Ups and Downs of Type Theory

by

W. Peremans
This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology. Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review. Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. M. Philips
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

All rights reserved
editors: prof.dr.M.Rem
         prof.dr.K.M.van Hee.
Ups and Downs of Type Theory*

W. Peremans†

A global historical sketch of the development of formalized type theory in the 20th century will be given. No surprising historical insights will be presented as all the items discussed can be found in the current literature.

In the beginning of the 20th century, type theory was developed to deal with the paradoxes which were discovered in logic and set theory. This led to a great interest in types during the first decennia of this century. However, this interest flagged afterwards. There were two somewhat related reasons for this. Firstly, type theory began to play a less important role in logic because it became clear that first-order-logic, which is free of types, satisfied many criteria. In the second place, axiomatic set theory had been developed. This theory was free of types and its aim was to describe "the entirety of mathematics".

As a result, interest in types had diminished considerably by the middle of our century. It was the hey-day of Bourbaki, whose influence on the style of thinking and writing in mathematics must not be underestimated. Later, the tide turned again. Satisfaction with set-theoretical coding diminished and its unnaturalness was more strongly felt. Furthermore, the rise of computer science sparked a revival of interest in types.

This raises two questions: What are types? Why do we restrict ourselves to the 20th century? Undoubtedly, formalized type theory is a product of the 20th century. The intuitive notion of type is not clearly defined, but had undoubtedly been vivid much earlier. It was in fact so self-evident that there was no need to make it explicit. It is simply the awareness that there are different kinds of mathematical objects. For example, a triangle and a real number are mathematical objects. In the 19th century, one would not have understood the meaning of \{triangle, real number\}, and

---

*This text reflects the contents of a lecture held on May 18, 1990, in Eindhoven, which was an introduction to a symposium on type theory.

†The author is indebted to Marianne Sanders and Twan Laan for the translation from Dutch of his paper.
not only because this kind of notation was unknown.

We have been so indoctrinated with set theory that we are used to it. These days, "everything" is a set. One can wonder whether this is a blessing or not. The set-theoretical coding of mathematics surely is not the ultimate salvation message, but rather an incident of a temporary nature.

De-typing (the disappearance of types) was already evident in the 19th century, and perhaps even earlier. A triangle could be described by its three vertices, giving, with coordinates in the plane, six real numbers. The arithmetization of mathematics took place in the 19th century, introduced by the restraint of the crisis in analysis, led to the tightening up of notions such as limit, continuity and real numbers (Cauchy, Dedekind). The goal was to reduce everything to the concept of natural number. This reduction, however, was not canonical; one just has to realise the different ways in which real numbers can be constructed. The analogy to the set-theoretical coding in the 20th century is striking.

Arithmetization is a detyping which gave the mathematicians great satisfaction. Poincaré, for example, held a lecture at the international congress of mathematics in Paris in 1900 in which he stated that mathematics was finally resting on a solid base: Everything was reduced to natural numbers. In fact, the next crisis in the foundations of mathematics was drawing near.

Because of the arithmetization, there was great interest in the question of what natural numbers really are. Consider the works of Dedekind (1888), Peano (1903) and Frege (1893-1903). In Frege’s work we also find the origin of propositional and predicate logic. Besides, Cantor’s set theory had been developed since 1873.

At the turn of the century, contradictions (also called antinomies or paradoxes) were discovered. The oldest one is Burali-Forti’s (1897), which did not attract much attention, maybe because it refers to ordinal numbers, a subject in which mathematicians were not much interested in those days. Russell’s paradox of 1902 did attract attention, however, because it made logic as well as set theory topical again. Dedekind and Frege were shocked, and Poincaré changed his position and became an opponent of set theory.

There is a logical and a set-theoretical variant of the paradox. As the first one is the most important to us now, we will discuss it briefly.

Consider properties, or predicates. These can be self-referent or not. For instance, "red" is not red, but "abstract" is abstract. We will call "self-referent": "predicable", and its negation: "impredicable".

"Impredicable" is also a predicate itself, a predicate on predicates, to be precise.
We might wonder whether "impredicable" is predicable or impredicable. Assume that impredicable is impredicable. Then it refers to itself, so it is predicable. This is a contradiction. So, the assumption is not right, therefore impredicable is not impredicable. But that means that impredicable does not refer to itself, so: it is impredicable. We have another contradiction, but now without an assumption. It can also be written down in a less verbal manner. Let \( I \) denote the predicate "impredicable". Then, by definition, \( I(x) \iff \neg x(x) \). Now suppose \( I(I) \) holds, then, by definition, \( \neg I(I) \) holds, so: \( I(I) \) does not hold. Contradiction. So: \( I(I) \) is not true. But that means \( \neg I(I) \) is true, and, again by definition, \( I(I) \) holds. Contradiction. During the first decennium of our century, several other antinomies were discovered. An antinomy of quite another kind than Russell's is Richard's (1905), related to Berry's paradox (1906):

Consider the real numbers between 0 and 1, which can be described by a finite sentence in English. These form a countable set \( R \). Using an enumeration of all the English sentences, an enumeration of \( R \) can be constructed. Using a diagonal argument, one can construct a real number which does not belong to \( R \). Enumeration and diagonal argument can be described by a finite sentence and therefore provide a number in \( R \). Contradiction.

Historically, the description above is the oldest one. A variant is as follows: Write the real numbers between 0 and 1 as binary fractions, e.g., .01101..., and interpret such a sequence as a property of natural numbers: If the \( n \)th place of the sequence has a 1 (0) then \( n \) has (does not have) the property. Properties of natural numbers which can be described by finite sentences can be enumerated: \( W_0, W_1, \ldots \). Now consider \( \neg W_n(n) \), as a property of \( n \). This property can be formulated by a finite sentence, so a natural number \( q \) exists such that \( W_q(n) \iff \neg W_n(n) \) for all natural numbers \( n \). Substituting \( q \) for \( n \) leads to a contradiction.

Berry's antinomy is as follows. Regard the natural numbers which can be defined by an English sentence containing no more than 50 words. Because there are only finitely many of these sentences, this yields a finite collection of natural numbers. The smallest natural number not belonging to this set results in a contradiction.

We will not discuss other antinomies here. The antinomy of the liar is very old and well known.

How can these antinomies be avoided? Logic and set theory have each taken their own lines. We start with logic.

It appears to be troublesome to unbridledly apply principles which used to be allowed. We therefore exercise restraint in using logical principles. This is unsatis-
factory since we have to do it ad hoc: The antinomies which are to be avoided do not cogently imply what the restrictions should be. One simply imposes restrictions and states that the known arguments which lead to the antinomies can no longer be maintained. This is unsatisfactory on two points:

On the one hand, we can wonder whether it could have been done with less. The choice is sufficient for the target we set ourselves, but may not be necessary. This weighs even more heavily because reasonings which seemed to be "innocent" cannot be maintained or only maintained with much more effort. Sand has come into the machinery.

On the other hand, the question whether it should not be more arises. The fact that known antinomies are eliminated does not prove that there are no others, which still can be proved. A consistency proof of the system would be very welcome. However, this does not appear to be very successful.

In set theory the solution has been sought in another direction. In logic the formalism, the language itself, was restricted. On set theory the restrictions were not made on the language but on the axioms, especially when new sets are constructed.

We may ask ourselves what the problem is. In Russell's antinomy, self-reference comes up. In Richard's paradox, we have to do with impredicativity. A definition of an object is called impredicative if there is a totality in it, which contains that object, or a property fulfilled by that object. At the beginning of this century there was a lot of discussion on the permissibility of impredicative definitions. Poincaré and Russell, among others, took part in this debate. Simply forbidding them is not very attractive because mathematics is full of them. An example is "maximum of a continuous function on a closed interval", but also "the youngest of the class".

Russell solved the problem by not allowing self-reference and restricting impredicativity. This led to so-called ramified type-theory, treated in great detail in the classical three-volume work by Whitehead and Russell, "Principia Mathematica" (1910-1913). Ramifying has gone out of use, which resulted in simple type theory.

Below is a general description of the idea.

There are objects of type 0. Properties of objects of type 0 have type 1; properties of properties of objects have type 2, and so on. There are no other properties. In this way, self-reference is eliminated and, together with it, Russell's antinomy. Types which are \( \geq 0 \) are divided into orders. To simplify matters, let us look at type 1. Properties of type 1 which are defined without referring to an entity of objects are of order 0. Properties containing entities of order \( \leq k \) in their definition are of order \( k+1 \). As a consequence, only entities of properties of bounded order may occur in
the definition of a property. Thus, Richard's paradox is eliminated, but, in fact, this holds for all impredicative definitions. Because of that, the system is hardly useful for common mathematics. In "Principia Mathematica" this problem is solved by a bold intervention, the axiom of reducibility: To every property of higher order, there is a property of order 0 which is coextensive with the given property.

Later, another solution to the problems was found. It started with Ramsey, who, in 1926, divided antinomies into two classes:

1. Logical (e.g., Burali-Forti, Russell)
2. Semantical (e.g., Richard, Berry, the Liar)

For logical antinomies, ramifying is not necessary, and simple type theory is sufficient. Semantic antinomies have in their language references to expressions in that language like definability in that language, or truth of sentences in that language. This is what we are going to forbid. We begin with a language. That language has to be spoken about in another language, which will be called meta-language.

This procedure has been completely accepted nowadays. The hierarchy of orders is substituted by a hierarchy of languages because the process can be iterated: Meta-language has a metametalanguage, and so on. That this idea was quickly accepted was probably due to the fact that ramifying in type theory could be thrown overboard.

We now arrive at the situation in the '30s. But first something must be said about the other development: set theory. In the meantime set theory had grown and had been perfected (Zermelo, Fraenkel, Von Neumann, Bernays, Gödel). One saw the realization of the vision to reflect and code "the entirety of mathematics" come closer. Everything was built on the empty set and types did not come up at all. At the end of the '30s, the Bourbaki group started the execution of this plan.

On the other hand, the type-theoretical design of logic had also been streamlined and simplified. With the use of subtle tricks it was even possible to build type theory on a first order language. Even type theory itself attempted to make the intuitive notion of type superfluous.

A well-known and later much used way of formalizing (simple) type theory is Church's (1940). He connected types with the use of lambda-calculus, which came to be usual in later developments as well. Types are no longer numbers as in "Principia Mathematica". Type symbols are used, reflecting composition of functions
and forming of predicates. Furthermore, as usual in lambda-calculus, functions of more than one variable are considered as functions of one variable by repetition of function application.

Below is a short sketch of the build-up:

- There are two basic types: \( \iota \) (individuals) and \( \sigma \) (propositions, truth values).

There are also type symbols:

- \( \iota \) and \( \sigma \) are type-symbols;
- if \( \alpha \) and \( \beta \) are type-symbols, then \( \alpha \beta \) is also a type-symbol. \( \alpha \beta \) is interpreted as the type of functions with domain \( \beta \) and range \( \alpha \).

The obvious omission rules for brackets are used.

- There are variables and constants, all having a type, written as a subscript. The following primitive symbols occur:

  - Improper: \( \lambda, (,); \)
  - Proper: \[ N_{\alpha \sigma}, D_{\alpha \sigma}, \Pi_{\alpha(\sigma)}; \iota_{\alpha(\sigma)} \] constants
  - \( a_\alpha \) variables

  Here \( \alpha \) can be any type symbol.

- (Well-formed) formulas have a type (written as subscript) and are built up from primitive symbols using formation rules:

  - Proper primitive symbols;
  - \( (\lambda x_\beta . M_\alpha) \), having type \( \alpha \beta \) for variables \( x_\beta \) and formulas \( M_\alpha \);
  - \( (F_\alpha \beta A_\beta) \), having type \( \alpha \) for formulas \( F_\alpha \beta \) and \( A_\beta \).

The interpretation of the constants is as follows:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Interpretation</th>
<th>( x )</th>
<th>( x ) is also written as</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N_{\alpha \sigma} )</td>
<td>Negation</td>
<td>( N_{\alpha \sigma} A_\sigma )</td>
<td>( \neg A_\sigma )</td>
</tr>
<tr>
<td>( D_{\alpha \sigma} )</td>
<td>Disjunction</td>
<td>( D_{\alpha \sigma} A_\sigma B_\sigma )</td>
<td>( A_\sigma \lor B_\sigma )</td>
</tr>
<tr>
<td>( \Pi_{\alpha(\sigma)} )</td>
<td>Universal quantification</td>
<td>( \Pi_{\alpha(\sigma)}(\lambda x_\alpha A_\sigma) )</td>
<td>( (\forall x_\alpha)A_\sigma )</td>
</tr>
<tr>
<td>( \iota_{\alpha(\sigma)} )</td>
<td>Choice-operator</td>
<td>( \iota_{\alpha(\sigma)}(\lambda x_\alpha A_\sigma) )</td>
<td>( (\iota x_\alpha) A_\sigma )</td>
</tr>
</tbody>
</table>

In a further construction, combinators, natural numbers of each type, equality, successor function, and so on, are made.

Rules are: lambda conversion, substitution, modus ponens, generalization.

Axioms are the axioms of predicate calculus and the axiom of infinity. For use in analysis, the axiom of extensionality and the axiom of choice are added.
Another build-up of type theory, stratification (Quine), is not based on lambda-calculus, but on a first order predicate logic. It can be found in a simple form in A. Robinson’s book “Non Standard Analysis”.

Now, types are directly connected with predicates with more variables:

- 0 is a type;
- if $\tau_1, \ldots, \tau_n$ are types, then also $(\tau_1 \ldots \tau_n)$.

We use a predicate calculus of first order with a $(n+1)$-ary relation symbol $\Phi_\tau$ for each type $\tau \neq 0$; $\tau = (\tau_1, \ldots, \tau_n)$. Furthermore, there are variables and constants (without type).

We assign types to the open places in the relation symbols: If $\tau_0 = (\tau_1, \ldots, \tau_n)$, then $\tau_i$ is the type of place $(i+1)$ in $\Phi_{\tau_0}$, for $i = 0, \ldots, n$. Doing so, the thing put on that place gets a type as well. To do this unambiguously, we restrict ourselves to stratified formulas, i.e., formulas in which each occurring constant and variable has the same type in each occurrence. Consider the following example: Let $x, y$ be different variables.

- $\Phi_{(0)}(x, y)$ is stratified;
- $(\forall x)\neg\Phi_{(0)}(x, x)$ is not stratified: In $\Phi_{(0)}(x, x)$, the first occurrence of $x$ is of type (0) and the second one is type 0.

In a stratified formula, a type can be assigned to each variable and each constant appearing in it. When we construct a model, a predicate of the right structure must be chosen for each variable.

We will not work this out in detail.

Because of all these developments, elimination of types was not complete, but it was far-reaching. Set theory was completely type-free and in higher order logic, types played only a modest rôle. A smart modification of the concept of “model” resulted in a completeness theorem that was also valid for higher order logics (Henkin 1950), though, because of Gödel, 1931, this is impossible for “ordinary” models. Consequently, the concept of “non-standard model” became the vogue. In relation to this, we only mention Skolem’s paradox and non-standard analysis.

There were a number of circumstances which led to a renewed interest in types in the second half of this century.

Slowly, there grew a discontent with the straitjacket of Bourbaki, which prescribed how mathematics should be practised and presented. In particular, the identification
"mathematics = set theory" was increasingly challenged. Pure set theory provides at most a coding of mathematics which is sometimes unnatural. A simple example is the ordered pair \((a, b)\), usually defined as \(\{\{a\}, \{a, b\}\}\) in set theory. The normal properties of ordered pairs are guaranteed by this definition, but one cannot hold that this is the ordered pair, or that there is a compelling reason for defining it in this way and not in another.

The rise of computer science has also been of influence. Types had already appeared in early programming languages. Meanwhile a complete specialism has developed, viz., the theory of data types with the corresponding algebras. Typed lambda calculus is enjoying growing interest.

Another line of development originates from practising the foundations of mathematics. De Bruijn’s AUTOMATH project arrived in the process of the development of a language in which this could be realised, at typed lambda calculus. Moreover, a unity of logic and mathematics came into existence, in which classes of proofs could also function as types, and proofs were objects. This is often indicated as “propositions as types”. De Bruijn considers this unfortunate. He prefers speaking about “proofs as objects”. At about the same time, this thought was also put forward by others.

Martin-Löf’s type theory obtained a great reputation because he managed to interest computer scientists for his theory.

The notion of “propositions as types” did not completely rise from nothing. It is connected with older developments in intuitionism, but has followed its own path. The relation can be seen in the so-called BHK interpretation of intuitionism, where a proof of an implication \(A \rightarrow B\) is, in fact, a construction which turns every proof of \(A\) into a proof of \(B\).

Because of these developments, type theory is currently very much alive again.

We conclude with two remarks.

The first one is that typed lambda calculus is also used for the analysis of natural languages in R. Montague’s school.

Secondly, we want to call attention to topos theory as an alternative for type theory. A topos is a category satisfying certain extra conditions. Category theory was established by Eilenberg-MacLane in 1945 in the context of algebraic topology. Originally it was, of course, founded on set theory. Later, the thought grew that the category could serve as an alternative for the set as a building block for mathematics. The breakthrough was realised by Lawvere’s work in 1963, introducing the topos as a category with some extra conditions, analogous to the Grothendieck topos. For
Lawvere, topos theory is a kind of higher-order logic. Originally there was not much response from logicians. A personal memory is that on a conference on logic in Varenna (1968), a guest-lecture of Lawvere was hardly understood by the logicians present. However, afterwards appreciation grew quickly and topos theory became a recognized specialism in logic. For example, the "Handbook of Mathematical Logic" (1977) contains a full chapter on topos theory.

In fact, topos theory as well as type theory is a kind of higher order logic. They can also be translated into each other, though not easily. Because of the different frameworks, we will not go into detail here. We only mentioned topos theory because it is an alternative for type theory which some people consider more natural.
In this series appeared:

91/01 D. Alstein
91/02 R.P. Nederpelt
    H.C.M. de Swart
91/03 J.P. Katoen
    L.A.M. Schoenmakers
91/04 E. v.d. Sluis
    A.F. v.d. Stappen
91/05 D. de Reus
91/06 K.M. van Hee
91/07 E. Poll
91/08 H. Schepers
91/09 W.M.P.v.d.Aalst
91/10 R.C. Backhouse
    P.J. de Bruin
    P. Hoogendijk
    G. Malcolm
    E. Voermans
    J. v.d. Woude
91/11 R.C. Backhouse
    P.J. de Bruin
    G. Malcolm
    E. Voermans
    J. van der Woude
91/12 E. van der Sluis
91/13 F. Rietman
91/14 P. Lemmens
91/15 A.T.M. Aerts
    K.M. van Hee
91/16 A.J.J.M. Marcelis

Implication. A survey of the different logical analyses "if...,then...", p. 26.
Parallel Programs for the Recognition of P-invariant Segments, p. 16.
Performance Analysis of VLSI Programs, p. 31.
An Implementation Model for GOOD, p. 18.
SPECIFICATIEMETHODEN, een overzicht, p. 20.
CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.
Terminology and Paradigms for Fault Tolerance, p. 25.
Interval Timed Petri Nets and their analysis, p. 53.
POLYNOMIAL RELATORS, p. 52.
Relational Catamorphism, p. 31.
A note on Extensionality, p. 21.
The PDB Hypermedia Package. Why and how it was built, p. 63.
An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.
91/17 A.T.M. Aerts  
P.M.E. de Bra  
K.M. van Hee  

Transforming Functional Database Schemes to Relational Representations, p. 21.

91/18 Rik van Geldrop

Transformational Query Solving, p. 35.

91/19 Erik Poll

Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.

91/20 A.E. Eiben  
R.V. Schuwer


91/21 J. Coenen  
W.-P. de Roever  
J.Zwiers

Assertional Data Reification Proofs: Survey and Perspective, p. 18.

91/22 G. Wolf


91/23 K.M. van Hee  
L.J. Somers  
M. Voorhoeve

Z and high level Petri nets, p. 16.

91/24 A.T.M. Aerts  
D. de Reus

Formal semantics for BRM with examples, p. 25.

91/25 P. Zhou  
J. Hooman  
R. Kuiper

A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and completeness, p. 52.

91/26 P. de Bra  
G.J. Houben  
J. Paredaens

The GOOD based hypertext reference model, p. 12.

91/27 F. de Boer  
C. Palamidessi

Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.

91/28 F. de Boer

A compositional proof system for dynamic process creation, p. 24.

91/29 H. Ten Eikelder  
R. van Geldrop

Correctness of Acceptor Schemes for Regular Languages, p. 31.

91/30 J.C.M. Baeten  
F.W. Vaandrager

An Algebra for Process Creation, p. 29.

91/31 H. ten Eikelder

Some algorithms to decide the equivalence of recursive types, p. 26.

91/32 P. Struik


91/33 W. v.d. Aalst

The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.

91/34 J. Coenen

Specifying fault tolerant programs in deontic logic, p. 15.
Asynchronous communication in process algebra, p. 20.

A note on compositional refinement, p. 27.

A compositional semantics for fault tolerant real-time systems, p. 18.

Real space process algebra, p. 42.

Program derivation in acyclic graphs and related problems, p. 90.

Conservative fixpoint functions on a graph, p. 25.

Discrete time process algebra, p. 45.

The fine-structure of lambda calculus, p. 110.

On stepwise explicit substitution, p. 30.


Composition and decomposition in a CPN model, p. 55.

Demonic operators and monotype factors, p. 29.


Set theory and nominalisation, Part II, p. 22.

The total order assumption, p. 10.

A system at the cross-roads of functional and logic programming, p. 36.

Integrity checking in deductive databases; an exposition, p. 32.

Interval timed coloured Petri nets and their analysis, p. 20.

A unified approach to Type Theory through a refined lambda-calculus, p. 30.

Axiomatizing Probabilistic Processes: ACP with Generative Probabilities, p. 36.

Are Types for Natural Language? P. 32.
92/21  F.Kamareddine  Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.

92/22  R. Nederpelt F.Kamareddine  A useful lambda notation, p. 17.


92/24  M.Codish D.Dams Eyal Yardeni  Bottom-up Abstract Interpretation of Logic Programs, p. 33.

92/25  E.Poll  A Programming Logic for Fö, p. 15.


93/01  R. van Geldrop  Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.

93/02  T. Verhoeff  A continuous version of the Prisoner's Dilemma, p. 17.

93/03  T. Verhoeff  Quicksort for linked lists, p. 8.

93/04  E.H.L. Aarts J.H.M. Korst P.J. Zwictering  Deterministic and randomized local search, p. 78.

93/05  J.C.M. Baeten C. Verhoef  A congruence theorem for structured operational semantics with predicates, p. 18.

93/06  J.P. Veltkamp  On the unavoidability of metastable behaviour, p. 29.

93/07  P.D. Moerland  Exercises in Multiprogramming, p. 97.

93/08  J. Verhoosel  A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.


93/10  K.M. van Hee  Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.


| 93/14 | J.C.M. Baeten  
| 93/15 | J.C.M. Baeten  
    | J.A. Bergstra  
| 93/16 | H. Schepers  
    | J. Hooman | A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27
| 93/17 | D. Alstein  
    | P. van der Stok | Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.
| 93/18 | C. Verhoef | A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.
| 93/19 | G-J. Houben | The Design of an Online Help Facility for ExSpect, p. 21.
| 93/21 | M. Codish  
    | D. Dams  
    | G. Filé  
| 93/22 | E. Poll | A Typechecker for Bijective Pure Type Systems, p. 28.
| 93/23 | E. de Kogel | Relational Algebra and Equational Proofs, p. 23.
| 93/24 | E. Poll and Paula Severi | Pure Type Systems with Definitions, p. 38.
| 93/26 | W.M.P. van der Aalst | Multi-dimensional Petri nets, p. 25.
| 93/27 | T. Kloks and D. Kratsch | Finding all minimal separators of a graph, p. 11.
| 93/28 | F. Kamareddine and R. Nederpelt | A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.
| 93/29 | R. Post and P. De Bra | GOLD, a Graph Oriented Language for Databases, p. 42.
| 93/30 | J. Deogun  
    | T. Kloks  
    | D. Kratsch  
    | H. Müllner | On Vertex Ranking for Permutation and Other Graphs, p. 11.
| 93/31 | W. Körver | Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.
<table>
<thead>
<tr>
<th>ID</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>93/33</td>
<td>L. Loyens and J. Moonen</td>
<td>ILIAS, a sequential language for parallel matrix computations, p. 20.</td>
</tr>
<tr>
<td>93/34</td>
<td>J.C.M. Baeten and J.A. Bergstra</td>
<td>Real Time Process Algebra with Infinitesimals, p.39.</td>
</tr>
<tr>
<td>93/36</td>
<td>J.C.M. Baeten and J.A. Bergstra</td>
<td>Non Interleaving Process Algebra, p. 17.</td>
</tr>
<tr>
<td>93/38</td>
<td>C. Verhoef</td>
<td>A general conservative extension theorem in process algebra, p. 17.</td>
</tr>
<tr>
<td>93/41</td>
<td>A. Bijlsma</td>
<td>Temporal operators viewed as predicate transformers, p. 11.</td>
</tr>
<tr>
<td>93/42</td>
<td>P.M.P. Rambags</td>
<td>Automatic Verification of Regular Protocols in P/T Nets, p. 23.</td>
</tr>
<tr>
<td>93/43</td>
<td>B.W. Watson</td>
<td>A taxonomy of finite automata construction algorithms, p. 87.</td>
</tr>
<tr>
<td>93/44</td>
<td>B.W. Watson</td>
<td>A taxonomy of finite automata minimization algorithms, p. 23.</td>
</tr>
<tr>
<td>93/48</td>
<td>R. Gerth</td>
<td>Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.</td>
</tr>
</tbody>
</table>
| 94/01 | P. America  
M. van der Kammnen  
R.P. Nederpelt  
O.S. van Roosmalen  
H.C.M. de Swart | The object-oriented paradigm, p. 28. |
| 94/02 | F. Kamareddine  
R.P. Nederpelt | Canonical typing and \( \Pi \)-conversion, p. 51. |
| 94/03 | L.B. Hartman  
| 94/04 | J.C.M. Baeten  
J.A. Bergstra | Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18. |
| 94/05 | P. Zhou  
| 94/06 | T. Basten  
T. Kunz  
J. Black  
M. Coffin  
D. Taylor | Time and the Order of Abstract Events in Distributed Computations, p. 29. |
| 94/07 | K.R. Apt  
| 94/08 | O.S. van Roosmalen | A Hierarchical Diagrammatic Representation of Class Structure, p. 22. |
| 94/09 | J.C.M. Baeten  
J.A. Bergstra | Process Algebra with Partial Choice, p. 16. |
| 94/10 | T. verhoeff | The testing Paradigm Applied to Network Structure, p. 31. |
| 94/11 | J. Peleska  
C. Huizing  
| 94/12 | T. Kloks  
D. Kratsch  
| 94/13 | R. Seljée | A New Method for Integrity Constraint checking in Deductive Databases, p. 34. |