Fixed-point calculus

Eindhoven University of Technology

Department of Mathematics and Computing Science

Fixed-Point Calculus

by

Mathematics of Program Construction Group

94/48

ISSN 0926-4515

editors: prof.dr. J.C.M. Baeten
prof.dr. M. Rem

Computing Science Report 94/48
Eindhoven, October 1994

Fixed-Point Calculus

Mathematics of Program Construction Group*

October 26, 1994

Abstract

The aim of this paper is to present a small calculus of extreme fixed points and to show it in action. The fixed-point theorem that was the main incentive for writing this paper is the fusion theorem presented in Section (3). It exploits the calculational properties of Galois connections.

Keywords Program derivation, programming calculi, theory of computation.

1 Introduction

Solving equations is fundamental to computing. Yet, rules for doing so are seldomly explicitly taught or used, and certainly not in a calculational way. This paper summarizes a small selection of such rules and shows their use in a series of examples. Most results obtained in these applications are well-known; it is the method —purely equational reasoning— that is novel.

Our universes of discourse are complete lattices — in some applications augmented with a regular-algebra structure — and all functions considered are monotonic. We present a calculus of least fixed points; its counterpart for greatest fixed points follows by dualization.

The following notations and notational conventions are used

- function application is denoted by a right-associative infix dot
- μf and $\mu\langle x \mapsto f.x\rangle$ denote the least fixed point of function $f : x \mapsto f.x$, which maps x to $f.x$

*Chritiene Aarts, Roland Backhouse, Eerke Boiten, Henk Doornbos, Netty van Gasteren, Rik van Geldrop, Paul Hoogendijk, Ed Voermans and Jaap van der Woude. Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

- lifting: for any binary relation $\sqsubseteq$ the lifted version $\dot{\sqsubseteq}$ is defined by

$$f \dot{\sqsubseteq} g \quad \equiv \quad \forall(x :: f.x \sqsubseteq g.x)$$

- sections: for binary infix operator $\oplus$, sections $(b\oplus)$ and $(\oplus b)$ are the functions defined for all x by $(b\oplus).x = b\oplus x$ and $(\oplus b).x = x\oplus b$.

2 The calculus

The basic fixed-point theorem, and our main reason for confining attention to complete lattices, is the Knaster-Tarski theorem dealing with the *existence* of extreme fixed points.

(1) **[Knaster-Tarski]** Every monotonic endofunction (on a complete lattice) has a least fixed point, which coincides with its least prefix point ("x is prefix point of f" means $f.x \sqsubseteq x$). Thus μf is characterized by

[computation] $f\,.\,\mu f = \mu f$.

[induction] $\mu f \sqsubseteq x \;\Leftarrow\; f.x \sqsubseteq x$ for all x .
□

As a consequence of its shape, the induction rule is the first candidate for application in case of demonstranda of the form "$\mu f \sqsubseteq x$".

Three further useful, simple, and probably well-known, fixed-point rules are

(2) **[μ monotonic]** $\mu f \sqsubseteq \mu g \;\Leftarrow\; f \dot{\sqsubseteq} g$.

(3) **[rolling rule]** $\mu(f \circ g) = f\,.\,\mu(g \circ f)$.

(4) **[diagonal rule]** $\mu\langle x \mapsto x\oplus x\rangle = \mu\langle x \mapsto \mu\langle y \mapsto x\oplus y\rangle\rangle$, for every (monotonic) binary operator $\oplus$.

Proofs of these rules is left to the reader.

3 μ-Fusion and Galois connections

As remarked earlier, we know how to deal with demonstranda like $\mu f \sqsubseteq x$. We have, for instance,

(5) [**simple μ-fusion**] $\mu f \sqsubseteq g \,.\, \mu h \;\Leftarrow\; f \circ g \dot{\sqsubseteq} g \circ h$.

What, however, should one do with demonstranda like $f \,.\, \mu g \sqsubseteq \mu h$ (and $f \,.\, \mu g = \mu h$), in which the f- application has to disappear in order to make the formula fit for μg-induction? This is where Galois connections enter the picture.

(6) [**Galois connection**] Functions F and G form a Galois connection with respect to $\sqsubseteq$ and $\leq$ precisely when, for all x and y,

$$F.x \sqsubseteq y \;\equiv\; x \leq G.y \quad .$$

Function G is the *upper adjoint* of F, denoted by $F^{\#}$; dually, F is the *lower adjoint* $G^{\flat}$ of G. (Note that $\#$ and $\flat$ can be defined because adjoints are unique.)
□

The two most frequently used properties of Galois connections are

(7) [**cancellation**] Functions F and G that form a Galois connection (6) satisfy $F \circ G \dot{\sqsubseteq} id$ and $id \dot{\leq} G \circ F$, where id denotes the (appropriate) identity function.

(8) [**adjoint**] A function is a lower adjoint precisely when it distributes over all suprema; a function is an upper adjoint precisely when it distributes over all infima.

Now we are ready to formulate and prove the μ-fusion rule.

(9) [**μ-fusion**] Functions f, g, and h, where f is a lower adjoint, satisfy

(a) $f \,.\, \mu g \sqsubseteq \mu h \;\Leftarrow\; f \circ g \dot{\sqsubseteq} h \circ f$

(b) $f \,.\, \mu g = \mu h \;\Leftarrow\; f \circ g = h \circ f$.

Proof (b) follows from (a) and simple μ-fusion. For (a) we calculate

$$\begin{array}{ll} & f \,.\, \mu g \sqsubseteq \mu h \\ \equiv & \quad \{ \quad \text{Galois connection (6), with } F := f \quad \} \\ & \mu g \leq f^{\#} \,.\, \mu h \\ \Leftarrow & \quad \{ \quad \text{simple } \mu\text{-fusion (5)} \quad \} \\ & g \circ f^{\#} \dot{\leq} f^{\#} \circ h \\ \equiv & \quad \{ \quad \text{identities} \quad \} \\ & id \circ g \circ f^{\#} \dot{\leq} f^{\#} \circ h \circ id \end{array}$$

$$
\begin{array}{ll}
\Leftarrow & \{ \quad \text{cancellation (twice): } id \dot{\leq} f^\# \circ f \text{ and } f \circ f^\# \dot{\sqsubseteq} id; \\
& \quad f^\# \circ h \text{ is monotonic} \quad \} \\
& (f^\# \circ f) \circ g \circ f^\# \dot{\leq} f^\# \circ h \circ (f \circ f^\#) \\
\Leftarrow & \{ \quad \circ \text{ is associative and } f^\# \text{ is monotonic} \quad \} \\
& f \circ g \dot{\sqsubseteq} h \circ f \quad .
\end{array}
$$

□

Note that in order to apply the μ-fusion theorem we do not need to *know* the upper adjoint $f^\#$ of function f, we only need to know that it exists. Property (8) was included because it constitutes the most common way to guarantee that existence.

As an aid in memorizing the μ-fusion theorems note that the order of f, g, $\sqsubseteq$ or $=$, and h is the same in the consequent and the antecedent, be it that in the antecedent the lower adjoint occurs twice, in different argument positions of $\circ$.

We conclude the presentation of the calculus with a useful corollary of the rules presented so far.

(10) **[exchange rule]** Functions f,g, and h, where g and h are lower adjoints, satisfy

$$\mu(f \circ g) = \mu(f \circ h) \Leftarrow g \circ f \circ h = h \circ f \circ g \quad .$$

The calculus of greatest fixed points is obtained from the above rules for least fixed points by the interchanges $\mu \leftrightarrow \nu$, $\sqsubseteq \leftrightarrow \sqsupseteq$, and lower adjoint $\leftrightarrow$ upper adjoint.

4 The calculus in action I

Any fixed point of a function that maps pairs of values to pairs of values is, of course, a pair. In this section we illustrate the calculus by showing how to calculate the individual components of the least fixed point of such a function, given that it is possible to calculate least fixed points of functions mapping single elements to single elements. The fixed-point equation to be dealt with is the following.

(11) $x\ ,\ y \ :: \ x = x \odot y \ \wedge \ y = x \otimes y \quad .$

We first consider two special cases in which $\odot$ and $\otimes$ depend on only one of their arguments .

Lemma 12

(a) $\mu\langle(x\ ,\ y)\ \mapsto\ (f.x\ ,\ g.y)\rangle\ =\ (\mu f\ ,\ \mu g)$

(b) $\mu\langle(x\ ,\ y)\ \mapsto\ (f.y\ ,\ g.x)\rangle\ =\ (\mu(f\circ g)\ ,\ \mu(g\circ f))$.

Proof of (a). With Π_1 denoting projection on the first component, we have to prove

(13) $\Pi_1\,.\,\mu\langle(x\ ,\ y)\ \mapsto\ (f.x\ ,\ g.y)\rangle\ =\ \mu f$.

Since Π_1 distributes over all suprema, this is an occasion for applying μ-fusion: doing so, we see that (13) follows from

$$\Pi_1.(f.x\ ,\ g.y)\ =\ f.\Pi_1.(x\ ,\ y)\quad \text{for all } x \text{ and } y,$$

which is true.

The second component is dealt with similarly.
□

Proof of (b)

$$\begin{array}{ll}
 & \mu\langle(x\ ,\ y)\ \mapsto\ (f.y\ ,\ g.x)\rangle\ =\ (\mu(f\circ g)\ ,\ \mu(g\circ f)) \\
\equiv & \quad \{\quad \text{(a) on RHS}\quad \} \\
 & \mu\langle(x\ ,\ y)\ \mapsto\ (f.y\ ,\ g.x)\rangle\ =\ \mu\langle(x\ ,\ y)\ \mapsto\ (f.g.x\ ,\ g.f.y)\rangle \\
\equiv & \quad \{\quad \text{define } \phi\text{: } \phi(x\ ,\ y) = (f.y\ ,\ g.x)\quad \} \\
 & \mu\phi = \mu(\phi\circ\phi) \\
\equiv & \quad \{\quad \mu(\phi\circ\phi)\sqsubseteq\mu\phi\quad \} \\
 & \mu\phi\sqsubseteq\mu(\phi\circ\phi) \\
\Leftarrow & \quad \{\quad \text{induction on } \phi\quad \} \\
 & \phi\,.\,\mu(\phi\circ\phi)\sqsubseteq\mu(\phi\circ\phi) \\
\equiv & \quad \{\quad \text{rolling rule}\quad \} \\
 & \mathit{true}\ .
\end{array}$$

□

With the aid of lemma (12), we now compute the least solution of (11), viz. we prove

Lemma 14

$$\mu\langle(x\ ,\ y)\ \mapsto\ (x\odot y\ ,\ x\otimes y)\rangle\ =\ (\mu\langle x\ \mapsto\ x\odot p.x\rangle\ ,\ \mu\langle y\ \mapsto\ q.y\otimes y\rangle)\ \ ,$$

where $p.x = \mu\langle v\ \mapsto\ x\otimes v\rangle$ and $q.y = \mu\langle u\ \mapsto\ u\odot y\rangle$, i.e. $p.x$ and $q.y$ are the least fixed points of the individual equations.

Proof

$$\begin{array}{ll}
 & \mu\langle(x\ ,\ y)\ \mapsto\ (x\odot y\ ,\ x\otimes y)\rangle \\
= & \quad \{\quad \text{diagonal rule, (heading for lemma (12a)) with } x\ :=\ (x\ ,\ y) \text{ and} \\
 & \qquad \oplus \text{ defined by } (u\ ,\ v)\oplus(x\ ,\ y) = (u\odot y\ ,\ x\otimes v)\quad \} \\
 & \mu\langle(x\ ,\ y)\ \mapsto\ \mu\langle(u\ ,\ v)\ \mapsto\ (u\odot y\ ,\ x\otimes v)\rangle\rangle \\
= & \quad \{\quad \text{lemma (12a) and definition of } p \text{ and } q\quad \} \\
 & \mu\langle(x\ ,\ y)\ \mapsto\ (q.y\ ,\ p.x)\rangle \\
= & \quad \{\quad \text{lemma (12b)}\quad \} \\
 & (\mu\langle x\ \mapsto\ q.p.x\rangle\ ,\ \mu\langle y\ \mapsto\ p.q.y\rangle) \\
= & \quad \{\quad \text{definition } q,\ p\quad \} \\
 & (\mu\langle x\ \mapsto\ \mu\langle u\ \mapsto\ u\odot p.x\rangle\rangle\ ,\ \mu\langle y\ \mapsto\ \mu\langle v\ \mapsto\ q.y\otimes v\rangle\rangle) \\
= & \quad \{\quad \text{diagonal rule twice: } u := x,\ v\ :=\ y\quad \} \\
 & (\mu\langle x\ \mapsto\ x\odot p.x\rangle\ ,\ \mu\langle y\ \mapsto\ q.y\otimes y\rangle)\ \ .
\end{array}$$

□

5 The calculus in action II

Next we consider an algebra $(+\ ,\ 0\ ,\ \cdot\ ,\ 1)$, where $+$ denotes binary supremum with identity 0 in a complete lattice and $\cdot$ is an associative composition operator with identity 1 that distributes over all suprema. Note that, in view of theorem (8), we have

(15) Functions $(b\cdot)$ and $(\cdot b)$ are lower adjoints.

Operator $*$ is defined by

(16) [**definition** $b*$] $\quad b* = \mu\langle x\ \mapsto\ 1+x\cdot b\rangle\quad .$

Given this fixed-point definition of $b*$, the calculus now helps us prove all kinds of basic properties in regular algebra very concisely, such as

(17) $[a \cdot b*] \quad a \cdot b* = \mu\langle x \;\mapsto\; a + x \cdot b\rangle$.

(18) **[star decomposition]** $\quad (a+b)* = b* \cdot (a \cdot b*)*$.

Note that (17) immediately invites the use of μ-fusion. Its proof reads

$$
\begin{array}{ll}
 & a \cdot b* = \mu\langle x \;\mapsto\; a + x \cdot b\rangle \\
\Leftarrow & \qquad \{ \quad \mu\text{-fusion, } (a\cdot) \text{ is a lower adjoint, } b* = \mu\langle x \;\mapsto\; 1 + x \cdot b\rangle \quad \} \\
 & \forall(x ::\; a \cdot (1 + x \cdot b) = a + (a \cdot x) \cdot b) \\
\equiv & \qquad \{ \quad (a\cdot) \text{ over } +, \text{ associativity of } \cdot \quad \} \\
 & \mathit{true} \quad .
\end{array}
$$

The proof of star decomposition is a prize application of the diagonal rule:

$$
\begin{array}{ll}
 & (a+b)* \\
= & \qquad \{ \quad \text{definition of } * \quad \} \\
 & \mu\langle x \;\mapsto\; 1 + x \cdot (a+b)\rangle \\
= & \qquad \{ \quad \cdot \text{ over } + \text{ and diagonal rule} \quad \} \\
 & \mu\langle x \;\mapsto\; \mu\langle y \;\mapsto\; 1 + x \cdot a + y \cdot b\rangle\rangle \\
= & \qquad \{ \quad \text{associativity of } +\text{, (17) with } a \;:=\; 1 + x \cdot a \quad \} \\
 & \mu\langle x \;\mapsto\; (1 + x \cdot a) \cdot b*\rangle \\
= & \qquad \{ \quad \cdot \text{ over } + \quad \} \\
 & \mu\langle x \;\mapsto\; b* + x \cdot a \cdot b*\rangle \\
= & \qquad \{ \quad \text{associativity of } \cdot \text{ and (17)} \quad \} \\
 & b* \cdot (a \cdot b*)* \quad .
\end{array}
$$

Note how the associativity of the operators guides the construction of the above proofs: if a calculational step creates the possibility for regrouping arguments, the next step is to be performed on a subexpression that is the result of such a regrouping. This is a phenomenon we encounter over and over again.

The theorems given above suffice to prove other well-known properties like $b* \cdot b* = b*$, $(b*)* = b*$, etc.. The "dual" $*b$ of $b*$ is defined by

(19) [**definition** $*b$] $*b = \mu\langle x \;\mapsto\; 1 + b\cdot x\rangle$.

Its most prominent property $b* = *b$ is an immediate corollary of the leapfrog rule, which reads

(20) [**leapfrog**] $a\cdot(b\cdot a)* = *(a\cdot b)\cdot a$.

The exchange rule (10) comes in handy for the proof of the above leapfrog rule . We conclude this section with an exercise :

(21) $*b\cdot a* = \mu\langle x \;\mapsto\; 1 + x\cdot a + b\cdot x\rangle$.

6 The calculus in action III

In the preceding section we investigated the least solution of equation $x \;::\; x = a + x\cdot b$. Here we consider its largest solution $\nu\langle x \;\mapsto\; a + x\cdot b\rangle$. In particular we do so for a lattice that is completely distributive, so that, among other, things $(y+)$ and $(+y)$ distribute over all infima and, hence, are upper adjoints. Then ν-fusion yields a simple proof of the following theorem.

(22) If $(y+)$ is an upper adjoint, then we have, for all a and b,

$$\nu\langle x \;\mapsto\; a + x\cdot b\rangle = y + \nu\langle x \;\mapsto\; x\cdot b\rangle \;\Leftarrow\; y = a + y\cdot b \quad .$$

Proof

$$\begin{array}{ll} & \nu\langle x \;\mapsto\; a + x\cdot b\rangle = y + \nu\langle x \;\mapsto\; x\cdot b\rangle \\ \Leftarrow & \qquad \{ \quad (y+) \text{ is upper adjoint: } \nu\text{-fusion} \quad \} \\ & \forall(x ::\; a + (y + x)\cdot b = y + x\cdot b) \\ \Leftarrow & \qquad \{ \quad (\cdot b) \text{ over } + \text{ , associativity of } + \quad \} \\ & a + y\cdot b = y \quad . \end{array}$$

□

In other words, if + distributes over all infima, the *largest* solution of inhomogeneous equation $x \;::\; x = a + b\cdot x$ is the sum (i.e. supremum) of an arbitrary solution and the

largest solution of the "homogeneous" equation. Note that a special choice for y in theorem (22) is $y = a \cdot b*$.

An immediate corollary of theorem (22) is that if $\nu\langle x \mapsto x \cdot b\rangle = 0$, function $x \mapsto a + x \cdot b$ has a unique fixed point. This is the rule we call the Unique Extension Property (UEP) of Regular Algebra. It was first mentioned in [3]. In view of (17), the UEP can be formulated as

(23) **[UEP of regular algebra]** If $\nu\langle x \mapsto x \cdot b\rangle = 0$, then for all a and x

$$a + x \cdot b = x \quad \equiv \quad x = a \cdot b* \quad .$$

□

The UEP shows the importance of property $\nu\langle x \mapsto x \cdot b\rangle = 0$. In language theory it has the interpretation that "b does not possess the empty-word property". In relation algebra we say "b is well-founded": the property expresses that there are no infinite sequences of b-related elements (thus, if relation b represents a finite directed graph, $\nu\langle x \mapsto x \cdot b\rangle = 0$ means that the graph is acyclic).

The above UEP has many applications in programming, because equations like $a + x \cdot b = x$ and $a + x \cdot b \sqsubseteq x$ abound. Inductively defined sets, for instance, provide an example: the base is represented by a and the step by $(\cdot\, b)$. If applicable, the UEP then expresses that $x \mapsto a + x \cdot b$ has a unique fixed point, as a result of which it is irrelevant whether the semantics is a least-fixed-point or greatest-fixed-point or any other fixed-point-semantics.

As a more detailed example we consider recursively defined functional programs, viz. we consider the following recursive definition of the factorial function and transform it into another one:

(24)
$$\begin{array}{lcl} fac.0 & = & 1 \\ fac.(n+1) & = & (n+1) * fac.n \quad , \text{for } n \geq 0 \quad . \end{array}$$

First we show that a relational representation F of fac can be constructed that satisfies $F = T \sqcup F \circ R$, for some T and some well-founded R. (As a result the UEP is applicable, with $+$ instantiated to relational union $\sqcup$ and $\cdot$ instantiated to relational composition). With F and R defined by

$$\begin{array}{rcl} z\langle F\rangle(n\ ,\ y) & \equiv & y = fac.n \quad , \text{for all } z \\ (a\ ,\ b)\langle R\rangle(n\ ,\ y) & \equiv & a + 1 = n \ \wedge \ (a+1) * b = y \quad , \end{array}$$

we have $(n-1\ ,\ fac.(n-1))\langle R\rangle(n\ ,\ fac.n)$ and so some relational calculus yields

$$z\langle F\rangle(n\ ,\ y) \quad \equiv \quad (0\ ,\ 1) = (n\ ,\ y) \ \vee \ z\langle F \circ R\rangle(n\ ,\ y) \quad .$$

Hence $F = T \sqcup F \circ R$, for T defined by

$$z\langle T\rangle(n\ ,\ y) \quad \equiv \quad (0\ ,\ 1) = (n\ ,\ y) \quad \text{, for all } z.$$

Note that, indeed, R is well-founded : there are no (left-) infinite sequences of R-related elements. So by the UEP, $F = T \circ R*$. Exploiting $R* = *R$ (!), we see that $F = T \circ Q$, where $Q = *R$, that is, Q is the (least) solution of

$$Q \ = \ I \sqcup R \circ Q \ .$$

Using the definitions of F and T, $F = T \circ Q$ reads

$$(25) \quad y = fac.n \quad \equiv \quad (0\ ,\ 1)\langle Q\rangle(n\ ,\ y)$$

and, using R, $Q = I \sqcup R \circ Q$ reads

$$(26) \quad (a\ ,\ b)\langle Q\rangle(n\ ,\ y) \quad \equiv \quad (a\ ,\ b) = (n\ ,\ y) \ \vee \ (a+1\ ,\ (a+1) * b)\langle Q\rangle(n\ ,\ y) \ .$$

Writing $(a\ ,\ b)\langle Q\rangle(n\ ,\ y)$ as $y = q_n.(a\ ,\ b)$, (25) and (26) are simplified to

$$fac.n = q_n.(0\ ,\ 1)$$

where

$$\begin{array}{lll} q_n.(a\ ,\ b) = & \text{if} & \\ & \quad n = a & \rightarrow \quad b \\ & \Box & \\ & \quad n \neq a & \rightarrow \quad q_n.(a+1\ ,\ (a+1) * b) \\ & \text{fi} \ . & \end{array}$$

The latter is quite a different recursive definition of factorial from (24).

This example was inspired by Augusteijn [2]. The transformation given here is a special case of a more general transformation of a recursive-descent algorithm into a recursive-ascent algorithm. For more on relational calculus —and Galois connections— see Aarts *et al.* [1].

7 Epilogue

We have found it encouraging to experience how a small calculus like the one presented here can have a variety of applications (although inevitably it also has its limitations). Since fixed points are so prominent in computing, it is our hope that the use of such a calculus may help in making program design (yet) more calculational.

Acknowledgements

Thanks go to David Gries and Rustan Leino for their helpful comments.

References

[1] C.J. Aarts, R.C. Backhouse, P. Hoogendijk, T.S. Voermans, and J. van der Woude. A relational theory of datatypes. Available via anonymous ftp from `ftp.win.tue.nl` in directory `pub/math.prog.construction`, September 1992.

[2] A. Augusteijn. Functional Programming, Program Transformations and Compiler Construction. Ph.D. Thesis. Eindhoven, 1993.

[3] Backhouse, R.C. and B.A. Carré. Regular Algebra applied to path-finding problems. Journal of the Institute of Mathematics and its Applications, Vol 15, 161–186, 1975.

Computing Science Reports

Department of Mathematics and Computing Science
Eindhoven University of Technology

In this series appeared:

91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.
91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	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	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
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 complete ness, 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 proces 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	Techniques for designing efficient parallel programs, p. 14.
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.

91/35	F.S. de Boer J.W. Klop C. Palamidessi	Asynchronous communication in process algebra, p. 20.
92/01	J. Coenen J. Zwiers W.-P. de Roever	A note on compositional refinement, p. 27.
92/02	J. Coenen J. Hooman	A compositional semantics for fault tolerant real-time systems, p. 18.
92/03	J.C.M. Baeten J.A. Bergstra	Real space process algebra, p. 42.
92/04	J.P.H.W.v.d.Eijnde	Program derivation in acyclic graphs and related problems, p. 90.
92/05	J.P.H.W.v.d.Eijnde	Conservative fixpoint functions on a graph, p. 25.
92/06	J.C.M. Baeten J.A. Bergstra	Discrete time process algebra, p.45.
92/07	R.P. Nederpelt	The fine-structure of lambda calculus, p. 110.
92/08	R.P. Nederpelt F. Kamareddine	On stepwise explicit substitution, p. 30.
92/09	R.C. Backhouse	Calculating the Warshall/Floyd path algorithm, p. 14.
92/10	P.M.P. Rambags	Composition and decomposition in a CPN model, p. 55.
92/11	R.C. Backhouse J.S.C.P.v.d.Woude	Demonic operators and monotype factors, p. 29.
92/12	F. Kamareddine	Set theory and nominalisation, Part I, p.26.
92/13	F. Kamareddine	Set theory and nominalisation, Part II, p.22.
92/14	J.C.M. Baeten	The total order assumption, p. 10.
92/15	F. Kamareddine	A system at the cross-roads of functional and logic programming, p.36.
92/16	R.R. Seljée	Integrity checking in deductive databases; an exposition, p.32.
92/17	W.M.P. van der Aalst	Interval timed coloured Petri nets and their analysis, p. 20.
92/18	R.Nederpelt F. Kamareddine	A unified approach to Type Theory through a refined lambda-calculus, p. 30.
92/19	J.C.M.Baeten J.A.Bergstra S.A.Smolka	Axiomatizing Probabilistic Processes: ACP with Generative Probabilities, p. 36.
92/20	F.Kamareddine	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.
92/23	F.Kamareddine E.Klein	Nominalization, Predication and Type Containment, p. 40.
92/24	M.Codish D.Dams Eyal Yardeni	Bottum-up Abstract Interpretation of Logic Programs, p. 33.
92/25	E.Poll	A Programming Logic for Fω, p. 15.
92/26	T.H.W.Beelen W.J.J.Stut P.A.C.Verkoulen	A modelling method using MOVIE and SimCon/ExSpect, p. 15.
92/27	B. Watson G. Zwaan	A taxonomy of keyword pattern matching algorithms, p. 50.
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. Zwietering	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.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach

		Part V: Specification Language, p. 89.
93/14	J.C.M. Baeten J.A. Bergstra	On Sequential Composition, Action Prefixes and Process Prefix, p. 21.
93/15	J.C.M. Baeten J.A. Bergstra R.N. Bol	A Real-Time Process Logic, p. 31.
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.
93/20	F.S. de Boer	A Process Algebra of Concurrent Constraint Programming, p. 15.
93/21	M. Codish D. Dams G. Filé M. Bruynooghe	Freeness Analysis for Logic Programs - And Correctness?, p. 24.
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.
93/25	H. Schepers and R. Gerth	A Compositional Proof Theory for Fault Tolerant Real-Time Distributed Systems, p. 31.
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. Müller	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.
93/32	H. ten Eikelder and H. van Geldrop	On the Correctness of some Algorithms to generate Finite Automata for Regular Expressions, p. 17.

93/33	L. Loyens and J. Moonen	ILIAS, a sequential language for parallel matrix computations, p. 20.
93/34	J.C.M. Baeten and J.A. Bergstra	Real Time Process Algebra with Infinitesimals, p.39.
93/35	W. Ferrer and P. Severi	Abstract Reduction and Topology, p. 28.
93/36	J.C.M. Baeten and J.A. Bergstra	Non Interleaving Process Algebra, p. 17.
93/37	J. Brunekreef J-P. Katoen R. Koymans S. Mauw	Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.
93/38	C. Verhoef	A general conservative extension theorem in process algebra, p. 17.
93/39	W.P.M. Nuijten E.H.L. Aarts D.A.A. van Erp Taalman Kip K.M. van Hee	Job Shop Scheduling by Constraint Satisfaction, p. 22.
93/40	P.D.V. van der Stok M.M.M.P.J. Claessen D. Alstein	A Hierarchical Membership Protocol for Synchronous Distributed Systems, p. 43.
93/41	A. Bijlsma	Temporal operators viewed as predicate transformers, p. 11.
93/42	P.M.P. Rambags	Automatic Verification of Regular Protocols in P/T Nets, p. 23.
93/43	B.W. Watson	A taxomomy of finite automata construction algorithms, p. 87.
93/44	B.W. Watson	A taxonomy of finite automata minimization algorithms, p. 23.
93/45	E.J. Luit J.M.M. Martin	A precise clock synchronization protocol,p.
93/46	T. Kloks D. Kratsch J. Spinrad	Treewidth and Patwidth of Cocomparability graphs of Bounded Dimension, p. 14.
93/47	W. v.d. Aalst P. De Bra G.J. Houben Y. Kornatzky	Browsing Semantics in the "Tower" Model, p. 19.
93/48	R. Gerth	Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.

94/01	P. America M. van der Kammen 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 Π-conversion, p. 51.
94/03	L.B. Hartman K.M. van Hee	Application of Marcov Decision Processe to Search Problems, p. 21.
94/04	J.C.M. Baeten J.A. Bergstra	Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.
94/05	P. Zhou J. Hooman	Formal Specification and Compositional Verification of an Atomic Broadcast Protocol, p. 22.
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 R. Bol	Logic Programming and Negation: A Survey, p. 62.
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 C. Petersohn	A Comparison of Ward & Mellor's Transformation Schema with State- & Activitycharts, p. 30.
94/12	T. Kloks D. Kratsch H. Müller	Dominoes, p. 14.
94/13	R. Seljée	A New Method for Integrity Constraint checking in Deductive Databases, p. 34.
94/14	W. Peremans	Ups and Downs of Type Theory, p. 9.
94/15	R.J.M. Vaessens E.H.L. Aarts J.K. Lenstra	Job Shop Scheduling by Local Search, p. 21.
94/16	R.C. Backhouse H. Doombos	Mathematical Induction Made Calculational, p. 36.
94/17	S. Mauw M.A. Reniers	An Algebraic Semantics of Basic Message Sequence Charts, p. 9.

94/18	F. Kamareddine R. Nederpelt	Refining Reduction in the Lambda Calculus, p. 15.
94/19	B.W. Watson	The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.
94/20	R. Bloo F. Kamareddine R. Nederpelt	Beyond β-Reduction in Church's λ→, p. 22.
94/21	B.W. Watson	An introduction to the Fire engine: A C++ toolkit for Finite automata and Regular Expressions.
94/22	B.W. Watson	The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.
94/23	S. Mauw and M.A. Reniers	An algebraic semantics of Message Sequence Charts, p. 43.
94/24	D. Dams O. Grumberg R. Gerth	Abstract Interpretation of Reactive Systems: Abstractions Preserving ∀CTL*, ∃CTL* and CTL*, p. 28.
94/25	T. Kloks	$K_{1,3}$-free and W_4-free graphs, p. 10.
94/26	R.R. Hoogerwoord	On the foundations of functional programming: a programmer's point of view, p. 54.
94/27	S. Mauw and H. Mulder	Regularity of BPA-Systems is Decidable, p. 14.
94/28	C.W.A.M. van Overveld M. Verhoeven	Stars or Stripes: a comparative study of finite and transfinite techniques for surface modelling, p. 20.
94/29	J. Hooman	Correctness of Real Time Systems by Construction, p. 22.
94/30	J.C.M. Baeten J.A. Bergstra Gh. Ştefanescu	Process Algebra with Feedback, p. 22.
94/31	B.W. Watson R.E. Watson	A Boyer-Moore type algorithm for regular expression pattern matching, p. 22.
94/32	J.J. Vereijken	Fischer's Protocol in Timed Process Algebra, p. 38.
94/33	T. Laan	A formalization of the Ramified Type Theory, p.40.
94/34	R. Bloo F. Kamareddine R. Nederpelt	The Barendregt Cube with Definitions and Generalised Reduction, p. 37.
94/35	J.C.M. Baeten S. Mauw	Delayed choice: an operator for joining Message Sequence Charts, p. 15.
94/36	F. Kamareddine R. Nederpelt	Canonical typing and Π-conversion in the Barendregt Cube, p. 19.

94/37	T. Basten R. Bol M. Voorhoeve	Simulating and Analyzing Railway Interlockings in ExSpect, p. 30.
94/38	A. Bijlsma C.S. Scholten	Point-free substitution, p. 10.
94/39	A. Blokhuis T. Kloks	On the equivalence covering number of splitgraphs, p. 4.
94/40	D. Alstein	Distributed Consensus and Hard Real-Time Systems, p. 34.
94/41	T. Kloks D. Kratsch	Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph, p. 6.
94/42	J. Engelfriet	Concatenation of Graphs, p. 7.
94/43	R.C. Backhouse Bijsterveld	Category Theory as Coherently Constructive Lattice M . Theory: An Illustration, p. 35.
94/44	E. Brinksma J. Davies R. Gerth S. Graf W. Janssen B. Jonsson S. Katz G. Lowe M. Poel A. Pnueli C. Rump J. Zwiers	Verifying Sequentially Consistent Memory, p. 160
94/45	G.J. Houben	Tutorial voor de ExSpect-bibliotheek voor "Administratieve Logistiek", p. 43.
94/46	R. Bloo F. Kamareddine R. Nederpelt	The λ-cube with classes of terms modulo conversion, p. 16.
94/47	R. Bloo F. Kamareddine R. Nederpelt	On Π-conversion in Type Theory, p. 12.