieved by the following, almost self-explanatory program:

```
:- lib(fd).

send(List):-
    List = [S,E,N,D,M,O,R,Y],
    List :: 0..9,
    alldistinct(List), % a built-in that generates
                       % the inequality constraints
    1000*S + 100*E + 10*N + D +
    1000*M + 100*O + 10*R + E
#= 10000*M + 1000*O + 100*N + 10*E + Y,
    M ## 0,
    S ## 0,
    labeling(List). % a built-in that generates the successive
                    % possible values for all variables in List.
```

8 Future?

Another problem Paul and I discussed in those early days was the following one. Suppose that you solved a famous problem, say Fermat's Last Theorem, and want to keep it secret but still want to have a proof that you solved the problem before a given date (for example, earlier than another scientist who eventually publishes the solution). First we wanted to use as the only means a stamp machine at a post office by means of which you can put a time stamp on your piece of paper. But then there is no satisfactory solution — it suffices to go to a post office, put a time stamp on a blank sheet of paper and then to write the relevant proof on this sheet of paper any time later.

But with the additional help of a copying machine there is a satisfactory solution. Can you see why? We assume here that there is a way to distinguish between an original and a photocopy and also that there is a way to certify that a photocopy is indeed a copy of the original (for example, by using handwriting).

First note that the following scheme does not work: you write your solution down, then go to the post office, time stamp it, and then photocopy it. Indeed, you can again go to the post office with a blank sheet of paper and time stamp it. Several years later you impudently copy the proof from a journal onto this sheet of paper, photocopy it and subsequently claim that this is your proof written down several years ago.

The right solution is: you write your solution down, photocopy it and *then* you put a stamp on the original. Namely, this proves that you put the stamp *after* you photocopied the original (the photocopy shows no trace of a time stamp), and you photocopied the original *after* you wrote your proof. So the proof was written before the date of the time stamp.

Again, we did not pursue the matter how to formalize these arguments. Paul, perhaps after all we should?

Peter Asveld

An Inefficient Representation of the Empty Word

1 Preface

From July 1978 till the summer of 1980 Paul Vitányi and I shared an office at Mathematical Centre (former name of C.W.I.), located at Tweede Boerhaavestraat 49 in the Oosterpark-neighborhood of Amsterdam. A couple of weeks after I arrived, Paul and I moved together with Arie de Bruin to a freshly painted room which happened to be the half of a former class room, like many offices in that old school building, separated from its companion by a rather thin wall. Without difficulty or any intention we could hear at least 50% of our neighbors' conversations as they could from ours. And the view from our room was something special too: a 19th century part of the former Amstel brewery which happened to be of too little interest to industrial archaeology in order to survive in later days.

Another well-known feature of the old Mathematical Centre was its library: famous for its outstanding collection of books, journals and reports on

[0] Department of Computer Science, Twente University of Technology, P.O. Box 217, 7500 AE Enschede, the Netherlands (infprja@cs.utwente.nl)

mathematics and computer science, and remarkable for the dishes of rat poison on the floors, inviting the nightly intruders to commit suicide rather than nibbling at the precious volumes.

In 1978 both Paul and I finished our Ph.D. work on parallel rewriting, although our theses have very little in common; cf. [1, 7]. But we both had the intention to change our subject into the direction of computational complexity: a subject in which Paul achieved much more than I did; see e.g. [4], which also gives me the opportunity to turn to a more serious matter that is related to the title of this note.

2 Introduction

As we all know from [4], Kolmogorov complexity deals with short and efficient descriptions of objects, e.g., descriptions of strings over the alphabet $\{0, 1\}$. But in some cases there is a long, involved way to go before we arrive at the object to be described even in a context that looks anything but complex.

To be more precise, we will show in this note that it is possible to describe a very simple object (viz. the empty string) by means of a simple method (viz. a well-known elementary instance of Post's tag system) in a very complicated way. To illustrate this point we need a few definitions and some notation.

3 Definitions, Examples and a Conjecture

Post's tag system is a rewriting system $T = (\Sigma, d, P, \omega_0)$ which consists of an alphabet Σ, a natural number d, a function $P : \Sigma \to \Sigma^*$ and an initial string ω_0 over Σ. We obtain the string ω_{i+1} from ω_i —denoted by $\omega_i \Rightarrow \omega_{i+1}$— for $i = 0, 1, \ldots$ as follows. If ω_i equals the empty word λ, then $\omega_{i+1} = \lambda$. If σ is the first symbol of a nonempty word ω_i, then we append $P(\sigma)$ to the right end of ω_i and we delete the first d symbols from $\omega_i P(\sigma)$; this yields ω_{i+1}.

As an example—to which we will restrict our attention in the sequel—consider $T_0 = (\{0, 1\}, 3, P_0, \omega_0)$ with $P_0(0) = 00$ and $P_0(1) = 1101$. This instance T_0 has originally been introduced by Post in [5, 6], but there are still a lot of open questions in connection with T_0; cf. e.g. [8, 2, 3].

From a dynamical system point of view[1], there are three possibilities for

[1] It is legitimate to consider T_0 as a non-linear discrete deterministic dynamical system;

the ultimate behaviour of T_0 (as function of the initial string ω_0, of course); viz.

1. T_0 explodes: for each natural n there is an ω_i such that its length exceeds n;
2. T_0 reaches a fixed point: T_0's only fixed point is the empty string λ;
3. T_0 enters a periodic cycle of which the period must be an even number. (And indeed, for each even natural number there exists an initial string that ends in such a cycle; cf. [3]).

As examples of (2) we mention that each string ω_0 of the form 0^k reduces in k steps to λ: since $0^k \Rightarrow 0^{k-1}$, we have $0^k \Rightarrow^k \lambda$. Less trivially, $\omega_0 = 000100000$ yields in 13 steps λ because $000100000 \Rightarrow 10000000 \Rightarrow 000001101 \Rightarrow 00110100 \Rightarrow 1010000 \Rightarrow 00001101 \Rightarrow 0110100 \Rightarrow 010000 \Rightarrow 00000 \Rightarrow^5 \lambda$.

The case $\omega_0 = 100000100$ is an example of (3); the (smallest) period is 6, since $\omega_{12} = \omega_{18} = \omega_{24} = \ldots$; cf. [2].

Since no examples of (1) are known, we have the following conjecture.

Conjecture 1 For each initial string ω_0 over $\{0, 1\}$, the ultimate behaviour of Post's tag system T_0 is either (2) or (3).

There are a few similarities between this conjecture and the famous $3n+1$-problem[2]; see [3]. But apart from the validity of Conjecture 1, it is clear that the only string that we can describe in terms of ultimate behaviour of T_0 is T_0's single fixed point: the empty string λ.

4 The Result

The advance of modern computers enables us to simulate rewriting systems for quite a number of rewriting steps. A few years ago some people reported in the appropriate news groups of Internet that, according to their simulations, T_0 would probably explode for $\omega_0 = (100)^{110}$ or, equivalently, for $\omega_0 = 1^{330}$,

cf. [3] for an explanation.

[2] Also known as Ulam's problem, Collatz's problem, or Syracuse problem.

thereby providing a possible counterexample to Conjecture 1. Note that for the invalidity of Conjecture 1 we need a proof and not a simulated initial part of a possibly infinite derivation.

But continuing the simulation to the very end shows a different picture: after 43 913 328 040 672 rewriting steps T_0 reaches the empty string. The maximal length of intermediate strings is 31 299 218; it occurs after 14 392 308 412 264 rewriting steps. Of course, the first 110 steps are trivial: for $i = 1, \ldots, 110$ we have $\omega_i = (100)^{110-i}(1101)^i$, whereas the end of the rewriting process looks like

1328040598 01110100001101110100000000001101000011011101
1328040599 10100001101110100000000001101000011011110100
1328040600 00001101110100000000001101000011011101001101
1328040601 01101110100000000001101000011011101001101000
1328040602 01110100000000001101000011011101001101000000
1328040603 10100000000001101000011011101001101000000000
1328040604 00000000001101000011011101001101000000001101
1328040605 00000001101000011011101001101000000110100
1328040606 00001101000011011101001101000000011010000
1328040607 01101000011011101001101000000011010000000
1328040608 01000011011101001101000000110100000000000
1328040609 00011011101001101000000110100000000000000
1328040610 11011101001101000000110100000000000000000
1328040611 11101001101000000110100000000000001101
1328040612 01001101000000110100000000000011011101
1328040613 01101000000110100000000000001101110100
1328040614 01000000110100000000000001101110100000
1328040615 00000110100000000000001101110100000000
1328040616 00110100000000000001101110100000000000
1328040617 10100000000000001101110100000000000000
1328040618 00000000000001101110100000000001101
1328040619 00000000011011101000000000110100
1328040620 00000011011101000000000011010000
1328040621 00011011101000000000001101000000
1328040622 11011101000000000001101000000000
1328040623 11101000000000001101000000001101

1328040624 01000000000011010000000011011101
1328040625 00000000011010000000011011110100
1328040626 00000011010000000011011110000
1328040627 00011010000000011011101000000
1328040628 11010000000011011110100000000
1328040629 10000000011011101000000001101
1328040630 00000011011101000000011011101
1328040631 00011011101000000011011110100
1328040632 11011101000000011011110000
1328040633 11101000000011011110100001101
1328040634 01000000011011110100011011101
1328040635 00000011011110100011011110100
1328040636 00001101110100011011110000
1328040637 01101110100011011101000000
1328040638 01110100011011110100000000
1328040639 10100011011110100000000000
1328040640 00001101110100000000001101
1328040641 01101110100000000000110100
1328040642 01110100000000011010000
1328040643 10100000000001101000000
1328040644 00000000011010000001101
1328040645 00000011010000000110100
1328040646 00001101000000011010000
1328040647 01101000000011010000000
1328040648 01000000110100000000
1328040649 00000110100000000000
1328040650 00110100000000000000
1328040651 10100000000000000
1328040652 00000000000000001101
1328040653 0000000000110100
1328040654 0000000011010000
1328040655 000001101000000
1328040656 00110100000000
1328040657 1010000000000
1328040658 000000000001101
1328040659 0000000110100

1328040660 000011010000
1328040661 01101000000
1328040662 0100000000
1328040663 000000000
1328040664 00000000
1328040665 0000000
1328040666 000000
1328040667 00000
1328040668 0000
1328040669 000
1328040670 00
1328040671 0
1328040672 λ

of which the last 9 steps are again trivial. The first column in this table shows i mod $(2 \cdot 10^9)$. An overview of the string length as function of the discrete time parameter i is depicted in Figure 1.

The simulation of T_0 for $\omega_0 = (100)^{110}$ is no simple task; it takes about 80 MIPSyears. Clearly, this number depends on the way the simulation is coded but, as far as I can see, there is not much to be gained. Note that, due to the sequential character of the rewriting process, parallelism is of no use either. Storing all intermediate strings is out of the question, since this would take about $214 \cdot 10^9$ Gb. And still you want to store some useful information for making snapshots like Figure 1, and for the unlucky moments that your system crashes.

5 Conclusion

We showed that it is possible to describe the empty string in a very inefficient way using rather simple means only, and that Conjecture 1 stands firm as ever.

Bibliography

[1] Asveld, P.R.J.: *Iterated Context-Independent Rewriting - An Algebraic Approach to Families of Languages* (1978), Twente University of Technology,

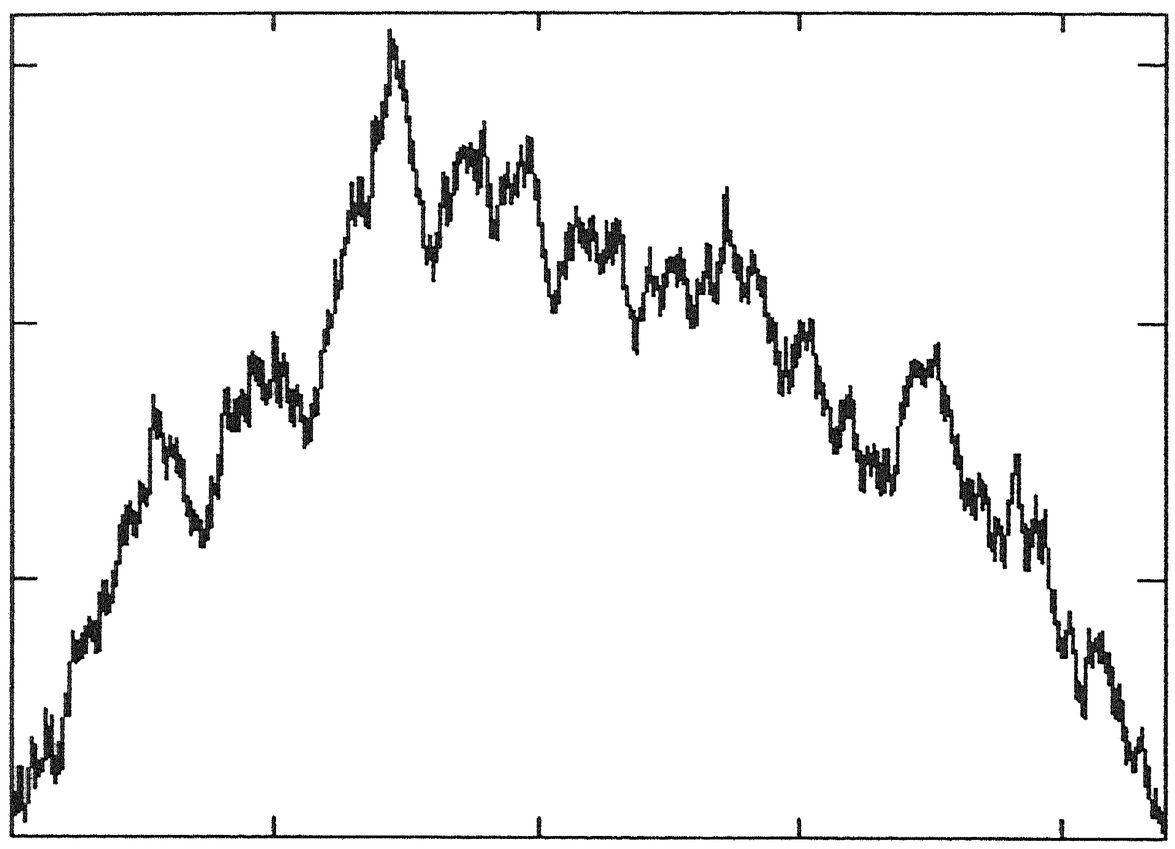


Figure 1 *string length oscillations*

Enschede.

[2] Asveld, P.R.J.: On a Post's system of tag, *Bull. Europ. Assoc. Theor. Comput. Sci.* No. 36 (1988) 96–102.

[3] Asveld, P.R.J.: Post's system of tag—A simple discrete nonlinear system, pp. 26–31 in: J.P. van der Weele (ed.): *Proceedings of Nonlinear Dynamics—Twente 91* (1992), Center for Theoretical Physics, University of Twente, Enschede, The Netherlands.

[4] Li, M. & Vitányi, P.: *An Introduction to Kolmogorov Complexity and Its Applications* (1993), Springer-Verlag, New York, Berlin, Heidelberg, etc.

[5] Post, E.M.: Formal reductions of the general combinatorial decision problem, *Amer. J. Math.* 65 (1943) 197–215.

[6] Post, E.M.: Absolutely unsolvable problems and relatively undecidable propositions—An account of an anticipation, pp. 340–433 in: M. Davis (ed.): *The Undecidable—Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions* (1965) Raven, New York.

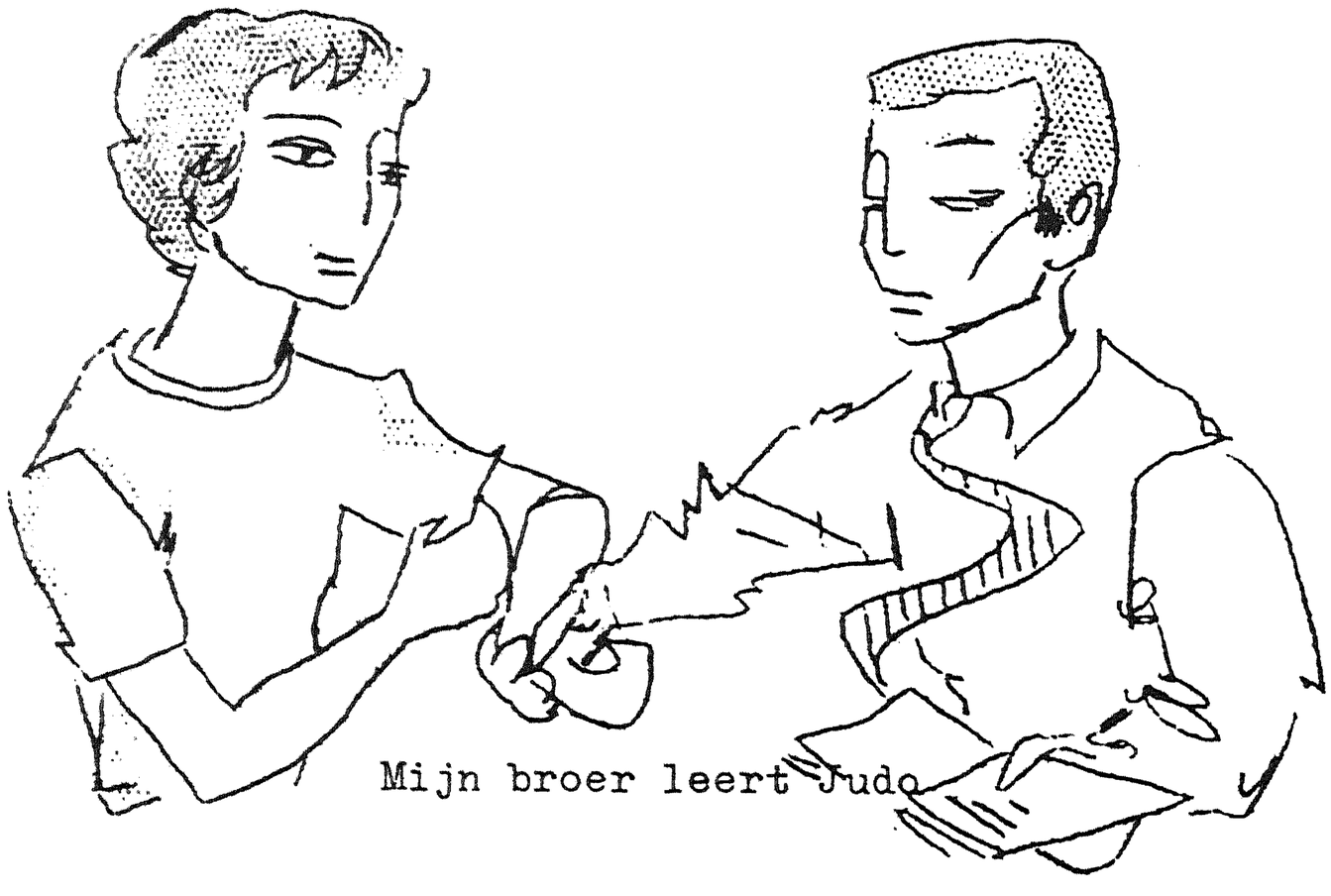
[7] Vitányi, P.M.B.: *Lindenmayer Systems—Structure, Languages, and Growth Functions* (1978), Free University, Amsterdam.

[8] Wanatabe, S.: Periodicity of Post's normal process of tag, pp. 83–99 in: *Proc. Symp. on Math. Theory of Automata 1962* (1963), Polytechnic Press, Brooklyn, N.Y.

Grzegorz Rozenberg **Dear Paul**

The very best wishes on the occasion of the "25th anniversary". I still remember the L-days when your scientific interest was "on the cellular level" in biology. Since then you have achieved amazingly much! Now you went back to nature to study quantum computing. With my interest in DNA computing I may now be your neighbour in the world of natural computing. Perhaps one day our research interests will cross again—if so, then I look forward to it.

All the best, Grzegorz



Mijn broer leert Judo

Bea Vitányi

Mijn broer leert Judo

Het vrouwvolk ringeloort en knevelt
mannekracht
Joost van den Vondel

Mijn broer leert Judo, uit een boekje. "Geef me je pols 'ns", zei hij laatst tegen me. In vol vertrouwen stak ik hem m'n pols toe. Hij draaide er eens flink aan en zei: "Zie je, als ik even doordraai is hij gebroken". Toen pakte hij mijn been en begon daaraan te draaien. Daarna probeerde hij of hij me kon wurgen. Dat werd me te gortig, ik vloog met gebalde vuisten op hem af.

"Ha", zei hij, "dacht je soms dat je tegen me op kon?" Ik haalde goed uit, met een diepe kniebuiging, zoals je dat in boeken leest. Maar voor ik hem kon raken, kreeg ik een klap tegen m'n kin. Ik verloor nu alle voorzichtigheid uit het oog en liep met maaiende vuisten op hem af. "Wacht even", riep m'n broertje, in 't nauw gedreven: "even m'n boekje pakken".

[0] 5 Gymnasium

Johan van Benthem

Bisimulation: The Never-Ending Story

1 Paul Vitányi

This booklet celebrates Paul Vitányi's 25 years of employment at CWI. But my connection with him concerns what may be just a footnote in his life. Around 1990, Paul diversified, staying 80% of his working life as a key CWI researcher, while becoming a UvA professor, and hence a colleague of mine, for the remaining 20%. This puts me in the position of a mistress being invited to contribute a speech to a silver wedding anniversary. Unabashed by all of this, I will go on to say that Paul is one of the stars of our ILLC. His research is profound, productive, and highly original. His silver book with Ming Li on Kolmogorov complexity documents an amazing research program, shedding new light on an impressive range of algorithmic and logical topics, down to things I thought settled once and for all (such as basic formal language theory). Among his many other interests, I was impressed by his rapid mastery of quantum computing, resulting in the first joint research project between an FWI and an FNS professor in the emerging faculty WINS, which

[0] Institute for Logic, Language & Computation, Universiteit van Amsterdam

landed on the desk of our provisional board, far ahead of any established procedures. (This un-Dutch panache was then matched by our granting the project straightaway...)

2 Modal Logic and Invariance for Bisimulation

My offering is a small observation, testifying to the overlap in research at ILLC and CWI. It concerns a result from my 1976 dissertation which seems the earliest connection between modal logic and what is now the central process equivalence of bisimulation. Consider a first-order language making assertions about, in UvA-speak, poly-modal Kripke models (in CWI-speak, 'labeled transition systems').

Theorem 1 (Modal Invariance Theorem) *For first-order formulas $\phi(x)$, the following are equivalent:*

1. *$\phi(x)$ is invariant under bisimulations*
2. *$\phi(x)$ is definable by a modal formula.*

A key proof step for the MIT replaces the 'linguistic' relationship of 'modal equivalence' between two Kripke models by a 'structural' one of bisimulation, among elementarily equivalent models (satisfying the same first-order sentences). Thus, we can pass back-and-forth between the CWIworld of bisimulation and the UvA-world of modal equivalence:

Lemma 2 (First Switching Lemma) *For rooted models M, x and N, y, the following are equivalent:*

1. *M, x and N, y satisfy the same modal formulas*
2. *M, x and N, y have elementary extensions M^+, x and N^+, y, respectively, which bisimulate (with x connected to y).*

In a picture, this observation gives us the following square of related notions:

$$
\begin{array}{ccc}
M, x & \textit{modal equivalence} & N, y \\
\textit{first-order equivalence} & & \textit{first-order equivalence} \\
M^+, x & \textit{bisimulation} & N^+, y
\end{array}
$$

Maarten de Rijke's 1993 dissertation used walks through this diagram for a systematic comparison between modal logic and first-order logic. Even so, I do not think we have the full picture yet: further UvA-CWI 'Gestalt switches' are useful. Here is a new one.

3 Boosting Modal Equivalence to First-Order Equivalence

The expressive power of a language may sometimes be 'boosted' to that of a normally richer one. Here is another CWI-UvA square, from Andréka, van Benthem & Németi 1996.

Lemma 3 (Second Switching Lemma) *For rooted models M, x and N, y, the following are equivalent:*

1. *M, x and N, y satisfy the same modal formulas*
2. *M, x and N, y have bisimilar models M^*, x and N^*, y, respectively, which are elementarily (i.e., first-order) equivalent.*

This time, the picture has turned around—allowing us different back-and-forth trips:

$$
\begin{array}{ccc}
M, x & \textit{modal equivalence} & N, y \\
\textit{bisimulation} & & \textit{bisimulation} \\
M^*, x & \textit{first-order equivalence} & N^*, y
\end{array}
$$

One application of this new style of thinking is my constructive thought for this occasion:

3.1 A Quick New Proof of the Modal Invariance Theorem

Let $\phi(x)$ be a first-order formula which is invariant for bisimulation, and define mod(ϕ) to be the set of all modal consequences of ϕ. We show that mod(ϕ) $\models \phi$, from which fact a modal equivalent for ϕ follows by Compactness (namely, as the conjunction of some finite subset of mod(ϕ)). So, let $M, x \models$ mod(ϕ). By standard reasoning, the full modal type of M, x together with $\phi(x)$ is finitely satisfiable. Compactness then gives a model N, y for ϕ which is modally equivalent to M, x. Now consider the two models M^*, x, N^*, y given by clause (ii) in our Lemma. As N^*, y is bisimilar to N, y, ϕ holds there (by its bisimulation invariance). Hence, ϕ (being first-order) holds in the elementarily equivalent model M^*, x, too, and thus also $M, x \models \phi$ (again by ϕ's bisimulation invariance). ■

3.2 Coda

If we analyze the proof of the last Lemma more precisely, we can find out more. The models M^*, N^* are what are often called 'tree unravelings' of Kripke models or LTSs (cf. Bergstra & Klop 1989), with duplication of nodes. Let an *extended modal formula* be any modal formula constructed using also the 'universal modality' of truth "in all worlds". Then we get a reduction between first-order logic and modal logic of the following type.

Fact 4 *There exists an effective translation taking first-order formulas ϕ to extended modal formulas $m(\phi)$ such that, for all models M, x and their duplicated tree unravelings M^*, x, $M, x \models m(\phi)$ iff $M^*, x \models \phi$.*

This suggests an intriguing generalization of the relevant notion of 'logical translation'. The MIT presupposes a well-known effective translation taking modal formulas to first-order formulas equivalent to them on the class of Kripke models. There is no effective converse, however—as this would reduce first-order logic (undecidable) to modal logic (decidable). But this situation changes when we broaden the notion of translation, allowing equivalences across *different models*. Then, as we have just seen, the game becomes wide open again.

4 Conclusion

I conclude with a reflection on my title, and obliquely also on Paul Vitányi's double career. Dear Reader, there is more between CWI and UvA than is dreamt of in your philosophy...

Bibliography

[1] H. Andréka, J. van Benthem & I. Németi, *Modal Logics and Bounded First-Order Fragments*, 1996. To appear in the Journal of Philosophical Logic.

[2] J. van Benthem, *Exploring Logical Dynamics*, CSLI Publications, Stanford, 1996. Distributed by Cambridge University Press, Cambridge.

[3] J.A. Bergstra & J-W Klop, *Process Theory Based on Bisimulation Semantics*, in J.W. de Bakker, W-P de Roever & G. Rozenberg, eds., *Linear Time, Branching Time and Partial order in Logics and Models for Concurrency*, Springer, Berlin, Lecture Notes in Computer Science 354, 50-122, 1989.

[4] M. de Rijke, 1993, *Extending Modal Logics*, Ph.D. dissertation, Institute for Logic, Language & Computation, University of Amsterdam.

Peter van Emde Boas

Are Sheets obsolete?

For readers of this contribution who are involved with a profession which requires regular presentations of technical material I bring some bad news: in the near future the efforts and time required for preparing such a presentation will be multiplied by a substantial constant factor, assuming that you are sufficiently trendy. New technologies in audio-visual tools provide you with completely new possibilities and at the same time increase the volume of non-subject related decisions to make while preparing a talk.

As Paul certainly will remember there was a time that a lecturer would prepare a class and give it using just two basic tools: a blackboard and a piece of chalk. If you were very careful you might even use colored chalk; this improvement was, however, at some time prohibited at the Math department of the University of Amsterdam on behalf of the cleaning ladies who had problems wiping the colored dust from the floor.

It was during the time (early 70-ies) of my move to Computer Science that I discovered that there existed such a marvelous device as an overhead

[0] dedicated to Paul Vitányi on the occasion of his 25th Anniversary at the CWI. ILLC, FWINS, Universiteit van Amsterdam, Plantage Muidergracht 24, 1018 TV Amsterdam.

projector. The Computer Scientists were giving great talks and presentations using clear hand written slides in several colors. Quality of the presentations was evidently enhanced by the fact that the work of designing the proper lay-out of information was off-loaded from the actual presentation to the preparation stage. Hence preparing a talk became more work than just understanding the material and designing an outline on what you wanted to present; the actual visual part of the talk was prepared entirely off-line.

For more than a decade the use of transparencies on an overhead projector was a prime discriminator between Mathematicians and Computer Scientists. Mathematicians would consistently use chalk and blackboards and Computer Scientists would use the projector. Other scientists, if using a projection screen at all, would primarily prepare dias. There were other differences as well. For a mathematician it was evident that the public was motivated for the subject, knew the basic backgrounds in advance and was interested in proofs. The computer scientists typically would provide motivation, introduce the hearer to the field and rarely give more than proof outlines in the presentation. Contrary to Mathematics, selling the result was more important than proving it.

If a mathematician would use transparencies at all he would typically use it as a blackboard, writing his formulas with the same density of symbols. The backbenchers in the classroom could never read what was written on the screen. However the speed of presentation would be much faster since the mathematician no longer had to write these formulas on the blackboard.

I myself made the complete switch from blackboard to slides in the late seventies. Occasionally this would cause problems, for example when traveling to Eastern conferences where the unique overhead projector was a scarcely available object and a blown lamp was fatal. I also remember the trip in 1981 to Israel where I gave four talks at four different locations, provoking a blown fuse everywhere.

The next stage in the development was the invention of the word processor and the laser printer. At some point the evident discovery was made that these devices could not just generate papers but also slides for a presentation. But basically the lecturers would prepare the same lay-out they used when preparing their slides by hand. Most WYSIWYG word processors had the option to increase the font size from 12 to 18 or 24 (never use fontsize below 18 for slides), making use of this option become the discriminator between

Mathematical and Computer Science use of this tool. Some Mathematicians even today seem not to have discovered this option. For the tag based wordprocessors like TEX and LATEX (or the Script product at IBM where I prepared my first Computer Generated slides back in 1985) special formats and style files for slides were invented.

Evidently the slides became cleaner, clearer but also more sterile. The time for presentation on the other hand became dependent on how effective the lecturer was in manipulating the computer tools connected to the word processor (editors, shells and operating systems). Personally I never reached a stage where preparing slides on a computer went faster than preparing the same slides by hand.

One feature these computer generated slides were missing was the use of color. The color printer still had to be invented or be sufficiently reduced in price to become available as a tool for preparing a presentation. Some impoverished locations like the ILLC department in the Euclid building even today don't have access to a colour printer on which slides can be prepared. You could in principle always print a slide in black and enhance it by adding colour by hand.

However, in the meantime even this technology seems to have become obsolete. The new breakthrough is caused by two devices: laptop computers, and reliable transviewers and/or the video projectors generating a full color image on a large screen.

Technology is unstoppable. I still remember the evening presentation during the 1991 LICS conference at the Aula Major of my university where a rather new piece of technology was used: an overhead projector where the image was captured by a camera in order to be subsequently projected by video projector. This forward and backward transformation from optical to electronic signal caused sufficient loss to make the densely hand written slides unreadable, which, in combination with a defective audio system, sufficed to cause disaster. The speaker involved is a well known elder Dutch Mathematician and Computer Scientist.

Another typical example was the opening plenary talk at the 1994 OOPSLA in Portland, where the organisers had the great idea of sharing this talk with another conference in North Carolina. It happened to be the case that the speaker was at the east coast and we were faced with the video presentation. The sound installation was rather weak, and the computer screens on which

the software system described in the talk was being demonstrated involved an unlucky choice of colours which were completely invisible by the time they appeared on our projection screen. I have been told that the poor speaker was never informed about the true nature of this disastrous failure of technology.

Originally the transviewers were monochromatic but rather sharp, whereas the video projectors always succeeded in blurring the images and misaligning the colours. However these devices today seem to work OK. At the 1996 OOPSLA in San Jose this October both the Audio and Video equipment worked and the various computer screens could be read throughout the room (which as usual housed over 2400 people). Since about four years LCD screens and transviewers provide clear colors and their resolution has increased to the level where you hardly notice that the pixels are coarser than a corresponding monitor screen.

Which brings me to the conclusion of the history and to the present. So far people preparing talks at a laptop seemed to use this device for generating coloured slides as before, bypassing the process of printing the slides and/or xeroxing them on plastic sheets. The laptops were used also to present demo's of computer programs discussed in the talk. The screens produced by these programs show dynamic visualisations of systems, but that depended on the application at hand. The only dynamic feature at the word processor level which I saw being used is the option to let a sequence af bulleted items appear in sequential order on the screen—let the slide grow in time rather than being completed at once. Also various tricks which we used to perform by placing slides on top of each other nowadays can be done on the laptop screen in a dynamic way.

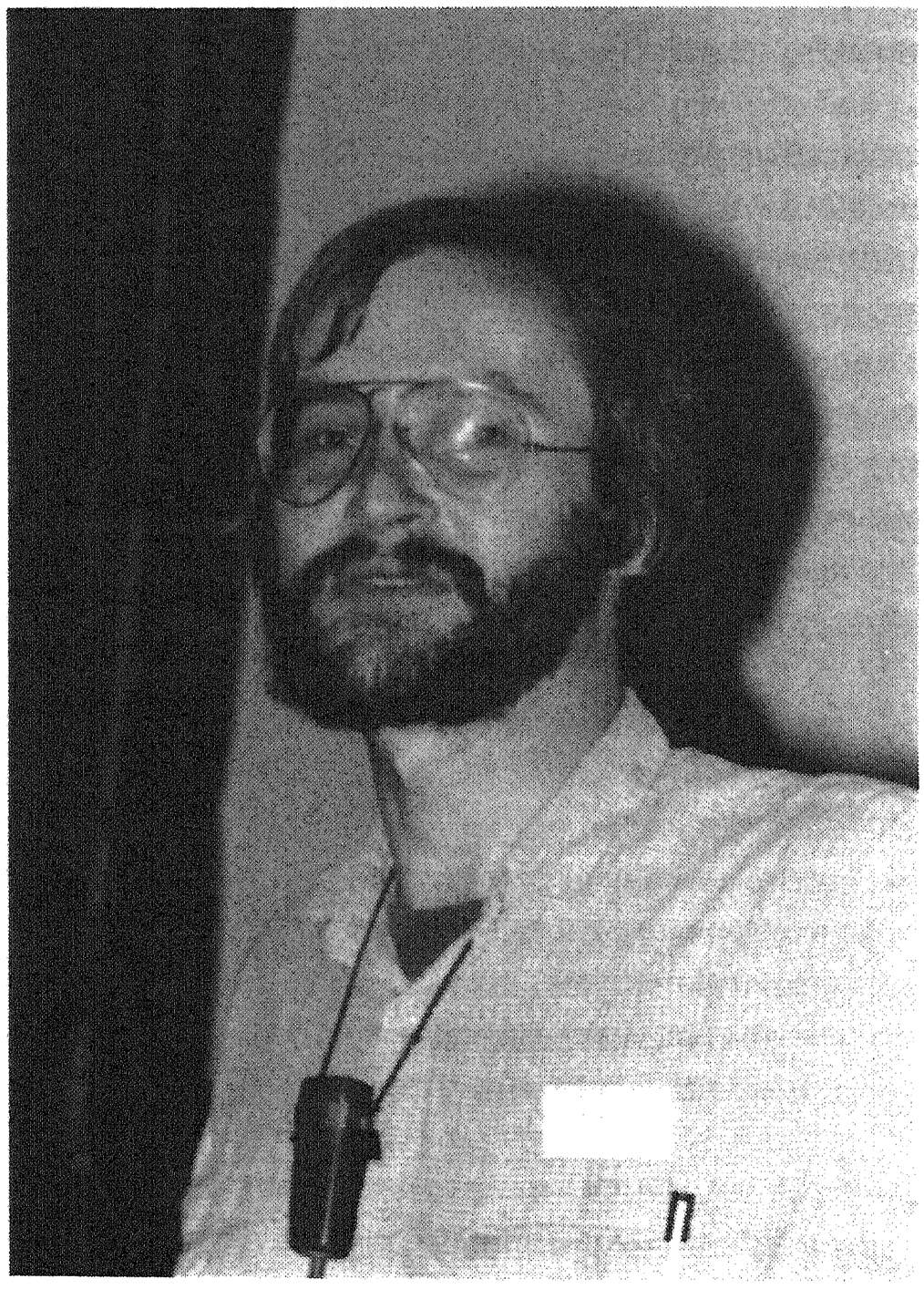
The new development which I noticed at OOPSLA'96 is that the boundary between word processing and the application program in these presentation is dissolving. Presumably the appearance of the JAVA language and the portable dynamic applets supported by the language is to blame. As a consequence ingredients in a slide no longer appear on their intended position but can be seen to move to it, bumping into and occasionally replacing existing material. These dynamizations are moreover supported using computer generated audio signals ranging from broken windows to screaming cars.

Evidently, preparing a talk using this new technology is not going to be business as usual. At OOPSLA'96 I experienced both talks where this technology was used in a meaningful way and talks where it was just a

distracting gimmick. The challenge is on us, both as producers and consumers to discover and propagate the guidelines on how to use these new techniques and tools in a responsible and informative manner. One thing however is evident: preparing a talk in this way will consume far more time than we are used to.

It may be a comforting message to learn that in this regard this year's opening address at OOPSLA was exceptional: the organizers had invited the well known Architect Christopher Alexander, whose ideas about patterns and pattern languages have been transferred from the world of Architecture to the world of Object Technology. Within only a few years Patterns has become a hot subject, leaving Alexander surprised about this strange community suddenly giving his ideas a far wider scope than he ever imagined himself. In his talk Alexander used no visual tools at all (no slides, no laptops): he read a presentation by heart including long pauses, leaving the audience fascinated for the 75 minutes the presentation took.

The old fashioned way therefore still is feasible, if you can manage it. And as long as Paul remains the scientist he currently is, he evidently won't need any of the above innovations.



Hellooo Boys

Harry Buhrman
Peter Grünwald
Jaap-Henk Hoepman
Barbara Terhal
John Tromp

A typical conversation among Paul Vitányi and his post-docs and Ph.D. students

A late summer afternoon. CWI, rooms M231-232, the door between both is open as usual. The south-faced offices overlook a small park on the WCW premises. Jaap-Henk, Harry, John, Peter and Barbara appear to be working (hard). To their annoyment, the sun blinds appear to move up and down more or less randomly. It is 5.00 PM.

ENTER Paul Vitányi, plastic cup of coffee in his hands.

PAUL: Hellooo Boys!
ALL: Hi Paul!
PAUL: Darn, that stupid Park Control Service decided my car was wrongly parked and towed it away last night.
JOHN: What the heck do you need a car for in Amsterdam anyway? Why not ride a bike to work everyday?
PAUL: Surely that's not healthy, John! *(turns to look at Harry's Koga Myata racing bike.)* Say, Harry, isn't that your racing bike? Or is it one of yours,

John?

HARRY: Yes Paul, it's mine.

PAUL: Why didn't you buy a 'RIH' racing bike. It's a much better brand.

HARRY: Well, Koga Myata is o.k. 'RIH' is very expensive Paul...

PAUL: That's only because you pay too much taxes, Harry!

HARRY: You tell me what else I can deduct then, Paul...

PAUL: Your daily cup of coffee at the office, for example!

PETER: A friend of mine who makes a lot of money at a commercial bank tries to deduct every single penny he earns. Would you believe he voted labour all his life?

PAUL: Peter, being your supervisor, I think I should tell you the following important fact of life: *Socialists are only in it for the money!*

JOHN: Come on, it is not as simple as that!

PAUL: John, listen to Newton: "Nature is pleased with simplicity, and affects not the pomp of superfluous causes."

PETER: but Paul, you took part in the *Maagdenhuisbezetting* yourself. I've even been told you possess Mao's Little Red Book! Where did your ideals go?

PAUL: I only joined the Maagdenhuisbezetting because of all the beautiful girls taking part in it!

JAAP-HENK: But don't you think it's more just if the rich transfer some of their possessions to the poor?

PAUL: Jaap, now I think...

JAAP-HENK: *(interrupting)* It's Jaap-Henk, not Jaap.

PAUL: Yes, I know, but 'Jaap' is much easier for foreigners and I think Jaap-Henk is not such a nice name anyway. But returning to what you just said, Jaap, I think that now *you* are the one whom I should teach something about the world we are living in. It is called the 'Matthew Effect'.

ALL: The Matthew Effect?????

PAUL: Listen to Matthew 29-30: "For to every one who has, more will be given, and he will have in abundance; but from him who has not, even what he has will be taken away. And cast the worthless servant into the outer darkness; there men will weep and gnash their teeth."

PETER: How come you know all these things?

PAUL: You should not forget that I have been educated by the Jesuits. I went to the school where they prepared the Dutch Catholic Establishment: Ruud

Lubbers and Hans van Mierlo went there too! I remember we had this very difficult final physics exam, and I had much higher marks than anybody...

HARRY: *(interrupting)* so what about this job interview you had to conduct today. Was the guy any good?

PAUL: Well, I do not think I'll take the guy. He did not even know what the Gamma function was!

BARBARA: Why would you need to know about the Gamma function if you're going to work on Kolmogorov Complexity?

PAUL: Whenever somebody applies for a job with me, I ask him whether he knows the Gamma function and whether he knows Newton's binomium. When I was a student...

JAAP-HENK: : ...yes I know: you knew all these things.

PAUL: And much more besides.

JOHN: Look Paul, I figured something out. Look—if we have a Turing Machine with three heads and square tails, we can simulate a multi-tape mind-reader with only three states using a quantum oracle.

PAUL: Hmmmm...simple.

JOHN: Maybe so Paul, but now it turns out that a quantum oracle can separate the heads from the bodies. Assuming such an oracle, if you have two heads but more than two bodies, you get an unsolvable n-body problem from which the separation follows immediately.

PAUL: That can't be right. In 1978 I was the first to prove that separating a head from a body requires an unbounded number of tails, even in the presence of an oracle. Which means that what you say is impossible.

BARBARA: But quantum oracles did not exist back then, so your result does not apply anymore! Trouwens, waarom spreken we eigenlijk geen Nederlands?

PAUL: Ja waarom niet? But it *does* apply.

HARRY: I think Paul's right.

JOHN: Wanna bet a loempia on that?

BARBARA: Harry, Paul, look: with quantum oracles, you can bring the tails in superposition, so you only need a bounded number of them now.

PAUL: That's not right.

JOHN: Yes it is, Paul.

PAUL: Well, probably not.

BARBARA: Sure is!

PAUL: Ok, well, you might be right.

PETER: Whether right or wrong, this way of doing it does seem a bit clumsy.

PAUL: Peter, listen to Dr. Johnson - when Boswell and he watched a bear dancing, Boswell remarked that the bear did not dance very well. Dr. Johnson replied: "My dear Boswell - the point here is not that the bear dances well. The point is that the bear dances *at all*." Anyway, I've got some email to catch up on.

Paul throws his cup into the waste bin and leaves.

Jan van Eijck

Simpleton Domains

Abstract The paper proposes an almost trivial denotational semantics for imperative programs.

1 Introduction

Favourite topics of my occasional lunch discussions with Paul Vitányi include (i) strategies for house buying in Amsterdam, (ii) dark streaks in the characters of people of local fame, and (iii) the triviality of much computer science research.

As I have just bought a house in the centre of Amsterdam and have no intention of selling it again in the near future, topic (i) has lost much of its relevance for me. Topic (ii) is inappropriate as a subject for this contribution, because I would like to avoid libel suits. This leaves me with topic (iii).

Experience during our lunch meetings has taught me that arguing against Paul's prejudices is hard going indeed. This time, therefore, I will go with the flow of Paul's opinions, and demonstrate that defining a denotational semantics for imperative programs can be an almost trivial exercise.

2 Extending Typed Logics with States

The following is a special case of a construction described in [1]. Let t be the type of truth values, and i the type of integers. Then the set of standard types over $\{i, t\}$ is given by:

$$T \quad ::= \quad i \mid t \mid (T_1, T_2)$$

Intuitively, the standard types are the types of all function spaces that can be constructed out of the basic domains of truth values and integers. Extend the set of standard types as follows:

$$U \quad ::= \quad * \mid T \mid (U_1, U_2)$$

The type $*$ is meant to be the type of all memory states of a machine with a given set of storage cells (or: registers, or: locations). Call the members of $U - T$ extended types. Intuitively, an extended type is a type in which $*$ occurs. Thus, the extended types denote function spaces constructed out of memory states (and possibly also truth values and integers). The details of the construction are given below.

Constants for all types T are allowed. For present purposes, we will assume that a constant *zero* of type i is available for naming the number 0, plus a constant *succ* of type (i, i) for the successor function, plus constants $\times$, $+$ and $-$, all of type $(i, (i, i))$, for multiplication, addition and subtraction, and finally constants $=$, $<$, $\leq$, all of type $(i, (i, t))$, for the relations of identity, less than and less or equal than on the integers.

For the extended types, only constants of types $(*, i)$ are allowed. Let $C_{(*,i)}$ be the set of all constants of type $(*, i)$. For our application we need that the set $C_{(*,i)}$ is countably infinite. Members of $C_{(*,i)}$ are called *store constants* because they are the stores or registers out of which the domain of states will be construed.

We allow variables in all types U, indeed we assume a countably infinite supply V_U of them for every type U.

The set of expressions of our language of typed logics with states (TLS) is given by the rule (we use C for the constants of the language, and V for the

variables):

$$E \quad ::= \quad C \mid V \mid (E_1, E_2) \mid (\lambda V.E) \mid (C_{(*,t)}|E_t)$$

The formation of applications (E_1E_2) is constrained by the requirement that E_1, E_2 must have types (U_1, U_2) and U_1, respectively. If this requirement is met, (E_1E_2) is well-formed, and its type is U_2. If V has type U_1 and E type U_2, then the type of $(\lambda V.E)$ is (U_1, U_2). The type of $(C_{(*,t)}|E_t)$ is $(*, *)$. An expression $(c|e)$ is called a *state changer*, because it is to be interpreted as a function from states to states which changes the input state by assigning a new value e to register c.

If the types of v, e and f are known, the types of $(\lambda v.e)$ and (ef) are uniquely determined, so we will not always write all subscripts for types. When writing down expressions, we will commit the usual sins of sloppiness in the interest of readability. An expression $\lambda v_1.(\lambda v_2.(\ldots(\lambda v_n.e)\ldots))$ will be written as $\lambda v_1 v_2 \ldots v_n.e$, and $(\ldots((ef_1)f_2)\ldots f_n)$ as $ef_1 \ldots f_n$, i.e., we assume that application, indicated by parentheses (and), binds more strongly on the left than on the right.

Assume that the domains for t and i are given by: $D_t = \{0, 1\}$ and $D_i = Z$ (the set of integers). Then the other domains are constructed as follows:

$$\begin{aligned} D_{(T_1,T_2)} &:= D_{T_1} \to D_{T_2} \\ D_* &:= C_{(*,i)} \to D_i \\ D_{(U_1,U_2)} &:= D_{U_1} \to D_{U_2}. \end{aligned}$$

If $s \in D_*$, $c \in C_{(*,i)}$ and $z \in Z$, we use $s[c \to z]$ for the function $f \in D_*$ which is given by $f(c') = z$ if $c \approx c'$ and $f(c') = s(c')$ otherwise (here $\approx$ is used for syntactic identity).

Let I be an interpretation for the standard constants of the language satisfying $I(c_T) \in D_T$ for $c_T \in C_T$ (we will call a set of TLS domains D_U together with such an interpretation I a TLS model), and let g be an assignment mapping variables in V_U to objects in D_U. Then the interpretation function $[[\cdot]]_{I,g}$

is defined as follows:

$$
\begin{aligned}
[[c_T]]_{I,g} &:= I(c_T) \\
[[v_U]]_{I,g} &:= g(v_U) \\
[[c_{(*,i)}]]_{I,g} &:= \text{the function given by } \lambda s.s(c) \\
[[(c_{(*,i)}|e_i)]]_{I,g} &:= \text{the function given by } \lambda s.s[c_{(*,i)} \mapsto [[e_i]]_{I,g}] \\
[[(e_{(U_1,U_2)}e_{U_1})]]_{I,g} &:= [[e_{(U_1,U_2)}]]_{I,g}([[e_{U_1}]]_{I,g}) \\
[[(\lambda v_{U_1}.e_{U_2})]]_{I,g} &:= \text{the function given by } \lambda d.[[e_{U_2}]]_{I,g[v_{U_1} \mapsto d]}
\end{aligned}
$$

Note in particular that the interpretation of an expression in $C_{(*,i)}$ is indeed a function in $D_* \to D_i$.

3 Notions of Reduction

Recall that the notion of beta reduction is the relation between an expression of the form $((\lambda v.e)e')$ and the expression e'' which is the result of substitution of e' for all free occurrences of v in e, with renaming of bound variables in e if necessary to prevent accidental capture of free variables in e'. A concrete example: $((\lambda x.(Px)c)$ is β related to (Pc). On the basis of this, the relations $\xrightarrow{\beta}$ (one step beta reduction) and $\xrightarrow{\beta}\!\!\!\!\!\rightarrow$ (the reflexive transitive closure of one step beta reduction) are defined for TLS expressions in the standard way. The relation $\xrightarrow{\beta}\!\!\!\!\!\rightarrow$ is called the relation of β reduction.

Because of the presence of state changers, we can also define a notion of reduction σ given by $\sigma = A \cup B$, where A is the relation between $(c\ ((c|e_i)e_*))$ and e_i (intuitively, this says that the result of looking up the value of c in the state which has e_i assigned to c is simply the value e_i), and B the relation between $(c\ ((c'|e_i)e_*))$, where $c \not\equiv c'$ and c and c' are both in $C_{(*,i)}$, and (ce_*) (intuitively, this says that looking up the value of c in a state $s[c' \mapsto d]$ boils down to looking up the value of c in s). Let $\xrightarrow{\sigma}\!\!\!\!\!\rightarrow$ be the relation of σ reduction. It is not difficult to show that σ reduction is sound, confluent and strongly normalizing.

Let $\beta\sigma$ reduction be the relation $\xrightarrow{\beta\sigma}\!\!\!\!\!\rightarrow$, i.e., the relation $\xrightarrow{R}\!\!\!\!\!\rightarrow$ for $R = \beta \cup \sigma$. $\beta\sigma$ reduction is sound, confluent, and strongly normalizing. Confluence follows from the fact that the two notions of reduction β and σ are orthogonal (i.e.,

β reduction steps do not spoil opportunities for σ reduction and vice versa). Here is an example $\beta\sigma$ reduction:

$$\begin{aligned}(\lambda p.(p\ ((c|n)s))(\lambda s.(y(cs))) &\xrightarrow{\beta} ((\lambda s.(y(cs)))\ ((c|n)s)) \\ &\xrightarrow{\beta} (y(c\ ((c|n)s))) \\ &\xrightarrow{\sigma} (yn).\end{aligned}$$

4 Equations as Terms

Equations in typed logic have the form $e = f$, where e, f are assumed to be of the same type. Since we assume a basic type t, with $D_t = \{1, 0\}$, we can consider equations as terms of type t, by extending the language with a rule:

$$E_t \quad ::= \quad E_U = E'_U$$

Note that this is not a real extension of the syntactic format, for it boils down to adding constants $Q_{(U,(U,t))}$ for every type U, and writing the term $((Q_{(U,(U,t))}e_U)f_U)$ as $e_U = f_U$. The interpretation of $e_U = f_U$ is given by:

$$[\![e_U = f_U]\!]_{I,g} \quad := \quad \begin{cases} 1 & \text{if } [\![e_U]\!]_{I,g} = [\![f_U]\!]_{I,g}, \\ 0 & \text{if } [\![e_U]\!]_{I,g} \neq [\![f_U]\!]_{I,g}. \end{cases}$$

We agree to write $e_t = f_t$ as $e \leftrightarrow f$. Call an expression of type t a formula. Some useful abbreviations for formulas and formula-forming operators can now be given:

$$\begin{aligned} \top &:= (\lambda x_t.x_t) = (\lambda x_t.x_t) \\ \bot &:= (\lambda x_t.x_t) = (\lambda x_t.\top) \\ \neg &:= (\lambda x_t.(x_t \leftrightarrow \bot)) \\ \wedge &:= (\lambda x_t y_t.(\lambda z_{(t,t)}.((zx) \leftrightarrow y)) = (\lambda z_{(t,t)}.(z\top))) \\ \forall x_U e &:= (\lambda x_U.e) = (\lambda x_U.\top) \end{aligned}$$

It is not difficult to see that these abbreviations all mean what the logical symbolism suggests.

We write $(e \wedge f)$ for $((\wedge e)f)$, and we abbreviate $\neg(\neg e \wedge \neg f)$ as $(e \vee f)$, $\neg(e \wedge \neg f)$ as $(e \to f)$, and $\neg\forall x_U \neg e$ as $\exists x_U e$, and, in general, we will omit brackets where such practice doesn't create ambiguity.

A formula f_t is valid if $[[f_t]]_{I,g} = 1$ for every choice of I, g.

By the construction of the domains, for every store constant $c_{(*,i)}$ of the language the following formula is valid:

$$\forall a_* \forall x_i \exists b_* (b = ((c|x)a))).$$

This says that TLS models have enough states to model assignment of any appropriate new value to any register.

On the other hand, the construction of the TLS models ensures that there are not too many states. This is expressed by the validity of the following formula:

$$(a_* = b_*) \leftrightarrow \forall y_{(*,i)} (ya = yb).$$

This says that the identity of a state is fully determined by how functions in $(*, i)$ act on it. In fact, the identity of a state is fully determined by how the set of functions corresponding to the registers in $C_{(*,i)}$ act on it, but of course the property of being a function corresponding to a register $c_{(*,i)}$ is not expressible in the language.

5 Denotational Semantics in TLS

Assume that $c_{(*,i)}$ ranges over $C_{(*,i)}$. Then the following defines the language of *while* programs with program variables taken from the set of symbols $C_{(*,i)}$.

$$\begin{aligned}
N &::= \mathbf{0} \mid \cdots \mid \mathbf{9} \mid N\mathbf{0} \mid \cdots \mid N\mathbf{9} \\
A &::= c_{(*,i)} \mid N \mid (A_1 + A_2) \mid (A_1 - A_2) \mid (A_1 * A_2) \\
B &::= A_1 = A_2 \mid A_1 < A_2 \mid A_1 \leq A_2 \mid \neg B \mid (B_1 \wedge B_2) \\
S &::= c_{(*,i)} := A \mid \text{skip} \mid (S_1; S_2) \\
&\quad \mid \text{if } B \text{ then } S_1 \text{ else } S_2 \mid \text{while } B \text{ do } S
\end{aligned}$$

The meanings of the statements of the *while* language are partial functions over $C_{(*,t)} \to Z$, i.e., members of $(C_{(*,t)} \to Z) \to (C_{(*,t)} \to Z)$, where Z is used for the set of integers. Recall that the domain D_* has the form $C_{(*,t)} \to Z$. Thus, the members of $(C_{(*,t)} \to Z) \to (C_{(*,t)} \to Z)$ are in fact members of $D_* \to D_*$. We can represent a member of $D_* \to D_*$ (a partial function from states to states) by means of its graph, which is a member of $D_{(*,(*,t))}$. Meanings of *while* statements can be represented as members of $D_{(*,(*,t))}$. We will now show that one can refer to them by means of TLS expressions.

First define a relation $\sqsubseteq$ on $D_{(*,(*,t))}$ by means of:

$$f \sqsubseteq g := \forall su(fsu \to gsu),$$

where f, g are expressions of type $(*,(*,t))$, and s, u variables of type $*$.

Next, define a function FIX in $(D_{(*,(*,t))} \to D_{(*,(*,t))}) \to D_{(*,(*,t))} \to D_t$ by means of:

$$\text{FIX} := \lambda Fg.\forall su(Fgsu = gsu),$$

where F is a variable of type $((*,(*,t)),(*,(*,t)))$, g a variable of type $(*,(*,t))$, and s, u are variables of type $*$.

If H has type $D_{(*,(*,t))} \to D_{(*,(*,t))}$ and h type $D_{(*,(*,t))}$, then FIX Hh has type D_t. The expression FIX Hh reduces to:

$$\forall su(Hhsu = hsu),$$

which is true iff (the interpretation of) h is the graph of a fixed point of (the interpretation of) H.

Finally, define a function μ in $(D_{(*,(*,t))} \to D_{(*,(*,t))}) \to D_{(*,(*,t))}$ by means of:

$$\mu := \lambda Fsu.\exists g(\text{FIX } Fg \wedge \forall d(\text{FIX } Fd \to g \sqsubseteq d) \wedge gsu),$$

where F a variable of type $((*,(*,t)),(*,(*,t)))$, g, d are variables of type $(*,(*,t))$, and s, u variables of type $*$.

If H has type $D_{(*,(*,t))} \to D_{(*,(*,t))}$, then μH has type $D_{(*,(*,t))}$, and μH reduces to

$$\lambda su.\exists g(\text{FIX } Hg \wedge \forall d(\text{FIX } Hd \to g \sqsubseteq d) \wedge gsu).$$

This expression denotes the graph of the least fixed point of (the interpretation of) H, if it exists, and the empty relation on D_* otherwise.

The translation of *while* to TLS now proceeds in four stages: first we translate numerals into type z, next arithmetical expressions into type $(*, i)$, then boolean expressions into type $(*, t)$, and finally statements into type $(*, (*, t))$.

Translation of numerals into type z, using the constants for zero, successor, multiplication, addition and subtraction mentioned before (we write the binary operators in infix notation, and we abbreviate *succ zero* as *one*, *succ one* as *two*, ..., and *succ nine* as *ten*):

$$\begin{aligned}
\mathbf{0}^\circ &:= zero \\
&\vdots \\
\mathbf{9}^\circ &:= nine \\
(N\mathbf{0})^\circ &:= N^\circ \times ten \\
&\vdots \\
(N\mathbf{9})^\circ &:= (N^\circ \times ten) + nine
\end{aligned}$$

Translation of arithmetical expressions into type $(*, i)$, using the functions $*, +, -$ on the domain Z:

$$\begin{aligned}
N^\bullet &:= \lambda s.N^\circ \\
c^\bullet_{(*,i)} &:= c_{(*,i)} \\
(A_1 + A_2)^\bullet &:= \lambda s.(A_1^\bullet s + A_2^\bullet s) \\
(A_1 - A_2)^\bullet &:= \lambda s.(A_1^\bullet s - A_2^\bullet s) \\
(A_1 * A_2)^\bullet &:= \lambda s.(A_1^\bullet s \times A_2^\bullet s)
\end{aligned}$$

Note that the translation of $c_{(*,i)}$ is c itself. This is correct, for $c_{(*,i)}$ is a variable in the programming language, but a store constant in the TLS language, and it has type $(*, i)$.

Translation of boolean expressions into type $(*, t)$ (as we mentioned already, we assume that we have relations $=, <, \leq$ in $(i, (i, t))$ available; we

will write them in infix notation):

$$\begin{aligned}
(A_1 = A_2)^\bullet &:= \lambda s.(A_1^\bullet s = A_2^\bullet s)\\
(A_1 < A_2)^\bullet &:= \lambda s.(A_1^\bullet s < A_2^\bullet s)\\
(A_1 \leq A_2)^\bullet &:= \lambda s.(A_1^\bullet s \leq A_2^\bullet s)\\
(\neg B)^\bullet &:= \lambda s.\neg(B^\bullet s)\\
(B_1 \wedge B_2)^\bullet &:= \lambda s.(B_1^\bullet s \wedge B_2^\bullet s)
\end{aligned}$$

Translation of statements into type $(*,(*,t))$:

$$\begin{aligned}
(c_{(*,i)} := A)^\bullet &:= \lambda s u.((c_{(*,i)}|(A^\bullet s)s) = u)\\
\text{skip}^\bullet &:= \lambda s u.(s = u)\\
(S_1;S_2)^\bullet &:= \lambda s u.\exists v(S_1^\bullet s v \wedge S_2^\bullet v u)\\
(\text{if } B \text{ then } S_1 \text{ else } S_2)^\bullet &:= \lambda s u.((B^\bullet s \wedge S_1^\bullet s u) \vee (\neg B^\bullet s \wedge S_2^\bullet s u))\\
(\text{while } B \text{ do } S)^\bullet &:= (\mu\lambda g s u.((B^\bullet s \wedge \exists v(S^\bullet s v \wedge g v u))\\
&\qquad \vee(\neg B^\bullet s \wedge s = u))))
\end{aligned}$$

Proposition: The translation $S^\bullet$ is correct in the sense that the interpretation of $S^\bullet$ corresponds to the semantic function specified for S by the standard semantics for the *while* language.

The proposition can be proved by first specifying the semantics of *while* by some other means, e.g., in operational style, in terms of transition rules, and then engaging in a lengthy induction exercise, too boring for the present festive occasion (see, e.g., [2]).

The proposition shows that TLS provides a cheap way of representing the meanings of *while* programs, by means of the simpleton domains of the title of this contribution. TLS can also be used for defining a compositional dynamic semantics for natural language, but this is a topic for another paper.

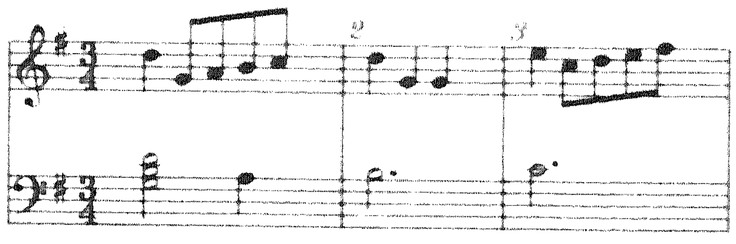
Bibliography

[1] J. van Eijck, *Typed logics with states*, Manuscript, CWI and ILLC, Fall 1996.

[2] H.R. Nielson and F. Nielson, *Semantics with Applications*, John Wiley and Sons, 1992.

Anton Eliëns

Abstraction and Complexity



1 Abstract

Living in one af the ancient quarters of Amsterdam, I regularly meet Paul Vitányi, drinking a beer, shopping on the market, or just strolling around. As inhabitants of our beloved city, we both admire its richness and diversity. As scientists, no doubt, we are both in awe with the richness and diversity of our discipline, Computer Science. As a tribute to Paul's achievements in this field, I would like to take an intellectual stroll around the notions of Abstraction and Complexity, the ABC of Computer Science. On the Web. Where else? To follow the links, see http://www.cs.vu.nl/~eliens/amice/vitanyi/.

2 Introduction

The short musical phrase above is taken from the Notebook of Anna Magdalena Bach. I have used twelve of these phrases in my book on the *Principles*

[0] Vrije Universiteit, Department of Mathematics and Computer Science, De Boelelaan 1081, 1081 HV Amsterdam. (eliens@cs.vu.nl)

of Object-Oriented Software Development [1], as decorations and, to some extent, for pedagogical reasons. As I say in the Preface of my Book: *Despite their apparent simplicity, they are acknowledged by experienced scientists as being hard to play properly, yet they are among the standard exercises for beginning pianists.* I also explain how these phrases reflect the concepts underlying object-oriented programming, easy to learn but hard to play.

Apparently, it is not very hard to acquire some fluency in *object-speak*, yet true understanding requires more serious effort. Deceptively simple, yet intrinsically difficult.

Some still regard Object-Oriented Programming merely as a trend, to be replaced sooner or later by some other trend. Perhaps particular programming languages will be replaced by some others. For example C++, which is by many regarded as too complex, might be replaced by Java. Time will tell. Perhaps, some methods of software development will be replaced by others. Undoubtedly. Nevertheless, in the history of Computer Science, Object-Orientation is definitely a step forward, building on the control-flow abstractions from the Structured Programming doctrine and the data abstractions taught by the Abstract Data Type school.

Object-Orientation allows for mastering complexity by abstraction. Reflecting on my experience in teaching Object-Oriented Programming, I am painfully aware of the difficulty of transferring this knowledge. Object-Oriented Programming is not about Programming. It is not about Algorithms. It is about how to organize programs, how to represent programs in a flexible manner making use of Object-Oriented Technology. Teaching Object-Oriented Programming requires abstracting from the complexities of a technology that enables one to master complexity by employing abstractions. Enough! The lingo of Computer Science seems to be playing a game with us. What is Abstraction? What is Complexity? And how are they related?

For those who wish to bear with me, I will explore the meaning of the ABC of Computer Science, Abstraction and Complexity, by investigating how they are deployed in various contexts of usage. This investigation includes some empirical research, that is an inventory of their occurrence on the Web, as well as sucking my favorite resource, my thumb. For those who doubt the fruitfulness of this endeavor, have a beer. And read on.

somewhat more detail how I stumbled upon the subject. Being a bit lazy, I simply incorporate two fragments from my Book. The first fragment comes from the Preface. It explains the approach I've taken in my book to present the material. The second fragment comes from one of the introductory chapters in part I, where I explain in what sense object-oriented programming affects the complexity of programs, and in what sense it doesn't.

Abstraction The approach taken ... may be characterized as *abstract*, in the sense that attention is paid primarily to concepts rather than particular details of a solution or implementation language. By chance, in response to a discussion in my class, I looked up the meaning of *abstract* in a dictionary, where to my surprise I learned that one of its meanings is *to steal, to take away dishonestly*. Jokingly, I remarked that this meaning sheds a different light on the notion of *abstract data types*, but at a deeper level I recognized the extent to which the ideas presented ... have profited from the ideas originally developed by others. My rendering of these ideas in a more abstract form is, however, not meant to appropriate them in a dishonest way, but rather to give these ideas the credit they deserve by fitting them in a context, a framework encompassing both theoretical and pragmatical aspects of object-oriented computing. As one of the meanings of the adjective *abstract*, the dictionary also lists the word *abstruse* (not easy to understand). There is no need to say that, within the limits of my capabilities, I have tried to avoid becoming abstruse.

Complexity Our model of a computing device [2] does quite precisely delimit the domain of computable problems, and gives us an indication of what we can expect the machine to do for us, and what not. Also, it illustrates what means we have available to program such a device, in order to let it act in the way we want. Historically, the Turing machine model may be regarded as a mathematical description of what is called the Von Neumann machine architecture, on which most of our present day computers are based. The Von Neumann machine consists of a memory and a processor that fetches data from the memory, does some computation and stores the data back in

memory. This architecture has been heavily criticized, but no other model has yet taken its place. This criticism has been motivated strongly by its influence on the practice of programming. Traditionally, programs for the Von Neumann architecture are conceived as sequences of instructions that may modify the state of the machine. In opposition to this limited, machine oriented view of programming a number of proposals have been made that intended to arrive at a more abstract notion of programming, where the machine is truly at the service of the programmer and not the other way around.

One of these proposals to arrive at a more abstract notion of programming is advocated as the *object-oriented approach*. Before studying the intrinsics of the object-oriented approach, however, it may be useful to reflect on what we may expect from it. Do we hope to be able to solve more problems, or to solve known problems better? In other words, what precisely is the contribution of an object-oriented approach?

Based on the characterization of a computing device, some answers are quite straightforward. We cannot expect to be able to solve more problems, nor can we expect to reduce the computational complexity of the problems that we can solve. What an object-oriented approach can contribute to, however, is simply in providing better means with which to program the machine. Better means to reduce the chance of (human) errors, better means also, to manage the complexity of the task of programming (but not to reduce the computational complexity of the problem itself). In other words, by providing abstractions that are less machine oriented and more human oriented, we may enlarge the class of problems that we can tackle in the reality of software engineering. However, we simply cannot expect that an object-oriented approach may in any sense enlarge our notion of what is computable.

4 Empirical Results

The material presented before has been taken directly from my book. As I am working on the second edition, I might change it or omit it altogether. The real effort in writing this essay went into searching the Web for enlightenment. Somehow, abstraction and complexity seem to be two sides of a single coin, dual notions in other words. Well, are they? Or do they belong together in a

more intricate manner?

In searching the Web with *Alta Vista* I got 9000 documents matching the query for Abstraction. A brief selection of those will be presented below. For Complexity I got 30000 hits. To my surprise the titles for Complexity did not contain a reference to Paul Vitányi's work on Kolgomorov complexity. That is no real surprise since he did work on Kolmogorov Complexity. Due to sloppy spelling I learned that both Kolmogorov exists (2000 hits) as well as Kolgomorov (13 hits), including a document referring to a paper discussing Kolgomorov complexity and Hausdorff dimension. Searching for Vitányi did result in over 500 documents including *Paul's Homepage* and a fairly large number related to *Li's and Vitányi's book*. As a side effect, this tribute affects Paul's visibility on the Web.

Before taking a look at the material found, let me note that there is an interesting difference, apart from the number of hits, between the notions of Abstraction and Complexity, which becomes evident when looking at the documents found. Apparently, Complexity is a much more respectable notion since there is a *Journal of Complexity* as well as a *Faculty of Complexity*. In contrast, I did not find any evidence for the existence of a similar *Journal of Abstraction*, let alone a *Faculty of Abstraction*. Yet, there is a company named *Abstraction, Ltd*, dealing with abstraction. What else?

5 Abstraction

Since this is merely an exploratory tour, there are no results in the sense of *categories of usage* or a deeper analysis in any sense. Nevertheless, the list below more or less truthfully reflects what can be found on the Web and it does give an impression of the variety of meanings that adhere to abstraction. Abstraction is an abstract concept, indeed.

Surprisingly, there is only one page that carries *Abstraction* as it sole title. It explains that *Abstraction is a design technique that focuses on the essential aspects of an entity and ignores or conceals less important or non-essential aspects.*

The next reference explains the opposition of *Functional Abstraction versus Structural Decomposition*, which is explained by saying that: *The behavior of a system can be analyzed in two ways. One is by decomposition into parts the behavior of which is known in isolation, The behavior of the system is described*

in terms of cause-and-effect relations. Another is by abstraction to a functional level and separation of the loops of interaction of interest. In other words, there is (just) a difference in focus, so it seems.

The next document found introduced *The Tool Abstraction Concept*, but alas, the idea of Tool Abstraction remains unexplained because this is (or was) a dead link!

Looking outside the realm of Computer Science may be informative as well. The page about *Painterly Abstraction in Modernist American Poetry* with the intriguing subtitle *Infinite Incantations of Ourselves* argues that *modernist poets have tended to resist the received values of their contemporary culture by finding idealizing principles in modes of pure abstraction.* Interesting, but complex. No doubt, Computer Science may be characterized as *Applied Abstraction* at best.

The next page, promising to pave the road *Towards a New Model of Abstraction in Software Engineering* looks somewhat more familiar: *The view of abstraction on which software engineering is based does not support the reality of practice: it suggests that abstractions hide their implementation, whereas the evidence is that this is not generally possible. This discrepancy between our basic conceptual foundations and practice appears to be at the heart of a number of portability and complexity problems.*

Could it be that the Abstraction commonly applied in Software Engineering is close to *The Lyrical Abstraction* which is aptly characterized by Jackson Pollock's utterance: *"When I'm in my painting, I'm not aware of what I'm doing".*

Well, it's time to sober up. So let's look at a new promising trend in Software Engineering *Design Patterns : Abstraction and Reuse of Object-Oriented Design* which proposes *design patterns as a new mechanism for expressing object-oriented design expertise. Design patterns identify, name, and abstract common themes in object-oriented design. They capture the intent behind a design by identifying objects, their collaborations, and the distribution of responsibilities.* The abstraction of a craft, in a nutshell.

For convenience I have omitted links to papers about the role of *abstraction in semantics*, documents referring to research such as *Planning: Complexity and Expressivity and Abstraction in Problem Solving*. Also, I did not mention that there is a *CS455 Manual: Abstraction* nor that there is a *Thematic guide on Flexibility, Abstraction and Accessibility*.

From the examples discussed, it is clear to me that abstraction does not merely conceal or hide irrelevant aspects. Abstraction captures something important, it discloses another layer of reality. We only have to find out what that is, precisely.

6 Complexity

Apparently, Complexity comes in many guises. After scanning the list of references I tend to agree with the quote on the page about *Chaos and Complexity* which characterizes Complexity by saying *"I can't define it, but I know it when I see it."* Or was it Chaos?

Apart from the aforementioned *Journal of Complexity* (which publishes original research papers that contain substantial mathematical results on complexity as broadly conceived) there is a load of pages from *CS180: Algorithms and Complexity.*

An interesting reference is the *Hypertext Bibliography of Measures of Complexity* covering *some of the philosophical and practical references to the concept and measurement of Complexity. It does not attempt to cover all the new sciences that might come under the "Complexity" banner, unless they are relevant to the idea of Complexity per se.*

It is also interesting to ponder about the *Mathematical Complexity of Simple Economics* since, *relative to the physical and biological sciences, one tends to think of economics and other social and...* Well, you may fill this in to your own taste. I wonder in what respect this differs from the *Complexity of Japanese Business Culture.* Naturally, *business culture, or corporate culture, is a difficult and subtle topic to discuss.*

Complexity International is another journal about Complexity. It deals with *the field of complex systems, the generation of complex behavior from the interaction of multiple parallel processes. Relevant topics include (but are not restricted to): artificial life, cellular automata, chaos theory, control theory, evolutionary programming, fractals, genetic algorithms, information systems, neural networks, non-linear dynamics, and parallel computation.*

With the *Complexity of Methods and Classes*, we are again on familiar ground: *The choice of an implementation out of several possible alternatives is usually governed by efficiency considerations. Depending on the structure of the problem at hand, different implementations will have different runtime*

characteristics and space requirements. We refer to these as the (time or space) complexity. Other measures exist. Fortunately so. Although the title is at first sight somewhat misleading this falls clearly within the area covered by *Automata, Formal Languages, and Complexity* [2], about which it is said: *Since God created computer science, the standard textbook in this area has been Hopcroft and Ullman.*

By the way, the Complexity treated in [2] has nothing to do with *Software Complexity* which goes back to 1960s: *In general, the complexity of an object is a function of the relationships among the components of the object.*

Most relevant to our investigation is the title *Abstraction-Induced Complexity.* As the author says: *Abstraction reduces the apparent complexity of an implementation by hiding all but "the most relevant" details. However, no interface is suitable for all the users of the implementation for exactly the same reason: each user has a slightly different view of what is "most relevant." Thus, although abstractions can reduce complexity, working around their limitations can introduce other complexities. Some abstractions are designed to minimize the added complexity.* Well said, but it needs to be digested.

For refreshment look at *Simple Complexity*, which combines *fantasy, satire, philosophy ...* It is a review of The Little Prince by Antoine de Saint-Exupry. Read it.

7 Conclusions

Abstraction without Complexity is like the simplistic euphoria of a drugged mind. Complexity without Abstraction is like the neurotic obsessions of daily life. Ruminating on C++, the authors begin with stating that "Abstraction is selective ignorance" [3]. It is. But it is more!

Abstraction is disclosure. It is *adaptive.* It is a vehicle for expressing *viewpoints*, for taking a *probe*, for changing *perspective*, but foremost Abstraction is a continuous search for *robustness* and *meaning.*

There is a myriad of Complexities, ranging from *Simple Complexity* and *Structural Complexity* to the *Complexity of Self-Organization and Emergent Phenomena.* I quote:

Complexity - What Is it Good For ? Absolutely Everything ... (to the tune of "War")

Abstraction comes in many *forms* too. Abstraction brings light to the

darkness of Complexity. But beware, Complexity is everywhere. It might present itself to you as *Black Abstraction*. Have another beer.

Bibliography

[1] A. Eliëns, *Principles of Object-Oriented Software Development*, Addison-Wesley, 1995.

[2] J.E. Hopcroft and J.D. Ullman, *Introduction to automata theory, languages and computation*, Addison-Wesley, 1979.

[3] A. Koenig and B. Moo, *Ruminations on C++*, Addison-Wesley, 1996.

Kai Engelhardt

New Winde for Old Bags

Willem-Paul de Roever

1 Introduction

Paul Vitanyi and the senior author have known each other as long as Hoare logic exists. We met shortly after Dana Scott formulated his induction rule. The present paper contains a new result on the combination of these two topics, obtained by taking the specification statement $\phi \rightsquigarrow \psi$ as basic statement instead of assignments—here ϕ and ψ stand for recursively enumerable first order assertions. This result owes its significance to [4], in which the general case of data refinement is shown to be expressible using specification statements. Consequently the resulting Hoare logic enables data refinement proofs, and the interesting question arises to which extent this logic can be simplified without sacrificing completeness. We present a rather simple sound and relatively complete logic in which the recursion rule does not require a deduction in its antecedent, in contrast to all but one formulation of Scott's induction rule, namely the one by Jerald Schwarz[6].

[0] Christian-Albrechts-Universität zu Kiel, Preußerstraße 1–9, 24105 Kiel, Germany. ({ke|wpr}@informatik.uni-kiel.de). Supported by ESPRIT BRA project REACT (no. 6021).

2 A Language with Specification Statements

We introduce an assertion language, a programming language, and a language of correctness formulae. Our set-up is rather standard, except for *specification statements* $\pi \rightsquigarrow \rho$ (introduced as *generic commands* $\{\pi \Rightarrow \rho\}$ in [6]). We refer to the literature for details omitted here[2, 3].

2.1 Assertions

Our assertion language $Pred \ni \phi, \psi, \pi, \rho$ is first order predicate logic over Peano arithmetic. The free variables of such an assertion stem from two disjoint sets: a finite set of *program variables* ($Var \ni x$), and a countably infinite set of *logical variables* ($Lvar \ni x_0$). Program variables are meant to be accessed by programs while logical variables occur in assertions only. They are not accessible to programs and are introduced to increase the expressiveness of assertions. Without them, we could not even specify exactly the program **skip** (in case the carrier of the interpretation has at least two elements).

To ease the exposition below, we assume that vector $\vec{z}$ is a totally ordered version of Var and that $\vec{z_0}$ is a logical variable vector of the same length without duplicates.

Assertions are interpreted in an interpretation $\mathcal{I}$ of Peano arithmetic in carrier set $\mathbb{V}$. We use a pairs of a program state $\sigma \in \Sigma = [Var \longrightarrow \mathbb{V}]$ and a logical state $\gamma \in \Gamma = [Lvar \longrightarrow \mathbb{V}]$ to evaluate free variables in assertions. The semantics function $C[[.]]_{\mathcal{I}} : Pred \longrightarrow 2^{\Gamma \times \Sigma}$ maps each assertion to the set of state pairs satisfying it in interpretation $\mathcal{I}$.

2.2 Programs

We consider a simple programming language with specification statements, sequential composition, non-deterministic choice and recursion. Formally we define the syntactic category $Prog \ni S ::= \phi \rightsquigarrow \psi \mid X \mid S_1 ; S_2 \mid S_1 \sqcap S_2 \mid \mu X.S$ of *programs*, where X is a *recursion variable*.

More common language constructs like assignments, guards, conditionals, loops, and guarded commands can be easily expressed using specification

statements.[1]

In the sequel we will consider only well-formed programs, i.e., ones that do not contain free occurrences of recursion variables like X, to avoid difficulties arising from free occurrences of recursion variables. This restriction allows us to define the semantics of programs without giving an explicit meaning to program X.

Given an interpretation $\mathcal{I}$, each well-formed program $S \in \mathit{Prog}$ is mapped to a binary relation $\mathcal{P}[\![S]\!]_{\mathcal{I}} \subseteq 2^{\Sigma \times \Sigma}$ on program states.

$$\mathcal{P}[\![\phi \rightsquigarrow \psi]\!]_{\mathcal{I}} \stackrel{\text{def}}{=} \bigcap_{\gamma \in \Gamma} \{\, (\sigma, \tau) \mid (\gamma, \sigma) \in C[\![\phi]\!]_{\mathcal{I}} \Rightarrow (\gamma, \tau) \in C[\![\psi]\!]_{\mathcal{I}} \,\}$$

As usual, $\mathcal{P}[\![S_1 ; S_2]\!]_{\mathcal{I}} \stackrel{\text{def}}{=} \mathcal{P}[\![S_1]\!]_{\mathcal{I}} ; \mathcal{P}[\![S_2]\!]_{\mathcal{I}}$, $\mathcal{P}[\![S_1 \,[\!]\, S_2]\!]_{\mathcal{I}} \stackrel{\text{def}}{=} \mathcal{P}[\![S_1]\!]_{\mathcal{I}} \cup \mathcal{P}[\![S_2]\!]_{\mathcal{I}}$, and $\mathcal{P}[\![\mu X.S]\!]_{\mathcal{I}} \stackrel{\text{def}}{=} \bigcup_{i \in \mathbb{N}} \mathcal{P}[\![S^i(\Omega)]\!]_{\mathcal{I}}$, where Ω abbreviates the never terminating program **true** $\rightsquigarrow$ **false**. We sloppily use $S(S')$ as shorthand for $S[^{S'}/_X]$ whenever the recursion variable X subject to substitution is clear from the context. Using this convention, we define $S^0(\Omega) \stackrel{\text{def}}{=} \Omega$ and $S^{i+1}(\Omega) \stackrel{\text{def}}{=} S(S^i(\Omega))$ for $i \in \mathbb{N}$. Substitution $S_1[^{S_2}/_X]$ of program S_2 for recursion variable X in program S_1 is defined in the literature (see, e.g., [3, 2]).

For later reference, we note the *fixed point property*: $\mathcal{P}[\![\mu X.S]\!]_{\mathcal{I}} = \mathcal{P}[\![S(\mu X.S)]\!]_{\mathcal{I}}$.

2.3 Correctness Formulae

We use partial correctness Hoare formulae (*Hoare-triples*) as correctness formulae. Informally, validity of a Hoare-triple $\{\pi\}\, S\, \{\rho\}$ means that, whenever S terminates after being started in a state satisfying precondition π, postcondition ρ holds, however, termination is not guaranteed. Formally, $[\![\{\pi\}\, S\, \{\rho\}]\!]_{\mathcal{I}} \stackrel{\text{def}}{=} C[\![\pi]\!]_{\mathcal{I}} ; \mathcal{P}[\![S]\!]_{\mathcal{I}} \subseteq C[\![\rho]\!]_{\mathcal{I}}$, which is equivalent to $\mathcal{P}[\![S]\!]_{\mathcal{I}} \subseteq \mathcal{P}[\![\pi \rightsquigarrow \rho]\!]_{\mathcal{I}}$, i.e., our definition of the meaning of specification

[1] *Assignment* $x := e$ of an arithmetic expression e to a program variable $x \in \mathit{Var}$ is expressed by $(\vec{z} = \vec{z_0})[^e/_x] \rightsquigarrow \vec{z} = \vec{z_0}$ and *guard* $b \rightarrow$ for a Boolean expression b over Var is expressed by $\vec{z} = \vec{z_0} \rightsquigarrow b \wedge \vec{z} = \vec{z_0}$. Using the definition of guards, **skip** is expressed by **true** $\rightarrow$ and the never terminating program Ω is also expressed by **false** $\rightarrow$. The conditional **if** b **then** S_1 **else** S_2 **fi** is modeled as $(b \rightarrow ; S_1) \,[\!]\, (\neg b \rightarrow ; S_2)$ and the while loop **while** b **do** S **od** is expressed by $\mu X.((b \rightarrow ; S ; X) \,[\!]\, \neg b \rightarrow)$

statements ensures that $\pi \rightsquigarrow \rho$ is the maximal program that is partially correct w.r.t. precondition π and postcondition ρ in interpretation $\mathcal{I}$.

2.4 Weakest Liberal Preconditions

Definition 1 (Weakest liberal precondition) *For a class of programs S, a class of assertions $\mathcal{A}$, each equipped with a semantics function like Prog and Pred, define the* weakest liberal precondition, *$wlp : S \times \mathcal{A} \longrightarrow 2^{\Gamma \times \Sigma}$, by*

$$wlp(S, \rho) \stackrel{def}{=} \{ (\gamma, \sigma) \in \Gamma \times \Sigma \mid \forall \tau \in \Sigma ((\sigma, \tau) \in \mathcal{P}[\![S]\!]_{\mathcal{I}} \Rightarrow (\gamma, \tau) \in C[\![\rho]\!]_{\mathcal{I}}) \} .$$

Obviously, $[\![\{\pi\} S \{\rho\}]\!]_{\mathcal{I}} \Leftrightarrow C[\![\pi]\!]_{\mathcal{I}} \subseteq wlp(S, \rho)$. For later reference, we note the *wlp-specification property*: $\mathcal{P}[\![\pi \rightsquigarrow \vec{z} = \vec{z_0}]\!]_{\mathcal{I}} = \mathcal{P}[\![S]\!]_{\mathcal{I}}$ if π expresses $wlp(S, \vec{z} = \vec{z_0})$.

3 A Hoare-style Proof System

Let $\vec{x}$ express the vector of free program variables inside assertions ϕ, ψ, and ρ, let $\vec{x_0}$ express the vector of free logical variables inside ϕ and ψ, and let $\vec{y_0}$ express a vector of fresh logical variables of the same length as $\vec{x}$.

wlp-adaptation axiom $\quad \vdash \left\{ \forall \vec{y_0} \left(\forall \vec{x_0} \left(\phi \Rightarrow \psi[^{\vec{y_0}}/_{\vec{x}}] \right) \Rightarrow \rho[^{\vec{y_0}}/_{\vec{x}}] \right) \right\} \phi \rightsquigarrow \psi \{\rho\}$

$$\text{composition rule} \quad \frac{\{\pi\} S_1 \{\phi\}, \{\phi\} S_2 \{\rho\}}{\{\pi\} S_1 ; S_2 \{\rho\}}$$

$$\text{choice rule} \quad \frac{\{\pi\} S_1 \{\rho\}, \{\pi\} S_2 \{\rho\}}{\{\pi\} S_1 \,[\!]\, S_2 \{\rho\}}$$

$$\text{consequence rule} \quad \frac{\pi \Rightarrow \phi, \{\phi\} S \{\psi\}, \psi \Rightarrow \rho}{\{\pi\} S \{\rho\}}$$

$$\rightsquigarrow\text{-substitution rule} \quad \frac{\{\phi\} S_1 \{\psi\}, \{\pi\} S_2(\phi \rightsquigarrow \psi) \{\rho\}}{\{\pi\} S_2(S_1) \{\rho\}} ,$$

Our Hoare style rule for recursion is based on Scott's induction rule [7].[2]

$$\text{recursion rule} \quad \frac{\{\pi\}\, S(\pi \rightsquigarrow \rho)\, \{\rho\}}{\{\pi\}\, \mu X.S\, \{\rho\}}$$

We let $\mathfrak{H}$ denote the set of instances of the above axiom scheme and five rules. Soundness of $\mathfrak{H}$ is straightforward. Olderog's rule of adaptation[5] can be derived using our axiom and the $\rightsquigarrow$-substitution rule.[3]

Let us turn to the issue of (relative) completeness.

Definition 2 (Expressiveness) *An interpretation $\mathcal{I}$ is* expressive *for a class S of programs and a class $\mathcal{A}$ of assertions iff, for all $S \in S$ and $\rho \in \mathcal{A}$, $wlp(S, \rho)$ can be expressed in $\mathcal{A}$.*

$$\forall S \in S, \rho \in \mathcal{A}\, (\exists \pi \in \mathcal{A}\, (C[[\pi]]_{\mathcal{I}} = wlp(S, \rho)))$$

Let $\mathfrak{I}$ denote the standard interpretation of Peano arithmetic in the natural numbers $\mathbb{N}$.

Theorem 3 (Expressiveness) *$\mathfrak{I}$ is expressive for Prog and Pred.*

Proof by induction on the structure of the program. The weakest liberal precondition of specification statement $\phi \rightsquigarrow \psi$ w.r.t. postcondition ρ is expressed by $\forall \vec{y}_0\, (\forall \vec{x}_0\, (\phi \Rightarrow \psi[^{\vec{y}_0}/_{\vec{x}}]) \Rightarrow \rho[^{\vec{y}_0}/_{\vec{x}}])$, whence its appearance as precondition in the *wlp*-adaptation axiom. The induction steps are standard, in case of recursion involving Gödel encoding. ■

Definition 4 (Relative completeness) *A proof system $\mathcal{H}$ is* complete relative to an interpretation *$\mathcal{I}$ iff*

$$[[\{\pi\}\, S\, \{\rho\}]]_{\mathcal{I}} \Rightarrow \vdash_{\mathcal{H} \cup \mathcal{T}_{\mathcal{I}}} \{\pi\}\, S\, \{\rho\}$$

for all assertions $\pi, \rho \in Pred$ and programs $S \in Prog$, where the theory $\mathcal{T}_{\mathcal{I}} \stackrel{def}{=}$

[2] Observe that we do not need premises of the form $\{.\}\,.\,\{.\} \vdash \{.\}\,.\,\{.\}$.

[3] Similarly, the standard axioms and rules for loops, conditionals, assignments, guarded commands, etc. can be derived from our rules.

$\{ \pi \in \mathit{Pred} \mid C[[\pi]]_{\mathcal{I}} = \Gamma \times \Sigma \}$ *of valid assertions under* $\mathcal{I}$ *is assumed as set of additional axioms.*

Theorem 5 (Relative completeness) *$\mathfrak{H}$ is complete relative to $\mathfrak{I}$.*

Proof Let $\pi, \rho \in \mathit{Pred}$ and $S \in \mathit{Prog}$ such that $[[\{\pi\}\, S\, \{\rho\}]]_{\mathfrak{I}}$. Let π' express $\mathit{wlp}(S, \rho)$, then $\pi \Rightarrow \pi' \in \mathcal{T}_{\mathfrak{I}}$. We show by induction on the structure of S how to prove $\{\pi'\}\, S\, \{\rho\}$. An instance of the consequence rule then finishes the proof. Consider the cases of S:

$S = \pi \rightsquigarrow \rho$: $\{\pi'\}\, S\, \{\rho\}$ is an instance of the *wlp*-adaptation axiom.

$S = S_1 ; S_2$: Let ψ express $\mathit{wlp}(S_2, \rho)$, then $\vdash_{\mathfrak{H} \cup \mathcal{T}_{\mathfrak{I}}} \{\psi\}\, S_2\, \{\rho\}$ and $\vdash_{\mathfrak{H} \cup \mathcal{T}_{\mathfrak{I}}} \{\pi'\}\, S_1\, \{\psi\}$, by induction hypotheses. An instance of the composition rule with these two premises shows $\{\pi'\}\, S\, \{\rho\}$.

$S = S_1 \,[]\, S_2$: $\vdash_{\mathfrak{H} \cup \mathcal{T}_{\mathfrak{I}}} \{\pi'\}\, S_1\, \{\rho\}$ and $\vdash_{\mathfrak{H} \cup \mathcal{T}_{\mathfrak{I}}} \{\pi'\}\, S_2\, \{\rho\}$, by induction hypotheses. An application of the choice rule then proves $\{\pi'\}\, S\, \{\rho\}$.

$S = \mu X.T$: Let $\psi \equiv \vec{z} = \vec{z_0}$ and ϕ express $\mathit{wlp}(\mu X.T, \psi)$. Then, $\phi \rightsquigarrow \psi = \mu X.T$ holds semantically by the *wlp*-specification property. Thus, π' is equivalent to $\forall \vec{y_0}\, (\forall \vec{x_0}\, (\phi \Rightarrow \psi[{}^{\vec{y_0}}/_{\vec{x}}]) \Rightarrow \rho[{}^{\vec{y_0}}/_{\vec{x}}])$, where $\vec{y_0}$, $\vec{x_0}$, and $\vec{x}$ are as in the *wlp*-adaptation axiom. To apply the $\rightsquigarrow$-substitution rule where S_2 is just X, two premises remain to be shown: $\{\pi'\}\, \phi \rightsquigarrow \psi\, \{\rho\}$, which is an instance of the *wlp*-adaptation axiom scheme, and $\{\phi\}\, \mu X.T\, \{\psi\}$. By the induction hypothesis, the latter is provable using the recursion rule since its premise, $\{\phi\}\, T(\phi \rightsquigarrow \psi)\, \{\psi\}$, is valid. This can be seen as follows. By the fixed point property $\mu X.T = T(\mu X.T)$, thus, by the *wlp*-specification property, $\phi \rightsquigarrow \psi = T(\phi \rightsquigarrow \psi)$. Hence $[[\{\phi\}\, T(\phi \rightsquigarrow \psi)\, \{\psi\}]]_{\mathfrak{I}}$.

■

Bibliography

[1] S. A. Cook, *Soundness and completeness of an axiom system for program verification*, SIAM Journal of Computing, 7(1):70–90, Feb. 1978.

[2] P. Cousot, *Handbook of Theoretical Computer Science: Formal Models and Semantics*, volume B, chapter 15, pages 841–993, Elsevier/MIT Press,

1990.

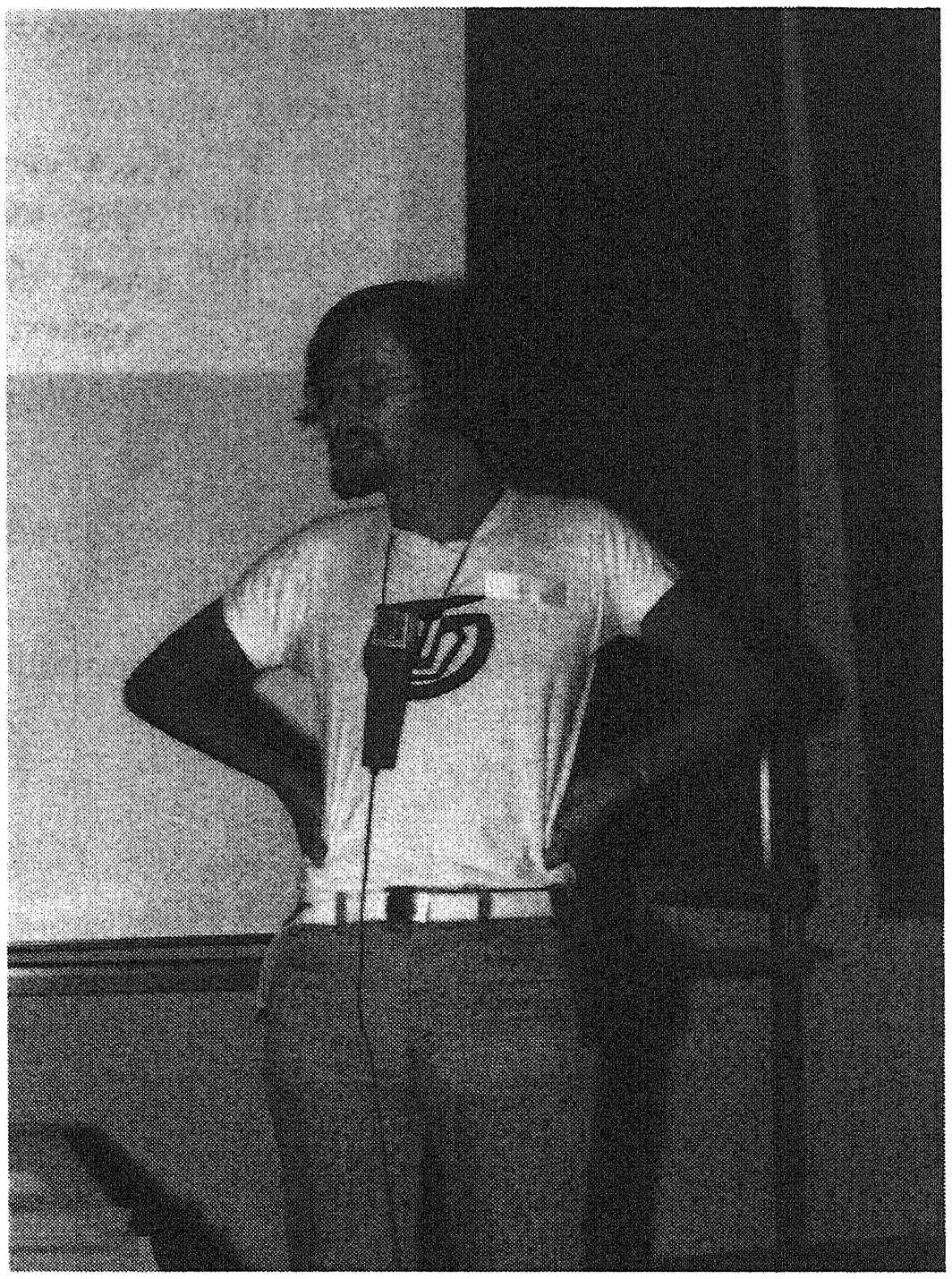
[3] J. de Bakker, *Mathematical Theory of Program Correctness*, Prentice Hall, 1980.

[4] K. Engelhardt and W.-P. de Roever, *Simulation of specification statements in Hoare logic*, in W. Penczek and A. Szałas, editors, *Mathematical Foundations of Computer Science* 1996, 21st International Symposium, MFCS '96, Cracow, Poland, Proceedings, LNCS volume 1113, pages 324–335, Springer-Verlag, Sept. 1996.

[5] E.-R. Olderog, *On the notion of expressiveness and the rule of adaptation*, Theoretical Computer Science, 24:337–347, 1983.

[6] J. Schwarz, *Generic commands—a tool for partial correctness formalisms*, The Computer Journal, 20(2):151–155, 1977.

[7] D. Scott and J. de Bakker, *A theory of programs*, unpublished seminar notes, IBM, Vienna, 1969.



Frederic Gruau

A Snapshot of Paul

Paul likes computers,
He also knows other things,
He talks about complexity,
And prostitution in Amsterdam,
With the same light in the eyes.
He likes to work
He likes to drink
When he can afford it.
He sees far away,
So his footprints are durable.
When you talk to him,
You had better know your classics.
You can ask him for advice,
If you catch him.
You can listen to him for a long time,
You will not get bored.
Paul respects the great scientists,
of the ancient times

of the modern times
like we have a patriarch in Zen.
He speaks about them
With hope in the eyes.
Time only can judge.
Let time be the judge.

Jan Heering

Klassieke Fysica en Berekenbaarheid

Bij de totstandkoming van de Church-Turing these hebben fysische overwegingen nauwelijks een rol gespeeld en het is dan ook niet uitgesloten dat de—bekende—fysica de Church-Turing these aan zijn laars lapt. Zullen computers gebaseerd op bekende maar nog niet in computers benutte fysische processen de Church-Turing barrière even gemakkelijk doorbreken als de Concorde door de geluidsbarrière gaat? En zo ja, om welke processen zou het dan kunnen gaan?

Volgens Pour-El en Richards[1] heeft de klassieke golfvergelijking in twee of meer dimensies voor bepaalde berekenbare beginvoorwaarden een unieke maar niet-berekenbare oplossing. Dat is verrassend, maar wat betekent het? De meningen zijn verdeeld.

In zijn boek *The Emperor's New Mind* meent Penrose dat de oplossingen in kwestie fysisch niet relevant zijn omdat ze niet twee maal continu differentieerbaar zijn. Het zijn dus gegeneraliseerde oplossingen, maar deze

[1] Marian B. Pour-El en J. Ian Richards, *Computability in Analysis and Physics*, Springer-Verlag, 1989. Zie ook mijn bespreking in *Mededelingen van het Wiskundig Genootschap* **34** 8 (november 1991) 331-332.

treden in (theorieën van) fysische verschijnselen vaker op. Schokgolven zijn het bekendste voorbeeld.

Een andere mogelijkheid—geopperd door Kreisel—is dat de beginvoorwaarden, alhoewel berekenbaar, niet experimenteel realiseerbaar zijn zodat het proces nooit op de vereiste wijze gestart kan worden.

En misschien gaat het inderdaad om een fysisch realiseerbaar proces. Dan is de volgende vraag of en hoe het in een computer benut kan worden om de berekeningsmacht principieel te vergroten.

Het resultaat van Pour-El en Richards mag dan wiskundig gezien volstrekt duidelijk zijn, de fysische interpretatie ervan roept een hoop interessante vragen op.

Evangelos Kranakis

Problems on Domino Tilings and 2 by 2 Squares

Abstract We consider domino tilings of rectangular checkerboards. We define the graph of domino tilings and give a simple algorithm for computing the distance between any two tilings. We compute the combinatorial function of the number of domino tilings of an $n \times 2$ rectangle with a prescribed number of non-overlapping 2×2 squares. We also consider random tilings. Several open problems are proposed.

1 Graph of domino tilings

A domino consists of two unit squares touching on an edge and has the shape of a horizontal ▭ 2×1 or vertical ▯ 1×2 rectangle. Such dominoes can be used to tile checkerboards (these are the black-white boards used in chess) of arbitrary polygonal shapes.

Let n denote an integer greater than or equal to 2. We consider the set $\mathcal{T}_n$ of domino tilings of an $n \times 2$ rectangular checkerboard. Every domino tiling

[0] School of Computer Science, Carleton University, Ottawa, Ontario, K1S 5B6, Canada (kranakis@scs.carleton.ca). Research supported in part by an NSERC grant.

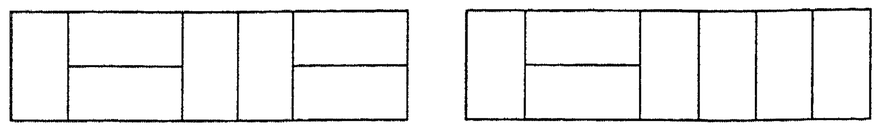


Figure 1 *Two adjacent domino tilings of a* 7×2 *checkerboard. One is obtained from the other by flipping the rightmost* 2×2 *square.*

$T \in \mathcal{T}_n$ must have a 2×2 square of the form ⊟ or ⊞. We define a graph on the set of tilings as follows. The set of vertices is $\mathcal{T}_n$ and two tilings are adjacent if one can be obtained from the other via a flip of the form

⊟ → ⊞ or ⊞ → ⊟.

The resulting graph is connected and bipartite. Connectivity follows from the fact that for any domino tiling there is a path to the tiling with all dominoes vertical. To prove that the graph is bipartite note that there are two kinds of tilings depending on the parity of the number of squares of the form ⊟.

This raises the following interesting question. How do you compute the distance of any two domino tilings in this graph? Let $d(T, T')$ denote the distance between T, T'. There are two kinds of domino tilings: those whose two leftmost squares are occupied by a vertical domino ▯ and those occupied by ⊟. It can now be argued that the following theorem holds.

Theorem 1 *The distance function between any two domino tilings of an* $n \times 2$ *rectangle can be computed from the following recursive identities.*

1. $d(▯\,T, ▯\,T') = d(T, T')$.
2. $d(⊟\,T, ⊟\,T') = d(T, T')$.
3. $d(▯\,T, ⊟\,T') = 1 + d(T, ▯\,T')$.

Although we omit details of the proof we urge the reader to delve into its elegant subtleties. This theorem easily gives a linear algorithm for computing the distance function.

The same analysis will work for $n \times 3$ checkerboards except that there are

more cases to consider. A sophisticated study (using homotopy theory) on the distance function can be found in [6], however no precise algorithmic analysis of the complexity of this problem has ever been carried out for arbitrary checkerboards. Another interesting question concerns finding an algorithm to determine whether a given graph G is the bipartite graph of the set of domino tilings of an $m \times 2$ (or more generally, $m \times n$ rectangle).

2 Number of 2×2 squares

Graham et al [2] show that if $f(n)$ denotes the number of domino tilings of an $n \times 2$ rectangle then $\sum_{n \geq 0} f(n)x^n = \frac{1}{1-x-x^2}$. Moreover, the number of domino tilings of an $n \times 2$ rectangle with exactly s squares of the form ⊟ is shown to be equal to $\binom{n-s}{s}$. In the sequel we compute the generating functions for the number of non-overlapping 2×2 squares and squares of the form ⊞. We can prove the following theorems.

Theorem 2 *Let $f(n,s)$ be the number of domino tilings of an $n \times 2$ rectangle with exactly s non-overlapping 2×2 squares of the form ⊟ or ⊞. Then*

$$\sum_{n,s \geq 0} f(n,s)x^n y^s = \frac{y(2x^2 + 3x^3 + x^4)}{1 - 2x^2y - x^3y}.$$

Moreover for $n \geq 8$ or $s \geq 3$

$$f(n,s) = 2^{3s-n}\left(\binom{s-1}{3s-n-1} + 3\binom{s-1}{3s-n} + 2\binom{s-1}{3s-n+1}\right).$$

Theorem 3 *Let $f(n,s)$ be the number of domino tilings of an $n \times 2$ rectangle with exactly s non-overlapping squares of the form ⊞. Then*

$$\sum_{n,s \geq 0} f(n,s)x^n y^s = \frac{x^2 + x^2y}{1 - x^2 - x^3 - x^2y}.$$

Moreover

$$f(n,s) = \sum_{\substack{n=2(r+1)+m \\ m+s \leq r+1}} \binom{r}{r-m}\binom{r-m+1}{s}.$$

We will not give details of the proofs. However the main idea is as follows. Let t_n^k denote the number of domino tilings of an $m \times 2$ rectangle having exactly k non-overlapping 2×2 squares. We note that the two leftmost squares of a domino tiling may be occupied with either of the following three configurations ⊟, ⊟⊟, ⊞. The remaining tiling has exactly $k-1$ non-overlapping squares. Therefore this gives rise to the following recursive formula $t_n^k = 2t_{n-2}^{k-1} + t_{n-3}^{k-1}$. Using the obvious initial conditions $t_1^1 = 0, t_2^1 = 2, t_3^1 = 3, t_4^1 = 1, t_5^1 = 0$ we can verify easily the generating function in the first theorem. The proof of the second theorem is similar. Let f_n^k denote the number of domino tilings of an $m \times 2$ rectangle having exactly k non-overlapping squares of the form ⊞. The resulting recursive formula is $f_n^k = f_{n-2}^k + f_{n-3}^k + f_{n-2}^{k-1}$. In both theorems it is now easy to derive precise formulas for the coefficients of the infinite series by expanding the generating function into infinite series [7]. We leave the details to the reader.

There is a well-known formula for the number of domino tilings of an $m \times n$ rectangle:

$$2^{mn/2} \prod_{1 \le i \le m, 1 \le j \le n} \left(\left(\cos \frac{i\pi}{m+1} \right)^2 + \left(\cos \frac{j\pi}{n+1} \right)^2 \right)^{1/4},$$

which is derived by analyzing 1-factors in rectangular meshes (see [2][Chapter 7, Exercise 51] and [5][Exercise 4.29]). We may consider the combinatorial function $\mathrm{SQ}_{m \times n}(k)$ = "the number of different domino tilings of an $m \times n$ rectangle which have exactly k pairwise non-overlapping sub-tilings each consisting of a 2×2 square". Our previous study computes $\mathrm{SQ}_{n \times 2}(k)$ exactly. However, in general, no combinatorial formula is known for this function.

3 Random tilings

For the case of domino tilings of $m \times 2$ rectangles we can define a notion of randomness by taking advantage of the following simple observation: a horizontal domino in such a tiling occurs always as part of a 2×2 square of the form ⊟. However, a vertical domino may occur isolated. Therefore we

consider the following correspondence

$$\begin{array}{ccc} \square & \mapsto & 0 \\ \square\square & \mapsto & 1 \end{array}$$

Keeping this correspondence in mind we see that a domino tiling of an $n \times 2$ rectangle corresponds to a sequence $s_1 s_2 \cdots s_m$ where $s_i \in \{0, 1\}$. Call such a tiling Kolmogorov random if the corresponding sequence is. We can derive easily the following theorem from results in [4].

Theorem 4 *A Kolmogorov random tiling generated by a sequence of bits of length m forms an $n \times 2$ rectangle with $n = \frac{3m}{2} + O(\sqrt{m})$. In this tiling there exist exactly*

1. $\frac{n}{3} + O(\sqrt{n})$ *squares of the form* $\square$,
2. $\frac{n}{6} + O(\sqrt{n})$ *squares of the form* $\square\square$ *(possibly overlapping), and*
3. $\frac{n}{9} + O(\sqrt{n} \log n)$ *non-overlapping squares of the form* $\square\square$.

An interesting challenge is to define random tilings for $n \times 3$ checkerboards and more generally for $m \times n$ checkerboards. It would seem that this requires the developments of "Kolmogorov randomness with constraints".

4 Conclusion

It would be interesting to study the distance, enumeration, and randomization questions proposed, for domino tilings of more general checkerboards. Another challenge is to study the same questions when domino tiles are replaced with tiles of the form $1 \times p, p \times 1$, for some $p \geq 3$ and the rectangular checkerboard replaced by polygonal checkerboards with or without holes. It may come as a surprise but it is not straightforward (and sometimes even outright difficult) to generalize these results to more general tiles.

5 Acknowledgements

Many thanks to John Tromp for pointing out misprints.

Bibliography

[1] S. W. Golomb, *Polyominoes*, Princeton Science Library, Princeton University Press, 1994. (Original edition published by Charles Scribner's Sons, 1965.)

[2] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*, Addison-Wesley, 2nd edition, 1994.

[3] B. Grünbaum and G. C. Shephard, *Tilings and Patterns*, W. H. Freeman and Company, New York, 1987.

[4] M. Li and P. Vitanyi, *An Introduction to Kolmogorov Complexity and its Applications*, Springer Verlag, 1993.

[5] L. Lovasz, *Combinatorial Problems and Exercises*, North-Holland, 1979.

[6] N. C. Saldanha, C. Tomei, M. A. Casarin, Jr., and D. Romualdo, *Spaces of Domino Tilings*, Discrete and Computational Geometry, 14:207–233, 1995.

[7] H. S. Wilf, *Generatingfunctionology*, Academic Press, 2nd edition, 1994.

Leslie Lamport

How to Tell a Program from an Automobile

History *I am told that, in the world of* objets d'art, *anything over twenty-five years old is considered an antique. It seems appropriate to celebrate Paul's entry into antiquity with a suitably aged* objet. *More* objet trouvé *than* objet d'art, *this is a litte note I wrote not quite twenty years ago—just a year or two before I met Paul. That was also five years before Paul started to drive. Had I known he was going to embark on that radical change of lifestyle, I would have sent it to him then, perhaps with the title reversed. I hope it can still be of some service.*

This note was written when I was at Massachusetts Computer Associates. It was greeted with outrage at the time, so I did not distribute it further. But, Massachusetts Computer Associates was not as long-lived as Paul, and it passed away a couple of years ago. So the time has come to brush the dust off the manuscript and expose it to the subdued light of a fall day in Amsterdam.

Leslie Lamport
Palo Alto
September, 1996

Computer programs and automobiles are both important in modern life, and people should learn to distinguish one from the other. Let us consider how "the man in the street" might try to find out the difference between them. This is not as easy as it sounds. We might think that he need only be told that automobiles are made mostly of metal parts, whereas programs are made of things like subroutines and assignment statements. Unfortunately, if he has never seen a subroutine or an assignment statement, he might think that they are just some kinds of metal parts. Being told what programs and automobiles are used for might not help either. What something is used for is an expression of intention, and intentions are often hard to discover by looking at what something does. (Seeing an automobile in a traffic jam, it might be hard to guess that it is used to move people from one place to another.)

Our man in the street could try asking an auto mechanic to tell him about automobiles, then asking a programmer to tell him about programs, and comparing the results. The mechanic might reply as follows. "An automobile is constructed by people. After they have finished constructing it, it *runs*. However, it may have some *bugs*. [Most mechanics would use the term "problems", but everyone understands this one when he speaks of 'bugs'.] These are usually minor [such as non-functioning windshield wipers], but occasionally they are serious enough to make the automobile stop running. The bugs are found by *testing*, and are fixed. The automobile then runs properly, and is sold to the user. The user then operates the automobile. However, as he uses it, new bugs may appear. Therefore, the automobile must be *maintained*. Maintenance is also required if the user wants to modify the automobile—for example, to improve its performance [perhaps by 'souping up' the engine]. When the automobile becomes to difficult to maintain, the user has to buy a new one."

Unfortunately, the programmer might say almost exactly the same things about programs! Our poor man in the street would be no further towards his goal. The reader has undoubtedly guessed by now that I am not interested in explaining the difference between a program and an automobile to the man in the street. He was just a pedagogical device, and will now be returned to his street. My purpose is to explain the difference between a program and an automobile to programmers.

An automobile runs, a program does not. (Computers run, but I'm not discussing them.) An automobile requires maintenance, a program does not. A program does not need to have its stack cleaned every 10,000 miles. Its *if* statements do not wear out through use. (Previously undetected *errors* may need to be corrected, or it might be necessary to write a new but similar program, but those are different matters.) An automobile is a piece of machinery, a program is some kind of mathematical expression.

Programmers may be disheartened to learn that their programs are not like automobiles, but are mathematical expressions. Automobiles can be loveable—one can relate to them almost as if they were alive. Mathematical expressions are not loveable—they are hard to relate to. Many programmers may feel like clinging to their belief that programs are like automobiles. However, further thought reveals that mathematical expressions do have certain advantages over automobiles.

Unlike an automobile, a mathematical expression can have a meaning. We can therefore ask whether it has the *correct* meaning. One cannot talk about the correctness of an automobile—it may run properly, but it makes no sense to say that it is correct. Since a program is a mathematical expression, one can (at least in principle) decide if it is correct. The user and the programmer can agree in advance what the meaning of the program should be, and the programmer can prove mathematically that his program has that meaning. An automobile salesman can never prove that he sold a properly running automobile, he can only give enough evidence to satisfy a jury.

I have tried briefly to indicate the difference between an automobile and a program. The programmer should now apply his new knowledge in the field, and see if he can tell the difference by himself. He should first examine his automobile, and ask whether it is running properly, or if it has some bugs and requires maintenance. If he cannot answer this question, he may have to ask an auto mechanic to help him. He should then examine a program, and ask what its meaning should be, and whether he can prove that it has the correct meaning. If he can't answer these questions, it is probably because the program was written by someone who thought he was building an automobile. In that case, I suggest that the programmer ask these questions about the next program he writes—while he is writing it! In this way, any programmer can learn to tell a program from an automobile.

Jan van Leeuwen

On the Effectiveness of Search Engines

Abstract The evaluation of search engines on the Web naturally leads us to the classical notions of recall and precision as known from Information Retrieval. We introduce a new component in the definition of precision, and consider the trade-off between the two measures and a possible explanation.

1 Introduction

The way we search for information has changed dramatically. In the digital world we live in, we no longer consult a library with traditional books and journals: we search the World Wide Web. There are hundreds of electronic journals and many interesting, publicly accessible digital libraries, and the experienced cybersearcher knows the sites where he should look. Publishers are trying to adjust to the new ways scientific information is archived, and to the new requirements for information retrieval which result.

If you are not certain of where to go on the Web for the information you

[0] Department of Computer Science, Utrecht University, Padualaan 14, 3584 CH Utrecht, the Netherlands.

are looking for, the best way is to use a *search engine*. This is a facility that searches the Web for you. The input to a search engine is a 'query', usually a sequence of keywords that pertain to your information-need or some boolean combination of them. The output is a list of annotated 'hyperlinks' that point to Web-pages which presumably contain data relevant to your information-need. Well-known search engines are AltaVista, excite, HotBot, InfoSeek, Lycos, WebCrawler and Yahoo.

Search engines are useful but confront us with all the problems of effective information retrieval: it is difficult to specify a query so it covers the actual information-need precisely, and it is even more difficult to classify documents (web-pages) so an adequate matching process between queries and documents can be carried out. Many of these problems are well-known and are already described in the seminal book by van Rijsbergen[4] on information retrieval.

We see a strong and renewed interest for the problems of information retrieval in the context of the Web. The common problem is that search engines tend to return a mixture of relevant and non-relevant information and that they don't succeed in having all the relevant information 'up front'. On the contrary, even when the output is presented in some order sorted by 'confidence', the degree of pollution of the output by non-relevant material as experienced by the user can be quite disappointing.

Understanding and resolving the problems of information retrieval is not an easy task and many papers and Ph.D. theses have been written about it over the years. For a very recent study, see e.g. the Ph.D. thesis of Huibers[2]. In this note we look at two simple measures to evaluate the effectiveness of search engines and a trade-off between the two that is commonly reported in the literature. Intuitively this trade-off says that any attempt to increase the number of relevant documents in the output will also increase the degree of pollution by non-relevant documents in it. We will attempt to give a partial explanation for this phenomenon, for which there is only empirical evidence. Our main goal is to entertain the reader about the phenomenon.

2 Recall and Precision

Consider a search engine A and the output it generates on a query q. If we look at the retrieved documents, then we will automatically declare some of them to be relevant and other to be not. One might even distinguish

documents by 'degrees of relevance', but for the purpose of this note we will take relevance to be just a binary notion. This means that the output of a search engine can be viewed as a *0-1 sequence* $\omega_A(q)$, with a 1 corresponding to a retrieved document that we regard as relevant and a 0 for a retrieved document that we regard as non-relevant. For example, if we want to know something about Paul and submit the query 'Vitanyi' to Yahoo, then the result is a list of 582 items which, using my notion of relevance at least, translates to the following 582-bit binary sequence:

$$\omega_{\text{Yahoo}}(\text{Vitanyi}) = 10101111110000010000000010000000000100000011...$$

It is a good sign to have many ones at the beginning of the sequence. Note that the notion of 'relevance' is not as clear-cut as it seems. What is relevant and what is not, is ultimately determined by the individual user and his information-need, and thus your interpretation may lead to a sequence that is very different from mine.

Let $Rel_U(q)$ be the set of all documents available on the Web which the user (i.e. we) would deem to be relevant to q, and let $Ret_A(q)$ be the set of documents retrieved by A. It is the disparity between these two sets that we want to assess in some way. We assume that both sets are non-empty, although this is not essential. Note that $|Ret_A(q)| = |\omega_A(q)|$ and that there is a 1 in $\omega_A(q)$ precisely for every document in the set $Ret_A(q) \cap Rel_U(q)$. Let Pr denote the uniform probability distribution over the universe of all documents on the Web.

We now introduce the notions of recall and precision, where we will deviate from the literature in the definition of the latter. The original definitions date back to the nineteen sixties (cf. [4]) when the first systematic investigations of the performance of information retrieval systems began. The simplest measure of the effectiveness of a search engine is its 'recall'. It measures the proportion of relevant documents that are actually retrieved.

Definition 1 *The recall of A on q is*

$$Rec_A(q) = Pr(Ret_A(q)|Rel_U(q)).$$

The second measure we consider is termed 'precision'. It attempts to

measure the proportion of retrieved documents that are actually relevant. It can also be described as the proportion of ones in the sequence $\omega_A(q)$. This interpretation immediately shows that it is not just the proportion of ones that we should be after. We should also express something about the way the ones are distributed in the sequence: are the ones cluttered in the beginning of the sequence, or are they more spread out over the sequence. The idea is that the former should give a higher precision than the latter, as it is clearly desirable to have as many of the relevant documents at the beginning of the sequence as possible. We introduce a quality-factor f to express this and define precision as follows.

Definition 2 *The precision of A on q is*

$$Prec_A(q) = f(\omega_A(q))Pr(Ret_U(q)|Rel_A(q)).$$

The quality-factor only depends on the sequence ω that results from A and q only and not on any other feature of A or q. The traditional definition of precision ([4]) does not include a quality-factor and takes all sequence to be equal in the definition. We assume that $f(\omega)$ is 'high' when all ones of ω occur in the beginning of the sequence and that it is lower the farther the ones are away from it.

In the remainder we will assume that q is fixed but arbitrary and omit it from the notation. We will simplify the notation even further and write U for $Rel_U(q)$, A for $Ret_A(q)$ and f_A for $f(\omega_A(q))$. Note that $Rec_A = \frac{|U \cap A|}{|U|}$ and $Prec_A = f_A \frac{|U \cap A|}{|A|}$.

3 Comparing Search Engines

Consider two search engines A and B and assume that we want B to be some kind of improvement over A. If we look at recall and precision, there is an interesting trade-off phenomenon between the two. The phenomenon is well-known and reported at many occasions. For example, Buckland and Gey[1] write: "we note(d) the empirical finding that precision and recall appear, in practice, to be inversely related: improvement in either tends to be associated with poorer performance of the other." It is this phenomenon that we will

explore, by elementary means. In everything that follows we will assume that B retrieves more documents than A, i.e., that $|B| > |A||$.

It is useful to observe the following formula, easily proved from the definition of the various conditional probabilities involved:

$$|U|(Rec_B - Rec_A) = \frac{|B|}{f_B} Prec_B - \frac{|A|}{f_A} Prec_A.$$

From this formula one can immediately derive the following observation, for any query q.

Proposition 3 *Let B have a recall value less than or equal to the recall value of A. Then the precision of B will be less than or equal to the precision of A, unless the quality-factor of B is larger than the quality-factor of A.*

Proof $Rec_B \leq Rec_A$ implies that $Prec_B \leq \frac{f_B|A|}{f_A|B|} Prec_A$. It follows that $Prec_B < Prec_A$ unless $\frac{f_B|A|}{f_A|B|} > 1$. Using that $|B| \geq |A|$, a necessary condition for the latter to hold is that $f_B > f_A$. ■

This simple argument shows that in case recall decreases, precision will decrease as well unless B actually manages to achieve an increase in 'quality' by a factor of $\frac{|B|}{|A}$. This is unlikely.

It is common to want a new search engine to achieve a better (higher) recall value than the old one. So assume that $Rec_B > Rec_A$, implying that $|U \cap B| - |U \cap A| > 0$. In this case it is useful to consider the *incremental precision* of B over A, introduced in [2] for a more special case only.

Definition 4 *The incremental precision of B over A is* $\Delta_{B,A} = f_B \frac{|U \cap B| - |U \cap A|}{|B| - |A|}$.

The incremental precision measures the extra number of relevant documents obtained by B in proportion to the extra number of documents that B had to retrieve in order to obtain them. It is commonly assumed that, in practice, the incremental precision of B over A will not be higher than the precision of B itself and that it is even unlikely to be higher than the precision of A. The provable statement is this.

Proposition 5 *Let B have a higher recall value than A. Then a necessary condition for the precision of B to be higher than the precision of A is that either*

the incremental precision of B over A is higher than the precision of A or the quality-factor of B is larger than the quality-factor of A.

Proof Note that $Prec_B > Prec_A$ if and only if $(|B|Prec_B - |A|Prec_A) > (|B|Prec_A - |A|Prec_A)$ and thus if and only if $Prec_A < \frac{|B|Prec_B - |A|Prec_A}{|B|-|A|}$. Rewriting the right-hand side we obtain: $\frac{|B|Prec_B - |A|Prec_A}{|B|-|A|} = \frac{f_B|U \cap B| - f_A|U \cap A|}{|B|-|A|} = f_B\frac{|U \cap B|-|U \cap A|}{|B|-|A|} + (f_B - f_A)\frac{|U \cap A|}{|B|-|A|}$. It follows that $Prec_B > Prec_A$ if and only if $Prec_A < \Delta_{B,A} + (f_B - f_A)\frac{|U \cap A|}{|B|-|A|}$. If $f_B \leq f_A$, then a necessary condition for $Prec_B > Prec_A$ to hold is that $Prec_A < \Delta_{B,A}$. This proves the proposition. ■

The proposition has the following interpretation. The only way to obtain a search engine with a higher recall than a given one is to achieve a better incremental precision or a higher quality factor. As the former is hard to achieve in practice, one may conclude that a higher recall only leads to a higher precision if one can realize a higher quality factor. It means that the new search engine should output better quality sequences.

4 Conclusion

The evaluation of search engines on the Web provides a good illustration of the notions of recall and precision as they are classically known from the area of information retrieval. We gave some background to the reported trade-off between the two measures and introduced a new factor into the definition of precision that helps to explain this trade-off better. With the new definition one can argue that a higher recall leads to higher precision only if the quality of the output is increased.

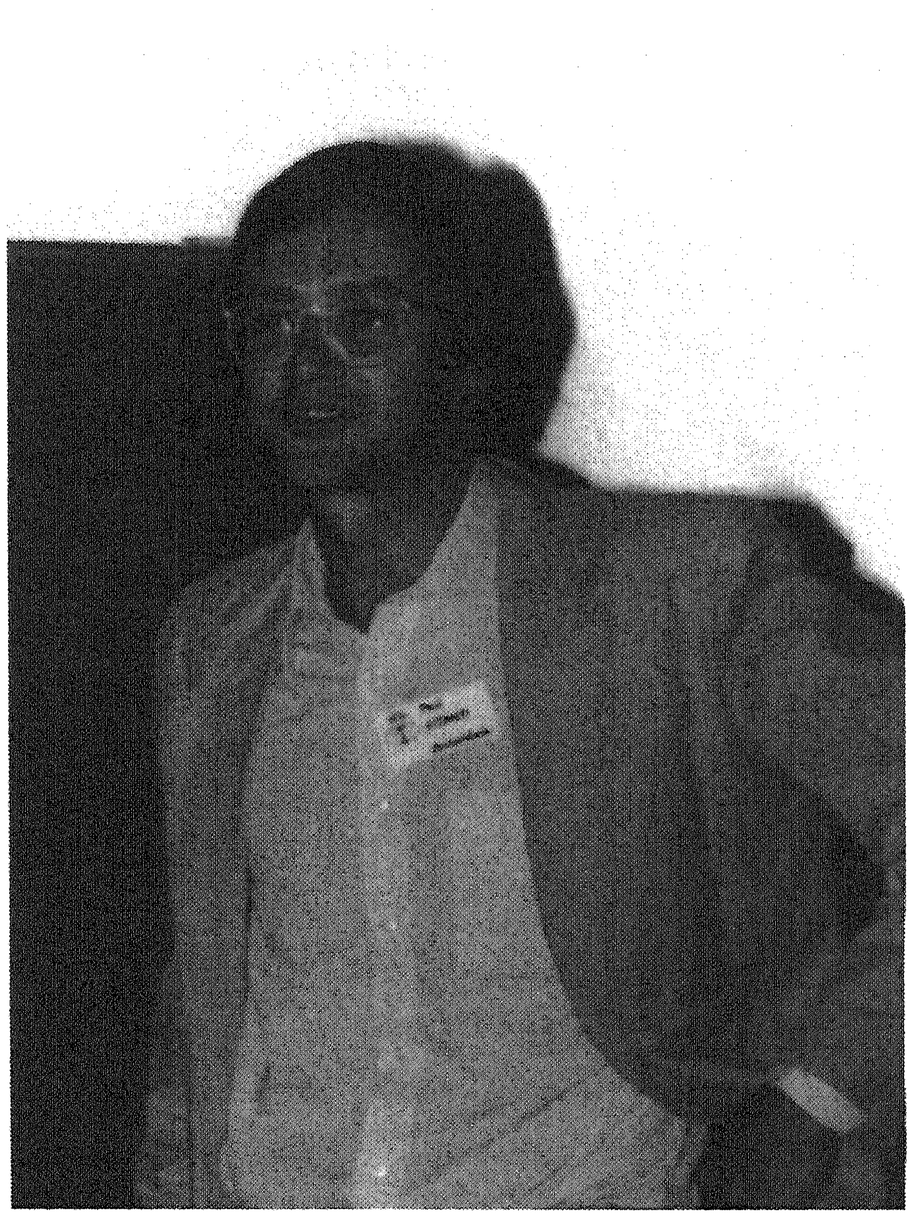
Bibliography

[1] M. Buckland, and F. Gey, *The Relationship between Recall and Precision*, J. Amer. Soc. for Information Sci. 45 (1994) 12-19.

[2] T.W.C. Huibers, *An Axiomatic Theory for Information Retrieval*, Ph.D. thesis, Dept. of Computer Science, Utrecht University, Utrecht, 1996.

[3] M. Li, and P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, Texts and Monographs in Computer Science, Springer-Verlag, New York, 1993.

[4] C.J. van Rijsbergen, *Information Retrieval*, Butterworth & Co., London, 2nd Edition, 1979.



Ming Li

Where Do We Go For Dinner, Paul?

I won't eat imperfect food
Confucius (552-479 B.C.)
The Analects:10

Abstract We won't write an imperfect book. But a book is never perfect, we need breaks for dinners.

1 The Appetizer

1984 STOC, Washington DC., I was helplessly fooling around, counting the big fish whose names I'd learned from textbooks, and trying to figure out what the heck everybody was talking about. A tall gentleman walked towards me. He was in his thirties, quite fit, sun-tanned, and beardless. He had a backpack on one shoulder and looked as if one shoulder was higher than the other. He spoke to me in a deep nasal voice: "Hi, I am Paul Vitányi." I increased my big fish counter by one.

He was destined to be my most prolific co-author, and mentor, in the next

[0] Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong. On leave from University of Waterloo. E-mail: mli@cs.cityu.edu.hk. The dinners (usually Paul's dinners) were partially supported by NSERC via operating grant OGP0046506 and two International Scientific Exchange Awards.

decade.

In the previous year, we had independently solved an open question, by showing that it takes quadratic time for a one-tape Turing machine to simulate a two-tape Turing machine (each with an extra input tape). This question was studied over the years by J. Hartmanis & R. Stearns, M. Rabin, S.O. Aanderaa, R. Reischuk & W. Paul, and P. Duris & Z. Galil. We each mailed our results to Joel Seiferas at Rochester and Joel put us into contact with each other. It turned out that Wolfgang Maass, then at Berkeley, was earlier than both of us. His paper, which also contained a stronger nondeterministic result, appeared in the 1984 STOC. We all used Kolmogorov complexity.

Earlier, as a graduate student at Cornell, I happened to read Paul's STOC'82 paper on simulating counter machines in real time, but I gave it up and turned my attention to a neighbouring paper in the proceedings by Duris and Galil which resulted in all this mess and our collaboration.

2 Knuth Paid!

As far as I can remember, our first "dinner" together was paid by Don Knuth. It was at the 1985 ICALP conference on the beautiful Greek peninsula of Nafplion. Paul and I went on a little tour with two Caltech graduates: Juris Hartmanis and Don Knuth.

As my PhD advisor, Hartmanis has given me lots of excellent advice. Two pieces of advice influenced my research profoundly. One was A.N. Kolmogorov, the other was Paul Vitányi. In 1987, after learning that I would be writing the Kolmogorov complexity book with Paul and knowing my shortcomings, he said affirmatively: "You need him!"

During the ICALP'85 meeting, Knuth gave an invited lecture entitled "Theory and Practice" in the ancient Epidaurus Theater, in the open air, without a microphone. We were supposed to hear clearly because they claimed the theater was so well-designed acoustically that if you dropped a needle in the middle, it could be heard clearly from the farthest seat. Knuth's voice must have been significantly lower than a needle dropping, as I did not hear a single word of his speech (or of Melina Mercouri's speech that followed). Later, Knuth wrote an article *Errors in TEX* modeled after a book of Paul's grand uncle Karoly Szechy, where he also mentioned his discussion with Paul on this trip.

But we did have a good time with him and Juris on our mini-tour and *heard* Don talking about his TEX and his plans for future volumes of his book. I still have the treasured pictures with them from this trip. On the way back, we stopped at a bar, and ordered drinks. I had a coke (that was probably half of my restaurant vocabulary). Knuth offered to pay and I was thrilled.

Okay, that was not exactly a dinner.

3 The Steak House in the Nappa Valley

Yaacov Yesha presented our paper at STOC'86 in Berkeley. I had a rental car. Paul said why don't we have a tour of the Nappa valley. It was a beautiful spring day. We drove through fields of grape vines. Through the trees, we saw a mother deer playing with a jumpy baby deer. We then drove to the valley of the moon where Jack London had lived and worked. Perhaps it was the wine of Nappa valley, perhaps it was the awakening life under the spring sun, perhaps it was the great writer himself, but something told us that we had to write a book on Kolmogorov complexity.

We planned the book [5] all the way. Then it was dinner time, we stopped at a steak house. It served us the best steak, which we would be talking about for years. Today, we still compare any steak we eat with that from the steak house in Nappa county.

A few months later, I moved to Harvard to do a post-doc with Les Valiant, when I was on leave from Ohio State, and Paul visited MIT. We started to prepare the book by collecting papers and "interviewing" various people including Ray Solomonoff, L. Levin, P. Gács. We invited Ray and his wife Grace for dinner. They were nice people. Peter Gács was one of the most generous and serious scientists I ever met. He not only gave us his own manuscript (which can be easily turned into a book on Kolmogorov complexity [3]), but in 1992 he also spent enormous amounts of time reading a preliminary version of our book and made about 40 pages of serious comments.

Paul likes walking. He would walk for an hour from his Boston apartment on the other side of Charles River near Kendall square to my Harvard office. There, we wrote our first survey on Kolmogorov complexity and submitted it to the 1988 Structural Complexity conference, thanks to Les Valiant who allowed me to do any research I liked. Then, Kolmogorov died in 1987 before we were able to go to Moscow. Paul later went to Moscow in 1991, just before

the downfall of the communist system there. He met with Kolmogorov's colleagues (Uspensky, Shen', etc.) and his widow and stayed in Kolmogorov's villa.

4 The Hottest Food on Earth

Paul likes spicy food. The hottest food we had was from Amsterdam. In order to push our book project forward, I visited Paul at CWI just before Christmas 1988. It was a memorable trip for many reasons. We went for dinner, sometimes with Pauline, sometimes without her. This was the first time I met her. Later, she and Paul helped me when I was going through difficult times. Pauline enthusiastically supported our book project from the beginning. Paul and I one day went to an Indonesian restaurant. The food was grouped into small piles around a big plate. Each pile is hotter than the previous pile by a factor of two. There were about 15 piles. You eat the piles in increasing hotness order.

I stopped at the fifth pile, and not before I emptied five big cups of water.

Paul stopped after the last pile.

Our book project did not stop. It was moving on, slowly.

5 Chi Chi, Lai Lai, Chi Chi

I alternate among foods and stay at different places
Confucius, The Analects:10

As our book project goes along, Paul comes and goes. At York it was easy to find places in various colleges for him to stay. But at Waterloo, I always had a big headache finding a reasonable place for him. When he was lucky, he stayed in the Waterloo Inn or the Married Student Apartments, or even the university's Presidential Apartment (and sometimes shared it with a female professor). When he was less lucky, he had to stay with Mr. Fairheads, or in Father Mano's apartment (where no female is allowed), or the female student dorms (where no male is allowed). Once (in 1996, when we were editing the second edition) he returned from my office to his room at midnight after work, as we always do. He was surprised at the building entrance. His whole floor had been taken over by hundreds of cheering girl scouts. After negotiation

with the police, they escorted him to his room and he was forbidden to come out during the night.

As many parts of our book had very high Kolmogorov complexity, or "depth", we had to constantly recharge ourselves with energy. We rotated among our favorite restaurants.

If the restaurant in Nappa valley had the best steak, then the New Market Place had the best beer (New Market Beer) and the best pizza; Lai Lai had the best Kung Pao Chicken; Chi Chi's had the best Margareta and tortilla chips; Koh-I-Nor had the best Tandori chicken; Taka had the best sushi (because that was the only Japanese restaurant in town); Olympia had the best gyros (but their Greek salad was terrible, John Tromp and I go there often eating while playing Go); Marina's had the best Tabuli salad and healthy food (Derick Wood recommended this place); Small Room (North York) had reasonable steak when we couldn't go to the Nappa valley (Amos Israeli liked Small Room).

We frequented these restaurants so often that when I go to dinner without Paul some of the restaurant owners would start to ask me: "Where is your Dutch friend?" Some missed him so much that they decided to shut down their restaurants. This happened to both Chi Chi's on Steele Street, North York, and on Fairway Rd, Kitchener. Well, maybe other factors also played some role, but they definitely missed him. Definitely!

Too bad that the Chi Chi's have gone. The name of the Mexican food chain sounds so nice in Chinese, meaning "eat eat", literally. ("Lai Lai" means "come come".)

6 Real Chinese Food

Finally, Paul had a chance to taste real Chinese food in China. This was in August 1994, at the ISAAC conference in the Friendship Hotel in Beijing.

He arrived a few days early for the meeting. I was there already, we were running an AI summer school in the Chinese Academy of Sciences. We stayed in the Welcome Hotel of the Institute of Computing Technology. The following day, Paul would give a seminar and then we would depart for Xi'an, so I just put him in the room of Yun Peng (Maryland) who went home that day. This was illegal. In China, a "Friendship" hotel is always meant for foreigners, but a "Welcome" hotel only welcomes Chinese. The girls at the Welcome hotel

desk treated Paul leniently. But after dinner, when we returned, Paul was seen by the fierce-looking manager Liu. The small man immediately shouted at the girls at the front desk: "What is that big-nose doing here?!" At 12 midnight, somebody knocked on my door and I was told that "The big-nose cannot stay here! You must move him out immediately." I made many phone calls, but all the hotels nearby were full. Finally, the girls agreed with me that we would leave before 7am the following morning before the manager showed up again. I went to wake up Paul at 6:30am, and we left before 7am. That was ok, he had six-hour jet-lag anyways.

He gave a talk on physical limitations of parallel computing, and then we were taken to the Ke Yuan Restaurant (a branch of Fong Shan) for a real Chinese feast. The restaurant specializes in Chinese royal food. They use the special menu of the Empress Ci Xi of the Qing dynasty. It was a magnificent feast with exotic foods like camel's feet. But Paul wasn't too impressed. He kept asking me if they had Kung Pao Chicken. I didn't think Ci Xi would eat lowly food like Kung Pao Chicken.

We headed for China's west by train the same day. His ticket was two times as expensive as mine because he had a bigger nose. (But a bigger nose was not always a disadvantage, a Karaoke club in Xi'an let him in for free!) In Xi'an, we ate the well-known local food "lamb-stew soaking bread" and "dumpling feast." Again none of these was comparable to Kung Pao Chicken for Paul. We of course also went to see the marvelous terra-cotta soldiers. Paul suspected that the whole site was faked by the Chinese government. But he bought a lot of miniature terra-cotta soldiers anyways for his mother, sister, and Pauline.

One day he returned from the street excitedly, with much shorter hair, unevenly cut. He had had a hair cut on the street, served by four girls. They charged him only two dollars. I mean two Chinese dollars, two Yuan, about a US quarter. That was the honest street price. Today, I still wonder how he bargained for this price.

On the way back to Beijing, he got sick. He blamed the pig ear he ate in my uncle's home. But I thought it was the grapes he ate without taking the skin off. Back to the Friendship Hotel, at the ISAAC conference, I brought yoghurt for him from street each day, and that was the only Chinese (or non-Chinese) food he ate for the next five days; he even stopped asking for Kung Pao Chicken.

7 A Missed Dinner

We sometimes miss dinners also. It was either 1991 or 1992, the Computer Science Department at the University of Toronto invited Paul to give a seminar on a Tuesday. But we thought it was on Wednesday (probably because Waterloo seminars were on Wednesdays). On Tuesday, when Wendy told us that the University of Toronto called, it was already too late. A full room of 50 people were waiting (and left), and we were still in Waterloo. The worst part is that we also didn't get the free dinner that goes with the seminar. (Two years later, Paul did make up the talk at U of T and got the dinner back.)

Seminars usually go better than this. We have given many seminars on Kolmogorov complexity and its applications all over the world. Paul's most recent seminar was a four-hour tutorial on Kolmogorov Complexity at the University of New South Wales, Sept. 27, 1996. Simultaneously, within 100 meters, the Dalai Lama talked. Still Paul drew an audience that filled the room. I wonder whether the Dalai Lama had more.

8 Open Questions

Although many research questions remain unanswered, "Where do we go for dinner" will always be the most frequently asked question. As far as I can figure it out, this question was first raised by Confucius [2] 2500 years ago. Other than his advice that you have to alternate among your favorite restaurants, he also said wisely: "Eat not too much, but drink no limit" [2]. I think this suits Paul extremely well.

First time readers of this article may decide to skip the following more technical paragraph which has nothing to do with food. Other questions we recently have been wondering about include using Kolmogorov complexity to analyze the average-case complexity of algorithms. For example, the current approach for approximation algorithms for NP hard problems might be misleading. The algorithm that achieves the best worst-case performance ratio might not be the one which works the best on average. We showed [4] that if the shortest common supersequence (SCS) can be approximated within a constant factor, then NP=P. However, on average a trivial algorithm approximates SCS with ratio $1 + O(\frac{1}{\sqrt{n}})$. This analysis uses Kolmogorov complexity [5]. Similarly, consider the Max Cut problem. It is approximable with ratio 1.139

but cannot be approximated better than 1.012 [1]. But on average, a trivial algorithm that just divides the graph arbitrarily into two halves does better than $1+\epsilon$ for any $\epsilon > 0$. Consider a Kolmogorov random graph. If you just partition it into two halves, the number of crossing edges is $n^2/4 \pm O(n \log n)$. This holds for any half-half partition. Partitioning the graph unevenly gives a smaller cut. Carefully averaging over all graphs, we obtain an approximation scheme on the average.

Due to space limitations, all "proofs" have been omitted. But the stories and quotations are all true.

9 Acknowledgements

I would like to thank John Tromp for editing this book. John's favorite food was the eggplant dish I cooked at home. But everything would taste well if you play Go with him (and do not smoke) while eating. Thanks also to Mary and Derick Wood for many corrections. Mary cooks the best Chinese food on earth.

Bibliography

[1] P. Crescenzi and V. Kann, *A compendium of NP optimization problems*, Oct. 1995.

[2] Confucius, *The analects*, Village Gang 10, 552-479 B.C.

[3] P. Gács, *Lecture notes on descriptional complexity and randomness*, manuscript, 1987/1988.

[4] T. Jiang and M. Li, *On the approximation of shortest common supersequences and longest common subsequences*, SIAM J. Comput., 24:5(1995), 1122–1139.

[5] M. Li and P. Vitányi, *An Introduction to Kolmogorov complexity and its applications*, Springer-Verlag, 1993. 2nd Edition to appear in 1997.

Lambert Meertens

A new lower bound on the period of 3x+1 cycles

Abstract. A lower bound of 17 million is established for the period of non-trivial cycles in the "$3x + 1$" problem, improving the best previous lower bound of 450,000.

1 Introduction

Take some positive integer n. If n is odd, replace it by $(3n + 1)/2$. If n is even, replace it by $n/2$. (In either case, the replacement is again a positive integer.) Repeat this procedure.

For example, starting with $n = 7$, we obtain:

$7 \mapsto 11 \mapsto 17 \mapsto 26 \mapsto 13 \mapsto 20 \mapsto 10 \mapsto 5 \mapsto 8 \mapsto 4 \mapsto 2 \mapsto 1 \mapsto 2 \mapsto 1 \ldots$

Once such a sequence reaches the number 1, it gets into the cycle $1 \mapsto 2 \mapsto 1 \ldots$, which is called the *trivial cycle*. The well-known "$3x + 1$" conjecture

[0] Department of Algorithmics and Architecture, CWI, Amsterdam, and Department of Computing Science, Utrecht University, The Netherlands (lambert@cwi.nl).

is that for all positive integers as starting value the sequence eventually gets into the trivial cycle[1]. No evidence is available that falsifies the conjecture, but attempts to prove it have thus far been fruitless. A comprehensive survey of what is known about the problem is given by Lagarias [1].

If the conjecture is false, there are two (not necessarily mutually exclusive) possibilities:

- for some starting value the sequence is unbounded;
- for some starting value the sequence gets into a non-trivial cycle.

Here we are concerned with the latter possibility, and show that in that case the cycle has a rather large period, namely at least 17,087,915.

Starting values that lead to 1 are called *stopping*. Nobuo Yoneda showed that there are no non-stopping values for $n < M = 2^{40}$. (For this and other results mentioned, see Lagarias.) This Yoneda bound will play an important role.

2 Preliminaries

The notation used is in general the same as that of Lagarias. We define

$$\begin{array}{lcll} up(n) & = & (3n+1)/2 & \text{for } n \text{ odd} \\ dn(n) & = & n/2 & \text{for } n \text{ even} \end{array}$$

$$T(n) = \begin{cases} up(n) & \text{if } n \text{ is odd} \\ dn(n) & \text{if } n \text{ is even} \end{cases}$$

The procedure that generates the sequence consists of the repeated replacement $n \mapsto T(n)$. If such a replacement step takes the form $n \mapsto up(n)$ we call it an *up-step*, while a replacement $n \mapsto dn(n)$ is called a *down-step*.

[1] In the usual statement of this problem there is a clause "Stop when 1 is reached", and the conjecture is then that all starting values lead to a stop. This is clearly equivalent to the statement above.

Define, for a sequence $a = [a_1, a_2, \ldots, a_q]$ of natural numbers,

$$S(a) = \sum_{i=1}^{q} 2^{a_i} 3^{q-i}$$

Let, for a as above, $a \triangleleft p$ stand for the sequence a extended to the right with an element $a_{q+1} = p$, so that $(a \triangleleft p)_i = a_i$ for $i \leq q$, and $(a \triangleleft p)_{q+1} = p$.

Proposition 1 $S(a \triangleleft p) = 3S(a) + 2^p$.

Proof

$$\begin{aligned}
& S(a \triangleleft p) \\
= & \quad \text{(definition of } S \text{)} \\
& \sum_{i=1}^{q+1} 2^{(a \triangleleft p)_i} 3^{q+1-i} \\
= & \quad \text{(range split)} \\
& \sum_{i=1}^{q} 2^{(a \triangleleft p)_i} 3^{q+1-i} + 2^{(a \triangleleft p)_{q+1}} \\
= & \quad \text{(definition of } a \triangleleft p \text{)} \\
& \sum_{i=1}^{q} 2^{a_i} 3^{q+1-i} + 2^p \\
= & \quad \text{(factor out 3)} \\
& 3 \sum_{i=1}^{q} 2^{a_i} 3^{q-i} + 2^p \\
= & \quad \text{(definition of } S \text{)} \\
& 3S(a) + 2^p
\end{aligned}$$

■

Proposition 2 *For all integers n and natural numbers p, $T^p(n)$ can be expressed in the form*

$$\frac{3^q n + S(a)}{2^p}$$

in which q is some natural number and a is some sequence of q natural numbers $[a_1, a_2, ..., a_q]$ with the property that

$$a_1 < a_2 < ... < a_q < p$$

Proof By induction on p. For the base case $p = 0$, $T^p(n) = n$, which can be obtained by taking $q = 0$. The sequence a is then empty, so that $S(a) = 0$.

For the induction step from p to $p + 1$ the inductive hypothesis is $T^p(n) = \frac{3^q n + S(a)}{2^p}$ for some a of length q and whose elements satisfy the required inequalities. We distinguish two cases, depending on whether $T^{p+1}(n)$ was reached by an up-step or a down-step.

Case $T^{p+1}(n) = up(T^p(n))$. We calculate:

$$\begin{aligned}
& up(T^p(n)) \\
= & \quad \text{(definition of } up \text{)} \\
& \frac{3T^p(n) + 1}{2} \\
= & \quad \text{(inductive hypothesis)} \\
& \frac{3\left(\frac{3^q n + S(a)}{2^p}\right) + 1}{2} \\
= & \quad \text{(simplify)} \\
& \frac{3^{q+1} n + 3\,S(a) + 2^p}{2^{p+1}} \\
= & \quad \text{(Proposition 1)} \\
& \frac{3^{q+1} n + S(a \triangleleft p)}{2^{p+1}}
\end{aligned}$$

which is again of the desired form. Note that the required inequalities on the elements of $a \triangleleft p$ are satisfied.

Case $T^{p+1}(n) = dn(T^p(n))$. This case is simpler:

$$\begin{aligned}
& dn(T^p(n)) \\
= & \quad \text{(definition of } dn \text{)} \\
& \frac{T^p(n)}{2}
\end{aligned}$$

$$= \qquad \text{(inductive hypothesis)}$$
$$\frac{3^q n + S(a)}{2^{p+1}}$$

■

From the proof it is immediate that q equals the number of up-steps in the sequence $n \mapsto T(n) \mapsto \cdots \mapsto T^p(n)$. Note further that $S(a)$ under the requirements on a is odd only if p (and therefore q) is positive and $a_1 = 0$.

3 Cycles

In this section n stands for an integer that occurs in a cycle, that is, for some $p > 0$ (the period of the cycle) $n = T^p(n)$.

Proposition 3 *If n occurs in a cycle of period p, then*

$$n = \frac{S(a)}{2^p - 3^q}$$

for some sequence a of length q satisfying the inequalities of Proposition 2.

Proof

$$n = T^p(n)$$
$$\equiv \qquad \text{(Proposition 2)}$$
$$n = \frac{3^q n + S(a)}{2^p}$$
$$\equiv \qquad \text{(solve for } n \text{)}$$
$$n = \frac{S(a)}{2^p - 3^q}$$

■

A fact that we shall not need is that, conversely, whenever $2^p - 3^q$ divides $S(a)$, the quotient is in a cycle of period p.

All numbers in a given cycle share the period and the number of down-steps in a full cycle. So if n in Proposition 3 is replaced by $T(n)$, the same formula

should apply with a replaced by some sequence $L(a)$ but keeping the same values for p and q.

We synthesize a definition for $L(a)$. We make a case distinction depending on whether $n \mapsto T(n)$ is an up-step or a down-step, which corresponds to, respectively, $a_1 = 0$ and $a_1 > 0$.

Case $T(n) = up(n)$.

$$\frac{S(L(a))}{2^p - 3^q} = up(n)$$

$\equiv$ (definition of up)

$$\frac{S(L(a))}{2^p - 3^q} = \frac{3n+1}{2}$$

$\equiv$ (use assumption on n)

$$\frac{S(L(a))}{2^p - 3^q} = \frac{3\frac{S(a)}{2^p - 3^q} + 1}{2}$$

$\equiv$ (simplify)

$$S(L(a)) = \frac{3S(a) + 2^p - 3^q}{2}$$

$\equiv$ (put $a = [a_1, a_2, \ldots, a_q]$; definition of S)

$$S(L(a)) = \frac{3\sum_{i=1}^{q} 2^{a_i} 3^{q-i} + 2^p - 3^q}{2}$$

$\equiv$ (bring in factor 3)

$$S(L(a)) = \frac{\sum_{i=1}^{q} 2^{a_i} 3^{q+1-i} + 2^p - 3^q}{2}$$

$\equiv$ (use $a_1 = 0$)

$$S(L(a)) = \frac{\sum_{i=2}^{q} 2^{a_i} 3^{q+1-i} + 2^p}{2}$$

$\equiv$ (factor out 2 from numerator and cancel)

$$S(L(a)) = \sum_{i=2}^{q} 2^{a_i - 1} 3^{q+1-i} + 2^{p-1}$$

$\equiv$ (shift index)

$$S(L(a)) = \sum_{i=1}^{q-1} 2^{a_{i+1} - 1} 3^{q-i} + 2^{p-1}$$

$\equiv$ (definition of S)

$$S(L(a)) = S([a_2 - 1, a_3 - 1, \ldots, a_q - 1, p - 1])$$

Case $T(n) = dn(n)$.

$$\frac{S(L(a))}{2^p - 3^q} = dn(n)$$

$\equiv$ (definition of dn)

$$\frac{S(L(a))}{2^p - 3^q} = \frac{n}{2}$$

$\equiv$ (use assumption on n)

$$\frac{S(L(a))}{2^p - 3^q} = \frac{S(a)}{2(2^p - 3^q)}$$

$\equiv$ (simplify)

$$S(L(a)) = \frac{S(a)}{2}$$

$\equiv$ (put $a = [a_1, a_2, \ldots, a_q]$; definition of S)

$$S(L(a)) = \frac{\sum_{i=1}^{q} 2^{a_i} 3^{q-i}}{2}$$

$\equiv$ (factor out 2 from numerator and cancel)

$$S(L(a)) = \sum_{i=1}^{q} 2^{a_i - 1} 3^{q-i}$$

$\equiv$ (definition of S)

$$S(L(a)) = S([a_1 - 1, a_2 - 1, \ldots, a_q - 1])$$

Since $L(a)$ turns out to depend on p, we add a subscript p to L. From the calculations we see that the equation is solved by:

$$L_p([a_1, a_2, \ldots, a_q]) = \begin{cases} [a_2 - 1, a_3 - 1, \ldots, a_q - 1, p - 1] & \text{if } a_1 = 0 \\ [a_1 - 1, a_2 - 1, \ldots, a_q - 1] & \text{if } a_1 > 0 \end{cases}$$

It is easy to see that if $a_1 = 0$, the values $k \geq 0$ for which the first element of $L_p^k(a)$ is 0 are given by $k = a_1$, $k = a_2$, and so on. In particular, we have for $k = a_i$:

$$L_p^k(a) = [a_i - a_i, a_{i+1} - a_i, \ldots, a_q - a_i, p + a_1 - a_i, p + a_2 - a_i, \ldots]$$

where we take, of course, only the first q elements. For a uniform treatment we extend the a_i virtually beyond the index q by putting $a_{mq+i} = mp + a_i$,

$0 < i \leq q$, so that we may write:

$$L_p^k(a) = [a_i - a_i, a_{i+1} - a_i, \ldots, a_{i+q-1} - a_i]$$

Now assume that $n > 0$ occurs in a non-trivial cycle, and that it is the least number among the p numbers n, $T(n)$, $T^2(n)$, ... $T^{p-1}(n)$ that make up the cycle. So the step $n \mapsto T(n)$ is an up-step, and n is odd. From Proposition 3 together with $n > 0$ and $S(a) > 0$ we see that in the equation

$$T^k(n) = \frac{S(L_p^k(a))}{2^p - 3^q}$$

the denominator $2^p - 3^q$ is positive, and that $T^k(n)$ varies monotonically with $S(L_p^k(a))$. So the latter expression also assumes its least value for $k = 0$.

The fact that n is odd implies that $a_1 = 0$.

Proposition 4 *If, for n in a non-trivial cycle with period p,*

$$n = \frac{S(a)}{2^p - 3^q}$$

where

$$a = [a_1, a_2, \ldots, a_q]$$

then, for some k:

$$T^k(n) \leq 1/(2^{p/q} - 3)$$

Proof Take $k = a_j$ where j is chosen such that $a_j - (j-1)\frac{p}{q}$ is maximized, and put $b = L_p^k(a)$. Then

$$\begin{aligned}
& b_i \\
= & \quad \text{(definition of } b \text{ and fact about } L_p^k \text{)} \\
& a_{j+i-1} - a_j \\
= & \quad \text{(elementary algebra)} \\
& \left(a_{j+i-1} - (j+i-2)\frac{p}{q}\right) - \left(a_j - (j-1)\frac{p}{q}\right) + (i-1)\frac{p}{q} \\
\leq & \quad \text{(} j \text{ maximizes } a_j - (j-1)\tfrac{p}{q} \text{)}
\end{aligned}$$

$$(i-1)\frac{p}{q}$$

So

$$\begin{aligned}
& S(b) \\
= & \quad \text{(definition of } S \text{)} \\
& \sum_{i=1}^{q} 2^{b_i} 3^{q-i} \\
\leq & \quad \text{(upper bound on } b_i \text{)} \\
& \sum_{i=1}^{q} 2^{(i-1)\frac{p}{q}} 3^{q-i} \\
\leq & \quad \text{(geometric series)} \\
& \frac{2^p - 3^q}{2^{p/q} - 3}
\end{aligned}$$

Now we compute

$$\begin{aligned}
& T^k(n) \\
= & \quad \text{(relationship between } T^k \text{ and } L^k \text{)} \\
& \frac{S(L_p^k(a))}{2^p - 3^q} \\
= & \quad \text{(definition of } b \text{)} \\
& \frac{S(b)}{2^p - 3^q} \\
\leq & \quad \text{(upper bound on } S(b);\ 2^p - 3^q > 0 \text{)} \\
& 1/(2^{p/q} - 3)
\end{aligned}$$

■

4 Moving in for the kill

Proposition 5 *If n is the least number in a non-trivial cycle with period p, with $n = \frac{S(a)}{2^p - 3^q}$, then*

$$n \leq 1/(2^{p/q} - 3)$$

Proof Since n is the least number, $n \leq T^k(n)$ for all k, which together with Proposition 4 establishes the result. ■

Proposition 6 *The length of a non-trivial* $3x + 1$*-cycle is at least 17,000,000.*

Proof Let n be the least number in a non-trivial cycle. Then n is definitely non-stopping, so $n \geq M$, the Yoneda bound. Define ϵ by $\epsilon = p/q - \log_2 3$. Then

$$\begin{aligned} & M \\ \leq & \quad (\ n \text{ is non-stopping}\ ) \\ & n \\ \leq & \quad (\ \text{Proposition 5}\ ) \\ & 1/(2^{p/q} - 3) \\ = & \quad (\ \text{definition of } \epsilon\ ) \\ & \frac{1}{3(2^{\epsilon} - 1)} \end{aligned}$$

We continue as follows:

$$\begin{aligned} & M \leq \frac{1}{3(2^{\epsilon} - 1)} \\ \equiv & \quad (\ \text{switch-over}\ ) \\ & 2^{\epsilon} \leq 1 + \frac{1}{3M} \\ \equiv & \quad (\ \text{logarithms}\ ) \\ & \epsilon \leq \log_2(1 + \frac{1}{3M}) \end{aligned}$$

With the Yoneda bound $M = 2^{40}$ we obtain $\epsilon < 0.44 \times 10^{-12}$, so p/q is close to $\log_2 3$. The first convergent of the continued-fraction expansion of $\log_2 3$ (see the table below) differing less than ϵ from the limit is number 15, 16785921/10590737; but $2^{p_{15}} - 3^{q_{15}} < 0$. The next possible candidate is 17087915/10781274, giving us

$$p \geq 17,087,915$$

■

Remark The best previous lower bound of 450,000 is due to Crandall. Crandall defined the period as what is called q here, but his argument can

be equally applied to p. The definition of period here is the same as that of Lagarias, except that he uses Crandall's definition locally. In the continued fraction table in Lagarias there is a misprint: in the 14-th convergent "190737" should read "190537".

i	w_i	p_i / q_i		$p_i/q_i - \log_2 3$
1	1	1 / 1	$= 1$	-0.58496
2	1	2 / 1	$= 2$	$+0.41504$
3	1	3 / 2	$= 1.5$	$-0.84963\,\mathrm{E}^{-1}$
4	2	8 / 5	$= 1.6$	$+0.15037\,\mathrm{E}^{-1}$
5	2	19 / 12	$= 1.58333\ 33333\ 33333^{+}$	$-0.16292\mathrm{E}^{-2}$
6	3	65 / 41	$= 1.58536\ 58536\ 58536^{+}$	$+0.40335\mathrm{E}^{-3}$
7	1	84 / 53	$= 1.58490\ 56603\ 77358^{+}$	$-0.56840\mathrm{E}^{-4}$
8	5	485 / 306	$= 1.58496\ 73202\ 61437^{+}$	$+0.48195\mathrm{E}^{-5}$
9	2	1054 / 665	$= 1.58496\ 24060\ 15037^{+}$	$-0.94706\mathrm{E}^{-7}$
10	23	24727 / 15601	$= 1.58496\ 25024\ 03692^{+}$	$+0.16825\mathrm{E}^{-8}$
11	2	50508 / 31867	$= 1.58496\ 25003\ 92255^{+}$	$-0.32890\mathrm{E}^{-9}$
12	2	125743 / 79335	$= 1.58496\ 25007\ 87798^{+}$	$+0.66642\mathrm{E}^{-10}$
13	1	176251 / 111202	$= 1.58496\ 25006\ 74448^{+}$	$-0.46708\mathrm{E}^{-10}$
14	1	301994 / 190537	$= 1.58496\ 25007\ 21644^{+}$	$+0.48843\mathrm{E}^{-12}$
15	55	16785921 / 10590737	$= 1.58496\ 25007\ 21149^{+}$	$-0.71245\mathrm{E}^{-14}$
16	1	17087915 / 10781274	$= 1.58496\ 25007\ 21157^{+}$	$+0.16335\mathrm{E}^{-14}$
17	4	85137581 / 53715833	$= 1.58496\ 25007\ 21156^{+}$	$-0.93266\mathrm{E}^{-16}$
∞		$\log_2 3$	$= 1.58496\ 25007\ 21156^{+}$	0

Bibliography

[1] Jeffrey C. Lagarias, *The 3x+1 Problem and its Generalizations*, American Mathematical Monthly, vol. 92 no. 1 (January 1985), pp. 3–23.

Steven Pemberton

A Contribution to the Computational Theory of Big Game Hunting

1 Introduction

In [1], several methods are propounded for the hard problem of catching a lion in the Sahara Desert.

Here is an example:

Example 1 (The geometrical inversion method) *We place a spherical cage in the desert, enter it and lock it from inside. We then perform an inversion with respect to the cage. Then the lion is inside the cage, and we are outside.*

or

Example 2 (The Cauchy method) *We examine a lion-valued function $f(z)$. Let ζ be the cage. Consider the integral*

$$\frac{1}{2\pi i}\int_C \frac{f(z)}{z-\zeta}\,dz$$

where C represents the boundary of the desert. Its value is $f(\zeta)$, i.e. there is a lion in the cage.

However, due to an unfortunate case of bad timing, several important methods were overlooked. We include here, for historical completeness, one of them.

2 The Dijkstra Method

The way the problem reached me was: catch a wild lion in the Sahara Desert. Another way of stating the problem is:

Axiom 0 Sahara $\in$ deserts
Axiom 1 Lion $\in$ Sahara
Axiom 2 $\neg$(Lion $\in$ cage)

We observe the following invariant:

$$P1: \ C(L) \vee \neg(C(L))$$

where $C(L)$ means: the value of L is in the cage.
Establishing C initially is trivially accomplished with the statement

;cage := {}

Note 3 *This is easily implemented by opening the door to the cage and shaking out any lions that happen to be there initially.*

The obvious program structure is then:

;cage:={}
;do $\neg(C(L)) \rightarrow$
    ;"approach lion under invariance of P1"
    ;if $P(L) \rightarrow$
        ;"insert lion in cage"
[] $\neg P(L) \rightarrow$

```
                ;skip
            ;fi
        ;od
```

where $P(L)$ means: the value of L is within arm's reach.

Note 4 *Axiom 1 ensures that the loop terminates.*

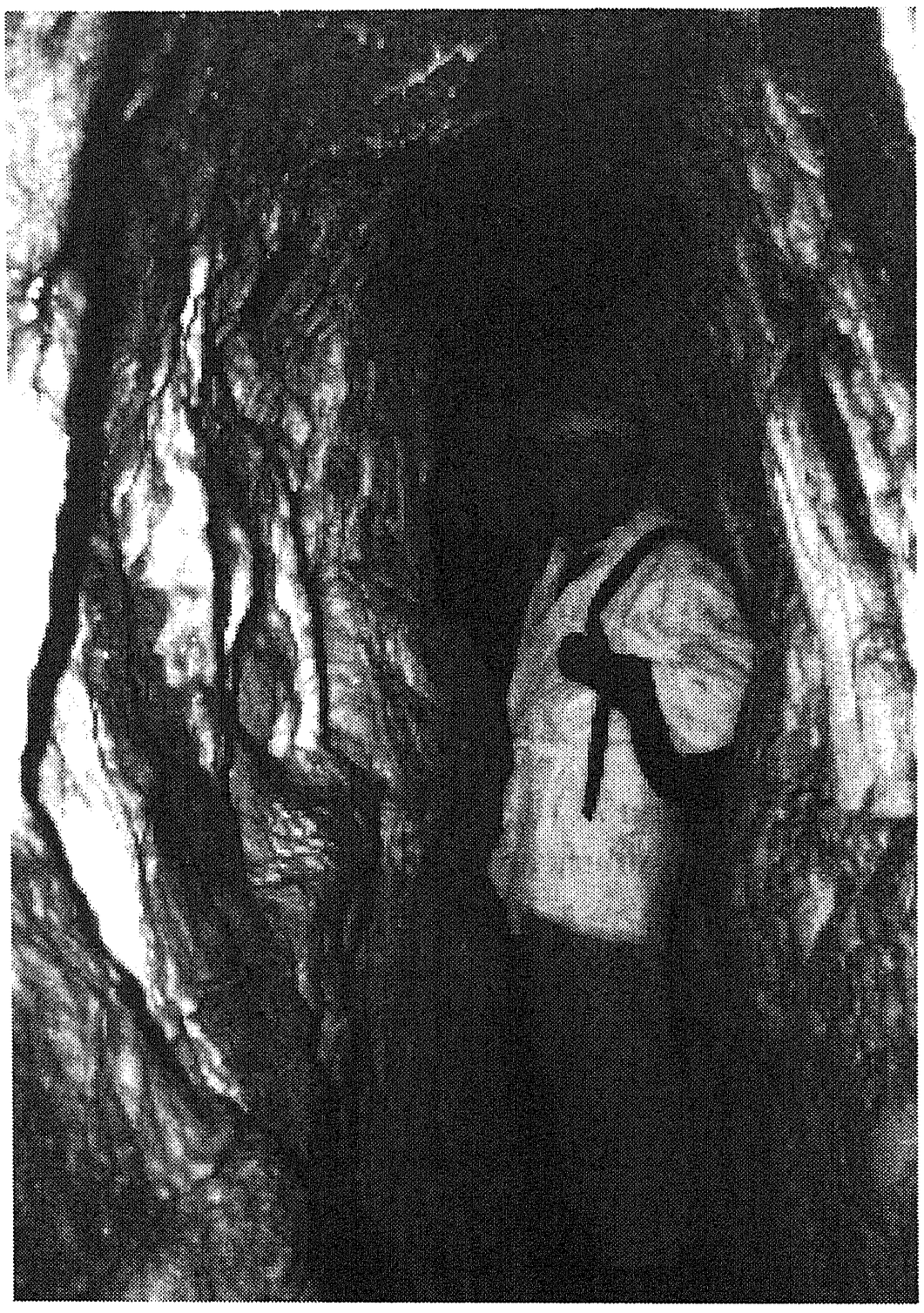
Exercise 5 *Refine the step "Approach lion under invariance of P1".*

Note 6 *The program is robust in the sense that it will lead to abortion if the value of L is "tiger". (This may be a new sense of the word "robust" for you.)*

Note 7 *From observation we can see that the above program leads to the desired goal. It goes without saying that we therefore do not have to run it.*

Bibliography

[1] H. Petard, *A Contribution to the Mathematical Theory of Big Game Hunting*, Princeton, N. J., in American Mathematical Monthly, August, 1938.



Herman te Riele **Beste Paul**

Graag wil ik je langs deze weg van harte feliciteren met je 25-jarig jubileum op het CWI. Wij hebben nooit zo veel met elkaar te maken gehad maar ik vind wel dat het complexiteitsonderzoek dat jij hier doet bij uitstek thuishoort op een modern en "FIT" CWI, als basisonderzoek voor veel ander meer toegepast onderzoek dat hier wordt gedaan. Fast Factoring via Quantum Computing is natuurlijk voor mijn onderzoek zeer interessant. Ik wens je dan ook toe dat je nog vele jaren met vrucht en plezier op het CWI kunt en wilt blijven werken.

Ik weet niet of jij je het volgende voorval nog kunt herinneren, maar het verdient hier zeker gememoreerd te worden. Ik neem zelden of nooit lifters mee maar op een goede dag, ruim 25 jaar geleden, bracht iets me er toe om dat toch te doen. Waarschijnlijk was het het vriendelijke, betrouwbare en intelligente uiterlijk van die lifter. We hadden in ieder geval een plezierige reis samen waarin heel wat is afgekletst. Van waar naar waar de reis ging weet ik niet zeker meer maar ik vermoed van Delft naar Amsterdam.

Wat ik nog wel goed weet is dat ik die lifter enige tijd later tot mijn verrassing op het CWI tegenkwam, en dat bleek jij dus te zijn! Je was net op het CWI begonnen en ik vlij me dan ook met de gedachte dat ik, door jou ooit die lift te geven, een heel klein beetje heb bijgedragen aan het tot stand

komen van dit jubileum!

Als je nog eens een lift nodig hebt ...

Carl Smith

For Paul Vitányi

When I first agreed to write some words about my friend Paul, I responded enthusiastically. It sounded like a simple thing to do. After all, we have maintained contact since we met in the late 1970's and have enjoyed each others company and hospitality on many occasions. After looking at the text editor window for a very long time, I realized that I could write about Paul's research career, which every one knows about or I could relate some personal, potentially embarrassing anecdote. We have had a few adventures, some of which I am sure Paul would rather forget.

When I first met Paul, I noticed, and admired, the enjoyment he got out of research. At the time, I thought that this was due solely to his then current topic of sexually reproducing automata. However, time has shown me that while most scientists find enjoyment in research, Paul finds more than most. Paul has a eye for quality that he focuses on everything from research, to restaurants and even the bicycle he rarely rides.

Research wise, Paul is a theoretical computer scientist. However, he does have a well hidden experimental side. On my first visit to Amsterdam, Paul wanted to teach me the difference between "white bars" and "brown bars." Being curious, I consented to what turned out to be a three day mini course.

Knowing that I was interested in inductive inference, Paul proceeded to show me several examples. After completing this exercise, I knew quite well the difference between the colors of the bars, but was quite unprepared to do anything else for at least 72 hours. It took over a decade to get to the second lesson in the series on "P" bars.

We often met at conferences. At the STOC meeting in Providence, Rhode Island, there was an outing to Newport for the afternoon session and banquet. During the bus ride, Paul convinced me that it would be more interesting to try and find the seaside mansion that was the site of banquet by ourselves, rather than wait for the session to end and take the bus with everyone else. To me this sounded like a nice challenge. The logical thing to do would be to buy a map, but Paul had another idea. He bought a postcard that pictured the mansion in question and we walked to the shore and proceeded to implement a linear search algorithm on the numerous mansions by the sea in Newport. In spite of the known inefficiencies of the algorithm, and the side effect of having to scale a 2 meter high wall, we ended up at the banquet site a full half hour before the busses. Despite this positive performance of Paul's algorithm, it was not optimal, as about 30 people from the conference had arrived at the banquet site before we did.

One of Paul's more amazing characteristics is that he appears to have hardly aged at all in the time I have known him. I look forward to coming back in another 25 years to repeat this enjoyable exercise. By then, I will be old and cranky enough to tell some really embarrassing stories.

John Tromp

As simple as possible, but no simpler

1 Introduction

Paul enjoys considerable fame as one of the leading researchers in Kolmogorov complexity, the theory of ultimate effective compression of data.

Curiously absent from the 'bible' [3] on the subject, that he co-authored with Ming Li, is a concrete definition of its central object of study. Well, perhaps not all that curious, considering the myriad of nasty little details that require dealing with in order to make precise the notion of effective description. Besides which, a concrete definition has only very limited utility, as many of the unspecified $O(1)$ constants, present in the bible in great abundance, could easily fill a few pages when written down to the last bit. Still, having a concrete definition is an interesting exercise in itself, and is the very topic of this chapter.

Central in any conceived concrete definition must be a universal machine, one that interprets part of its input bits as a description of an arbitrary machine, which it proceeds to simulate on the remainder of the input. The beauty of this is that the machine encoding prefix can make the remaining input look like a C program, or a PASCAL program, or a Lempel-Ziv encoding,

or whatever it takes to make the total description as small as possible.

So for starters, one has to pick a specific machine model. Everybody will be familiar with the Turing Machine which has traditionally set the standard. However, a TM has no 'natural' encoding into a bit-string, and the requirement for this encoding to be self-delimiting (so the universal machine can figure out where it ends) further complicates matters. In his book [1], Roger Penrose goes through the gory details, and even presents a 5495 bit encoding of a universal machine, which, if fed as input to a universal machine, will make it simulate itself on the rest of the input!

2 The Functional Programming Approach

In [2], Gregory Chaitin paraphrases John McCarthy, LISP inventor, as "This is a better universal Turing machine. Let's do recursive function theory that way!" Later on Chaitin says "So I've done that using LISP because LISP is simple enough, LISP is in the intersection between theoretical and practical programming. Lambda calculus is even simpler and more elegant than LISP, but it's unusable. Pure lambda calculus with combinators S and K, it's beautifully elegant, but you can't really run programs that way, they're too slow."

For the sake of simplicity though, I will attempt to do just that: base a concrete definition of Kolmogorov complexity on a combinatory logic machine. Slow it may be, but only by a constant factor, and this is one constant I permit myself to ignore!

3 Combinatory Logic

The basic combinators S and K are defined by the rewrite rules

$$\begin{aligned} Sxyz &= xz(yz) \\ Kxy &= x \end{aligned}$$

Other combinators are constructed from these two by application. We assume that application of x on y, written (xy), associates to the left,

and omit parentheses accordingly. Thus, $Sxyz$ should be interpreted as $(((Sx)y)z)$.

Perhaps surprisingly, S and K suffice to define any function definable in standard lambda calculus.

As an example, we can check that $I = SKz$, for any z, is the identify function $\lambda x.x$:

$$SKzx = Kx(zx) = x.$$

The combinator

$$\infty = SSK(SSK)(SSK) = S(SSK)(K(SSK))(SSK)$$

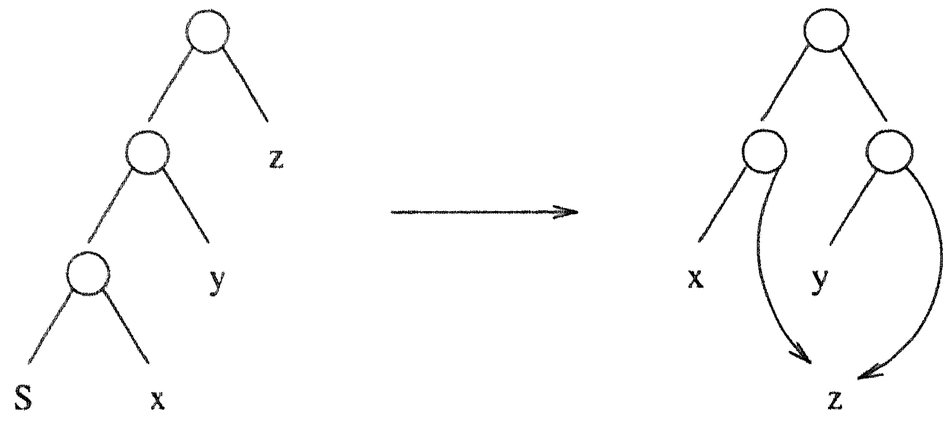
$$= SSK(SSK)(K(SSK)(SSK)) = SSK(SSK)(SSK) = \infty$$

provides another interesting example—it leads to an infinite rewriting chain.

In our machine model, we want to consider any combinator to be a machine, capable of processing input bits and producing output bits. This entails representing an input binary string by a combinator, to which the machine can be applied, and a way of interpreting the result, if possible, as an output binary string.

A rewrite of S or K will form the basic 'machine cycle'. If no more rewrites are possible, then the machine is said to halt, and the resulting combinator is said to be a *normal form* of the combinator starting the rewrite chain. The question remains of which rewrite to perform when different ones are possible. For instance, $KK(KKK)$ may be rewritten to either K or KKK. It turns out that so called 'outermost first' reduction yields a normal form if there is one, and is what we take to be the operation mode of our combinator 'machines'.

In actual implementations, efficiency demands the use of *graph-reduction*, which represents combinators by acyclic directed graphs, avoiding the duplication of z as in



4 Encoding combinators as binary strings

The nice thing about combinators is that they have a wonderfully simple encoding as binary strings, that's self-delimiting to boot: encode S as 00, K as 01, and application (its left parenthesis) 1. Formally, we define the encoding $\langle C \rangle$ of a combinator C as

$$\begin{aligned} \langle S \rangle &= 00 \\ \langle K \rangle &= 01 \\ \langle C_0 C_1 \rangle &= 1\langle C_0 \rangle \langle C_1 \rangle \end{aligned}$$

For instance, the combinator $S(KSS)$, $(S((KS)S))$ in full, is encoded as 10011010000. The length of this encoding is $3n - 1$, where n is the number of S's and K's it contains.

5 Encoding binary strings as combinators

This bit is a little trickier. We proceed, as LISP does, by constructing a pairing function, and then using this to build lists. A pairing function is, of course, not complete without corresponding selector functions, like `car` and `cdr` in LISP. These will be derived from our 'boolean' combinators which we conveniently denote by 0 and 1.

5.1 Itsy little bits

Continuing on our path of uncompromised simplicity, we take

$$\begin{aligned} 0 &= K \quad \texttt{(false)} \\ 1 &= SK \quad \texttt{(true)} \end{aligned}$$

These will be the building blocks of everything. The booleans can be tested conveniently as follows: Bxy evaluates to x if $B = 0$, and to y if $B = 1$. This is analogous to the conditional expression in C: "test ? expression_1 : expression_2".

5.2 Pairing things up

The simplest way to pair two combinators x and y is to have something that, when applied to 0 gives x, and when applied to 1 gives y. What we want then is a combinator, call it V, that satisfies

$$Vxyz = zxy.$$

We leave it as an exercise to the reader that

$$V = S(S(KS)(S(KK)(S(KS)(S(S(KS)(SK))K))))(KK)$$

fits the bill.

To make the step from a pairing function to a list, we'll need some way to represent the empty list (or empty string). The essential requirement on this combinator, call it \$, is that it be easily distinguishable from a non-empty list. A short quest brought forth $\$ = K(KS)$ as an ideal candidate: $z000$ yields 0 when $z = Vxy$ and yields 1 when $z = \$$.

Now we're in business! We can make lists of 0s and 1s all we want. For instance, the string $s = 0111001$ maps to the list $V0(V1(V1(V1(V0(V0(V1\$))))))$, conveniently denoted by $s(\$)$.

Fortunately, the combinators $Vb, b = 0, 1$, look a bit simpler than V itself: $S(K(S(S(SKS)(Kb))))K$.

6 What goes in, must come ...

Unfortunately, the output of a machine is much harder to define than its input. If the machine halts, then how do we interpret the result as a binary string? It is unreasonable to expect the result to be exactly of the form $s(\$)$ for some string s. At least we should be able to recognize if a combinator C is equivalent to some $s(\$)$. This part of my combinatory machine model requires further investigation.

What I currently do is to reduce $C001$ to a normal form, if possible, and see whether this gives 0, 1, or $S1$. If $S1$, then C is equivalent to \$ and interpreted as an empty string (end of output). If 0 or 1 then C is interpreted as a string starting with that bit, and consideration turns to $C1$ for the remaining output. If none of the above cases hold, then the computation is considered invalid.

Having defined now how a combinator can be seen as a machine taking binary strings as input and producing binary strings as output, we return to our main feature:

7 The search for the Universal Combinator

As hinted at before, a machine in our model is a combinator which is applied to the input combinator, i.e. the list that the input string maps to. For machines that are supposed to be able to detect the end of the input, an input string s is mapped to the list $s(\$)$.

For 'prefix machines', that need to decide when to stop reading more input based on what they've read so far, we can present them lists of the form $s(\infty)$ for increasingly longer s, until they halt. Recall that ∞ leads to an infinite computation. Therefore, halting of the machine on $s(\infty)$ implies it hasn't 'read past' string s.

A universal combinator acts like a prefix machine when interpreting the initial part of the input as an encoded combinator, and thereafter can act anyway it likes.

8 Decoding a combinator encoding

We need combinators that, given a list,

1. produce the combinator encoded on the initial part of that list
2. return the remaining part of the list

We'll leave verification that the following combinators do just that as another exercise for the reader: A binary encoding of a combinator can be decoded using the two combinators

$$\begin{aligned} Ax &= x0(x10SK)(A(x1)(A(B(x1)))) \quad \text{(initial combinator)} \\ Bx &= x0(x11)(B(B(x1))) \qquad \text{(skip combinator)} \end{aligned}$$

It may look like cheating to define these combinators recursively, but the fixpoint operator $Y = SSK(S(K(SS(S(SSK))))K)$ (found by studying the rewrite $SSKLx = SL(KL)x = LxL$) comes to the rescue. Y has the nice property that $Yx = x(Yx)$, allowing us to take $A = YA'$ and $B = YB'$, where

$$\begin{aligned} A'xy &= ya(ybaSK)(x(yb)(x(B(yb)))) \\ B'xy &= ya(ybb)(x(x(yb))) \end{aligned}$$

Putting the pieces together, we form the universal combinator

$$U = SAB = S(YA')(YB'),$$

which has the property that

$$U(\langle C\rangle x) = SAB(\langle C\rangle x) = A(\langle C\rangle x)(B(\langle C\rangle x)) = Cx.$$

At long last, we can define Kolmogorov Complexity!

Definition 1 *The plain Kolmogorov Complexity $KC(x)$ of a string x is the length of a minimal string p such that $U(p(\$))$ outputs x. The prefix Kolmogorov Complexity $KP(x)$ of a string x is the length of a minimal string p such that $U(p(\infty))$ outputs x.*

To define Kolmogorov Complexity conditional to a string y, one needs to apply a machine C to a pair of input list and $y(\$)$, using either $C(V(s(\$))(y(\$)))$ or $C(V(s(\infty))(y(\$)))$. This requires a somewhat modified universal combinator

$$U'x = A(x0)(V(B(x0))(x1))$$

which parses a combinator encoding from $x0$ (instead of from x as U does), and then applies this combinator to the pair of the remainder of $x0$ and the conditional argument.

Definition 2 *The plain Kolmogorov Complexity $KC(x|y)$ of a string x conditional to a string y is the length of a minimal string p such that $U'(Vp(\$)y(\$))$ outputs x. The prefix Kolmogorov Complexity $KP(x)$ of a string x conditional to a string y is the length of a minimal string p such that $U'(Vp(\infty)y(\$))$ outputs x.*

9 Conclusion

We've seen that combinatory logic makes an excellent vehicle for defining Kolmogorov Complexity concretely. As a result we can prove statement like

Theorem 3 $K(s) \leq l(s) + 8$

which follows immediately from the observation that the length of the encoded identity combinator is $l(\langle SKK \rangle) = 3 \cdot 3 - 1 = 8$.

I'd like to conclude by showing the universal combinator U in its full 488 bit encoded glory:

```
11100110010100110011000100101011100110001001011000111111000001
11001011100001001100000101110010110011001010011001011001100011
00010010101110011001010011001100101000110111001100110011001100
00100101100011010110100101011011100110011000100101100011011000
01110011001010011001011001010011001011001010111001100101001100
01011001010010011001100110001001011100010010101110011001010011
10010110010100100101011100110010100110010110011000100110010100
11100110001001011000110110111000100101101110011000100101100001
```

Bibliography

[1] R. Penrose, *The Emperor's New Mind*, Oxford University press, 1989.

[2] G. Chaitin, *An Invitation to Algorithmic Information Theory*, DMTCS'96 Proceedings, Springer Verlag, in press (http://www.research.ibm.com/xw-people-chaitin/inv.html).

[3] M. Li, and P. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Texts and Monographs in Computer Science, Springer-Verlag, New York, 1993.


Ulysses
R%ven wonga beach'96

husband dreaming

achter bedauwde raampjes schemert
en daagt het, schemert en daagt het
een koele grijze doos die Frankfurt heet
opent en sluit zich in een doolhof van
tijd en plaats die zich vastklinkt aan
een groter labyrint van tijd en plaats
dat zonder uitgang is

en verderop die waterval met
waterlelies die bij nader inzien
toch echt zijn dat is Singapore. liever
samen thuis wij en de dieren maar goed
toch wij, al is het ook voorbij de Daintree
River in regenwoud en zonder stroom
(niet lang meer zonder stroom dit bos)

met roodrugspinnen en zoutwater
krokodillen die roerloos dromen in
de vreemde stilte van mangroves
tussen Victorian-look ladies
a bird of paradise!— riflebirds
fruitduifjes geelgekroonde kakatoes

en een zeldzame Ulysses, blauw maar
nooitvertoond elektrisch blauw
Ulysses de reiziger net zoals
jij dream husband op weg naar
huis voorgoed op weg naar huis

P. v/d Ven 14.10.1996
horseshoe bay, magnetic isl.

Vladimir Uspensky

Don't even dare to think, let alone say, anything about *A Woman's* Logic

1 Setting the problem.

On September 30, 1925, Andreyj Nikolaevich Kolmogorov, then 22 years old, finished his famous paper on intuitionistic logic [1]. It was in Russian and was published, in the same year, in a Russian journal; luckily, an English translation [2] of it can be found in van Heijenoort's well known book [3].

Kolmogorov's paper was not only historically the first Russian work on logic that contained real mathematical results. It was also the historically first attempt to formalize intuitionistic logic by proposing an axiom system. The Kolmogorov system of five axioms (consisting of four Hilbert's axiom of implication and one Kolmogorov's axiom of negation) can be seen, e. g., on p. 387 of [4].

There arises a natural question belonging to the history of mathematics and logic. Was the said attempt of Kolmogorov to formalize intuitionistic logic his only attempt to formalize a logic? Or do there exist other logics having undergone the procedure of mathematical formalization by Kolmogorov? As we shall see in the next item, the answer is positive: yes, they do exist.

2 Kolmogorov's inference rule

Indeed, Kolmogorov's axiom system for intuitionistic logic does not constitute the only case when he offered a formalization for some nonclassical logic. However, in his attempt to formalize another logic, different from the intuitionistic one, Kolmogorov avoided writing down an axiom system. Instead, he restricted himself to introducing a rule of inference for that logic. The logic in consideration is the so called *a woman*'s logic. In the early eighties, in a private conversation with me, he suggested the following rule of inference for that logic:

if $P \to Q$ and Q is pleasant, then P.

Now I am copying that formulation from the set of proofs of my paper [4]. But one will seek in vain for the formulation in the Journal of Symbolic Logic.

3 Logic Colloquim '89

The 1989 European Summer Meeting of the Association for Symbolic Logic was *Logic Colloquim '89*, held at the Technische Universität and at the Freie Universität in Berlin (West). I was invited to that conference to present an expository talk "A. N. Kolmogorov and Mathematical Logic". It was the first time that I got permission from the Soviet authorities to cross the Iron Curtain. In reality, it turned out that crossing the Curtain was not a mere metaphor since a Soviet citizen should fly to East Berlin and then cross the Wall by foot.

My talk was fixed at 7.30 p.m. on July 28 with no other talk or social event at the same time; for that reason I got a rather large audience. My joke on Kolmogorov's formalization of *a woman*'s logic got a good response. The people laughed. Nobody of about two hundred Colloquim participants who were present in the auditorium expressed any dissatisfaction. On the contrary, in a brief discussion that ran after my talk, Professor Doctor Ernst-Jochen Thiele of the Technische Universität Berlin, who was the Chair of the Logic Colloquim's Program Committee, said that he had only one remark, and that remark was that the talk was good. I do understand that it is inappropriate for me to cite here those complimentary words of Prof. Dr.

Thiele but I do so to explain and even to partially justify two of my following actions. The first of them was to hand to Prof. Thiele the handwritten manuscript with the text of my talk. It was the first action, and it was done just after my talk. The second one was done later, in Massachusetts.

Prof. Thiele was nice enough to arrange both for typing my manuscript and for its submission to the Journal of Symbolic Logic; I believe that without his generous care the paper would have never appeared. In my text there was a passage concerning Kolmogorov's formalization of a woman's logic. That passage included Kolmogorov's inference rule for that logic.

4 Boston faux pas, or Never repeat the same joke even to a different audience

To mention Kolmogorov's rule of inference in a serious scientific talk, it seemed to be so witty! So I was delighted with my own wittiness, and I decided to repeat the joke in Boston, Massachusetts. I gave a lecture "Kolmogorov's Contribution to Logic and Computer Science" at Boston University on November 16, 1989. There and then I repeated the joke (it was the second of the two actions mentioned in the preceding item). The response was quite different from Berlin's. No laughs. Instead, some confusion. And after my lecture the organizers told me that my words were as politically incorrect as had they been racial. It was my first visit to the USA, I was an absolute novice in the American problem of political correctness and even had no knowledge of that problem at all; so that event struck me very much. I was sure that no person in my country would be offended if she or he has heard such a joke. And I hope that, fortunately, up to now (October of 1996) the situation in Russia has not changed substantially.

5 The fate of my paper

Meanwhile, typing of my paper proceeds in Berlin. The procedure took rather a long time, and I am fully responsible for all delays since I inserted corrections in many successive versions that were sent to me by Prof. Thiele. He remained kind and patient. At last, the final version of my paper "Kolmogorov and Mathematical Logic" was sent by Prof. Thiele to Prof. Ward Henson of the University of Illinois who was the Secretary-Treasurer of the Association for Symbolic Logic (ASL). That association publishes the Journal of Symbolic Logic

(JSL). By the letter of April 11, 1991, Ms. Betsy Gillies, ASL Staff Associate from the Urbana Office of the Secretary-Treasurer, informed Prof. Thiele that my paper had been forwarded to Prof. John Burgess of Princeton University who was one of the JSL editors. And Prof. Burgess informed me by his letter of September 24, 1991, that my Kolmogorov paper had been accepted by JSL.

6 Reading galley proofs

On November 21, 1991, an envelope from the American Mathematical Society (AMS), Providence, Rhode Island, reached me. It is AMS which does all the type setting for ASL. The envelope was sent by the Editorial Department of ASL, and it contained the set of proofs of my paper. I was impressed by the care with which the text was printed (not being accustomed to such care), and especially with the elaboration with which the list of references was worked up. Every entry was checked and rechecked, and in some cases amended. I am indebted to Mr. Ralph Sizer, Production Editor, JSL, for his attention to the bibliographical details. Here is a typical nice citation from a letter of him: "On rechecking I noticed that the page span should be 217–245 (instead of 244)".

In volume 57 of the JSL, one can see a paragraph running from page 387 to 388. It ends (on page 388) with the sentence: "Thus we have the historically first attempt to formalize intuitionistic logic by proposing an axiom system". In the proofs, after that sentence, there stood in parentheses in the same paragraph:

> (By the way, this is not the only case when Kolmogorov offered an axiomatization for some nonclassical logic. In a private conversation with me he suggested the following rule of inference for a "woman's logic": "if $P \to Q$ and Q is pleasant, then P".)

I believe the text would appear as it was had I not brought attention to that joke. Unfortunately, I had. Twice even.

First, I posed some grammatical problem. I marked the indefinite article before "woman's logic" and wrote on the margins of proofs:

> Proof-reader—It seems that 'a' relates to 'woman', not to 'logic'—so it must be *inside* the quotation marks: "a woman's logic". Or there must be no 'a' at all? My English is poor, so the decision is up to you. Thanks.

Second, I suggested a footnote. The place of the footnote: just before the closing parenthesis. The text of the footnote:

> My remark on Kolmogorov's formalization of "a woman's logic" was met with a well-disposed laugh by the Berlin audience on July 28, 1989. The same remark met an ill-disposed silence when I gave a talk "Kolmogorov's contribution to logic and computer science" at Boston University on November 16, 1989. I dare to believe that even the great struggle for women's rights might tolerate some sense of humour.

The total amount of suggested footnotes was two. Another footnote expressed my gratitude to Prof. Thiele.

7 The fate of corrected proofs

I was asked to return the set of proofs as soon as possible. Which I did. Mr. Sizer answered me with a letter of December 12, 1991. He wrote:

> Your corrected proofs arrived here today. Thank you for the prompt return. ⟨...⟩ The footnote will be added where you requested.
>
> All the things seemed to be OK. Nothing portended thunder.

But the weather changed very quickly. Soon I received a new letter from Providence. That letter was dated December 23 and was handwritten by an executive of JSL's production office. I am afraid I have no right to bring up his name here. I neither have the right to name another, more senior person whose title, name, and position in JSL were mentioned in the letter, so I denote this second person by *Z*. Here is said letter of December 23, 1991:

> Dear Professor Uspensky:
>
> I passed the problem of the "woman's logic" joke to *Z*, and he decided it would be better to simply delete the joke. We both regret the present lack of humor in the USA which makes this necessary.

8 "Conditions here" or, better to say, there

Z also wrote a letter to me but I have not received it. Nothing surprising: ours is a special country like VSOP (that is, "very special old product"), and from

time to time some Russian newspaper publishes a story about a sack having been found in a suburban forest. Somebody threw the sack out, and the sack is filled with undelivered letters from abroad.

Presuming that I have received his letter which was actually lost, *Z* wrote to me a second, very short note dated January 17, 1992:

> Found this in the newspaper the day after I wrote you. It will give you some idea of conditions here.

A xerox-copy of an article from a newspaper was attached to that note. Neither the newspaper's title nor the date were indicated. But the article's title and author were indicated: it was "Francisco Jose de Goya Convicted of Sexual Harassment" by Nat Hentoff. The story told of officials at Penn State's Schuykill (Skool'-Kihl) campus being forced to take away from the wall a copy of Goya's *Naked Maja*. A sentence from the article was marked by *Z*:

> Sexual harassment is being found by courts to exist when a "hostile work environment" has been created.

I took the marked sentence about courts as a threat. The term in quotation marks was explained in the newspaper's article as follows:

> A "hostile work environment" means words, as well as pictures ⟨and rules of inference, I shall add⟩. It means the steady weakening of the First Amendment—and its spirit—in this huge arena of American life.

9 Should I be convicted as Goya was, or be lynched without a trial?

I don't know. But if there be a trial I will invite witnesses to the court to prove that I respect women. I hope two Dutch women will agree to witness for me. These two persons are respected by me to the highest degree among all the women of the Koninkrijk der Nederlanden (not counting Her Majesty, of course). One is called Pauline, and she is a writer. Another is called Marja, and she is a secretary at the CWI. And each of them, within the bounds of their competence, secure a friendly and favourable environment for the scientific research activity of Paul Vitányi.

Bibliography

[1] A. N. Kolmogorov. *On the principle "tertium non datur"*, Matematicheskiyj Sbornik, vol. 32 (1924/25), pp. 646–667 (In Russian).

[2] A. N. Kolmogorov, *On the principle of excluded middle* (an English translation of [1]), in [3], pp. 416–437.

[3] J. van Heijenoort (editor), *From Frege to Gödel: a Source Book in Mathematical Logic, 1879–1931*, Harvard University Press, Cambridge, Massachusetts, 1967.

[4] V. A. Uspensky, *Kolmogorov and mathematical Logic*, The Journal of Symbolic Logic, vol. 57 (1992), no. 2, pp. 385–412.

00 B xx

00 A 08