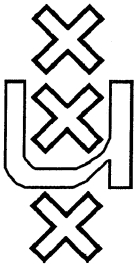
Institute for Language, Logic and Information

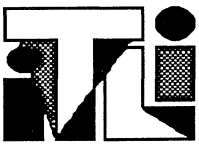
NONDETERMINISM, FAIRNESS AND A FUNDAMENTAL ANALOGY

Edith Spaan
Leen Torenvliet
Peter van Emde Boas

ITLI Prepublication Series
For Computation and Complexity Theory CT-88-10



University of Amsterdam



Institute for Language, Logic and Information
Instituut voor Taal, Logica en Informatie

NONDETERMINISM, FAIRNESS AND A FUNDAMENTAL ANALOGY

Edith Spaan
Leen Torenvliet
Peter van Emde Boas
Department of Mathematics and Computer Science, University of Amsterdam

Received December 1988

Correspondence to:

Faculteit der Wiskunde en Informatica
(Department of Mathematics and Computer Science)
Roetersstraat 15
1018WB Amsterdam

or

Faculteit der Wijsbegeerte
(Department of Philosophy)
Grimburgwal 10
1012GA Amsterdam

Nondeterminism, Fairness and a Fundamental Analogy

Edith Spaan, Leen Torenvliet and Peter van Emde Boas[1]

Departments of Mathematics and Computer Science, University of Amsterdam,
Nieuwe Achtergracht 166, 1018 WV Amsterdam

Abstract

In this note we propose a model for unbounded nondeterministic computation which provides a very natural basis for the structural analogy between recursive function theory and computational complexity theory: $P : NP \cong REC : RE$ At the same time this model presents an alternative version of the halting problem which has been known for a decade to be highly intractable.

[1]also CWI-AP6, Amsterdam

1 Introduction

Structural complexity theory is often presented as the theory in which the results obtained for classes of languages recognized by Turing machines are transferred to a resource bounded setting. Notions like reduction, simplicity, immunity, the arithmetical hierarchy, relativizations etc. were all first defined in recursive function theory and later (relativizations of) these notions were introduced in complexity theory.

All of this work was inspired by the frustration originating from the difficulty of the fundamental problem in computational complexity theory which has become known as the $P \stackrel{?}{=} NP$ problem. It is now an 18 year old unresolved problem whether the class of languages recognized by nondeterministic Turing machines in polynomially bounded time is equal to the class of languages recognized by deterministic Turing machines in polynomial time.

The applications of ideas from the theory of recursive functions in this area originates from a formal analogy which might be expressed by the formula: $P : NP \cong REC : RE$ where the symbols REC and RE denote the class of recursive and recursively enumerable sets respectively.

In the theory of recursive functions life is easy. It is known since the days this theory was created that $REC \neq RE$. The *Halting problem* ($HALT$) is the best known example of a set in RE which is not a member of REC.

On the other hand it is not clear that this separation has anything to do with the $P \stackrel{?}{=} NP$ problem at all. In recursion theory the classes REC and RE are introduced by specifying a mode of accepting sets and not on the distinction between determinism and nondeterminism. Sets in REC are specified by machines which always terminate and which recognize their input by final state. Sets in RE are specified by machines which do not necessarily terminate and which accept their input by halting.

In automata theory it is shown that for both classes the choice of machine model is irrelevant; moreover it does not make a difference whether the machines are deterministic or nondeterministic. The situation is similar to the world of finite automata where deterministic and nondeterministic machines determine the same family of sets: *the regular sets.* Note that for the remaining classes in the Chomsky hierarchy the situation is different. For the PDA's it is known that the nondeterministic machines determine a larger class than the deterministic ones. For the remaining class, the LBA's, the problem is still unsolved, as for the polynomially time bounded machines (but in this case we have recently learned that at least the nondeterministic class is closed under complementation [11] [17].)

The present note arose out of a discussion between two of the authors provoked by the self evident observation uttered by one of us that in the world of unbounded computation nondeterministic devices are more powerful than deterministic ones as exemplified by the inequality $REC \neq RE$. According to the argument the nondeterministic devices could guess and verify the halting computations which a deterministic device cannot produce.

As indicated above this argument must be false in traditional recursion theory since the conclusion is incorrect. On the other hand, if it were correct, it would provide a very natural basis for the analogy between P and NP in the resource bounded case and REC and RE in the unbounded case which has inspired the subject of structural complexity theory from its earliest start. It would therefore be nice if the incorrect argument above

could be made valid. In this note we will propose a new interpretation for the sets accepted by nondeterministic devices which achieves this purpose. This interpretation will provide the fundamental analogy between complexity theory and recursion theory with its most natural basis: both sides of the equation $P : NP \cong REC : RE$ invoke the difference between determinism and nondeterminism.

2 Origin of the nondeterministic Turing Machine

When examining standard textbooks on recursive function theory like [14] [15] the reader will find that in traditional recursive function theory there *is no such thing* as a nondeterministic Turing Machine. The analogy of the the $P \stackrel{?}{=} NP$ problem does not exist in recursive function theory. Rogers [14] even *requires* the Turing machine to be deterministic:

> ...Finally, the device is so constructed that it behaves according to a finite list of deterministic rules...(p 13)

It is an interesting problem to find out where and when the concept of a nondeterministic Turing machine was introduced in computation theory. The possibility of having a program containing more than one instruction for a given state/symbol pair is considered in Turing's original paper [18], but he seems to consider this possibility to be an abberation of the notion of computability, where the resulting ambiguity has to be resolved by an external operator.

For the case of finite automata the introduction of the nondeterministic variant as we know it today is claimed by Rabin and Scott [13]. Information concerning the scientific climate during these years when automata theory was created can be found in the historic survey written by Greibach [9]. According to Greibach the Rabin and Scott paper was preceded by a paper by Chomsky and Miller [5] which contains the nondeterministic finite automaton in a rather hidden way. Introduction of nondeterminism was crucial for the machine based characterization of the two middle levels of the Chomsky hierarchy. Compare the Kuroda paper [12] which brings the problem of characterizing the context sensitive languages to an end: with hindsight one concludes that there really was no problem if only the people had thought nondeterministically ....

It seems therefore that the notion of a nondeterministic machine was introduced not before the end of the 50's (see the time table in [3]) and we strongly suspect that the contemporary proliferation of the use of the nondeterministic model is inspired by automata theory and computational complexity theory rather than by recursive function theory.

Even today, with an experience of several decades of thinking nondeterministically, it is still quite common to develop recursive function theory on the basis of deterministic devices only. The reason for not considering nondeterministic machines in the unbounded case is of course the fact that in this case a deterministic Turing machine can recompute the entire computation tree of a nondeterministic machine in a breadth first way (See exercise 7.3 in [10]).

With respect to the fundamental analogy between recursive function theory and structural complexity theory mentioned in the introduction this state of affairs is less fortunate. As the general feeling is that, if the $P \stackrel{?}{=} NP$ will be resolved at all, then it will be in the

direction of $P \neq NP$, such a simulation of nondeterministic computation is not very agreeable in a world in which we like computational complexity theory to behave as Lilliput [16] in the world of recursive function theory.

Given the difficulty of solving the $P \stackrel{?}{=} NP$ problem we have considered to modify the realm of recursive function theory instead. We propose in the following section the adoption of an alternative acceptation convention for the nondeterministic version of the Turing machine model for recursive function theory such that the difference we suspect between determinism and nondeterminism in complexity theory can easily be established in the unbounded case.

3 The Model

The dilemma mentioned above finds its roots in the acceptation convention for Turing machines. If we allow infinite computations, and say that a Turing machine rejects its input in case of an infinite computation, a deterministic Turing Machine can simply unravel the computation tree of a nondeterministic Turing Machine in a breadth first manner. If the computation tree has a finite accepting branch, then the deterministic machine will find this branch in finite time. If the branch does not exist the breadth first search simulation will diverge and consequently the machine will not accept its input.

If we don't allow for infinite computations and recognize by accepting state then the deterministic machine has even less trouble. It can simulate the nondeterministic computation in a depth first way and find out if there's an accepting computation. If there does not exist an accepting branch the machine will sooner or later discover that it has explored all branches in the tree without finding an accepting one. Due to the bounded nondeterminism of the Turing Machine model we can infer from König's lemma that the existence of arbitrarily large branches entails the existence of an infinite branch as well, and this is not allowed for a halting nondeterministic device.

The way out of this dilemma is both elegant and obvious. Of course we don't allow infinite computations. Hence the deterministic machine has to accept or reject its input in finite time. The nondeterministic machine has to say yes or no to its input in finite time also. However, in order to exploit the power of nondeterminism we give the nondeterministic machine the choice between a countable number of computations. The acceptation convention for nondeterministic machines is then standard. The machine accepts if and only if an accepting computation on the input exists.

Precisely, the acceptation condition is defined as follows:

Definition 1 a Turing Machine M is said to *accept* the language $L \subseteq \Sigma^*$ iff:

- its computation on any input $\sigma \in \Sigma^*$ is finite
- there exists a computation of M on input σ that halts in an accepting state iff $\sigma \in L$

And we achieve unbounded nondeterminism within these limits by:

Definition 2 A computation of a (nondeterministic) Turing Machine is called *unfair* iff it holds for an (infinite) suffix of this computation that if the machine is in configuration δ

infinitely often, and $\{\delta \mapsto \delta_1, \delta \mapsto \delta_2\}$ are elements of the transition relation then $\delta \mapsto \delta_1$ is never chosen in the computation.

A (nondeterministic) Turing machine is called *fair* if it produces no unfair computations.

The above notion of fairness originates from the theory of concurrent processes; see for example [1] [8]. In fact there exist a number of different fairness notions in the literature. The notion we use is called *strong fairness* in [1]. The guarded loop programs considered in that paper can be related to our Turing machine programs by considering a Turing program to be represented by a single guarded loop where for every quintuple in the program a branch is provided which is guarded by the corresponding state/symbol pair. Such a loop terminates when all guards evaluate to false which is precisely the case if for the current state/symbol pair no instruction is provided in the program. It is easy to see that under this translation our fairness notion does correspond with strong fairness in [1].

It is known that the notion of fairness has to be enforced from *outside the model.* There is nothing within the Turing Machine program which prevents the machine from performing an infinite unfair computation. The fairness notion is invoked in order to exclude some infinite computations, and in this way, by clever programming, forcing in fact all computations to become finite.

The traditional example illustrating this mechanism is Dijkstra's random number generator [6]. This machine initializes a counter at zero and next performs a loop where either the counter is incremented or a termination flag is set to true. The second choice will make the machine halt on the next move. By requiring the machine to be fair the unique infinite computation which increments the counter forever is excluded.

Without loss of generality we can assume that our nondeterministic machines perform nondeterministic moves at every step, and that the number of possible successor configuration is bounded by two (deterministic moves are incorporated by making a choice between two copies of the same transition).

We can moreover construct a universal machine both for the deterministic and nondeterministic version of the machines, which will simulate all other machines. In the nondeterministic case the fair computations of the universal machine correspond to the fair computations of the simulated devices. The latter result is less evident than it seems to be: it requires some mechanism by which the universal machine can ensure fairness with respect to a finite but unbounded number of state/symbol pairs given just the fairness with respect to the state/symbol pairs of the universal machine itself. As will be indicated in the sequel the theory presented in [1] suffices for performing such a construction.

Now enter a well-known language.

Definition 3 For a suitable universal Turing machine U, the language $HALT \subseteq \Sigma^* = \{\sigma \in \Sigma^* | \sigma$ when interpreted as a deterministic TM program by U halts on input σ when interpreted as a string$\}$

The difference between determinism and nondeterminism now is easily established.

Theorem 1 *There exists a fair nondeterministic Turing machine which accepts HALT by the acceptation convention given in definition 3*

Proof: The nondeterministic machine M has a single nondeterministic transition and works as follows. It enters state δ. From δ it continues nondeterministically in one of two

successor states: either it writes a 0 on its worktape or otherwise its starts to simulate. If the machine chooses to start simulating it will simulate the input when interpreted as a program on itself for a number of steps which is bounded by the number of symbols written previously on the worktape. If the simulated machine halts and accepts within this number of steps then M halts and accepts also, else M halts and rejects.

It is easily seen that an accepting computation of M can exist if and only if the input is an element of $HALT$. Otherwise all possible computations are rejecting.

As M is a fair machine however, it *must* at some time take the second possibility of the nondeterministic choices and enter a finite computation. Hence all possible computations of M are finite. □

So there is a fair nondeterministic machine which accepts $HALT$ by the acceptation conventions given in definition 3.

By simple diagonalization (just as in any textbook) it is shown that by the same acceptation convention no deterministic machine accepting $HALT$ can exist. A claimed deterministic M_j device that accepts $HALT$ is converted to a machine $M_{\tau(j)}$ which loops if and only if M_j accepts its input. Now consider the behavior of $M_{\tau(j)}$ on input $M_{\tau(j)}$: If $M_{\tau(j)}$ stops on input $M_{\tau(j)}$ then M_j would accept $M_{\tau(j)}$, hence $M_{\tau(j)}$ would diverge. On the other hand if $M_{\tau(j)}$ diverges on input $M_{\tau(j)}$ then M_j would reject in which case $M_{\tau(j)}$ stops (and rejects). So we have constructed a machine which stops if and only if it diverges, a contradiction.

Theorem 1 states—since $HALT$ is $\leq_m$ complete for RE—that for any language $L \in RE$ there exists a fair nondeterministic machine that accepts L. On the other hand:

Theorem 2 *If L is accepted by a fair nondeterministic Turing machine, then $L \in RE$.*

Proof: Suppose $L = L(M)$ then M is a fair nondeterministic Turing machine that halts on any $\sigma \in \Sigma^*$. Moreover M has an accepting computation on input σ if and only if $\sigma \in L$. A deterministic machine trying to find out if $\sigma \in L$ does not have to decide if all fair computations of M on σ stop. They do so *by definition* hence it suffices to explore M's computation tree (breadth first) and find out if there is an accepting computation. If such a computation exists then M accepts, $\sigma \in L$ and σ can be reported as a member of L. Thus a deterministic machine can for an ever growing number of steps explore M's computation tree for an ever growing number of σ and report those σ for which it finds an accepting computation. □

4 Halting behavior of fair nondeterministic machines

It is not hard to generalize the above construction for the Halting problem for the nondeterministic machines; rather than writing the length of the computation to be simulated in unary the machine accepting the halting problem will write a trace for the nondeterministic choices on its worktape. Simulation of the machine will consume this trace in no more step than its length since at every step of the simulated machine a nondeterministic choice is presumed.

However, by looking back at this generalization a curious problem is uncovered. The universal simulator invoked in this construction performs much better than the original nondeterministic devices simulated. The fairness condition will force all computations into termination. In doing so the simulator will never generate an accepting branch which does not belong to the original device but the nonterminating behavior of the machines has vanished into thin air.

Since our fairness condition does not enforce all computations to become terminating it becomes necessary to specify what it means that a nondeterministic machine halts on some input.

Definition 4 The fair nondeterministic machine M_i halts on input σ provided all fair computations of M_i on input σ terminate.

And an analogy to *HALT*:

Definition 5 For a suitable Universal Turing Machine U we define the language $HALTN \subseteq \Sigma^* = \{\sigma \in \Sigma^* | \sigma$ when interpreted as a nondeterministic TM program by U halts on input σ when interpreted as a string$\}$

This immediately invokes the problem of determining whether we have introduced an animal that can eat itself tail-first. In other words can there be a fair nondeterministic device accepting *HALTN*? An analogous, but unfortunately somewhat less transparent diagonal argument shows this is not the case.

Theorem 3 *There's no fair nondeterministic Turing machine that accepts HALTN.*

Proof: Suppose there exists such a machine M_j. Then on input $\sigma \in \Sigma^*$ all (fair) computations of M_j terminate, and moreover M_j has an accepting computation if and only if all fair computations of σ on input σ terminate. WLOG M_j has only one accepting state. Now we can construct from M_j a fair nondeterministic machine $M_{\tau(j)}$ by copying M_j's program, but replacing the accepting state by a (deterministic) loop. Let us examine the behavior of $M_{\tau(j)}$ on input $M_{\tau(j)}$.

If all fair computations terminate, then M_j would have accepted the input $M_{\tau(j)}$. Hence $M_{\tau(j)}$ has a computation which enters the former accepting state and then loops deterministically. Since this computation makes only a finite number of nondeterministic moves, it is a fair infinite computation. On the other hand if there exists an infinite fair computation of $M_{\tau(j)}$ on input $M_{\tau(j)}$ then M_j rejects input $M_{\tau(j)}$ this means by definition that all fair computations of M_j on input $M_{\tau(j)}$ are finite but none of these computations ends in the accepting state. But that means that the only state in which $M_{\tau(j)}$ can diverge fairly can never be reached, whence all fair computations of $M_{\tau(j)}$ on input $M_{\tau(j)}$ are finite. The conclusion is that $M_{\tau(j)}$ has a fair diverging computation on input $M_{\tau(j)}$ if and only if all fair computations on this input are finite, a contradiction. □

5 Scheduling for an upper bound

So far our new model has yielded results which fit nicely into recursive function theory as we know it. The deterministic version of the halting problem can't be solved by deterministic machines whereas it can be solved by their nondeterministic sisters. These

nondeterministic devices have their own version of a halting problem *HALTN* which can't be solved by these nondeterministic devices. So there remains the question: How big then is the problem *HALTN* ? It would fit nicely in the analogy if the problem *HALTN* could be shown to be complete for the class Π_2^0. It is not difficult to invent a reduction from the problem *TOTAL*, known to be complete for Π_2^0 to *HALTN* but a corresponding upper bound can not be found. In fact it turns out that the answer to the above question has been known in the literature for a decade: *HALTN* is complete for the class Π_1^1 [2] [4] [7]. This result illustrates the true complexity of deciding the halting behavior of fair nondeterministic computations.

The striking difference with the standard model can be explained as follows. In the traditional interpretation it follows by Königs lemma that the fact that all possible computations of a nondeterministic machine halt entails the existence of a finite upper bound for the length of these computations. Therefore the nondeterministic version of the halting problem is in Σ_1^0 and in fact Σ_1^0 complete. In our new interpretation we only have that all *fair* nondeterministic computations must end, so the computation tree may contain unfair infinite branches whence Königs lemma is no longer applicable.

Using a lemma from [1] (lemma 3, as extended on page 85/86) we first sketch a proof for a Π_1^1 upper bound for the problem *HALTN*.

In order to understand the argument some explanation about this lemma is required. We already indicated that our Turing programs could be represented in the guarded command language of [1] in such a way that our fairness notion corresponds to strong fairness in the paper. Lemma 3 states that there exists a systematic transformation T_{Strong} of guarded loop programs S into deterministic programs $T_{Strong}(S)$ where the program S is extended with random assignments of the form $z :=?$, such that removing the random assignments from a computation of $T_{Strong}(S)$ will produce a fair computation of S and such that all fair computations of S can be obtained in this way. The random assignments are used in order to generate input to a *deterministic scheduler* which will guide the computation of the original program along a fair course.

For our application to Turing machines this lemma entails that it suffices to consider programs with a single nondeterministic choice which is subjected to a fairness condition. It suffices to have a subroutine which generates a random integer and which is guaranteed to terminate. Dijkstra's random number generator presents us such a subroutine. Given the subroutine the Apt/Olderog scheduler does the rest of the job.

A consequence is the observation that *HALTN* belongs to Π_1^1. All fair computations of the original program terminate provided all computations of the transformed program terminate regardless the output of the random number generator. This output can be represented by an infinite sequence of integers ω. The halting behavior now can be expressed by a sentence *For every sequence ω there exists a number of steps k after which the machine, when scheduled according to ω must halt.* Clearly this is a Π_1^1-sentence.

By this reduction we also can explain the lacking argument for the construction of the universal machine for fair nondeterministic Turing machines. By use of a scheduler the universal machine can live with a single nondeterministic choice in its random number generator. The rest of its simulations are performed deterministically.

The Π_1^1 lower bound has been obtained originally by Chandra [4], but a very simple proof is given by Apt and Plotkin [2]. They provide a reduction from the problem *WFTREE* defined to be the set of (indices) of total characteristic functions of well-founded

infinitely branching trees, which is a well known Π_1^1-complete set.

The authors consider a transformation $T(i)$ which transforms a program M_i into a program $M_{T(i)}$ which operates as follows: First $M_{T(i)}$ generates by a random assignment $x :=?$ an argument and evaluates $M_i(x)$; when this evaluation has terminated the program generates, using random assignments a branch $<>$, $< n_1 >$, $< n_1, n_2 >$, ... , subjecting every node encountered as input to M_i and terminates when it locates a node which is rejected by M_i.

The transformed program may diverge during the first stage in case M_i does not compute a total function. So the termination during the first stage tests for the Π_2^0-complete property *TOTAL*. During the second stage the transformed program may moreover diverge by traversing an infinite branch in the tree, which will not exist if the tree is well-founded. So overall termination of $M_{T(i)}$ will test for the Π_1^1-complete property *WFTREE*. Again the lemma from Apt and Olderog will provide us the equivalence between halting of programs with random assignment and fair termination of finitely branching nondeterministic programs.

6 Conclusions

In this note we have, albeit by artificial means, established a model for recursion theory in which determinism differs from nondetermism. Surprisingly, the fairness notion known from concurrency theory had to be dragged in. Our fair nondeterministic machines have sufficient power for recognizing sets in *RE* by final states and, as we have seen in the previous section, sets in arbitraryly high levels in the arithmetic hierarchy and beyond there by halting.

Does the above hardness result mean that our new interpretation of nondeterminism has overshoot its target ? It is just a question whether the halting problem as considered (all fair computations must terminate) is felt to be relevant or not. In the traditional interpretation this problem turns out to be reducible to the usual version (there exists at least one terminating computation) and that version has not become harder for our new model. But Chandra [4] has made a strong case for considering the halting behavior of a program to be an intrinsic part of its ontology.

It is obvious that fairness is a serious restriction on nondeterminism in the sense that it can really prevent latent diverging computations by forcing them to abort. In our application the essential ingredient is the use of fairness for guessing the length of an accepting computation without risking infinite computations. Random assignment would have achieved the same purpose (but random assignment does not belong in the world of Turing Machine computations—it is more at home in the world of the RAM). And, as we have seen, random assignment and fairness represent two sides of the same medal.

Lately it has become popular to inflict all kinds of restrictions on nondeterminism in the polynomial time bounded case, thus introducing classes like *UP*, *FEWP*, *MAJORITY*, and even classes of nondeterministic programs governed by threshold functions. Of course if $P = NP$ then all these classes collapse to P. But if not, what are then precisely the restrictions on nondeterminism by saying that a nondeterministic option may not be ignored for more than a constant or logarithmic number of times? I.e., what about fair nondeterministic computation in Lilliput?

Bibliography

[1] Apt, K.R. and Olderog, E.R., *Proof rules and transformations dealing with fairness*, Sci. Comput. Progr. 3 (1983) 65-100

[2] Apt, K.R. and Plotkin, G.D., *Countable nondeterminism and random assignment*, J. Assoc. Comput. Mach. 33 (1986) 724-767

[3] Böhling, K.H. and Indermark, K., *Endliche Automaten I* Bibliographisches Institut (1969)

[4] Chandra A.K., *Computable nondeterministic functions*, FOCS 19 (1978) 127-131.

[5] Chomsky, N. and Miller, G.A., *Finite state languages*, Inf. and Control 1 (1958) 91-112

[6] Dijkstra, E.W., *A Discipline of Programming*, Prentice Hall Series in Automatic Computation 1976.

[7] Emerson, E.A. and Clarke, E.M., *Characterizing correctness properties of parallel programs using fixed points*, Proc. ICALP 7 (1980), Springer LCS 85, pp. 169-181

[8] Francez, N. *Fairness*, Springer (1986)

[9] Greibach, S.A., *Formal languages: origin and directions*, Ann. Hist. Comp. 3 (1981) 14-41

[10] Hopcroft, J.E. and Ullman, J.D., *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley (1979)

[11] Immerman, N., *Nondeterministic space is closed under complementation*, SICOMP 17 (1988) 935-938

[12] Kuroda, S.-Y., *Classes of languages and linear-bounded automata*, Inf. and Control 7 (1964) 207-223

[13] Rabin, M.O. and Scott, D., *finite automata and their decision problems*, IBM Journal 3 (1959) 114-125

[14] Rogers, H. Jr, *Theory of Recursive Functions and Effective Computability*, Mc Graw-Hill Series in Higher Mathematics (1967)

[15] Soare, R.I., *Recursively Enumerable Sets and Degrees*, Springer Verlag (1987)

[16] Swift J., *Gulliver's Travels*, Penguin Books (1967)

[17] Szelepcsényi, R., *The method of forcing for nondeterministic automata*, Bull. EATCS 33 (1987) 96-100

[18] Turing, A.M., *On computable numbers, with an application to the entscheidungsproblem*, Proc. London Math. Soc. ser. 2, 42 (1936) 230-265

The ITLI Prepublication Series

1986

86-01	The Institute of Language, Logic and Information
86-02 Peter van Emde Boas	A Semantical Model for Integration and Modularization of Rules
86-03 Johan van Benthem	Categorial Grammar and Lambda Calculus
86-04 Reinhard Muskens	A Relational Formulation of the Theory of Types
86-05 Kenneth A. Bowen, Dick de Jongh	Some Complete Logics for Branched Time, Part I Well-founded Time, Forward looking Operators
86-06 Johan van Benthem	Logical Syntax

1987

87-01 Jeroen Groenendijk, Martin Stokhof	Type shifting Rules and the Semantics of Interrogatives
87-02 Renate Bartsch	Frame Representations and Discourse Representations
87-03 Jan Willem Klop, Roel de Vrijer	Unique Normal Forms for Lambda Calculus with Surjective Pairing
87-04 Johan van Benthem	Polyadic quantifiers
87-05 Víctor Sánchez Valencia	Traditional Logicians and de Morgan's Example
87-06 Eleonore Oversteegen	Temporal Adverbials in the Two Track Theory of Time
87-07 Johan van Benthem	Categorial Grammar and Type Theory
87-08 Renate Bartsch	The Construction of Properties under Perspectives
87-09 Herman Hendriks	Type Change in Semantics: The Scope of Quantification and Coordination

1988

Logic, Semantics and Philosophy of Language:

LP-88-01 Michiel van Lambalgen	Algorithmic Information Theory
LP-88-02 Yde Venema	Expressiveness and Completeness of an Interval Tense Logic
LP-88-03	Year Report 1987
LP-88-04 Reinhard Muskens	Going partial in Montague Grammar
LP-88-05 Johan van Benthem	Logical Constants across Varying Types
LP-88-06 Johan van Benthem	Semantic Parallels in Natural Language and Computation
LP-88-07 Renate Bartsch	Tenses, Aspects, and their Scopes in Discourse
LP-88-08 Jeroen Groenendijk, Martin Stokhof	Context and Information in Dynamic Semantics
LP-88-09 Theo M.V. Janssen	A mathematical model for the CAT framework of Eurotra

Mathematical Logic and Foundations:

ML-88-01 Jaap van Oosten	Lifschitz' Realizabiility
ML-88-02 M.D.G. Swaen	The Arithmetical Fragment of Martin Löf's Type Theories with weak Σ-elimination
ML-88-03 Dick de Jongh, Frank Veltman	Provability Logics for Relative Interpretability
ML-88-04 A.S. Troelstra	On the Early History of Intuitionistic Logic
ML-88-05 A.S. Troelstra	Remarks on Intuitionism and the Philosophy of Mathematics

Computation and Complexity Theory:

CT-88-01 Ming Li, Paul M.B.Vitanyi	Two Decades of Applied Kolmogorov Complexity
CT-88-02 Michiel H.M. Smid	General Lower Bounds for the Partitioning of Range Trees
CT-88-03 Michiel H.M. Smid, Mark H. Overmars Leen Torenvliet, Peter van Emde Boas	Maintaining Multiple Representations of Dynamic Data Structures
CT-88-04 Dick de Jongh, Lex Hendriks Gerard R. Renardel de Lavalette	Computations in Fragments of Intuitionistic Propositional Logic
CT-88-05 Peter van Emde Boas	Machine Models and Simulations (revised version)
CT-88-06 Michiel H.M. Smid	A Data Structure for the Union-find Problem having good Single-Operation Complexity
CT-88-07 Johan van Benthem	Time, Logic and Computation
CT-88-08 Michiel H.M. Smid, Mark H. Overmars Leen Torenvliet, Peter van Emde Boas	Multiple Representations of Dynamic Data Structures
CT-88-09 Theo M.V. Janssen	Towards a Universal Parsing Algorithm for Functional Grammar
CT-88-10 Edith Spaan, Leen Torenvliet Peter van Emde Boas	Nondeterminism, Fairness and a Fundamental Analogy
CT-88-11 Sieger van Denneheuvel Peter van Emde Boas	Towards implementing RL

Other prepublications:

X-88-01 Marc Jumelet	On Solovay's Completeness Theorem