Eindhoven University of Technology
Department of Mathematics and Computing Science

Temporal operators viewed as
predicate transformers

by

A. Bijlsma

93/41
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.
Temporal operators viewed as predicate transformers

A. Bijlsma
Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

November 22, 1993

1 Introduction

In this paper we present a calculus of temporal operators that is intended as an alternative to temporal logic in program verification. Temporal logic is a form of modal logic, designed for the description of time-dependent phenomena. Several different kinds of temporal logic have been proposed, but in the present paper only the linear time version [13] is treated. A common characteristic of all kinds of temporal logic is that they require a large number of irregularly-shaped axioms; as a result, proofs within such a system tend to be tortuous and mystifying. The cause of this problem is that these logics were designed to satisfy theoretical demands such as completeness, not to optimize ease of manipulation. It is our contention that the calculus proposed below, while equivalent in its descriptive power, places less of a burden on the user. Our approach was inspired by work of Boute [3] and Hehner [8], but diverges from these in that we refrain from explicit mention of time coordinates.

In [5], E.W. Dijkstra and C.S. Scholten presented a version of predicate calculus based on equational reasoning rather than natural deduction. Their work, whose style owes more to algebra than to logic, leads to very smooth calculational proofs. We emphasize that their calculus, although quite rigorous in a mathematical sense, is not a formal system: for instance, it allows every theorem from mathematics to be used in proofs without the addition of anything like 'domain axioms'. This may be a disadvantage if one is interested in proving properties of the calculus itself, but it is a great advantage when the calculus is being used as a tool in proving properties of actual programs [4]. When the same principles are applied to program semantics, the axioms and inference rules of Floyd-Hoare logic are replaced with the definition of a class of predicate transformers called weakest preconditions. In the present paper, we shall follow a similar path and enrich the calculus of Dijkstra and Scholten with the definition of a class of predicate transformers called shift operators, to be used as a replacement for the axioms and inference rules of temporal logic.

The organization of the paper is as follows. In section 2 we define the concept of a shift operator, and in section 3 we show how a shift operator can be used to construct a predicate-transformer model for the axioms of temporal logic. In section 4 we prove that, in fact, every such model can be obtained from a shift operator, which establishes the equivalence of both approaches. Finally, in section 5, examples demonstrate that reasoning with shift operators often leads to proofs that are considerably shorter than those of temporal logic.
Our point of departure is the predicate calculus as developed in [5], whose notation and
terminology are adopted. In particular, square brackets denote universal quantification over
the anonymous state space. In addition, we establish the convention that the letters \( p, q, r, s, t \)
will stand for predicates and \( i, j \) for natural numbers.

## 2 Concepts

**Definition 1** A *shift operator* is a predicate transformer \( \delta \) such that

- \( \delta \) is universally conjunctive, i.e.,
  \[
  [\delta (\forall x :: p.x) \equiv (\forall x :: \delta(p.x))] \tag{1}
  \]
  
  for every predicate-valued mapping \( p \);

- \( \delta \) is its own conjugate, i.e.,
  \[
  [\neg \delta \neg p \equiv \delta.p] \tag{2}
  \]
  
  for every predicate \( p \).

\( \Box \)

For instance, the identity function is a shift operator, and so is any substitution. The term
'shift operator' is taken from [3], where it denotes the particular substitution \( t := t + \Delta t \), a
translation in time. However, as we shall see, it is not necessary to be that specific.

As every logical operator and quantifier may be constructed from conjunction and negation,
a shift operator distributes over all logical operators and quantifiers. In particular,

\[
[\delta . \text{true} \equiv \text{true}] \tag{3}
\]

for shift operator \( \delta \).

**Definition 2** Consider three unary predicate transformers denoted by prefix operators \( \Box \)
(pronounced 'next'), \( \square \) ('always') and \( \Diamond \) ('sometime'), and also one binary predicate
transformer denoted by the infix operator \( \land \) (pronounced 'until' and given the same binding power
as \( \land \) and \( \lor \)). The tuple \( (\Box, \square, \Diamond, \land) \) is said to satisfy the *axioms of temporal logic* if

\[
[\Box \neg p \equiv \neg \Box p], \tag{4}
\]
\[
[\Box (p \Rightarrow q) \Rightarrow (\Box p \Rightarrow \Box q)], \tag{5}
\]
\[
[(\forall x :: \Box p.x) \Rightarrow \Box (\forall x :: p.x)], \tag{6}
\]
\[
[p] \Rightarrow [\Box p], \tag{7}
\]
\[
[\Box p \Rightarrow \Box p], \tag{8}
\]
\[
[\Box p \Rightarrow p], \tag{9}
\]
\[
[\Diamond p \Rightarrow \Box \Diamond p], \tag{10}
\]
\[
[\Box(p \Rightarrow q) \Rightarrow (\Box p \Rightarrow \Box q)], \tag{11}
\]
Formulae (4) through (15) are the equivalent, in our notation, of the axioms of temporal logic as presented in [13]. This is immediately obvious except, perhaps, in the case of (7), which figures in [13] as an inference rule rather than an axiom. This discrepancy is caused by the absence of our square brackets in the system of [13], and, conversely, by the absence of metalinguage symbols for provability in the system of [5]. This axiom system has been selected because it seems to be the best-known one, not because it is in any sense minimal. (For instance, (8) can be derived from the other axioms.) Different and smaller axiomatizations can be found in the literature [9].

3 From shift operators to temporal logic

Theorem 3 Let $\delta$ be a shift operator. Define $\ominus, \square, \Diamond, \mathcal{U}$ by

\begin{align*}
[\ominus p & \equiv \delta \cdot p], \\
[\square p & \equiv (\forall j :: \delta^j \cdot p)], \\
[\Diamond p & \equiv (\exists j :: \delta^j \cdot p)], \\
[p \mathcal{U} q & \equiv (\exists j :: (\forall i :: i < j : \delta^i \cdot p) \land \delta^j \cdot q)].
\end{align*}

Then the tuple $(\ominus, \square, \Diamond, \mathcal{U})$ satisfies the axioms of temporal logic.

Proof Formulae (4), (5) and (6) follow immediately by distributing $\delta$ over the logical operators and quantifiers. Proof of (7): assuming validity of $\left[ p \right]$, we have

\begin{align*}
\Box p & \equiv \{ \text{definition of $\Box$} \} \\
& \equiv (\forall j :: \delta^j \cdot p) \\
& \equiv \{ \left[ p \right] \} \\
& \equiv (\forall j :: \delta^j \cdot \text{true}) \\
& \equiv \{ (3) \} \\
& \equiv \{ \text{term true} \} \\
& \equiv \text{true}.
\end{align*}

Proof of (8):

\begin{align*}
\Box p & \equiv \{ \text{definition of $\Box$} \} \\
& \equiv (\forall j :: \delta^j \cdot p) \\
& \Rightarrow \{ \text{instantiation } j := 1 \}
\end{align*}
\[ \delta.p \]
\[ \equiv \quad \text{(definition of } \bigcirc \text{)} \]
\[ \bigcirc p . \]

Proof of (9) and (10):

\[ \square p \]
\[ \equiv \quad \text{(definition of } \square \text{)} \]
\[ (\forall j :: \delta^j.p) \]
\[ \equiv \quad \{ \text{split off } j = 0; \text{ dummy transformation } j := j + 1 \} \]
\[ p \land (\forall j :: \delta^{j+1}.p) \]
\[ \equiv \quad \{ \delta \text{ distributes over logic} \} \]
\[ p \land \delta.(\forall j :: \delta^j.p) \]
\[ \equiv \quad \{ \text{definitions of } \bigcirc \text{ and } \square \} \]
\[ p \land \square \square p . \]

Proof of (11):

\[ \square(p \Rightarrow q) \]
\[ \equiv \quad \text{(definition of } \square \text{)} \]
\[ (\forall j :: \delta^j.(p \Rightarrow q)) \]
\[ \equiv \quad \{ \delta \text{ distributes over logic} \} \]
\[ (\forall j :: \delta^j.p \Rightarrow \delta^j.q) \]
\[ \Rightarrow \quad \{ \text{predicate calculus} \} \]
\[ (\forall j :: \delta^j.p) \Rightarrow (\forall j :: \delta^j.q) \]
\[ \equiv \quad \{ \text{definition of } \square \} \]
\[ \square p \Rightarrow \square q . \]

Proof of (12):

\[ \square(p \Rightarrow \bigcirc p) \]
\[ \equiv \quad \{ \text{definitions of } \square \text{ and } \bigcirc \} \]
\[ (\forall j :: \delta^j.(p \Rightarrow \delta.p)) \]
\[ \equiv \quad \{ \delta \text{ distributes over logic} \} \]
\[ (\forall j :: \delta^j.p \Rightarrow \delta^{j+1}.p) \]
\[ \Rightarrow \quad \{ \text{mathematical induction} \} \]
\[ p \Rightarrow (\forall j :: \delta^j.p) \]
\[ \equiv \quad \{ \text{definition of } \square \} \]
\[ p \Rightarrow \square p . \]

Proof of (13):

\[ \neg \square p \]
\[ \equiv \quad \{ \text{definition of } \bigcirc \} \]
\[ \neg(\exists j :: \delta^j.p) \]
\[ \equiv \quad \{ \text{de Morgan} \} \]
\[ (\forall j :: \neg \delta^j.p) \]
\[ \equiv \quad \{ \delta \text{ distributes over logic} \} \]
\[(\forall j :: \delta^j . \neg p) \equiv \square \neg p \quad \{ \text{definition of } \square \} \]

Proof of (14):
\[p \mathbin{\mathcal{U}} q \equiv \{ \text{definition of } \mathcal{U} \} \]
\[(\exists j :: (\forall i : i < j : \delta^i . p) \land \delta^j . q) \quad \Rightarrow \]
\[\{ \text{weakening term} \} \]
\[(\exists j :: \delta^j . q) \equiv \{ \text{definition of } \Diamond \} \]
\[\Diamond q \quad . \]

Proof of (15):
\[p \mathbin{\mathcal{U}} q \equiv \{ \text{definition of } \mathcal{U} \} \]
\[(\exists j :: (\forall i : i < j : \delta^i . p) \land \delta^j . q) \equiv \{ \text{split off } j = 0; \text{ dummy transformation } j := j + 1 \} \]
\[q \lor (\exists j :: (\forall i : i < j + 1 : \delta^i . p) \land \delta^{j+1} . q) \equiv \{ \text{split off } i = 0; \text{ dummy transformation } i := i + 1 \} \]
\[q \lor (\exists j :: p \land (\forall i : i < j : \delta^{i+1} . p) \land \delta^{j+1} . q) \equiv \{ \text{distribution } \land \text{ over } \exists \} \]
\[q \lor (p \land (\exists j :: (\forall i : i < j : \delta^{i+1} . p) \land \delta^{j+1} . q))) \equiv \{ \delta \text{ distributes over logic} \} \]
\[q \lor (p \land \delta . (\exists j :: (\forall i : i < j : \delta^i . p) \land \delta^j . q))) \equiv \{ \text{definitions of } \Diamond \text{ and } \mathcal{U} \} \]
\[q \lor (p \land \Diamond (p \mathbin{\mathcal{U}} q)) \quad . \]

4 From temporal logic to shift operators

In this section we prove that the shift operator approach is quite as general as the axioms of temporal logic. This is captured in the following theorem.

**Theorem 4** Assume the tuple \((\mathcal{O}, \square, \Diamond, \mathcal{U})\) satisfies the axioms of temporal logic. Then there is a shift operator \(\delta\) such that

\[
[\mathcal{O} p \equiv \delta . p] \quad , \quad (16)
\]
\[
[\square p \equiv (\forall j :: \delta^j . p)] \quad , \quad (17)
\]
\[
[\Diamond p \equiv (\exists j :: \delta^j . p)] \quad , \quad (18)
\]
\[
[p \mathbin{\mathcal{U}} q \equiv (\exists j :: (\forall i : i < j : \delta^i . p) \land \delta^j . q)] \quad . \quad (19)
\]

**Proof** We have no choice but to let \(\delta\) be defined by (16). To begin with, we show that this \(\delta\) is a shift operator, i.e., satisfies (1) and (2).

Proof of (1):
\[
\delta. (\forall x :: p.x) \equiv (\forall x :: \delta.(p.x))
\]
\[
\equiv \{ \text{definition of } \delta \}
\]
\[
[\Box (\forall x :: p.x) \equiv (\forall x :: \Box p.x)]
\]
\[
\equiv \{(6)\}
\]
\[
[\Box (\forall x :: p.x) \Rightarrow (\forall x :: \Box p.x)]
\]
\[
\equiv \{ \text{dummy renaming and distributive properties} \}
\]
\[
(\forall y :: [\Box (\forall x :: p.x) \Rightarrow \Box p.y])
\]
\[
\equiv \{(8)\}
\]
\[
(\forall y :: [\Box true])
\]
\[
\equiv \{(7)\}
\]
\[
(\forall y :: [\true])
\]
\[
\equiv \{ \text{term } \true \}
\]
\[
\true.
\]

Proof of (2):
\[
\neg \delta. \neg p
\]
\[
\equiv \{ \text{definition of } \delta \}
\]
\[
\neg \Box \neg p
\]
\[
\equiv \{(4)\}
\]
\[
\Box p
\]
\[
\equiv \{ \text{definition of } \delta \}
\]
\[
\delta. p .
\]

We have now proved that \( \delta \) is indeed a shift operator. It follows that we may apply Theorem 3 to it. Let us give names to the right hand sides of (17) through (19), i.e., define \( \Box, \Diamond, \mathcal{U} \) by
\[
[\Box p \equiv (\forall j :: \delta^j.p)] , \quad (20)
\]
\[
[\Diamond p \equiv (\exists j :: \delta^j.p)] , \quad (21)
\]
\[
[p \mathcal{U} q \equiv (\exists j :: (\forall i : i < j : \delta^i.p) \land \delta^j.q)] \quad ; \quad (22)
\]
then Theorem 3 informs us that the tuple \( (\Box, \Box, \Diamond, \mathcal{U}) \) satisfies the axioms of temporal logic. In other words, formulae (4) through (15) are also valid when the temporal operators are replaced by their 'dotted' analogues.

Our remaining proof obligations are
\[
[\Box p \equiv \Box p] , \quad (23)
\]
\[
[\Diamond p \equiv \Diamond p] , \quad (24)
\]
\[
[p \mathcal{U} q \equiv p \mathcal{U} q] \quad (25)
\]
for any \( p, q \). In discharging these, we shall not use (20) through (22), but only (4) through (15) and their 'dotted' analogues. On account of symmetry, it will then be sufficient to prove implications rather than equivalences.
The proof of (23) is as follows:

\[
\begin{align*}
\Box p & \Rightarrow \Box p \\
\Leftrightarrow & \quad \{\text{transitivity}\} \\
\Box p & \Rightarrow \Box p \land [\Box (\Box p \Rightarrow p)] \\
\Leftrightarrow & \quad \{(12), (11)\} \\
\Box (\Box p & \Rightarrow \Box p) \land [\Box (\Box p \Rightarrow p)] \\
\Leftrightarrow & \quad \{(7) \text{ on both conjuncts}\} \\
\Box p & \Rightarrow \Box p \land [\Box p \Rightarrow p] \\
\equiv & \quad \{(10), (9)\} \\
\text{true}. \\
\end{align*}
\]

Proof of (24):

\[
\begin{align*}
\Diamond p \\
\equiv & \quad \{(13)\} \\
\neg \Box & \neg p \\
\equiv & \quad \{(23)\} \\

\neg & \neg p \\
\equiv & \quad \{(13)\} \\

\Diamond p. \\
\end{align*}
\]

Proof of (25): In order to shorten the formulae, we introduce the abbreviations

\[
\begin{align*}
r & \text{ for } p \cup q, \\
s & \text{ for } p \cap q, \\
t & \text{ for } r \land \neg s. \\
\end{align*}
\]

Our proof obligation can now be written as \([\neg t]\), which we transform, with the aim of introducing more temporal operators, as follows:

\[
\begin{align*}
[\neg t] \\
\equiv & \quad \{\text{predicate calculus}\} \\
[t & \Rightarrow \Diamond q] \land [t \Rightarrow \neg \Diamond q]. \\
\end{align*}
\]

The first conjunct is immediately seen to be valid, as

\[
[ t \Rightarrow \Diamond q] \\
\Leftrightarrow \quad \{\text{definition of } t\} \\
[r & \Rightarrow \Diamond q] \\
\equiv & \quad \{\text{definition of } r; (14)\} \\
\text{true}. \\
\]

Now for the second conjunct.

\[
[ t \Rightarrow \neg \Diamond q] \\
\equiv & \quad \{(13)\} \\
[t & \Rightarrow \Box \neg q] \\
\Leftrightarrow & \quad \{\text{transitivity}\}
\]
\[ [t \Rightarrow \Box t] \land [\Box t \Rightarrow \Box \neg q] \]

\[ \{ (12), (11) \} \]

\[ [\Box (t \Rightarrow \Diamond t)] \land [\Box (t \Rightarrow \neg q)] \]

\[ \{ (7) on both conjuncts \} \]

\[ [t \Rightarrow \Diamond t] \land [t \Rightarrow \neg q] \]

\[ \{ predicate calculus \} \]

\[ [t \Rightarrow \Box t \land \neg q] \]

the final line follows from

\[ t \]

\[ \{ definition of t \} \]

\[ r \land \neg s \]

\[ \{ definition of r, s; (15) \} \]

\[ (q \lor (p \land \Diamond r)) \land \neg (q \lor (p \land \Diamond s)) \]

\[ \{ de Morgan \} \]

\[ (q \lor (p \land \Diamond r)) \land \neg q \land (\neg p \lor \neg \Diamond s) \]

\[ \{ predicate calculus \} \]

\[ p \land \Diamond r \land \neg q \land (\neg p \lor \neg \Diamond s) \]

\[ \{ predicate calculus \} \]

\[ p \land \Diamond r \land \neg q \land \neg \Diamond s \]

\[ \{ omitting first conjunct \} \]

\[ \Diamond r \land \neg \Diamond s \land \neg q \]

\[ \{ (4) \} \]

\[ \Diamond r \land \Diamond \neg s \land \neg q \]

\[ \{ (6) \} \]

\[ \Diamond (r \land \neg s) \land \neg q \]

\[ \{ definition of t \} \]

\[ \Diamond t \land \neg q \].

5 Applications

Having established the equivalence of the shift operator approach and temporal logic, we now give some examples showing that the former often gives shorter and simpler proofs.

Theorem 5. \[ \Diamond \Box p \equiv \Box \Diamond p \] .

Proof

\[ \Diamond \Box p \]

\[ \{ definitions of \Diamond and \Box \} \]

\[ \delta.(\forall j :: \delta j .p) \]

\[ \{ \delta distributes over logic \} \]

\[ (\forall j :: \delta.(\delta j .p)) \]

\[ \{ associativity of functional composition \} \]
This theorem is proved as Theorem 16 in [13]. There the proof has 14 steps and uses several derived inference rules.

**Theorem 6** \( [p \cup (q \lor r) \equiv (p \cup q) \lor (p \cup r)] \).

**Proof** With \( A.j \) short for \((\forall i : i < j : \delta^i.p)\), we have

\[
p \cup (q \lor r) \\
\equiv (\exists j :: A.j \land \delta^j.(q \lor r)) \\
\equiv (\exists j :: (A.j \land \delta^j.q) \lor (A.j \land \delta^j.r)) \\
\equiv (\exists j :: A.j \land \delta^j.q) \lor (\exists j :: A.j \land \delta^j.r) \\
\equiv (p \cup q) \lor (p \cup r).
\]

This theorem is proved as Theorem 30 in [13]. There the proof has 15 steps and uses an auxiliary theorem.

**Theorem 7** \( [\text{true} \cup p \equiv \Diamond p] \).

**Proof**

\[
\text{true} \cup p \\
\equiv (\exists j :: (\forall i : i < j : \delta^i.\text{true}) \land \delta^j.p) \\
\equiv (\exists j :: \delta^j.p) \\
\equiv (\Diamond p).
\]

The reader that remains sceptical as to the usefulness of our proposals is invited to write down for himself the proof of this theorem, using only the axioms of temporal logic.
6 Discussion

The preceding pages do not provide any new insight into temporal logic: indeed, the connection between linear-time temporal logic and the order type of the natural numbers is well-known [7, chap. 8].

What we have achieved is an equivalent alternative characterization of predicate transformers in the Dijkstra-Scholten calculus that satisfy the axioms of temporal logic. Since the latter resurface in our calculus as theorems, no power has been lost; since all predicate-transformer models can be described in this way, no generality has been lost either. On the other hand, as shown in the preceding section, the calculus often provides proofs that are considerably shorter and more straightforward; moreover, it eliminates the need to remember a large number of axioms.

In particular, verifying that a particular mathematical object satisfies the axioms now merely entails checking the two defining properties of a shift operator. This has been done in [2] for an operational semantics in the style of [1] in order to provide a language-independent definition of the progress predicate transformer proposed in [12]. How to lift the restriction to deterministic computations is discussed in [6].

What we have sacrificed is self-containedness: rather than imposing a formal proof system with its own inference rules, we have elected to work within the framework of mathematics and predicate algebra. From the point of view of applications, however, this can hardly be called a sacrifice at all, as it actually facilitates the embedding of temporal concepts in other systems of description and specification: our calculus constitutes an ‘open semantics’ in the sense of [4], whereas temporal logic does not. The advantages of this are argued at length in [3].

A valid criticism of our approach is that it corresponds to something stronger than ordinary temporal logic: in our system we might express phrases like ‘after k steps’, a possibility lacking in standard versions of temporal logic [10][14]. However, as we have shown, this is an immediate consequence of our decision not to deny ourselves the freedom to employ mathematical techniques.

Another possible criticism centres on the prominent role of the ‘next’ operator, which, although part of linear time temporal logic, enjoys a bad reputation because it does not correspond to macroscopically relevant program properties [11]. It seems that this objection is best addressed by prohibiting the use of ‘next’ in specifications; it does not follow that ‘next’ must be avoided in the construction of the underlying theory.

Acknowledgements Many colleagues at Eindhoven University contributed valuable comments on previous drafts of this paper. In particular, thanks are due to Anne Kaldewaij for first suggesting the need for Theorem 4, and to Ruurd Kuiper for patiently explaining the point of view of temporal logic.

References


### In this series appeared:

<table>
<thead>
<tr>
<th>Number</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>91/03</td>
<td>J.P. Katoen, L.A.M. Schoenmakers</td>
<td>Parallel Programs for the Recognition of P-invariant Segments, p. 16.</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>91/12</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>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>Citation</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>
<tr>
<td>Year</td>
<td>Author(s)</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>92/01</td>
<td>J. Coenen, J. Zwiers, W.-P. de Roever</td>
<td>A note on compositional refinement, p. 27.</td>
</tr>
<tr>
<td>92/02</td>
<td>J. Coenen, J. Hooman</td>
<td>A compositional semantics for fault tolerant real-time systems, p. 18.</td>
</tr>
<tr>
<td>92/03</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Real space process algebra, p. 42.</td>
</tr>
<tr>
<td>92/05</td>
<td>J.P.H.W.v.d.Eijnde</td>
<td>Conservative fixpoint functions on a graph, p. 25.</td>
</tr>
<tr>
<td>92/06</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Discrete time process algebra, p. 45.</td>
</tr>
<tr>
<td>92/07</td>
<td>R.P. Nederpelt</td>
<td>The fine-structure of lambda calculus, p. 110.</td>
</tr>
<tr>
<td>92/10</td>
<td>P.M.P. Rambags</td>
<td>Composition and decomposition in a CPN model, p. 55.</td>
</tr>
<tr>
<td>92/13</td>
<td>F. Kamareddine</td>
<td>Set theory and nominalisation, Part II, p. 22.</td>
</tr>
<tr>
<td>92/14</td>
<td>J.C.M. Baeten</td>
<td>The total order assumption, p. 10.</td>
</tr>
<tr>
<td>92/15</td>
<td>F. Kamareddine</td>
<td>A system at the cross-roads of functional and logic programming, p. 36.</td>
</tr>
<tr>
<td>92/16</td>
<td>R.R. Seljée</td>
<td>Integrity checking in deductive databases; an exposition, p. 32.</td>
</tr>
<tr>
<td>92/17</td>
<td>W.M.P. van der Aalst</td>
<td>Interval timed coloured Petri nets and their analysis, p. 20.</td>
</tr>
<tr>
<td>92/18</td>
<td>R.Nederpelt, F. Kamareddine</td>
<td>A unified approach to Type Theory through a refined lambda-calculus, p. 30.</td>
</tr>
<tr>
<td>92/20</td>
<td>F.Kamareddine</td>
<td>Are Types for Natural Language? P. 32.</td>
</tr>
<tr>
<td>92/21</td>
<td>F.Kamareddine</td>
<td>Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.</td>
</tr>
<tr>
<td>Year</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>92/22</td>
<td>R. Nederpelt, F. Kamareddine</td>
<td>A useful lambda notation, p. 17.</td>
</tr>
<tr>
<td>92/23</td>
<td>F. Kamareddine, E. Klein</td>
<td>Nominalization, Predication and Type Containment, p. 40.</td>
</tr>
<tr>
<td>92/24</td>
<td>M. Codish, D. Dams, Eyal Yardeni</td>
<td>Bottum-up Abstract Interpretation of Logic Programs, p. 33.</td>
</tr>
<tr>
<td>92/25</td>
<td>E. Poll</td>
<td>A Programming Logic for F0, p. 15.</td>
</tr>
<tr>
<td>93/01</td>
<td>R. van Geldrop</td>
<td>Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.</td>
</tr>
<tr>
<td>93/02</td>
<td>T. Verhoeff</td>
<td>A continuous version of the Prisoner’s Dilemma, p. 17</td>
</tr>
<tr>
<td>93/03</td>
<td>T. Verhoeff</td>
<td>Quicksort for linked lists, p. 8.</td>
</tr>
<tr>
<td>93/04</td>
<td>E.H.L. Aarts, J.H.M. Korst, P.J. Zwietering</td>
<td>Deterministic and randomized local search, p. 78.</td>
</tr>
<tr>
<td>93/05</td>
<td>J.C.M. Baeten, C. Verhoef</td>
<td>A congruence theorem for structured operational semantics with predicates, p. 18.</td>
</tr>
<tr>
<td>93/06</td>
<td>J.P. Veltkamp</td>
<td>On the unavoidability of metastable behaviour, p. 29</td>
</tr>
<tr>
<td>93/07</td>
<td>P.D. Moerland</td>
<td>Exercises in Multiprogramming, p. 97</td>
</tr>
<tr>
<td>93/08</td>
<td>J. Verhoosel</td>
<td>A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.</td>
</tr>
<tr>
<td>93/10</td>
<td>K.M. van Hee</td>
<td>Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.</td>
</tr>
<tr>
<td>Paper-ID</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>-------</td>
</tr>
<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>
</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>
</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>
</tr>
<tr>
<td>93/19</td>
<td>G-J. Houben</td>
<td>The Design of an Online Help Facility for ExSpect, p.21.</td>
</tr>
<tr>
<td>93/22</td>
<td>E. Poll</td>
<td>A Typechecker for Bijective Pure Type Systems, p. 28.</td>
</tr>
<tr>
<td>93/23</td>
<td>E. de Kogel</td>
<td>Relational Algebra and Equational Proofs, p. 23.</td>
</tr>
<tr>
<td>93/24</td>
<td>E. Poll and Paula Severi</td>
<td>Pure Type Systems with Definitions, p. 38.</td>
</tr>
<tr>
<td>93/26</td>
<td>W.M.P. van der Aalst</td>
<td>Multi-dimensional Petri nets, p. 25.</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>
</tr>
<tr>
<td>93/28</td>
<td>F. Kamareddine and R. Nederpelt</td>
<td>A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.</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>
</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>
</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>
</tr>
<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>
</tbody>
</table>