E. Börger G. Jäger H. Kleine Büning
S. Martini M. M. Richter (Eds.)

Computer Science Logic

6th Workshop, CSL '92
San Miniato, Italy
September 28 - October 2, 1992
Selected Papers

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Series Editors

Gerhard Goos
Universität Karlsruhe
Postfach 69 80
Vincenz-Priessnitz-Straße 1
D-76131 Karlsruhe, Germany

Juris Hartmanis
Cornell University
Department of Computer Science
4130 Upson Hall
Ithaca, NY 14853, USA

Volume Editors

Egon Börger
Simone Martini
Dipartimento di Informatica, Università di Pisa
Corso Italia, 40, I-56125 Pisa, Italy

Gerhard Jäger
Universität Bern, Institut für Informatik und angewandte Mathematik
Länggasstraße 51, CH-3012 Bern, Switzerland

Hans Kleine Büning
FB 17, Mathematik/Informatik, Universität - GH Paderborn
Postfach 1621, D-33095 Paderborn, Germany

Michael M. Richter
FB Informatik, Universität Kaiserslautern
Postfach 30 49, D-67653 Kaiserslautern, Germany

CR Subject Classification (1991):F, I.2.3-4, G.2, D.3

ISBN 3-540-56992-8 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-56992-8 Springer-Verlag New York Berlin Heidelberg

Typesetting: Camera-ready by authors
Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
45/3140-543210 - Printed on acid-free paper

Preface

The *Computer Science Logic* Workshop CSL'92 was held in San Miniato (Pisa) from September 28 to October 2, 1992. It took place in the charming environment of the Centro Studi *I Cappuccini*, a nicely restored monastery made available by the Cassa di Risparmio of San Miniato. CSL'92 was the sixth of the series and the first one which was held as Annual Conference of the *European Association for Computer Science Logic*, founded in Schloß Dagstuhl in July 1992 by computer scientists and logicians from 14 countries.

The workshop was attended by 78 participants from 15 countries; 8 invited lectures and 25 talks, selected from 72 submissions, were presented. Following the traditional procedure for CSL volumes, full versions of the original contributions have been collected after their presentation at the workshop and a regular reviewing procedure has been started. On the basis of 58 reviews, 26 papers were selected for publication. They appear here in revised final form.

We thank the referees, without whose help we would not have been able to accomplish the difficult task of selecting among the many valuable contributions.

We also gratefully acknowledge the generous sponsorship by the following institutions:

Consiglio Nazionale delle Ricerche (CNR)
Cassa di Risparmio di San Miniato
Università degli Studi di Pisa
Dipartimento di Informatica dell'Università di Pisa
Cassa di Risparmio di Pisa
Hewlett-Packard Italiana S.p.A., Pisa Science Center

Finally, we would like to thank the following persons who generously helped in various ways in the organization of the conference: Antonella D'Alessandro, Paola Glavan, Stefania Gnesi, Elvinia Riccobene.

March 1993

E. Börger G. Jäger H. Kleine Büning S. Martini M.M. Richter

List of Referees

K. Ambos-Spies, Heidelberg
H. Barendregt, Nijmegen
M. Bezem, Utrecht
E. Börger, Pisa
S. Buss, San Diego
T. Coquand, Göteborg
E. Dahlhaus, Sydney
R. De Nicola, Roma
M. Dezani-Ciancaglini, Torino
A. Goerdt, Paderborn
E. Grädel, Aachen
E. Grandjean, Caen
Y. Gurevich, Ann Arbor
M. Hanus, Saarbrücken
R. Hasegawa, Paris
F. Honsell, Udine
M. Hyland, Cambridge
N. Immerman, Amherst
G. Jäger, Berne
H. Jervell, Oslo
H. Kleine Büning, Paderborn
P. Kolaitis, Santa Cruz
P. Lincoln, Stanford
J. Makowsky, Haifa
S. Martini, Pisa
W. McCune, Argonne
E. Moggi, Genova
F. Montagna, Siena
L. Pacholski, Wroclaw
M. Parigot, Paris
C. Paulin-Mohring, Lyon
L. Priese, Koblenz
W. Reisig, München
M.M. Richter, Kaiserslautern
D. Rosenzweig, Zagreb
L. Roversi, Pisa
D. Sangiorgi, Edinburgh
A. Scedrov, Philadelphia
W. Schönfeld, Heidelberg
H. Schwichtenberg, München
D. Seese, Karlsruhe
J. Shepherdson, Bristol
E. Speckenmeyer, Düsseldorf
D. Spreen, Siegen
R. Stärk, Münich
W. Thomas, Kiel
K. Wagner, Würzburg

Table of Contents

A Universal Turing Machine
Stål Aanderaa 1

Recursive Inseparability in Linear Logic
Stål Aanderaa and Herman Ruge Jervell 5

The Basic Logic of Proofs
Sergei Artëmov and Tyko Straßen 14

Algorithmic Structuring of Cut-Free Proofs
Matthias Baaz and Richard Zach 29

Optimization Problems: Expressibility, Approximation Properties and Expected Asymptotic Growth of Optimal Solutions
Thomas Behrendt, Kevin Compton and Erich Grädel 43

Linear λ-Calculus and Categorical Models Revisited
Nick Benton, Gavin Bierman, Valeria de Paiva and Martin Hyland . 61

A Self-Interpreter of Lambda Calculus Having a Normal Form
Alessandro Berarducci and Corrado Böhm 85

An "Ehrenfeucht-Fraïssé Game" for Fixpoint Logic and Stratified Fixpoint Logic
Uwe Bosse 100

The Class of Problems that Are Linearly Equivalent to Satisfiability, or a Uniform Method for Proving NP-Completeness
Nadia Creignou 115

Model Building by Resolution
Christian G. Fermüller and Alexander Leitsch 134

Comparative Transition System Semantics
Tim Fernando 149

Reasoning with Higher Order Partial Functions
Antonio Gavilanes-Franco, Francisca Lucio-Carrasco and Mario Rodríguez-Artalejo 167

Communicating Evolving Algebras
Paola Glavan and Dean Rosenzweig 182

On the Completeness of Narrowing as the Operational Semantics of Functional Logic Programming
Juan Carlos González-Moreno, María Teresa Hortalá-González and Mario Rodríguez-Artalejo 216

Inductive Definability with Counting on Finite Structures
Erich Grädel and Martin Otto 231

Linear Time Algorithms and NP-Complete Problems
Etienne Grandjean 248

The Semantics of the C Programming Language
Yuri Gurevich and James K. Huggins 274

A Theory of Classes for a Functional Language with Effects
Furio Honsell, Ian A. Mason, Scott Smith and Carolyn Talcott 309

Logical Definability of NP-Optimization Problems with Monadic Auxiliary Predicates
Clemens Lautemann 327

Universes in the Theories of Types and Names
Markus Marzetta 340

Notes on Sconing and Relators
John C. Mitchell and Andre Scedrov 352

Solving 3-Satisfiability in Less than $1{,}579^n$ Steps
Ingo Schiermeyer 379

Kleene's Slash and Existence of Values of Open Terms in Type Theory
Jan M. Smith 395

Negation-Complete Logic Programs
Robert F. Stärk 403

Logical Characterizations of Bounded Query Classes II: Polynomial-Time Oracle Machines
Iain A. Stewart 410

On Asymptotic Probabilities of Monadic Second Order Properties
Jerzy Tyszkiewicz 425

A Universal Turing Machine

Stål Aanderaa

University of Oslo
staal.aanderaa@math.uio.no

Abstract. The aim of this paper is to give an example of a universal Turing machine, which is somewhat small. To get a small universal Turing machine a common constructions would go through simulating tag system (see Minsky 1967). The universal machine here simulate two-symbol Turing machines directly.

The Turing machine is defined by the Figure 1 or the Table 1. Suppose the universal Turing machine should simulate the Turing machine defined by the Figure 2 or by the Table 2, starting by the instantaneous description

$$010p_0100 \tag{1}$$

Then the universal Turing machine UTm should start by the instantaneous description

$$01AABq_0baac^3d^{599}cd^{14}c^3d^{708}cd^{15}c^3d^{599}cd^{10}c^3d^{644}cd^{11}c. \tag{2}$$

Here the first three symbols in (2): $01A$, code the first three symbols in (1): 010. The next two symbols in (2): AB code the state symbol p_0 in (1). q_0 in (2) denote the state of the universal Turing machine. The symbols baa in (2) code the last three symbols 100 in (1). The last part of (2):

$$c^3d^{599}cd^{14}c^3d^{708}cd^{15}c^3d^{599}cd^{10}c^3d^{644}cd^{11}c. \tag{3}$$

codes the Turing machine Tme defined in Table 2. The exponents are calculated as follows:

	A	B	C	D	E	F
(i)	$0p_00 \vdash 0H0$	$AABA$	$(0010)_2 = 2$	$RAAA0,$	$(21113)_4$	$= 599$
(ii)	$0p_01 \vdash_L p_101$	$AABB$	$(0011)_2 = 3$	$LBB01$	$(00032)_4$	$= 14$
(iii)	$0p_10 \vdash_R 01p_0$	$ABBA$	$(0110)_2 = 6$	$R0BAB$	$(23010)_4$	$= 708$
(iv)	$0p_11 \vdash_L p_100$	$ABBB$	$(0111)_2 = 7$	$LBB00$	$(00033)_4$	$= 15$
(v)	$1p_00 \vdash 1H0$	$BABA$	$(1010)_2 = 10$	$RAAA0,$	$(21113)_4$	$= 599$
(vi)	$1p_01 \vdash_L p_111$	$BABB$	$(1011)_2 = 11$	$LBB11$	$(00022)_4$	$= 10$
(vii)	$1p_10 \vdash_R 11p_0$	$BBBA$	$(1110)_2 = 14$	$R1BAB$	$(22010)_4$	$= 644$
$(viii)$	$1p_11 \vdash_L p_110$	$BBBB$	$(1111)_2 = 15$	$LBB10$	$(00023)_4$	$= 11$

In row (ii) the exponents of d is calculated to be 14, in order to simulate the move stated in column A. First we have to calculate where to put the information.

This is done coding $0p_01$ in the following way. Replace 0, p_0 and 1 by A, AB and B, respectively. Then we get the word $AABB$ in column B. This word is interpreted as a binary number which is calculated in column C to be 3. This means that the information about the move is to be located between the c number 4 and 5. The L in the word $LBB01$ means that the head moves to the left in this move. The rest $BB01$ of the word $LBB01$ codes the word p_101, where p_1 is replaced by BB and the rest of the word is kept unchanged. Then $LBB01$ is interpreted as a base 4 number in the following way: L, B, 0 and 1 are interpreted as the digits 0, 0, 3 and 2, respectively. The result is $(00032)_4$ which is the decimal number 14.

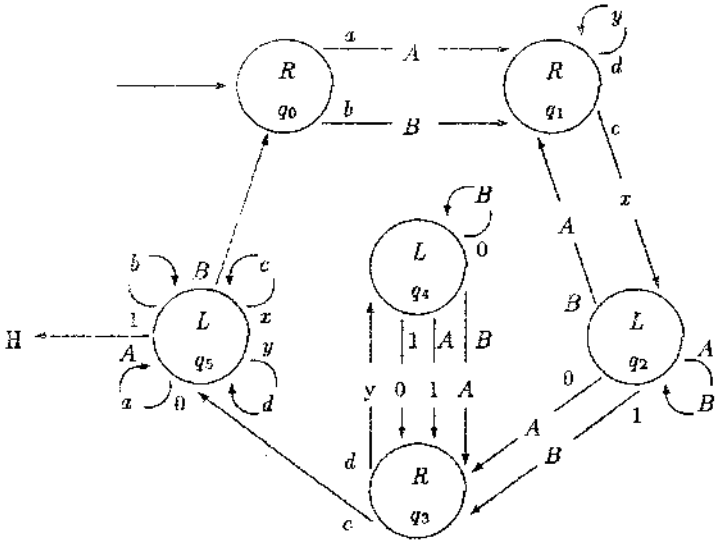
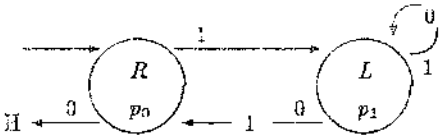


Fig. 1. Universal Turing machine: UTm



	p_0	p_1
0	HALT 1	$R\ p_0$
1	$1\ L\ p_1$	$0\ L\ p_1$

Fig. 2. Turing machine example: Tme

In row (iii) the columns A, B and C are made in the same way as in row (ii). In column D the word $R0BAB$ represents the subword $01p_0$ of column A in the following way. R means that the move is a right move. 0 is kept unchanged. 1 is

	q_0	q_1	q_2	q_3	q_4	q_5
a	$A\ R\ q_1$	$a\ R\ q_1$	$a\ L\ q_2$	$a\ R\ q_3$	$a\ L\ q_4$	$a\ L\ q_5$
b	$B\ R\ q_1$	$b\ R\ q_1$	$b\ L\ q_2$	$b\ R\ q_3$	$b\ L\ q_4$	$b\ L\ q_5$
c	$c\ R\ q_0$	$x\ L\ q_2$	$c\ L\ q_2$	$c\ L\ q_5$	$c\ L\ q_4$	$c\ L\ q_5$
d	$d\ R\ q_0$	$y\ R\ q_1$	$d\ L\ q_2$	$y\ L\ q_4$	$d\ L\ q_4$	$d\ L\ q_5$
x	$x\ R\ q_0$	$x\ R\ q_1$	$x\ L\ q_2$	$x\ R\ q_3$	$x\ L\ q_4$	$c\ L\ q_5$
y	$y\ R\ q_0$	$y\ R\ q_1$	$y\ L\ q_2$	$y\ R\ q_3$	$y\ L\ q_4$	$d\ L\ q_5$
A	$A\ R\ q_0$	$A\ R\ q_1$	$B\ L\ q_2$	$A\ R\ q_3$	$1\ R\ q_3$	HALT
B	$B\ R\ q_0$	$B\ R\ q_1$	$A\ R\ q_1$	$B\ R\ q_3$	$A\ R\ q_3$	$B\ R\ q_0$
0	$0\ R\ q_0$	$0\ R\ q_1$	$A\ R\ q_3$	$0\ R\ q_3$	$B\ L\ q_4$	$a\ L\ q_5$
1	$1\ R\ q_0$	$1\ R\ q_1$	$B\ R\ q_3$	$1\ R\ q_3$	$0\ R\ q_3$	$b\ L\ q_5$

Table 1. Universal Turing machine: UTm

replaced by B and p_0 is replaced by AB. Then R, 0, B and A are interpreted as the digits 2, 3, 0 and 1,respectively, in the number system of the base four. The result is $(23010)_4$ which is calculated to be 708 in decimal.

To simulate the move $010p_0100 \vdash 01p_10100$ of the Tme machine, the UTm machine will have to use about 20 000 steps. Among the instantaneous descriptions which will occur are the following instantaneous descriptions (Compare row (ii) above):

$$01AABq_0baac^3d^{599}cd^{14}c^3d^{708}cd^{15}c^3d^{599}cd^{10}c^3d^{644}cd^{11}c.$$
$$01AABBq_1aac^3d^{599}cd^{14}c^3d^{708}cd^{15}c^3d^{599}cd^{10}c^3d^{644}cd^{11}c.$$
$$01AABBaq_2axc^2d^{599}cd^{14}c^3d^{708}cd^{15}c^3d^{599}cd^{10}c^3d^{644}cd^{11}c.$$
$$0Bq_3BBBBaax^3y^{599}xd^{14}c^3d^{708}cd^{15}c^3d^{599}cd^{10}c^3d^{644}cd^{11}c.$$
$$0BBBBBaax^3y^{599}q_4xyd^{13}c^3d^{708}cd^{15}c^3d^{599}cd^{10}c^3d^{644}cd^{11}c.$$
$$0BBB01aax^3y^{598}xy^{13}q_5yc^3d^{708}cd^{15}c^3d^{599}cd^{10}c^3d^{644}cd^{11}c.$$
$$0BBBq_0abaac^3d^{599}cd^{14}c^3d^{708}cd^{15}c^3d^{599}cd^{10}c^3d^{644}cd^{11}c.$$

To simulate the next move $01p_10100 \vdash 011p_0100$ of the Tme machine, the following instantaneous descriptions will occur (Compare row (vii) above):

$$0BBBq_0abaac^3d^{599}cd^{14}c^3d^{708}cd^{15}c^3d^{599}cd^{10}c^3d^{644}cd^{11}c.$$
$$0BBBAq_1baac^3d^{599}cd^{14}c^3d^{708}cd^{15}c^3d^{599}cd^{10}c^3d^{644}cd^{11}c.$$
$$0BBBAbaq_2axc^2d^{599}cd^{14}c^3d^{708}cd^{15}c^3d^{599}cd^{10}c^3d^{644}cd^{11}c.$$
$$Aq_3BBBBbaax^3y^{599}xy^{14}x^3y^{708}xy^{15}x^3y^{599}xy^{10}x^3d^{644}cd^{11}c.$$
$$ABBBBbaax^3y^{599}xy^{14}x^3y^{708}xy^{15}x^3y^{599}xy^{10}x^2q_4xyd^{643}cd^{11}c.$$
$$01BABbaax^3y^{599}xy^{14}x^3y^{708}xy^{15}x^3y^{599}xy^{10}x^3y^{643}q_5ycd^{11}c.$$
$$01BABq_0baac^3d^{599}cd^{14}c^3d^{708}cd^{15}c^3d^{599}cd^{10}c^3d^{644}cd^{11}c.$$

The loop q_1 - q_2 converts from binary to unary in order to find the information about the next move.

Before the information is evaluated the UTm machine guesses that the next move will be a left move. Hence to be prepared for a left move, the UTm machine changes the content of a square when moving from state q_2 to the state q_3. (a 0

is changed to A and a 1 is canged to B). If the move turn out to be a right move, the guess was wrong, and in order to change back each R in colemn D above is replaced by 2 in colemn E. If the move was a left move, the guess was correct, and nothing has to be changed and aech L in colemn D above is replaced by 0 in colemn E.

The loop q_3 - q_4 converts from unary to a number in base four in order to code the next state and to code the new position of the head.

Reference

Minsky, Marvin *Computation : Finite and Infinite Machines.* Prentice-Hall 1967.

Recursive Inseparability in Linear Logic

Stål Aanderaa[1] and Herman Ruge Jervell[2]

[1] University of Oslo
staal.aanderaa@math.uio.no
[2] Universities of Oslo and Tromsø
herman.jervell@ilf.uio.no

Abstract. We first give our version of the register machines to be simulated by proofs in propositional linear logic. Then we look further into the structure of the computations and show how to extract "finite counter models" from this structure. In that way we get a version of Trakhtenbrots theorem without going through a completeness theorem for propositional linear logic. Lastly we show that the interpolant I in propositional linear logic of a provable formula $A \multimap B$ cannot be totally recursive in A and B.

We use results and notations about linear logic as given by Troelstra in his lectures [T].

1 ANDOR machines

We consider register machines with a finite number of registers $\mathbf{a}$ $\mathbf{b}$ $\mathbf{c}$ $\mathbf{d}$ .... The machines have a number of states say $\mathbf{p}$ $\mathbf{q}$ $\mathbf{r}$ $\mathbf{s}$ ... $\mathbf{t}$. The computations are controlled by instructions. There are four forms of instruction

$p : a+(q)$ add 1 to register a and proceed to state q
$p : a-(q)$ subtract 1 from register a if possible and proceed to state q
$p : and(q,r)$ spawn off two processes and proceed with states q and r
$p : or(q,r)$ proceed nondeterministically to one of the states q and r

There are the following important differences from ordinary register machines

1. In the ordinary register machine the subtraction instruction is combined with a branch. If the register is empty we proceed in one direction, else we subtract 1 and proceed in another direction. This is not the case with andor machines. If the subtraction is not possible, we go over into a waiting state from which the computation does not proceed.
2. In the and-branching we spawn off two new processes from a given process.
3. A process halts if it comes to a halting state and all registers are empty.
4. To get termination of the machine we demand that all spawned processes halt.

Let us see that for register machine computations with empty halting states we get the same computations with andor machines. This is done by showing that register machine instructions can be translated directly into andor machine instructions. To do this we need a simple trick (see [LMSS]). Assume we have one halting state **h** and two registers **a** **b** and have the following register machine instruction

$$p : \text{if } a = 0 \text{ then goto } q \text{ else } a - (r)$$

To translate this instruction we need some new auxiliary states **k l m n** . The following andor machine instructions do the job for the case of two registers **a** **b** and one halting state **h**. The general case with more registers and more halting states is done in a similar way. The instruction above is replaced by

$$\begin{array}{|l} p : or(k,l) \\ k : and(m,q) \\ m : or(n,h) \\ n : b - (m) \\ l : a - (r) \end{array}$$

The trick is to allow a number of garbage computations which does not matter. After having come to state **k** , you can only come to state **q** if it is possible to get to a halting state **h** after emptying all other registers than **a** (in our case register **b**).

Any computation on a register machine ending in a halting state with empty registers can then be transferred to a terminating computation on an andor machine.

2 The structure of computations

A computation in a register machine can be thought of as a transition between storage states. In an andor machine we must also take into account all the processes that are spawned. This is done by the following syntax

register :: **a** | **b** | **c** | **d** | ...
state :: **p** | **q** | **r** | **s** | ...
storagestate :: **empty** | **register** | **state** | **storagestate** · **storagestate**
configuration :: **setof storagestate**

The product of storage states given by · is assumed to be commutative. We are only interested in storage states which have at most one state present - even if we have notation for more. The storage state **aaabbq** indicates that we are in state **q** with 3 in register **a** and 2 in register **b**. The following is a configuration {**aabq**, **aar**, **abs**} . Configurations are interpreted conjunctively so that the configuration above can be thought of as three processes with storage states **aabq** **aar** **abs** . Configurations are sets and not multisets. A typical

halting state is a configuration { **h** } with empty registers. The computations in an andor machine can be thought of as transitions between configurations in an obvious way.

Recursive inseparability 1 *There is an instruction set I with two halting configurations C and D such that no state is transformed into both C and D, and the configurations transformed into C and those into D are recursively inseparable.*

To get to the algebraic structures we first remark that the storage state together with the concatenation $\cdot$ and **empty** makes a commutative monoid with unit. We use this to make an IL-algebra (see [T]) in the standard way

$$X * Y := \{x \cdot y : x \in X, y \in Y\}$$
$$X \multimap Y := \{z : \forall x \in X (z \cdot x \in Y)\}$$

Then $(\mathbf{configurations}, \cap, \cup, \emptyset, \multimap, *, \{\mathbf{empty}\},)$ is an IL-algebra. (See [T] Proposition 8.9). On the top of this algebraic structure we introduce the transition made by the andor machine. The transition $\rightarrow$ gives a relation between configurations. For the transition we have the following laws

T1 : If $C \rightarrow D$, then also $A * C \rightarrow A * D$
T2 : $C \rightarrow C$
T3 : If $C \rightarrow D$ and $D \rightarrow E$ then $C \rightarrow E$

Note T1. This is the crucial property in the construction of andor machines. It is not true for register machines.

An instruction set I is given by a finite number of relations of the following forms

I1 : $A * C \rightarrow D$ (from $a+$)
I2 : $C \rightarrow A * D$ (from $a-$)
I3 : $C \rightarrow D \cup E$ (from *and*)
I4 : $C \rightarrow D$ and $C \rightarrow E$ (from *or*)

Given an andor machine we can transform it faithfully into a configuration space with transitions and a finite number of equations of form I1-4. A configuration space with transition is a very simple structure. The laws are just sufficient to simulate computations in an andor machine given the instructions.

3 Linear logic

Now consider propositional linear logic. In most of our discussions we use the fragment MALLA — multiplicative additive propositional linear logic with assumptions. We write

$$A, B, C, \ldots \vdash F, G, H, \ldots$$

Here the assumptions are to the left of $\vdash$. The formulas A, B, C, F, G, H are from the multiplicative additive fragment of linear logic. The intended meaning is that assumptions can be brought into derivations any number of times – or possibly not at all. Using exponentials we can interpret this as

$$!A \ast !B \ast !C \multimap F + G + H$$

In MALLA there is an obvious interpretation of instruction sets for andor machines

- the letters in the instruction set is taken as atomic formulas
- $*$ into multiplicative conjunction $\star$
- $\cap$ into additive conjunction $\sqcap$
- $\cup$ into additive disjunction $\sqcup$
- $\rightarrow$ into linear implication $\multimap$
- {**empty**} into the multiplicative unit 1

Given an andor machine with the instruction set I. Let $I^{\#}$ be the corresponding formula in propositional linear logic. A configuration C can be transformed into a propositional linear formula $C^{\#}$ in the same way. Then

Theorem 1. *Let I be an instruction set and C and D configurations in an andor machine. The following are equivalent*

- *I transfers C into D*
- *from the equations I we can derive $C \rightarrow D$ using T1-3 and the laws of IL-algebra*
- *$I^{\#} \vdash C^{\#} \multimap D^{\#}$ is derivable in propositional linear logic*

The only slightly difficult part here is to prove that there are not more formulas of the form $I^{\#} \vdash C^{\#} \multimap D^{\#}$ provable in propositional linear logic than those given by the andor machine. This is proved in [LMSS] by a cut elimination argument.

Recursive inseparability 2 *There is an instruction set I and halting configuration D such that for configurations C we cannot recursively separate whether*

- *we can prove $I^{\#} \vdash C^{\#} \multimap D^{\#}$ in propositional linear logic*
- *a configuration structure with transition gives a counter model that $C \rightarrow D$ using instructions I*

4 Getting a phase structure

The configuration space is of course not much more than a convenient language for our syntactical transformations. The information about the particular andor machine we use is contained in the transitions. We now bring the transitions into the IL-algebra. A transition is a binary relation between configurations. We can interpret this in an IL-algebra by repeating the usual subset construction on the configuration space to get an IL-algebra with

Domain : Sets of configurations
Product : $X * Y = \{x * y : x \in X, y \in Y\}$
Implication : $X \multimap Y = \{z : \forall x \in X(z * x \in Y)\}$
Lattice : Union and intersection of sets

On this IL-algebra we can interpret transitions as operators

$$C(X) = \{y : y \rightarrow x \wedge x \in X\}$$

taking sets of configurations into sets of configurations.
A closure operation in an IL-algebra satisfies

1. $X \subseteq C(X)$
2. $X \subseteq Y \Rightarrow C(X) \subseteq C(Y)$
3. $CC(X) \subseteq C(X)$
4. $C(X) * C(Y) \subseteq C(X * Y)$

We observe that the crucial property T1 for transitions is exactly what is needed to get property 4 above.

Lemma 2. *C is a closure operation.*

Let M be an andor machine. The phase structure corresponding to M is the IL-algebra given by the transition closure C above. We denote this algebra by ΦM.

5 Finite parts of machines

Each particular computation uses only finite information. This is the extra trick used to get the Trakhtenbrot theorem. Let us introduce machines with bounded registers. Such a machine is given by a number MAX. Assume that register a contains the number N. If N < MAX the instructions a+ and a- behaves as usual. If N = MAX , then a+ gives MAX while a- gives a nondeterministic choice between MAX and MAX-1. If M is the original machine, we let M/MAX be the machine with registers bounded by MAX. This construction works both for register machines and andor machines.

Lemma 3. *For register machines or andor machines. Let C and D be configurations with all registers $\leq$ MAX . Then if M transfers C into D, so does also M/MAX.*

Lemma 4. *For a deterministic register machine M. Let D and E be halting configurations with empty registers. Assume that M transfer C into D, but not E. Then neither does M/MAX for large enough MAX.*

We would like to get lemma 3 for arbitrary machines. We do not need to do this in general. It is only necessary to do this for andor machines derived from deterministic register machines. The andor machines are going to be nondeterministic in general with nondeterminism introduced by the branching instructions in the deterministic register machine. The nondeterminism dervived from the branching

$$p : \text{if } a = 0 \text{ then goto } q \text{ else } a - (r)$$

is

$$\begin{array}{|l} \hline p : or(k,l) \\ k : and(m,q) \\ m : or(n,h) \\ n : b-(m) \\ l : a-(r) \\ \hline \end{array}$$

The branch going down from state k will only be used to get down to state q exactly when register a is empty. The same construction works equally well for an andor machine with bounded registers. Hence

Lemma 5. *Let M be an andor machine derived from a deterministic register machine and let D and E be halting configurations with empty registers. Assume that M transfers C into D, but not E. Then neither does M/MAX for large enough MAX.*

Recursive inseparability 3 *There is an instruction set I and halting configuration D such that for configuration C we cannot recursively separate whether*

- *we can prove in propositional linear logic that C gives D in I*
- *a finite configuration structure gives a countermodel that C→D in I*

The construction of the phase structure ΦM from andor machine M can also be repeated using bounded registers giving $\Phi \mathrm{M}_{MAX}$ for registers bounded by MAX.

6 Exponentials

To interpret exponentials in an IL-algebra we need what Troelstra calls a modality; we then get an IL-algebra with storage ([T] Definition 8.16). A modality is a subset F of the domain X of the IL-algebra satisfying certain conditions (idempotency and ≤ 1) and

$$\{\bot, 1\} \subseteq F \subseteq \{x : x * x = x \wedge x \leq 1\}$$

In all such modalities our transitions are going to be interpreted as 1. Because given an interpetation of !, a transition $C \rightarrow D$ and the phase structure ΦM (or $\Phi \mathrm{M}_{MAX}$), $C \rightarrow D$ is translated into a formula F in the multiplicative additive fragment of linear logic. In the phase structure we have $1 \leq F$, and hence $1 = !F$ for any modality.

Recursive inseparability 4 *It is recursively inseparable whether for formulas F in linear logic*

- *F is derivable in intuitionistic linear logic*
- *A finite phase structure with modality gives a counter example to F*

7 Introducing duality

So far the recursive inseparability is in intuitionistic linear logic. The outline of the argument is as follows. We started with an andor machine with two halting states where we cannot recursively separate whether a particular starting configuration ends in one or the other of the halting states. The computations can be represented as proofs in linear logic or by interpretations in a configuration space with transitions. The configuration space is later taken over to a phase structure. To get recursive inseparability in classical linear logic it is sufficient to show that an IL-algebra can be embedded into a CL-algebra.

We start with an IL-algebra L. Let $L^\perp$ be the dual lattice. We then construct a lattice $LL^\perp$ by using the sum of L and $L^\perp$ – put L and $L^\perp$ between the top and bottom elements $\top$ and $\bot$.The lattice operations in $LL^\perp$ are defined in the obvious way. We have also defined a duality $\bot$. We need to define products in the extended lattice. To do that we divide up into cases depending on whether the elements come from $\top$, L, $L^\perp$ or $\bot$. We define products by the following multiplication table

$\star$	$\top$	A_1	$B_1^\perp$	$\bot$
$\top$	$\top$	$\top$	$\top$	$\bot$
A_0	$\top$	$A_0 \star A_1$	$(A_0 \multimap B_1)^\perp$	$\bot$
$B_0^\perp$	$\top$	$(A_1 \multimap B_0)^\perp$	$\top$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

To prove that $LL^\perp$ is a CL-algebra we note

- $\star$ is commutative
- 1 is a multiplicative unit
- $\star$ is associative
- $X \multimap Y = (X \star Y^\perp)^\perp$
- $X^\perp = X \multimap 0$

Having constructed the embedding of an intuitionistic linear algebra into a classical linear algebra we can conclude

Recursive inseparability 5 *For formulas F in propositional linear logic we cannot recursively separate whether*

- *we can prove F in classical propositional linear logic*
- *a finite CL-algebra with modality gives a counter-model to F*

8 Interpolation

The interpolation theorem is true for classical and intuitionistic linear logic (see Roorda's thesis [R]), and we can show many of the usual properties of the interpolant. In particular, we can conclude that if we have two formulas A, B without any other common propositional symbols than 0 and 1 and if we can prove

$$\vdash A \multimap B$$

then there is an interpolant I built up from the propositional constants alone such that

$$\vdash A \multimap I$$

and

$$\vdash I \multimap B.$$

But the following argument shows that there is no total recursive function giving the interpolant as a function of the two formulas A and B. As in classical predicate logic we need to know the derivation of the formula $A \multimap B$.

Let A and B be the formulas expressing that an andor machine terminates in a halting state a respectively b. We know that there is no recursive function which can decide whether A or B is derivable. Consider now the following implication

$$C = !(!A \multimap \bot) \multimap !B$$

Observe

Lemma 6. *For A B and C as above we have (in both intuitionistic and classical logic) :*

- $\vdash A \Rightarrow \vdash C$
- $\vdash B \Rightarrow \vdash C$

Now let I be an interpolating formula for the implication C. I is built up from the propositional constants alone. We have

$$\vdash !(!A \multimap \bot) \multimap I$$

$$\vdash I \multimap !B$$

We have then

$$\vdash !(!A \multimap \bot) \multimap !I$$

$$\vdash !I \multimap !B$$

Observe that in our models of linear logic with the trivial modality – $\bot$ and 1 – we have

$$\vdash K \Rightarrow \models !K = 1$$

$$\vdash K \multimap L \Rightarrow \models K \leq L$$

Our model can separate between derivations terminating in a and terminating in b. We have

$$\not\vdash A \Rightarrow \models !A = \bot$$

$$\not\vdash B \Rightarrow \models !B = \bot$$

Applying this we get

$$\vdash A \Rightarrow \not\vdash B \rightarrow \models !B = \bot \Rightarrow !I = \bot$$

$$\vdash B \Rightarrow \not\vdash A \Rightarrow \models !A = \bot \Rightarrow !(!A \multimap \bot) = 1 \Rightarrow !I = 1$$

The interpolation formula $!I$ is built up from the propositional constants. The interpretation of $!I$ is given as $\bot$ or 1 depending on whether I as a classical propositional formula has truth value **false** or **true**. So the interpretation of $!I$ is recursively given by I.

Theorem 7. *In (classical or intuitionistic) propositional linear logic there is no totally recursive interpolation function.*

9 References

B : E Börger. *Berechenbarkeit, Komplexität, Logik.* Vieweg Verlag 1985. English translation published by North-Holland.

LMSS : P D Lincoln, J Mitchell, A Scedrov, N Shankar. *Decision problems for propositional linear logic.* In *Proc. 31st Annual IEEE Symposium on Foundations of Computer Science, St Louis, Missouri, October 1990.*

R : D Roorda *Resource Logics: Proof-theoretical Investigations.* Thesis. Amsterdam 1991.

T : A S Troelstra. *Lectures on linear logic.* CSLI Lecture Notes No 29. 1991

The Basic Logic of Proofs

Sergei Artëmov*
Steklov Mathematical Institute,
Vavilov str. 42,
117966 Moscow, Russia.
e-mail: art@log.mian.su

Tyko Straßen†
University of Berne, IAM,
Länggassstr. 51,
CH- 3012 Berne.
e-mail: strassen@iam.unibe.ch

Abstract

Propositional Provability Logic was axiomatized in [5]. This logic describes the behaviour of the arithmetical operator "y is provable". The aim of the current paper is to provide propositional axiomatizations of the predicate "x is a proof of y" by means of modal logic, with the intention of meeting some of the needs of computer science.

1 Introduction

The Propositional Provability Logic GL was axiomatized in [5]. This logic describes the behaviour of the arithmetical operator "y is provable" by means of modal logic. Although some properties of this logic are relevant for computer science (e.g. various forms of Gödel's incompleteness theorem for consistency proofs in databases), GL is rather a mathematical domain. One reason is that in computer science not only the *provability* of a statement is of interest, but also in many cases the *proofs themselves*, respectively informations about the time or memory expenditure for a proof are known. These considerations lead to a different situation. For example it is well-known that a powerful machine cannot prove its own consistency, but it is very well possible for such a machine to demonstrate that a given proof does not derive $0 = 1$, or that no computation within a fixed time comes to that answer. The studies on the Logic of Proofs have been initiated by a series of questions by G. Jäger related to this topic. One was to give an arithmetically complete propositional axiomatization of the predicate "x is a proof of y". The modal systems $\mathcal{P}$ and $\mathcal{PF}$ introduced below solve this problem.

Most definitions in this introduction are in accordance with those of the classical Provability Logic [5]. Nevertheless, the Basic Logic of Proofs is entirely different from the Provability Logic, and the arithmetical completeness proof for it does not use the Solovay argument.

*Supported by the Swiss Nationalfonds (project 21-27878.89) during a stay at the University of Berne in January 1992.

†Financed by the Union Bank of Switzerland (UBS/SBG) and by the Swiss Nationalfonds (projects 21-27878.89 and 20-32705.91).

1.1 Definition The modal language contains two sorts of variables,

$$p_0, p_1, \ldots \text{ (called } \textit{proof variables}\text{)},$$
$$S_0, S_1, \ldots \text{ (called } \textit{sentence variables}\text{)},$$

the usual boolean connectives, truth values $\top$ (for truth) and $\bot$ (for absurdity), and the labeled modality symbol $\Box_{p_i}$ for each proof variable p_i. The modal language is generated from the atoms $\top, \bot, S_0, S_1, \ldots$ by the boolean connective $\rightarrow$ as usual, and by the unary modal operators $\Box_{p_0}(\cdot), \Box_{p_1}(\cdot), \ldots$ as follows: if p is a proof variable and A a modal formula then $\Box_p(A)$ ($\Box_p A$ for short) is a modal formula.

Parentheses are avoided whenever possible by the usual conventions on precedence along with the modal convention that $\Box_{p_i}(\cdot)$ is given the minimal scope. Small letters $p, q, r, \ldots$ are used for proof variables, capital letters $S, T, \ldots$ for sentence variables and $A, B, C, \ldots$ for modal formulas.

The clear intention is to interpret $\Box_p A$ as "p is a proof of A". In order to allow iterations of modalities, which is an essential principle of the Logic of Proofs, the modal language must be interpreted in theories, which are able to link theorems and proofs after some natural coding. These considerations lead to the notion of the *arithmetical interpretation* of the modal language.

1.2 Definition Let the theory T be a recursive extension of $I\Sigma_1$ which is valid in the standard model of arithmetic, for example let T be Peano Arithmetic PA. Greek letters $\varphi, \psi, \ldots$ denote arithmetical formulas. In this paper it will not be distinguished between the number n and its numeral $\overline{n}$.

1.3 Definition An arithmetical formula $Prf(\cdot,\cdot)$ is called a *proof predicate* in T iff

- $Prf(x, y)$ is (provably-in-T equivalent to) a recursive formula in x and y.
- $\mathrm{T} \vdash \varphi \iff \exists n \in \mathbf{N}: \; Prf(n, \ulcorner \varphi \urcorner)$ for all arithmetical formulas φ.

A proof predicate is thus nothing but a recursive enumeration of the theorems of T.

1.4 Example

1. A *standard Gödel proof predicate* for the theory T is a recursive formula $\widetilde{Prf}(\cdot,\cdot)$, such that for every n, m: $\widetilde{Prf}(n, m)$ is true iff n codes a proof in T of the formula coded by m. Examples of such formulas can be found e.g. in [3] or [4]. Note that, according to $\widetilde{Prf}(\cdot,\cdot)$, (a) each proof codes exactly one theorem but each theorem has infinitely many proofs, and, (b) each proof is longer than the theorem proved by it, provided the usual Gödel numbering is used.
2. A modification $Prf_1(\cdot,\cdot)$ of $\widetilde{Prf}(\cdot,\cdot)$ is obtained if one allows not only proofs as first argument, but also programs. Note that property (b) of the first example fails for $Prf_1(\cdot,\cdot)$: $Prf_1(x, y)$ does not imply in general $x \geq y$ because short programs can compute long theorems.

3. In the context of *resource bounded reasoning* one can construct the proof predicate $Prf_2(x,y) := \exists p \leq x : Prf_1(p,y)$, i.e. which is true iff there is a a program of size not greater than x which computes y. Note that also property (a) of the first example fails for $Prf_2(\cdot,\cdot)$: for some x there can be several formulas φ such that $Prf_2(x, \ulcorner\varphi\urcorner)$ holds.

1.5 Definition A proof predicate is called *functional* iff for all $n, k_1, k_2 \in \mathbb{N}$:

$$\text{If } \; Prf(n,k_1) \; \text{ and } \; Prf(n,k_2) \; \text{ then } \; k_1 = k_2.$$

The standard Gödel proof predicate $\widetilde{Prf}(\cdot,\cdot)$ and $Prf_1(\cdot,\cdot)$ from example 1.4 are examples of such functional proof predicates.

1.6 Definition Let $Prf(\cdot,\cdot)$ be a proof predicate in T, and let ϕ be a function that assigns to each proof variable p_i some $n \in \mathbb{N}$ and to each sentence variable S_i a sentence of T. An *arithmetical interpretation* $(\cdot)^*$ is a pair $(Prf(\cdot,\cdot),\phi)$ of such $Prf(\cdot,\cdot)$ and ϕ. The arithmetical interpretation $(A)^*$ (A^* for short) of a modal formula A is the extension of ϕ to all modal formulas by:

- $\top^* := (0=0) \quad \bot^* := (0=1) \quad p_i^* := \phi(p_i) \quad S_i^* := \phi(S_i)$
- $(A \rightarrow B)^* := A^* \rightarrow B^*$
- $(\Box_p A)^* := Prf(p^*, \ulcorner A^* \urcorner)$

1.7 Definition Let $\equiv$ denote the syntactical identity of formulas (e.g. $\varphi \equiv \varphi$, but $\varphi \wedge \varphi \not\equiv \varphi$, $S_0 \not\equiv S_1$ and $\Box_{p_0}\top \not\equiv \Box_{p_1}\top$). An arithmetical interpretation $(\cdot)^* = (Prf(\cdot,\cdot),\phi)$ is called *functional* iff $Prf(\cdot,\cdot)$ is functional and $(\cdot)^*$ is injective, which means that $A^* \equiv B^*$ implies $A \equiv B$. The requirement of injectivity can be eliminated as it is demonstrated in [2].

The Basic Logic of Proofs is not concerned with occasional details about the coding of proofs in T by means of one fixed $Prf(\cdot,\cdot)$. Rather the Basic Logic of Proofs describes those basic principles which are true for *all* proof predicates of a given class.

In this paper the decidable modal logics $\mathcal{P}$ and $\mathcal{PF}$ are introduced, with axioms and rules of inference as follows:

$$\left.\begin{array}{ll} \textbf{(A1)} & \text{All (boolean) tautologies} \\ \textbf{(A2)} & \Box_p A \longrightarrow A \\ \textbf{(R1)} & \dfrac{A \qquad A \rightarrow B}{B} \\ \textbf{(A3)} & \Box_p A \longrightarrow \neg\Box_p B \quad (A \not\equiv B) \end{array}\right\} \mathcal{P} \Bigg\} \mathcal{PF}$$

where A and B are modal formulas and p is a proof variable.
(A2) is the *Reflexivity Axiom* and (A3) the *Functionality Axiom*.

The main result of the paper claims that for each modal formula A:

$$\begin{array}{rcl} \mathcal{P} \vdash A & \Longleftrightarrow & A^* \text{ is true for every interpretation } (\cdot)^* \\ \mathcal{PF} \vdash A & \Longleftrightarrow & A^* \text{ is true for every functional interpretation } (\cdot)^* \end{array}$$

Moreover, for each of these completeness theorems a proof predicate $Prf(\cdot,\cdot)$ can uniformly be chosen.

It is easy to see that neither $\mathcal{P}$ nor $\mathcal{PF}$ is closed under the necessitation $A \vdash \Box_p A$, or under the substitution rule $A \leftrightarrow B \vdash \Box_p A \leftrightarrow \Box_p B$.

The standard Gödel proof predicate $\widetilde{Prf}(\cdot,\cdot)$ enjoys an additional property, namely to be monotonic:

1.8 Definition A proof predicate is called *monotone* iff $Prf(n,k)$ implies $n \geq k$ for all $n, k \in \mathbb{N}$. An arithmetical interpretation $(\cdot)^* = (Prf(\cdot,\cdot), \phi)$ is called *monotone* iff $Prf(\cdot,\cdot)$ is a monotonic proof predicate.

A logic $\mathcal{PFM}$ containing all those formulas which are true under all interpretations based on any fixed functional and monotonic proof predicate has been established in [1], too. Thereby, $\mathcal{PFM}$ is the logic of the standard Gödel proof predicate $\widetilde{Prf}(\cdot,\cdot)$. $\mathcal{PFM}$ will not be presented in this paper.

In section 2 the soundness and completeness for $\mathcal{P}$ is proved, in section 3 the same is done for $\mathcal{PF}$ and section 4 is devoted to uniform proof predicates.

2 Arithmetical soundness and completeness of $\mathcal{P}$

The aim of this section is to prove soundness and completeness of the modal system $\mathcal{P}$ with respect to arithmetical interpretations.

2.1 Theorem Let A be a modal formula. Then

$$\begin{array}{rcll} \mathcal{P} \vdash A & \Longleftrightarrow & \forall^* : & \mathrm{T} \vdash A^* \\ & \Longleftrightarrow & \forall^* : & A^* \text{ is true} \end{array}$$

Proof of the soundness of $\mathcal{P}$: Let $(\cdot)^*$ be some arithmetical interpretation. One has to show that $\mathrm{T} \vdash A^*$. Induction on the complexity of the proof of A:

(A1) and (R1) straightforward.

(A2) 1st case: $\mathrm{T} \vdash Prf(p^*, \ulcorner A^* \urcorner)$. It follows that $\mathrm{T} \vdash A^*$,
hence $\mathrm{T} \vdash Prf(p^*, \ulcorner A^* \urcorner) \to A^*$.
2nd case: $\mathrm{T} \nvdash Prf(p^*, \ulcorner A^* \urcorner)$. As $Prf(\cdot,\cdot)$ is recursive $\mathrm{T} \vdash \neg Prf(p^*, \ulcorner A^* \urcorner)$,
hence $\mathrm{T} \vdash Prf(p^*, \ulcorner A^* \urcorner) \to A^*$.

■

The next task is to prove the arithmetical completeness of $\mathcal{P}$.