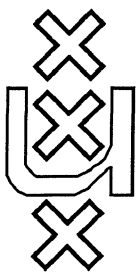
Institute for Language, Logic and Information

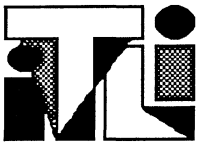
TOWARDS A UNIVERSAL PARSING ALGORITHM FOR FUNCTIONAL GRAMMAR

Theo M.V. Janssen

ITLI Prepublication Series
For Computation and Complexity Theory CT-88-09



University of Amsterdam



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

TOWARDS A UNIVERSAL PARSING ALGORITHM FOR FUNCTIONAL GRAMMAR

Theo M.V. Janssen
Department of Mathematics and Computer Science, Amsterdam

Received November 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

TOWARDS A UNIVERSAL PARSING ALGORITHM FOR FUNCTIONAL GRAMMAR

Theo M.V. Janssen
Depts. of Mathematics and Computer Science
University of Amsterdam

Abstract

A parsing algorithm is developed for Functional Grammar; it assigns to natural language sentences their underlying FG predications. The algorithm is based on two restrictions on rules in Functional Grammar, viz. 'invertibility' and 'increasing complexity', and it is argued that these restrictions follow from principles of FG theory. The algorithm is not designed for a particular grammar, but can be used for any Functional Grammar satisfying the restrictions.

To be published in:

J. Connally & S.C. Dik, eds., 'Functional Grammar and the Computer', Foris, Dordrecht.

TOWARDS A UNIVERSAL PARSING ALGORITHM FOR FUNCTIONAL GRAMMAR

Theo M.V. Janssen
Depts. of Mathematics and Computer Science
University of Amsterdam

1. Functional Grammar and Parsing

In Functional Grammar (henceforth FG) a central role is played by *predications*. An example of such a predication is:

(1) $eat_V(d1x_i : John(x_i))_{Ag}\ (i1x_j : sandwich(x_j))_{Go}$

Expression rules transform such predications into sentences. Predication (1) is expressed as sentence (2).

(2) *John eats a sandwich*

In the present paper the issue of parsing a functional grammar is considered; by this is understood the problem of finding the predications underlying a given sentence. The aim of the investigations is to develop a general parsing algorithm for FG.

Several approaches could be followed in order to obtain a parser for FG. One might take a specific FG grammar and develop somehow a parser for that grammar. As a next step one might try to generalize the program for other grammars. This approach has the advantage that one could design a fast algorithm that cleverly uses specific features of the given grammar, but probably each proposed grammar would require its own generalization. Another approach might be to take a wellknown parsing algorithm from computer science, and try to fit FG grammars into such an algorithm. This probably has the advantage that one obtains an efficient algorithm, but it is likely that not everything FG grammarians would like to express, will be allowed for by such an algorithm. Since such an algorithm is defined in another framework than FG, the results have to be reinterpreted in an FG fashion (especially when the parser is used for grammar testing).

The approach I will advocate here is to base the parser on the theory of functional grammar, especially on what the theory says about expression rules. This has the advantage that we obtain a parser which is generally applicable to FG grammars, and does not require reformulation of the rules into another framework. The consequence of this approach is that we cannot use specific features of a given grammar and will possibly not get the most efficient parser for that grammar. In the sequel I mainly ignore efficiency aspects, but in section 6 they will be considered briefly.

The basic idea of the algorithm is to invert the proces performed by the expression rules. This idea is due to Landsbergen, who developed a parser for Montague grammar that is based upon such an approach; more information will be given in section 6.

In order to turn that idea of inverting rules into a working algorithm, the FG rules have to obey two restrictions. There are, however, reasons for expecting that in practice all functional

grammars will obey the two restrictions: it will be argued that the restrictions follow in a certain sense from the FG theory.

Let me now describe briefly what FG theory says about expression rules. They are rules that describe (Dik 1978, p.20)

1. the forms by which the terms of a given predication are realized;
2. the form in which the predicate itself is realized;
3. the order of the constituents (the order of the realizations of terms and predicates);
4. stress assignment and intonation.

There are some general principles of functional grammar which say what expression rules should never do (Dik 1978, p.10-12)

1. no transformation: the rules may not perform deletions, substitutions or permutations of constituents;
2. no filtering: all structures formed by the rules are well formed, the rules may not reject structures;
3. no lexical analysis: the basic items in the grammar are not abstract entities, but entities that correspond to words in the language.

These FG principles will be appealed to in sections 3 and 4 where the restrictions on FG are described that make the parsing algorithm possible. Since the algorithm is a general algorithm, it is formulated in a rather abstract terminology. Therefore, I will not start with a description, but first present in the next section a very simple example that illustrates the idea underlying the algorithm. The description itself will be given in section 6.

2. A Simple Example

The parsing algorithm consists in inverting the generative process performed by the expression rules. Let us first consider this generative process. We will investigate the way from (3) to (4), following roughly the rules presented in Dik (1980). The treatment of the example is somewhat simplified.

(3) $walk_V(\text{every } x_i\text{: man}(x_i))_{Ag\ Subj}$

(4) *Every man walks.*

The steps in the process of expressing (3) will be labelled E1, E2, etc. The first stage in expressing (3) is the formation of the expressions corresponding with its constituents.

E 1. Expressing $walk_V$. This yields *walks.*

Next we express (every : man(x_i)). This requires step 2

E 2. Expressing man(x_i), which yields *man.*

E 3. Using the result of step E2 we can express (every x_i : man(x_i)), obtaining *every man.*

E 4. Finally we combine the results from steps 1 and 3. These constituents have to be put in the right order. The possible orders of constituents are described by placement rules, monitored by different principles of consituent ordering (see Dik 1978). For the present case it amounts

to the order S.V. (Subject followed by Verb). Placing the constituents in this order yields the sentence *Every man walks*.

The parsing algorithm will invert the steps of this production process. Therefore the steps of the parsing process will be labelled I4, I3 I2 and I1 in this order. In some cases the result of this inversion can immediately be described, but in most cases several options will have to be investigated.

I4. The input for the parser is the string of words *Every man walks*. As a first step we do the inverse of step E4: we will split up this sentence in accordance with the constituent ordering principles. Initially we do not know the type of the constituent ordering of the input sentence; maybe the input is not a sentence at all. So we have to try all possibilities. Let us start with investigating the possibility that it is an S.V.- sentence; other cases will be considered later. Having an S.V. sentence would mean that we can split the sentence into two parts: a Subject and a Verb. Let us first consider the split into *Every man* and *walks*.

I3. If *Every man* is a subject, it has to be a term. The expression rules for term formation tell us that this would be a term if *man* would be the expression of a some predicate frame P; then the term would be (every x_i : $P(x_i)$) .

I2. Investigating the table of basic predicate frames, we learn that there indeed is a predicate frame corresponding with *man*.

I1. Finally we investigate whether for *walks* there is a corresponding predicate frame , and that turns out to be the case.

If we combine the findings of I4 .. I1, we see that the input string indeed is a sentence, and we obtain the predication in (3) as the result of this parsing process.

Next we investigate the alternatives (the sentence might have more parses). In stage I4 we split the sentence into two parts. There is one other split into two parts: *every* and *man walks*. In the I3 stage we have then to investigate whether *every* is a term. The rules tell us that a single word is a term only in case it is a proper name; but *every* is not in the list of proper names. And an attempt to analyse *man walks* as a verb would show us that it is not obtainable from such a predicate. So there is only one SV parse for the given input. The constituent ordering principles allow for many more constituent orderings than only S.V., but for a three-word-sentence further only S.V.O. (Subject - Verb - Object) and P1.S.V. (Topic - Subject - Verb) are possible. Both options would require that *every* is a term, which is not the case. So we have found all parses of the given string.

3. Inversion of Rules

In this section I will discuss the first one of the two restrictions on FG that make it possible to turn the idea for parsing into an algorithm. It is the restriction that expression rules are invertible. Maybe it is useful to start with an example of a non-invertible rule. Consider rule F, where $F(\phi)$ is obtained by deleting ϕ from all symbols after the 10th one. Given an output of F, it not possible to say which symbols, or even how many, were deleted. Of course, this is an artificial rule, which is

not allowed for in FG. This has to do with the FG principles that require the expression rules do not perform deletions or transformations. These requirements cause the inputs of an expression rule to occur unchanged somewhere in the output. Conversely: if we have the output of an expression rule, we know that its inputs can be found somewhere in the output (although we we do not exactly know where). So there might be several options for further investigation (but only a finite number!). After further investigations some options may turn out to be no constituents at all, others may give rise to a successful parse. These considerations show that from the FG principles it follows in an intuitive sense that expressions rules can be inverted.

In order to express the requirement of invertability more precisely, a format for expression rules has to be introduced. We will consider the expression rules as being composed of several operators, each operating on a certain number of inputs. Examples of one place operators would be E_1 and E_2, as given in (5) and (6)

(5) E_1((d1: John)) = *John*

(6) E_2(run$_V$) = *runs* , provided that *run* occurs in the context of a singular subject.

A two place expression rule arises when two constiuents are put in the S.V.-order by E_{SV}:

(7) E_{SV} (*John, sings*) = *John sings*

definition

A one place operator E is called **invertible** if there is an operator I such that

if E (ϕ) = ψ and I(ψ) = {ϕ_1, ϕ_2, . . .,ϕ_k},

then $\phi \in$ {ϕ_1, ϕ_2, . . .,ϕ_k} and for each $\phi_j \in$ I(ψ) it holds that E(ϕ_j) = ψ.

So I_j gives us a finite list of candidates for further investigation in the parsing process, and it is required that the correct choice is among the candidates.

definition

An n-place operator E is called **invertible** if there is an operator I such that

if E(ϕ_1, ϕ_2, . . .,ϕ_n) = ψ

and I(ψ) = {($\phi^1{}_1$, $\phi^1{}_2$, . . .,$\phi^1{}_n$), ($\phi^2{}_1$, $\phi^2{}_2$, . . .,$\phi^2{}_n$), . . . ($\phi^k{}_1$, $\phi^k{}_2$, . . .,$\phi^k{}_n$)},

then (ϕ_1, ϕ_2, . . .,ϕ_n) $\in$ I(ψ) and for each i it holds that E ($\phi^i{}_1$, $\phi^i{}_2$, . . .,$\phi^i{}_n$) = ψ .

An example concerning a two place rule is I_{SV}, the inverse rule for E_{SV}. For I_{SV} holds:

I_{SV} (*Every man walks*) = {(*Every , man sings*), (*Every man, sings*)}.

We are now prepared to formulate the first restiction on functional grammars needed for our parsing algorithm.

Restriction of invertibility

All operators (in formation rules and expression rules) are invertible.

4. Increasing Complexity

The second restriction needed in order to turn the idea for parsing into a working algorithm, is that expression rules should build compound expressions from simpler ones. The relevance for

parsing is that this restriction guarantees termination of the inversion process. Let us first consider examples that do not fulfill this restriction. Suppose that F(ϕ) is defined as deleting the last # sign from ϕ. This operation is invertible and inversion would add a # to ϕ, thus yielding ϕ#, which can be inverted (for F) to ϕ## and so on. So such a rule could lead to a backward search that does not terminate. Of course this is an artificial rule that is not allowed for in FG, due to the fact that deletions are not allowed in FG. Also lexical decomposition could bring the danger of a non terminating search, e.g if one analyses *kill* as *cause to become not alive*, and next employs an analysis of *alive* which contains *kill*. But lexical decomposition is not allowed for in FG either.

The FG principles which disallow deletions and lexical decomposition, imply that the expression rules build complex expressions from simpler ones. This intuition will be formalized by the restriction on expression rules that they have to increase the complexity of the involved expressions. If we apply an inversion rule to ϕ, then we obtain expressions of lesser complexity than ϕ, and after the next inversion still lesser complexity. The process terminates when we finally have obtained elements of minimal complexity : the basic predicate frames and basic terms.

In most practical cases 'increase of complexity' will mean that the output expressions are of greater length than the input expressions. However, the notion of complexity has to used in an intelligent way. The term (d1:John) is expressed as *John*, so the output expression is shorter than the predication itself. We therefore have to stipulate that FG expressions are always considered as being less complex than natural language expressions. So 'complexity' becomes a theory dependent notion

Our aim is reached as follows. We assign to a sentence the complexity measure <1,*m*> where *m* is the number of words in the sentence, and to an FG predication the complexity measure <0,*n*>, where *n* is the number of basic FG items in the predication (counting e.g basic terms, variables , labels). When comparing two expressions (of whatever nature) we first compare the first components of their complexity measures, and if these are equal, the second components. So the minimal complexity of an expression is <0,1>. In order to account for this somewhat artificial way of measuring, the restriction will be formulated in an abstract terminology. This abstract approach is not only useful for cases like *John,* but it will turn out to be useful as well in a difficult case for parsing, to be discussed the next section.

Restriction of increasing complexity

A functional grammar is accompagnied with a complexity function μ that assigns a complexity measure to every expression (i.e every string over the vocabulary consisting of natural language words and basic FG items). The possible values of the complexity function form a well founded partial ordering with a minimal element, and basic FG items have the minimal complexity. It is required that if ($\phi_1, \phi_2, \ldots, \phi_k$) $\in$ I(ψ) *then for all* i *it holds that* $\mu(\phi_i) < \mu(\psi)$.

A functional grammar satisfying the two restrictions could be called a 'parsable' functional grammar. Since the restrictions intuitively follow from FG principles, it is to be expected that all grammars in FG are parsable. But one should not consider the two restrictions as some formalization of the FG principles: in their present form the restrictions are weaker (see section 5).

Further investigations might lead us to a concrete complexity function for FG that characterizes the possible FG rules.

5. A Difficult Example

In Hungarian it is possible that a constituent is expressed in a clause other than that in which it belongs according to the predication in which it occurs. An example (de Groot 1981, p. 47) is:

(8) *Mari nem hiszem, hogy ismeri Chomskyt.*
Mary$_{nom}$ not believe-I that knows-she Chomsky$_{acc}$.
'I do not believe that Mary knows Chomsky'.

For this phenomenon the following expression rule is proposed (de Groot 1981, p. 57).

> *Terms from embedded predications can optionally be displaced to the P1 and Pϕ positions of the higher clause when those terms carry the same pragmatic function as the embedded predication, with the exception of Focus from satellites.*

Note first that this rule does *not* first form the constituents, and puts them thereafter in the right order. The rule grasps a term from deep embeddings: a transformation takes place. So it seems that the rule does not obey the FG principles in several respects, but there appears to be strong linguistic motivation for it.

Since the rule deviates from the principles, it is not surprising that it causes problems for our algorithm that is based on them. The inversion of the above rule cannot start with splitting, and then investigating whether the parts are the desired constituents. We have to find another way.

There happens to be a solution. We start with selecting a candidate for the displaced term, take it away, and replace it in the position where it would have occurred if it had not been displaced. Since we do not know which position this is, we have to investigate all possibilities. For each possibility the parsing method as sketched in the previous sections can be used. However, two points have to be taken care of.

The length of the sentence with the replaced term is the same as of the original sentence. So we have to stipulate that the sentence with displaced term is more complex than the sentence where replacement has been performed. This can be dealt with due to our abstract approach to complexity. We add a third component to the complexity measure, set this component to 1 for all input sentences, and after replacement it becomes 0. It also becomes 0 if the level of sentences is left and we are parsing the constituents. The examples in the article make clear that for embedded sentences this component has to be reset on 1.

The expression rule quoted above, gives no information on what happens to the embedded term when no displacement has taken place. The idea of simply replacing only works if the displaced term has the same form as the non displaced term. If this is not the case, the replaced term has to be changed accordingly (and we have to hope that this can be done on the basis of the available input information). This illustrates that the design of inversion rules requires linguistic knowledge, and is preferably performed by the linguist designing the expression rules.

Note finally that there is one aspect of the quoted rule which makes the whole idea of replacement possible: the fact that the displacement was optional. In case of an obligatory

displacement rule, it would not be possible to replace the term and parse then as usual. Happily, an obligatory rule would not be in the spirit of FG (Dik, pers. comm.).

This example illustrates the power of an abstract complexity function. It seems always to be possible, given a rule of whatever wild character, to define a complexity function such that the rule obeys the requirement of increasing complexity. It is this flexibility which causes the requirements to be weaker than the FG principles. The challenge of the requirement lies in the aspect that all rules have to be consistently measured: there has to be a complexity function (with the desired properties) for the grammar as a whole.

6. The Algorithm

We are now prepared for a sketch of the parsing algorithm. The input for the parsing algorithm is a candidate sentence ψ. The constituent ordering principles determine a finite set of expression rules that might have been used to produce this sentence, say $E_1, \ldots, E_n$. Let $I_1, \ldots, I_n$ be the corresponding inversion rules. The first step of the parsing process is to produce $I_i(\psi)$ for all i. The elements in these sets (probably n-tuples) are candidates for constituents of ψ (candidates for being terms in certain roles, verbs, or satellites in certain roles). For each of these candidates, inversion rules of the appropriate type have to be applied, etc. In this way we build a search tree, in which the nodes are n-tuples of expressions, and the vertices describe the I-relation. Due to the requirement of increasing complexity this is a terminating process.

In order to define what a parse is, we define the auxialiary notion *parsed.* All basic FG items are called 'parsed'. We call an expression ϕ 'parsed', if there is a rule I_j such that there is in $I_j(\phi)$ an n-tuple of which all elements are parsed. So an input sentence is called parsed if there is a chain of parsed elements ending with basic FG items. The predication to be found is then obtained by combining these basic items in accordance with the expression rules found in the chain. One could terminate the process when one parse is found, or search for all possible parses. Depending on ones aim, a depth first way through the tree could be chosen, instead of the breadth first way described above.

Not all aspects of FG rules are covered by the examples we discussed. We did, for instance, not consider rules for tense, nor the formation of derived predicate frames and derived terms. I expect that such rules can be dealt with in the same way as the expression rules; it seems not too difficult to satisfy the two restrictions. Neither did we discuss the treatment of variables like x_i and x_j, whereas arsing *every man* requires the selection of a variable. I expect that this can be done in some canonical way (as we did in the example).Then we don't have to produce an infinite number of parses, only differing in the choice of the variable.

Next we come to the issue of efficiency. Several ideas for increasing the efficiency might be considered.

When a candidate sentence is to be parsed, it might be a good idea to start with searching for a finite verb. Such a verb is easily recognised, and if the verb is found, it is easier to guess which constituent ordering is used. However, this is knowledge about the language we are parsing, viz. English. Since we aimed at a generally applicable parsing algorithm, we abstinated from

incorporating in the algorithm information about specific languages. If we wish to improve the efficiency by implementing this way of searching, it has to be incorporated in the inversion rule for constituent ordering.

We allowed for any finite number of elements in I(ϕ) and that might cause an explosion of possibilities to investigate. This was for instance the case for the replacement rule discussed in section 6. Limiting this number to a only a few, say 2 or 3, seems in practice possible, but knowledge about the specific phenomenon described by the rule is needed in order to say how to do this. A general parsing algorithm can do nothing about such an explosion. As in the previous case, the designer of the inversion rules should take care of this aspect of efficiency. Since that requires knowledge of the phenomenon under discussion and of the way it is analysed in the grammar, the inversion rules are preferably made by the one who designed the expression rules.

There is at least one respect in which the described general algorithm can be improved. In the given description different options were investigated independently. For instance, in section 2 we investigated the S.V. analysis of *Every man walks*. After the split into *every* and *man walks,* we had to investigate whether *every* was a term. When testing for an S.V.O. form, the same question arose again. Efficiency would increase if we could guarantee that the same question will be investigated only once. Techniques for this strategy are possible.

The method of building a parsing algorithm on invertible rules and increasing complexity originates from Jan Landsbergen. Although his parsing algorithm for Montague Grammar (Landsbergen 1981) differs in some respects from the algorithm presented here, the two restrictions play the same role. In the Philips translation system 'Rosetta' (a variant of) the algorithm is used; experience with its efficiency is positive.

It is not by accident that a parsing algorithm could be transferred from Montague Grammar to Functional Grammar. The two grammatical systems have the same underlying mathematical structure: in both systems the semantic information is encoded in a term algebra (in FG the elements of this algebra are the predications, in MG the derivational histories). For a description of the algebraic framework underlying Montague grammar see Montague (1970) or Janssen (1986). Both in FG and in MG the natural language expressions are obtained by evaluating the elements in the term algebra, and in this evaluation procedure the two restrictions can be obeyed. That is due to the fact that in both systems the semantic relations between different syntactic forms need not to be captured on the syntactic level by means of transformations (as was the case in Transformational Grammar). The mathematical resemblance between the two systems is employed here to transfer a parsing algorithm, but more can be gained from it. In Montague grammar the semantic information encoded in the derivational histories is interpreted in a model theoretic way. Thus semantic relations (such as implication and equivalence) can be accounted for. A model theoretic interpretation can be assigned to Functional Grammar in precisely the same way (see Janssen (1981)). And these two transfers do not exhaust the possibilities; I expect that a cross-fertilization is possible in more respects.

References

DIK, Simon C.,

1978 *Functional Grammar*, Dordrecht: Foris .

1980 'Seventeen sentences: basic principles and application of functional grammar', in: E.A. MORAVCSIK and J.R. WIRTH (eds.) Syntax and Semantics 13, *Current approaches to syntax,* 45-75, New York: Academic Press.

GROOT, Caspar de

1981 'Sentence intertwining in Hungarian', in A.Machteld BOLKESTEIN et al., *Predication and expression in Functional Grammar,* 41-62, London: Academic Press.

JANSSEN, Theo M.V.,

1981 'Montague grammar and functional grammar', in Teun HOEKSTRA, Harry van der HULST and Michael MOORTGAT (eds), *Perspectives on functional grammar*, 273-297, Dordrecht : Foris. Also: GLOT 3, 1980, 273-297.

1986 *Foundations and applications of Montague grammar. Part 1: Foundations, logic, computer science',* CWI Tract 19, Amsterdam : Centre for Mathematics and Computer Science.

LANDSBERGEN, Jan,

1981 'Adaptation of Montague Grammar to the requirements of parsing', in Jeroen A.G. GROENENDIJK, Theo M.V. JANSSEN & Martin B.J. STOKHOF, *Formal methods in the study of language, part 2.*, 399-419, MC Tract 136, Amsterdam, Centre for Mathematics and Computer Science.

MONTAGUE, Richard,

1980 Universal Grammar, *Theoria 36*, 373-398. Reprinted in R.H. THOMASON (ed.) 1984, *Formal philosophy. Selected papers of Richard Montague*, 222-246, New Haven: Yale University Press.

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

Other prepublications:

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