Fixed-point calculus

Mathematics of Program Construction Group

Published: 01/01/1994

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
Eindhoven University of Technology

Department of Mathematics and Computing Science

Fixed-Point Calculus

by

Mathematics of Program Construction Group

Computing Science Report 94/48
Eindhoven, October 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

- \( \mu f \) and \( \mu(\, x \mapsto f.x) \) denote the least fixed point of function \( f : x \mapsto f.x \), which maps \( x \) to \( f.x \)
• lifting: for any binary relation \( \sqsubseteq \) the lifted version \( \sqsubseteq \) is defined by

\[
f \sqsubseteq g \iff \forall (x :: f.x \sqsubseteq g.x)
\]

• 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 \( \mu f \) is characterized by

\[\text{[computation]} \quad f.\mu f = \mu f \]

\[\text{[induction]} \quad \mu f \sqsubseteq x \iff f.x \sqsubseteq x \quad \text{for all } x .\]

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) [\( \mu \) monotonic] \( \mu f \sqsubseteq \mu y \iff f \sqsubseteq g \)

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

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

Proofs of these rules is left to the reader.

3 \( \mu \)-Fusion and Galois connections

As remarked earlier, we know how to deal with demonstranda like \( \mu f \sqsubseteq x \). We have, for instance,
What, however, should one do with demonstranda like $f \cdot \mu g \subseteq \mu h$ (and $f \cdot \mu g = \mu h$), in which the $f$-application has to disappear in order to make the formula fit for $\mu g$-induction? This is where Galois connections enter the picture.

Functions $F$ and $G$ form a Galois connection with respect to $\sqsubseteq$ and $\leq$ precisely when, for all $x$ and $y$,

$$F.x \sqsubseteq y \iff x \leq G.y .$$

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

The two most frequently used properties of Galois connections are

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

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 $\mu$-fusion rule.

Functions $f$, $g$, and $h$, where $f$ is a lower adjoint, satisfy

(a) $f \cdot \mu g \subseteq \mu h \iff f \circ g \subseteq h \circ f$

(b) $f \cdot \mu g = \mu h \iff f \circ g = h \circ f$ .

Proof (b) follows from (a) and simple $\mu$-fusion. For (a) we calculate

$$f \cdot \mu g \subseteq \mu h$$

$$\iff \{ \text{Galois connection (6), with } F := f \}$$

$$\mu g \leq f^\# \cdot \mu h$$

$$\iff \{ \text{simple } \mu\text{-fusion (5)} \}$$

$$g \circ f^\# \leq f^\# \circ h$$

$$\iff \{ \text{identities} \}$$

$$\text{id} \circ g \circ f^\# \leq f^\# \circ h \circ \text{id}$$
\( \Leftarrow \quad \begin{align*}
\text{cancellation (twice): } \text{id} & \leq f^\# \circ f \text{ and } f \circ f^\# \subseteq \text{id} ; \\
& f^\# \circ h \text{ is monotonic } \\
(f^\# \circ f) \circ g \circ f^\# & \leq f^\# \circ h \circ (f \circ f^\#) \\
\Rightarrow \quad \begin{align*}
\circ \text{ is associative and } f^\# & \text{ is monotonic } \\
f \circ g & \subseteq h \circ f
\end{align*}
\end{align*} \)

Note that in order to apply the \( \mu \)-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 \( \mu \)-fusion theorems note that the order of \( f, g, \subseteq \) 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

\[
\mu(f \circ g) = \mu(f \circ h) \Leftarrow g \circ f \circ h = h \circ f \circ g
\]

The calculus of greatest fixed points is obtained from the above rules for least fixed points by the interchanges \( \mu \leftrightarrow \nu, \subseteq \leftrightarrow \supseteq \), and lower adjoint \( \rightarrow \) 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) \( x, y :: x = x \odot y \land y = x \odot y \).

We first consider two special cases in which \( \odot \) and \( \odot \) depend on only one of their arguments.
Lemma 12

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

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

Proof of (a). With \( \Pi_1 \) denoting projection on the first component, we have to prove

\[(13) \quad \Pi_1 \cdot \mu((x, y) \mapsto (f.x, g.y)) = \mu f .\]

Since \( \Pi_1 \) distributes over all suprema, this is an occasion for applying \( \mu \)-fusion: doing so, we see that (13) follows from

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

which is true.

The second component is dealt with similarly.

\[\square\]

Proof of (b)

\[
\mu((x, y) \mapsto (f.y, g.x)) = (\mu(f \circ g), \mu(g \circ f)) \\
\equiv \{ \text{(a) on RHS} \} \\
\mu((x, y) \mapsto (f.y, g.x)) = \mu((x, y) \mapsto (f.g.x, g.f.y)) \\
\equiv \{ \text{define } \phi: \phi(x, y) = (f.y, g.x) \} \\
\mu \phi = \mu(f \circ \phi) \\
\equiv \{ \mu(f \circ \phi) \subseteq \mu \phi \} \\
\mu \phi \subseteq \mu(f \circ \phi) \\
\leftarrow \{ \text{induction on } \phi \} \\
\phi \cdot \mu(f \circ \phi) \subseteq \mu(f \circ \phi) \\
\equiv \{ \text{rolling rule} \} \\
\text{true} .
\]

\[\square\]

With the aid of lemma (12), we now compute the least solution of (11), viz. we prove
Lemma 14

\[ \mu((x, y) \mapsto (x \otimes y, x \otimes y)) = (\mu(x \mapsto x \odot p.x), \mu(y \mapsto q.y \otimes y)) \]

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

Proof

\[
\begin{align*}
\mu((x, y) & \mapsto (x \otimes y, x \otimes y)) \\
& = \{ \text{diagonal rule, (heading for lemma (12a)) with } x := (x, y) \text{ and} \\
& \quad \odot \text{ defined by } (u, v) \odot (x, y) = (u \odot y, x \odot v) \} \\
& \mu((x, y) \mapsto \mu((u, v) \mapsto (u \odot y, x \odot v)))) \\
& = \{ \text{lemma (12a) and definition of } p \text{ and } q \} \\
& \mu((x, y) \mapsto (q.y, p.x)) \\
& = \{ \text{lemma (12b)} \} \\
& (\mu(x \mapsto q.p.x), \mu(y \mapsto p.q.y)) \\
& = \{ \text{definition } q, p \} \\
& (\mu(x \mapsto \mu(u \mapsto u \odot p.x)), \mu(y \mapsto \mu(v \mapsto q.y \odot v))) \\
& = \{ \text{diagonal rule twice: } u := x, v := y \} \\
& (\mu(x \mapsto x \odot p.x), \mu(y \mapsto q.y \odot y)) \\
\end{align*}
\]

\[ \Box \]

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 \( b^* \)] \( b^* = \mu(x \mapsto 1 + x \cdot b) \) .
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) $[a \cdot b^*] \quad a \cdot b^* = \mu(x \mapsto a + x \cdot b)$.

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

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

$$a \cdot b^* = \mu(x \mapsto a + x \cdot b)$$
$$\Leftarrow \quad \{ \mu\text{-fusion, } (a \cdot) \text{ is a lower adjoint, } b^* = \mu(x \mapsto 1 + x \cdot b) \}$$
$$\forall(x :: a \cdot (1 + x \cdot b) = a + (a \cdot x) \cdot b)$$
$$\Rightarrow \quad \{ (a \cdot) \text{ over } +, \text{ associativity of } \cdot \}$$
$$\text{true}.$$

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

$$(a+b)^*$$
$$= \quad \{ \text{definition of } \ast \}$$
$$\mu(x \mapsto 1 + x \cdot (a+b))$$
$$= \quad \{ \cdot \text{ over } + \text{ and diagonal rule } \}$$
$$\mu(x \mapsto \mu(y \mapsto 1 + x \cdot a + y \cdot b))$$
$$= \quad \{ \text{associativity of } +, (17) \text{ with } a := 1 + x \cdot a \}$$
$$\mu(x \mapsto (1 + x \cdot a) \cdot b^*)$$
$$= \quad \{ \cdot \text{ over } + \}$$
$$\mu(x \mapsto b^* + x \cdot a \cdot b^*)$$
$$= \quad \{ \text{associativity of } \cdot \text{ and } (17) \}$$
$$b^* \cdot (a \cdot b^*)^*.$$

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" $\ast b$ of $b^*$ is defined by
(19) [definition \( *b \)] \( *b = \mu(x \mapsto 1 + b \cdot x) \).

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) \( *b \cdot a* = \mu(x \mapsto 1 + x \cdot a + b \cdot x) \).

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 \( \nu(x \mapsto a + x \cdot b) \). 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 \( \nu\)-fusion yields a simple proof of the following theorem.

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

\[ \nu(x \mapsto a + x \cdot b) = y + \nu(x \mapsto x \cdot b) \Leftarrow y = a + y \cdot b. \]

Proof

\[
\begin{align*}
\nu(x \mapsto a + x \cdot b) &= y + \nu(x \mapsto x \cdot b) \\
\Leftarrow &\quad \{ \quad (y+) \text{ is upper adjoint: } \nu\text{-fusion} \quad \} \\
\forall(x :: a + (y + x) \cdot b &= y + x \cdot b) \\
\Leftarrow &\quad \{ \quad (\cdot b) \text{ over } +, \text{ associativity of } + \quad \} \\
&\quad a + y \cdot b = y.
\end{align*}
\]

\( \square \)

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 $\nu(x \mapsto x \cdot b) = 0$, 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

\begin{equation}
\text{(23) [UEP of regular algebra] If } \nu(x \mapsto x \cdot b) = 0, \text{ then for all } a \text{ and } x, \quad a + x \cdot b = x \equiv x = a \cdot b^*. \tag{23}
\end{equation}

\(\Box\)

The UEP shows the importance of property $\nu(x \mapsto x \cdot b) = 0$. 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, $\nu(x \mapsto x \cdot b) = 0$ means that the graph is acyclic).

The above UEP has many applications in programming, because equations like $a + x \cdot b = x$ and $a + x \cdot b \subseteq x$ 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:

\begin{equation}
\text{(24) } \begin{align*}
fac.0 &= 1 \\
fac.(n+1) &= (n+1) \cdot fac.n, \text{ for } n \geq 0.
\end{align*}
\end{equation}

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{align*}
z(F)(n, y) &\equiv y = fac.n, \text{ for all } z \\
(a, b)(R)(n, y) &\equiv a + 1 = n \land (a+1) \cdot b = y,
\end{align*}

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

\begin{align*}
z(F)(n, y) &\equiv (0, 1) = (n, y) \lor z(F \circ R)(n, y).
\end{align*}
Hence $F = T \uplus F \circ R$, for $T$ defined by

$$z(T)(n, y) \equiv (0, 1) = (n, y)$$

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

$$Q = I \uplus R \circ Q.$$ Using the definitions of $F$ and $T$, $F = T \circ Q$ reads

$$y = \text{fac. } n \equiv (0, 1)(Q)(n, y)$$

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

$$Q(a, b)(Q)(n, y) \equiv (a, b) = (n, y) \lor (a + 1, (a + 1) \ast b)(Q)(n, y).$$

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

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

where

$$q_n.(a, b) = \begin{cases} b & n = a \\ q_n.(a + 1, (a + 1) \ast b) & n \neq a \end{cases}.$$

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


<table>
<thead>
<tr>
<th>Volume</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>91/02</td>
<td>R.P. Nederpelt</td>
<td>Implication. A survey of the different logical analyses &quot;if...,then...&quot;, p. 26.</td>
</tr>
<tr>
<td></td>
<td>H.C.M. de Swart</td>
<td></td>
</tr>
<tr>
<td>91/03</td>
<td>J.P. Katoen</td>
<td>Parallel Programs for the Recognition of $P$-invariant Segments, p. 16.</td>
</tr>
<tr>
<td></td>
<td>L.A.M. Schoenmakers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A.F. v.d. Stappen</td>
<td></td>
</tr>
<tr>
<td>91/05</td>
<td>D. de Reus</td>
<td>An Implementation Model for GOOD, p. 18.</td>
</tr>
<tr>
<td>91/06</td>
<td>K.M. van Hee</td>
<td>SPECIFICATIEMETHODEN, een overzicht, p. 20.</td>
</tr>
<tr>
<td>91/07</td>
<td>E. Poll</td>
<td>CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.</td>
</tr>
<tr>
<td>91/10</td>
<td>R.C. Backhouse</td>
<td>POLYNOMIAL RELATORS, p. 52.</td>
</tr>
<tr>
<td></td>
<td>P.J. de Bruin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P. Hoogendijk</td>
<td></td>
</tr>
<tr>
<td></td>
<td>G. Malcolm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E. Voernans</td>
<td></td>
</tr>
<tr>
<td></td>
<td>J. v.d. Woude</td>
<td></td>
</tr>
<tr>
<td>91/11</td>
<td>R.C. Backhouse</td>
<td>Relational Catamorphism, p. 31.</td>
</tr>
<tr>
<td></td>
<td>P.J. de Bruin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>G. Malcolm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E. Voernans</td>
<td></td>
</tr>
<tr>
<td></td>
<td>J. van der Woude</td>
<td></td>
</tr>
<tr>
<td>91/12</td>
<td>E. van der Sluis</td>
<td>A parallel local search algorithm for the travelling salesman problem, p. 12.</td>
</tr>
<tr>
<td>91/14</td>
<td>P. Lemmens</td>
<td>The PDB Hypermedia Package. Why and how it was built, p. 63.</td>
</tr>
<tr>
<td></td>
<td>K.M. van Hee</td>
<td></td>
</tr>
<tr>
<td>91/16</td>
<td>A.J.J.M. Marcelis</td>
<td>An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.</td>
</tr>
<tr>
<td>Number</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>91/18</td>
<td>Rik van Geldrop</td>
<td>Transformational Query Solving, p. 35.</td>
</tr>
<tr>
<td>91/19</td>
<td>Erik Poll</td>
<td>Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.</td>
</tr>
<tr>
<td>91/23</td>
<td>K.M. van Hee, L.J. Somers, M. Voorhoeve</td>
<td>Z and high level Petri nets, p. 16.</td>
</tr>
<tr>
<td>91/24</td>
<td>A.T.M. Aerts, D. de Reus</td>
<td>Formal semantics for BRM with examples, p. 25.</td>
</tr>
<tr>
<td>91/25</td>
<td>P. Zhou, J. Hooman, R. Kuiper</td>
<td>A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and completeness, p. 52.</td>
</tr>
<tr>
<td>91/27</td>
<td>F. de Boer, C. Palamidessi</td>
<td>Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.</td>
</tr>
<tr>
<td>91/28</td>
<td>F. de Boer</td>
<td>A compositional proof system for dynamic process creation, p. 24.</td>
</tr>
<tr>
<td>91/30</td>
<td>J.C.M. Baeten, F.W. Vaandrager</td>
<td>An Algebra for Process Creation, p. 29.</td>
</tr>
<tr>
<td>91/31</td>
<td>H. ten Eikelder</td>
<td>Some algorithms to decide the equivalence of recursive types, p. 26.</td>
</tr>
<tr>
<td>91/33</td>
<td>W. v.d. Aalst</td>
<td>The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.</td>
</tr>
<tr>
<td>91/34</td>
<td>J. Coenen</td>
<td>Specifying fault tolerant programs in deontic logic, p. 15.</td>
</tr>
</tbody>
</table>
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
A useful lambda notation, p. 17.
F.Kamareddine

92/23 F.Kamareddine
Nominalization, Predication and Type Containment, p. 40.
E.Klein

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

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

92/26 T.H.W.Beelen
A modelling method using MOVIE and SimCon/ExSpect, p. 15.
W.J.J.Stut
P.A.C.Verkoulen

92/27 B. Watson
A taxonomy of keyword pattern matching algorithms, p. 50.
G. Zwaan

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
Deterministic and randomized local search, p. 78.
J.H.M. Korst
P.J. Zwietering

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

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

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

93/11 K.M. van Hee

93/12 K.M. van Hee

93/13 K.M. van Hee
Systems Engineering: a Formal Approach
<table>
<thead>
<tr>
<th>Paper Number</th>
<th>Authors</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>93/16</td>
<td>H. Schepers, J. Hooman</td>
<td>A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27</td>
<td></td>
</tr>
<tr>
<td>93/17</td>
<td>D. Alstein, P. van der Stok</td>
<td>Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.</td>
<td></td>
</tr>
<tr>
<td>93/18</td>
<td>C. Verhoef</td>
<td>A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.</td>
<td></td>
</tr>
<tr>
<td>93/19</td>
<td>G.J. Houben</td>
<td>The Design of an Online Help Facility for ExSpect, p. 21.</td>
<td></td>
</tr>
<tr>
<td>93/22</td>
<td>E. Poll</td>
<td>A Typechecker for Bijective Pure Type Systems, p. 28.</td>
<td></td>
</tr>
<tr>
<td>93/23</td>
<td>E. de Kogel</td>
<td>Relational Algebra and Equational Proofs, p. 23.</td>
<td></td>
</tr>
<tr>
<td>93/24</td>
<td>E. Poll and Paula Severi</td>
<td>Pure Type Systems with Definitions, p. 38.</td>
<td></td>
</tr>
<tr>
<td>93/26</td>
<td>W.M.P. van der Aalst</td>
<td>Multi-dimensional Petri nets, p. 25.</td>
<td></td>
</tr>
<tr>
<td>93/27</td>
<td>T. Kloks and D. Kratsch</td>
<td>Finding all minimal separators of a graph, p. 11.</td>
<td></td>
</tr>
<tr>
<td>93/28</td>
<td>F. Kamareddine, R. Nederpelt</td>
<td>A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.</td>
<td></td>
</tr>
<tr>
<td>93/29</td>
<td>R. Post and P. De Bra</td>
<td>GOLD, a Graph Oriented Language for Databases, p. 42.</td>
<td></td>
</tr>
<tr>
<td>93/30</td>
<td>J. Deogun, T. Kloks, D. Kratsch, H. Müller</td>
<td>On Vertex Ranking for Permutation and Other Graphs, p. 11.</td>
<td></td>
</tr>
<tr>
<td>93/31</td>
<td>W. Körver</td>
<td>Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.</td>
<td></td>
</tr>
</tbody>
</table>
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.


93/36  J.C.M. Baeten and J.A. Bergstra  Non Interleaving Process Algebra, p. 17.


93/38  C. Verhoef  A general conservative extension theorem in process algebra, p. 17.


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 taxonomy of finite automata construction algorithms, p. 87.

93/44  B.W. Watson  A taxonomy of finite automata minimization algorithms, p. 23.


93/48  R. Gerth  Verifying Sequentially Consistently Consistent Memory using Interface Refinement, p. 20.
<table>
<thead>
<tr>
<th>Volume</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>94/01</td>
<td>P. America, M. van der Kammen, R.P. Nederpelt, O.S. van Roosmalen, H.C.M. de Swart</td>
<td>The object-oriented paradigm, p. 28.</td>
</tr>
<tr>
<td>94/02</td>
<td>F. Kamareddine, R.P. Nederpelt</td>
<td>Canonical typing and II-conversion, p. 51.</td>
</tr>
<tr>
<td>94/04</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.</td>
</tr>
<tr>
<td>94/06</td>
<td>T. Basten, T. Kunz, J. Black, M. Coffin, D. Taylor</td>
<td>Time and the Order of Abstract Events in Distributed Computations, p. 29.</td>
</tr>
<tr>
<td>94/08</td>
<td>O.S. van Roosmalen</td>
<td>A Hierarchical Diagrammatic Representation of Class Structure, p. 22.</td>
</tr>
<tr>
<td>94/09</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Process Algebra with Partial Choice, p. 16.</td>
</tr>
<tr>
<td>94/10</td>
<td>T. terboeck</td>
<td>The testing Paradigm Applied to Network Structure, p. 31.</td>
</tr>
<tr>
<td>94/13</td>
<td>R. Seljée</td>
<td>A New Method for Integrity Constraint checking in Deductive Databases, p. 34.</td>
</tr>
<tr>
<td>94/14</td>
<td>W. Peremans</td>
<td>Ups and Downs of Type Theory, p. 9.</td>
</tr>
<tr>
<td>94/16</td>
<td>R.C. Backhouse, H. Doombos</td>
<td>Mathematical Induction Made Calculational, p. 36.</td>
</tr>
<tr>
<td>Page</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>94/18</td>
<td>F. Kamareddine, R. Nederpelt</td>
<td>Refining Reduction in the Lambda Calculus, p. 15.</td>
</tr>
<tr>
<td>94/19</td>
<td>B.W. Watson</td>
<td>The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.</td>
</tr>
<tr>
<td>94/20</td>
<td>R. Bloo, F. Kamareddine, R. Nederpelt</td>
<td>Beyond β-Reduction in Church's λ→, p. 22.</td>
</tr>
<tr>
<td>94/22</td>
<td>B.W. Watson</td>
<td>The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.</td>
</tr>
<tr>
<td>94/23</td>
<td>S. Mauw and M.A. Reniers</td>
<td>An algebraic semantics of Message Sequence Charts, p. 43.</td>
</tr>
<tr>
<td>94/25</td>
<td>T. Kloks</td>
<td>K₁,₃-free and W₄-free graphs, p. 10.</td>
</tr>
<tr>
<td>94/29</td>
<td>J. Hooman</td>
<td>Correctness of Real Time Systems by Construction, p. 22.</td>
</tr>
<tr>
<td>94/30</td>
<td>I.C.M. Baeten, J.A. Bergstra, Gh. Ştefănescu</td>
<td>Process Algebra with Feedback, p. 22.</td>
</tr>
<tr>
<td>94/31</td>
<td>B.W. Watson, R.E. Watson</td>
<td>A Boyer-Moore type algorithm for regular expression pattern matching, p. 22.</td>
</tr>
<tr>
<td>94/33</td>
<td>T. Laan</td>
<td>A formalization of the Ramified Type Theory, p. 40.</td>
</tr>
<tr>
<td>94/34</td>
<td>R. Bloo, F. Kamareddine, R. Nederpelt</td>
<td>The Barendregt Cube with Definitions and Generalised Reduction, p. 37.</td>
</tr>
<tr>
<td>94/35</td>
<td>J.C.M. Baeten, S. Mauw</td>
<td>Delayed choice: an operator for joining Message Sequence Charts, p. 15.</td>
</tr>
<tr>
<td>94/36</td>
<td>F. Kamareddine, R. Nederpelt</td>
<td>Canonical typing and Π-conversion in the Barendregt Cube, p. 19.</td>
</tr>
</tbody>
</table>
| 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 Theory: An Illustration, p. 35. |
| 94/44 | E. Brinksma  
R. Gerth  
W. Janssen  
S. Katz  
M. Poel  
C. Rump  
J. Davies  
S. Graf  
B. Jonsson  
G. Lowe  
A. Pnueli  
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 \(\lambda\)-cube with classes of terms modulo conversion, p. 16. |
| 94/47 | R. Bloo  
F. Kamareddine  
R. Nederpelt | On \(\Pi\)-conversion in Type Theory, p. 12. |