Calculating the Warshall/Floyd path algorithm

Eindhoven University of Technology

Department of Mathematics and Computing Science

Calculating the Warshall/Floyd Path Algorithm

by

Roland C. Backhouse

92/09

Computing Science Note 92/09
Eindhoven, May 1992

COMPUTING SCIENCE NOTES

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. F. van Neerven
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

CALCULATING THE WARSHALL/FLOYD PATH ALGORITHM

Roland C. Backhouse
Department of Mathematics and Computing Science,
Eindhoven University of Technology,
P.O. Box 513,
5600 MB Eindhoven,
The Netherlands.

May 18, 1992

Abstract

A calculational derivation is given of an all-pairs path algorithm two instances of which are Warshall's reachability algorithm and Floyd's shortest-path algorithm. The derivation provides an elementary example of the importance of the so-called star-decomposition rule.

1 Algebraic Framework

This paper presents a calculational derivation of an all-pairs path algorithm, two well-known instances of which are Warshall's (reachability) algorithm and Floyd's shortest-path algorithm. The calculations presented here are essentially the same as those in [1, 2]. The presentation has been brought up-to-date in that explicit rather than implicit use is made of invariant properties. Moreover notational refinements enhance the clarity of the derivation.

Like [1, 2] the framework for the current derivation is regular algebra. The axioms of regular algebra — the algebra of regular languages — are now widely known and publicised. (See e.g. [6, 4].) The fact that the elementary operators involved in several path-finding algorithms obey the axioms of regular algebra is also widely known and this knowledge will be assumed.

The specific details of the framework are that $(S, +, \cdot, ^*, 0, 1)$ is a regular algebra. That is, S is a set on which are defined two binary operators $+$ and $\cdot$ and one unary operator * (written as a postfix of its argument). Addition $(+)$ is associative, commutative and idempotent. Multiplication $(\cdot)$ distributes over addition and is associative but is not necessarily commutative. The basic properties of * that we use here are, for all $a, b \in S$:

$$(1) \qquad a^* = 1 + a \cdot a^*$$

$$(2) \qquad a \cdot (b \cdot a)^* = (a \cdot b)^* \cdot a$$

$$(3) \qquad (a + b)^* = (a^* \cdot b)^* \cdot a^*$$

$$(4) \qquad 1^* = 1$$

Rule (2) will be referred to as the "leapfrog rule" whilst rule (3) will be called the "star-decomposition rule." The main contribution made by Backhouse and Carré [2] was to show that these four rules are at the heart of several elimination techniques for solving shortest-path and other path-finding problems. This paper provides the most elementary instance of this thesis. Backhouse and van Gasteren [3] have recently shown how a class of algorithms that includes Dijkstra's shortest-path algorithm [7] can be derived from these four rules. That derivation is longer than the one here since the

underlying assumptions are more complicated (and the algorithm is more efficient). The current paper can thus be viewed as an elementary introduction to the principal ideas.

An important theorem that we exploit is that if $(S, +, \cdot, ^*, 0, 1)$ is a regular algebra then so too is $(\mathcal{M}_N(S), +, \cdot, ^*, \mathbf{0}, \mathbf{1})$ where $\mathcal{M}_N(S)$ is the set of $N \times N$ matrices whose elements are drawn from S. In proving this theorem appropriate definitions must be given of matrix addition, multiplication and star, and of the null and identity matrices $\mathbf{0}$ and $\mathbf{1}$. For the first two the usual definitions of matrix addition and multiplication are taken; for the last two: $\mathbf{0}$ is the $N \times N$ matrix all of whose entries are 0 and the identity matrix is a 0-1 matrix that is everywhere 0 but for its diagonal elements (which are all 1). It is not so easy to explain the definition of A^*, for matrix A, in a few words. Appropriate definitions are given in [1, 4]. The former also includes a proof of the theorem. A proof has also recently been published by Kozen [8]. (As remarked by Kozen the proof of the theorem is an elementary exercise, although it does not appear explicitly in any of the standard references.)

With these preliminaries we can now proceed to the task at hand. Given is a matrix $A \in \mathcal{M}_N(S)$ and required is to derive an algorithm to compute $A^+ = A \cdot A^*$. By an "algorithm" we mean an (imperative) program in which in no assignment or test is the star operator applied to a matrix. (Application of star to matrix elements is however allowed.) The matrix A^+ is called the transitive closure of A.

The heuristic underlying the algorithm is to use the star-decomposition rule to reduce the given matrix A to the null matrix by successively nullifying columns of the matrix. In order to express this formally we need to introduce some additional notation. Specifically, for all integers k, $0 \leq k < N$ we define $\bullet k$ to be a $N \times 1$ matrix (i.e. a column vector) that is everywhere 0 but for its kth entry which is 1. (We index rows and columns beginning at index 0.) The notation $k\bullet$ is used for the transpose of $\bullet k$, that is a $1 \times N$ matrix (a row vector) whose only non-null entry is the kth. We also define $[k]$ to be the (matrix) product $\bullet k \cdot k\bullet$ and $[\geq k]$ to be the sum of $[i]$ over all i in the range $k \leq i < N$.

The specific properties that we assume of these expressions are as follows. First, by definition,

(5) $\quad [k] = \bullet k \cdot k\bullet$

Second, by range splitting on the summation defining $[\geq k]$, for all $k < N$,

(6) $\quad [\geq k] = [k] + [\geq (k+1)]$

Third, $[\geq N]$ is an empty summation. Thus

(7) $\quad [\geq N] = \mathbf{0}$

Fourth, $[\geq 0]$ is the identity matrix:

(8) $\quad [\geq 0] = \mathbf{1}$

Finally, because it is a criterion for deciding when we have an algorithm we remark that, for all $N \times N$ matrices X, $i\bullet \cdot X \cdot \bullet j$ is the (i,j)th element of X.

2 First Steps

We begin our calculation by noting that $[\geq k] \cdot X$, for $N \times N$ matrix X, is a copy of X but for its first k columns which are all null. Let us consider the expression $([\geq k] \cdot X)^*$ for $k < N$. We have:

$$\begin{array}{ll} & ([\geq k] \cdot X)^* \\ = & \quad \{\ k < N, (6) \text{ and distributivity } \} \\ & ([k] \cdot X \ + \ [\geq (k+1)] \cdot X)^* \\ = & \quad \{\ \text{star decomposition } \} \\ & ([k] \cdot X)^* \cdot ([\geq (k+1)] \cdot X \cdot ([k] \cdot X)^*)^* \end{array}$$

This little calculation is interesting because the pattern $([\geq i] \cdot Y)^*$ recurs in the first and last lines. In the first line i is just k and Y is X. In the last line i is $k+1$ and Y is $X \cdot ([k] \cdot X)^*$. It invites us to seek a particularly simple recurrence. Let us introduce the function M defined by

(9) $\quad M(X,k) \ = \ X \ \cdot \ ([\geq k] \ \cdot \ X)^*$

Then, for $k < N$ we have:

$$\begin{array}{ll} & M(X,k) \\ = & \quad \{\ \text{definition and above calculation } \} \\ & X \ \cdot \ ([k] \ \cdot \ X)^* \ \cdot \ ([\geq (k+1)] \ \cdot \ X \ \cdot \ ([k] \ \cdot \ X)^*)^* \\ = & \quad \{\ \text{definition } \} \\ & M(X \ \cdot \ ([k] \ \cdot \ X)^*, \ k+1) \end{array}$$

Noting also that

$$
\begin{array}{ll}
 & M(X,0) \\
= & \quad \{\ (8),\ \text{definition}\ \} \\
 & X \cdot (\mathbf{1} \cdot X)^* \\
= & \quad \{\ \mathbf{1}\ \text{is the unit of multiplication, definition}\ \} \\
 & X^+
\end{array}
$$

and

$$
\begin{array}{ll}
 & M(X,N) \\
= & \quad \{\ \text{definition, (7)}\ \} \\
 & X \cdot (\mathbf{0} \cdot X)^* \\
= & \quad \{\ \mathbf{0}\ \text{is zero of multiplication, } \mathbf{0}^* = \{(1)\}\ \mathbf{1}\ \} \\
 & X
\end{array}
$$

we have established the correctness of the following algorithm to compute A^+:

$$
\begin{array}{l}
\{\ A^+ = M(A,0)\ \} \\
\ X,k := A,0 \\
; \qquad \{\ \text{Invariant:}\quad A^+ = M(X,k)\ \} \\
\ \texttt{do}\ k \neq N \longrightarrow X,\ k := X \cdot ([k] \cdot X)^*,\ k+1 \\
\ \texttt{od} \\
\{\ A^+ = M(X,N)\ \} \\
\{\ A^+ = X\ \}
\end{array}
$$

3 The Algorithm

There is one more step to be taken before we have an algorithm in which * is applied only to elements and not to matrices. We take the expression $X \cdot ([k] \cdot X)^*$ and rewrite it using (5).

$$
\begin{array}{ll}
 & X \cdot ([k] \cdot X)^* \\
= & \quad \{\ (1)\ \} \\
 & X \cdot \mathbf{1} + X \cdot [k] \cdot X \cdot ([k] \cdot X)^* \\
= & \quad \{\ X \cdot \mathbf{1} = X,\ (5)\ \} \\
 & X + X \cdot \bullet k \cdot k \bullet \cdot X \cdot (\bullet k \cdot k \bullet \cdot X)^*
\end{array}
$$

$$= \quad \{ \text{ leapfrog rule: (2) } \}$$
$$X \ + \ X \ \cdot \ \bullet k \ \cdot \ (k \bullet \cdot X \ \cdot \ \bullet k)^* \ \cdot \ k \bullet \cdot X$$

As remarked earlier $k \bullet \cdot X \cdot \bullet k$ is the (k,k)th element of X and since it is the argument of the only application of * we have obtained our algorithm:

$\{ A^+ \ = \ M(A,0) \ \}$
$X, k \ := \ A, 0$
; $\quad \{ \text{Invariant:} \quad A^+ \ = \ M(X,k) \ \}$
do $k \neq N \ \longrightarrow$
$\quad X, \ k \ := \ X \ + \ X \ \cdot \ \bullet k \ \cdot \ (k \bullet \cdot X \ \cdot \ \bullet k)^* \ \cdot \ k \bullet \cdot X, \ k+1$
od
$\{ A^+ \ = \ M(X,N) \ \}$
$\{ A^+ \ = \ X \ \}$

4 Implementation Freedom

The algorithm we have obtained is not quite Warshall's algorithm or Floyd's algorithm (even after suitable interpretation of the operators). The reason is that at element level the assignment in the body of the loop is a *simultaneous* assignment to all matrix elements. Spelling this out in detail, the matrix assignment

$$X \ := \ X + X \ \cdot \ \bullet k \ \cdot \ (k \bullet \cdot X \ \cdot \ \bullet k)^* \ \cdot \ k \bullet \cdot X$$

is directly implemented as the simultaneous assignment

simultaneously_for $i \ := \ 0$ to $N-1$ and $j \ := \ 0$ to $N-1$ do
$\quad i \bullet \cdot X \cdot \bullet j \ :=$
$\quad\quad i \bullet \cdot X \cdot \bullet j + i \bullet \cdot X \ \cdot \ \bullet k \ \cdot \ (k \bullet \cdot X \ \cdot \ \bullet k)^* \ \cdot \ k \bullet \cdot X \ \cdot \bullet j$

(Writing $i \bullet \cdot X \cdot \bullet j$ conventionally as x_{ij} this takes on the more familiar appearance:

simultaneously_for $i \ := \ 0$ to $N-1$ and $j \ := \ 0$ to $N-1$ do
$\quad x_{ij} \ := \ x_{ij} + x_{ik} \ \cdot \ (x_{kk}) * \cdot x_{kj}$

But, of course, the problem of the simultaneous assignment remains.)

Exploitation of the, as yet unused, idempotency of addition and star, however, gives unlimited freedom in the order in which the matrix elements are assigned. They may be assigned sequentially as in Warshall's and Floyd's algorithms, or completely in parallel!

To explain why this is so consider the function

$$(10) \qquad X \;\mapsto\; X + X \cdot \bullet k \cdot (k \bullet \cdot X \cdot \bullet k)^* \cdot k \bullet \cdot X$$

which, as we know, is equal to the function

$$(11) \qquad X \;\mapsto\; X \cdot ([k] \cdot X)^*$$

Let this function be called f. The body of the loop is then the assignment

$$X, k \;:=\; f.X, k+1$$

Forget the matrix structure of X and just regard X as the name of the set of variables $\{i, j : 0 \leq i, j < N : i \bullet \cdot X \cdot \bullet j\}$. Then the fact that the elements of the set may be assigned in an arbitrary order rests on three key properties of the function:

(a) f is idempotent,

(b) $X \leq f.X$,

(c) f is monotone non-decreasing in each of its arguments.

The verification of the idempotency of f proceeds as follows:

$$\begin{array}{ll} & f.(f.X) \\ = & \qquad \{\ (11)\ \} \\ & X \cdot ([k] \cdot X)* \cdot ([k] \cdot X \cdot ([k] \cdot X)*)* \\ = & \qquad \{\ a* = (1+a)*\ \} \\ & X \cdot ([k] \cdot X)* \cdot (1 + [k] \cdot X \cdot ([k] \cdot X)*)* \\ = & \qquad \{\ (1)\ \} \\ & X \cdot ([k] \cdot X)* \cdot ([k] \cdot X)** \\ = & \qquad \{\ a** = a* \text{ and } a* = a* \cdot a*\ \} \\ & X \cdot ([k] \cdot X)* \\ = & \qquad \{\ (11)\ \} \\ & f.X \end{array}$$

Property (b) is immediate from (10), and (c) follows from the monotonicity of addition, multiplication and star.

The claim that the combination of these three properties permits the conversion of the simultaneous assignment to a parallel assigment may or may not be well known. (It is not well known among colleagues to whom I have spoken.) Its proof is remarkably simple and is given in the appendix.

This concludes the derivation of the Warshall/Floyd algorithm. Note that the total calculation (including the discussion of implementation freedom) takes roughly twenty elementary steps which is about what it should be for such a compact algorithm.

References

[1] R.C. Backhouse. *Closure algorithms and the star-height problem of regular languages.* PhD thesis, University of London, 1975.

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

[3] Roland Backhouse and A.J.M. van Gasteren. Calculating a path algorithm. Submitted for publication, 1992.

[4] B.A. Carré. *Graphs and Networks.* Oxford University Press, 1979.

[5] P. Chisholm. Calculation by computer. In *Third International Workshop Software Engineering and its Applications*, pages 713–728, Toulouse, France, December 3-7 1990. EC2.

[6] J.H. Conway. *Regular algebra and finite machines.* Chapman and Hall, London, 1971.

[7] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[8] Dexter Kozen. A completeness theorem for kleene algebras and the algebra of regular events. In *Proc. 6th Annual IEEE Symp. on Logic in Computer Science*, pages 214–225. IEEE Society Press, 1991.

Appendix

Let X be a finite set of variables and suppose the type of each of the variables is a set ordered by the (partial) relation $\leq$. Suppose f is an endofunction on the domain of X satisfying the properties:

(a) f is idempotent,

(b) $\forall(X :: X \leq f.X)$,

(c) f is monotone non-decreasing in each of its arguments.

Then the assignment

$$X := f.X$$

is equivalent to the assignment

$$\texttt{parfor } x \in X \texttt{ do } x := f_x.X$$

The proof is by induction on the size of X. The basis is of course trivial. For the induction step the following lemma suffices.

Lemma 12 Let $\oplus$ and $\otimes$ be binary operators such that

(a) $x \oplus y = (x \oplus y) \oplus (x \otimes y)$
$x \otimes y = (x \oplus y) \otimes (x \otimes y)$
(I.e. the function $(x, y \mapsto x \oplus y, x \otimes y)$ is idempotent.)
(b) $x \leq x \oplus y$
$y \leq x \otimes y$
(c) Both $\oplus$ and $\otimes$ are monotone in both their arguments.

Then the simultaneous assignment

$$x, y := x \oplus y, x \otimes y$$

can be implemented by the sequential assignment

$$\begin{array}{ll} & x := x \oplus y \\ ; & y := x \otimes y \end{array}$$

or by the sequential assignment

$$\begin{array}{ll} & y \;:=\; x \otimes y \\ ; & x \;:=\; x \oplus y \end{array}$$

Proof We have to show that

(13) $\quad (x \oplus y) \otimes y \;=\; x \otimes y$

and

(14) $\quad x \oplus (x \otimes y) \;=\; x \oplus y$

The first is proved as follows:

$$\begin{array}{ll} & x \otimes y \\ \leq & \qquad \{\ \text{(b) and (c)}\ \} \\ & (x \oplus y) \otimes y \\ \leq & \qquad \{\ \text{(b) and (c)}\ \} \\ & (x \oplus y) \otimes (x \otimes y) \end{array}$$

Hence, by (a),

$$x \otimes y = (x \oplus y) \otimes y$$

The second identity is proved similarly.
□

Acknowledgements

Thanks go to Wim Feijen, Joop van den Eijnde and Lambert Meertens for their critical comments and suggestions for improvement.

Preparation of the report was expedited by the use of the proof editor developed by Paul Chisholm [5].

In this series appeared:

89/1	E.Zs.Lepoeter-Molnar	Reconstruction of a 3-D surface from its normal vectors.
89/2	R.H. Mak P.Struik	A systolic design for dynamic programming.
89/3	H.M.M. Ten Eikelder C. Hemerik	Some category theoretical properties related to a model for a polymorphic lambda-calculus.
89/4	J.Zwiers W.P. de Roever	Compositionality and modularity in process specification and design: A trace-state based approach.
89/5	Wei Chen T.Verhoeff J.T.Udding	Networks of Communicating Processes and their (De-)Composition.
89/6	T.Verhoeff	Characterizations of Delay-Insensitive Communication Protocols.
89/7	P.Struik	A systematic design of a parallel program for Dirichlet convolution.
89/8	E.H.L.Aarts A.E.Eiben K.M. van Hee	A general theory of genetic algorithms.
89/9	K.M. van Hee P.M.P. Rambags	Discrete event systems: Dynamic versus static topology.
89/10	S.Ramesh	A new efficient implementation of CSP with output guards.
89/11	S.Ramesh	Algebraic specification and implementation of infinite processes.
89/12	A.T.M.Aerts K.M. van Hee	A concise formal framework for data modeling.
89/13	A.T.M.Aerts K.M. van Hee M.W.H. Hesen	A program generator for simulated annealing problems.
89/14	H.C.Haesen	ELDA, data manipulatie taal.
89/15	J.S.C.P. van der Woude	Optimal segmentations.
89/16	A.T.M.Aerts K.M. van Hee	Towards a framework for comparing data models.
89/17	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra.

90/1	W.P.de Roever- H.Barringer- C.Courcoubetis-D.Gabbay R.Gerth-B.Jonsson-A.Pnueli M.Reed-J.Sifakis-J.Vytopil P.Wolper	Formal methods and tools for the development of distributed and real time systems, p. 17.
90/2	K.M. van Hee P.M.P. Rambags	Dynamic process creation in high-level Petri nets, pp. 19.
90/3	R. Gerth	Foundations of Compositional Program Refinement - safety properties - , p. 38.
90/4	A. Peeters	Decomposition of delay-insensitive circuits, p. 25.
90/5	J.A. Brzozowski J.C. Ebergen	On the delay-sensitivity of gate networks, p. 23.
90/6	A.J.J.M. Marcelis	Typed inference systems : a reference document, p. 17.
90/7	A.J.J.M. Marcelis	A logic for one-pass, one-attributed grammars, p. 14.
90/8	M.B. Josephs	Receptive Process Theory, p. 16.
90/9	A.T.M. Aerts P.M.E. De Bra K.M. van Hee	Combining the functional and the relational model, p. 15.
90/10	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra, p. 30. (Revised version of CSNotes 89/17).
90/11	P. America F.S. de Boer	A proof system for process creation, p. 84.
90/12	P.America F.S. de Boer	A proof theory for a sequential version of POOL, p. 110.
90/13	K.R. Apt F.S. de Boer E.R. Olderog	Proving termination of Parallel Programs, p. 7.
90/14	F.S. de Boer	A proof system for the language POOL, p. 70.
90/15	F.S. de Boer	Compositionality in the temporal logic of concurrent systems, p. 17.
90/16	F.S. de Boer C. Palamidessi	A fully abstract model for concurrent logic languages, p. p. 23.
90/17	F.S. de Boer C. Palamidessi	On the asynchronous nature of communication in logic languages: a fully abstract model based on sequences, p. 29.

90/18	J.Coenen E.v.d.Sluis E.v.d.Velden	Design and implementation aspects of remote procedure calls, p. 15.
90/19	M.M. de Brouwer P.A.C. Verkoulen	Two Case Studies in ExSpect, p. 24.
90/20	M.Rem	The Nature of Delay-Insensitive Computing, p.18.
90/21	K.M. van Hee P.A.C. Verkoulen	Data, Process and Behaviour Modelling in an integrated specification framework, p. 37.
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.