An algebraic semantics of basic message sequence charts

Eindhoven University of Technology

Department of Mathematics and Computing Science

An Algebraic Semantics of Basic Message Sequence Charts

by

S. Mauw and M.A. Reniers

94/17

Computing Science Note 94/17
Eindhoven, April 1994

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

An Algebraic Semantics of Basic Message Sequence Charts

S. MAUW AND M. A. RENIERS

Dept. of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.
e-mail: sjouke@win.tue.nl, michelr@win.tue.nl

Message Sequence Charts are a widely used technique for the visualization of the communications between system components. We present a formal semantics of Basic Message Sequence Charts, exploiting techniques from process algebra. This semantics is based on the semantics of the full language as being proposed for standardization in the International Telecommunication Union.

1. INTRODUCTION

Message Sequence Charts are a graphical language, being standardized by the ITU–TS (the Telecommunication Standardization section of the International Telecommunication Union, the former CCITT), for the description of the interactions between entities. ITU recommendation Z.120 (CCI92) contains the syntax and an informal explanation of the semantics. The current goal in the process of standardization is the definition of a formal semantics of the language. The need for a formal semantics became evident since even experts in the field of Message Sequence Charts could not always agree on the interpretation of specific features. Furthermore validation of computer tools for Message Sequence Charts only makes sense if an exact meaning is available. Finally a formal semantics will help to harmonize the use of Message Sequence Charts.

There exist several attempts towards such a formal semantics. We mention approaches based on automaton theory (LL94), Petri net theory (GGR93) and on process algebra (dM93, MvWW93). None of these papers contain a formal semantics of the complete language. Although all approaches have their advantages and disadvantages, it has been decided by the standardization committee to use process algebra for the formal definition. The semantics in this paper is based on a complete algebraic semantics of Message Sequence Charts, which is the proposal for Z.120. We will not present the complete semantics here, but we restrict us to the core of the Message Sequence Charts language, which we will call Basic Message Sequence Charts.

This work is related to the formal semantics of Interworkings (MvWW93). A difference is that we will consider asynchronous communication whereas the theory of Interworkings only contains synchronous communication. Furthermore, Message Sequence Charts and Interworkings have a different approach with respect to their textual representation. Interworkings are event oriented, which means that an Interworking is a list of communications and other events, whereas Message Sequence Charts are instance oriented. This means that a Message Sequence Chart is described by giving the behavior of every instance in separation.

The formal semantics presented is based on the algebraic theory of process description *ACP* (Algebra of Communicating Processes) (BW90). *ACP* is an algebraic theory in many ways related to the algebraic process theories *CCS* (Calculus of Communicating Systems) (Mil80) and *CSP* (Communicating Sequential Processes) (Hoa85). This process algebra is a useful framework for the description of the formal semantics of Message Sequence Charts since all features incorporated in the theory of Message Sequence Charts are related to topics already studied in process algebra such as the state operator and the global renaming operator. Since we consider asynchronous communication and since Message Sequence Charts may be "empty", we use PA_ε, i.e. *ACP* without communication and with the empty process (BW90).

This paper is structured in the following way. First we will introduce Basic Message Sequence Charts. After that, we define the algebraic theory we use as a framework and the algebraic features specifically needed for Basic Message Sequence Charts. Next we will define the semantic function which maps Basic Message Sequence Charts into process terms and we will give an operational semantics. Finally we will prove a representation theorem which shows the relation between the instance oriented notation and an event oriented notation.

2. BASIC MESSAGE SEQUENCE CHARTS

2.1. Introduction

Message Sequence Charts provide a graphical notation for the interaction between system components. Their main application, in addition to SDL (CCI88), is in the area of telecommunication systems. Their use, however, is not restricted to the SDL methodology or to telecommunication environments.

A Message Sequence Chart is not a description of the complete behavior of a system, it merely expresses one execution trace. A collection of Message Sequence Charts may be used to give a more detailed specification of a system. Message Sequence Charts and related notations, such as Interworkings and Arrow Diagrams have been applied in systems engineering for quite some time. They are used in several phases of system development, such as requirement specification, interface specification, simulation, validation, test case specification and documentation.

A Message Sequence Chart contains the description of the asynchronous communication between instances. The complete Message Sequence Chart language, in addition, has primitives for local actions, timers (set, reset and time-out), process creation, process stop and coregions. Furthermore sub Message Sequence Charts and conditions can be used to construct modular specifications.

For brevity, we restrict ourselves in this paper to the core language of Message Sequence Charts, which we will call Basic Message Sequence Charts. A Basic Message Sequence Chart concentrates on communications and local actions only. These are the features encountered in most languages comparable to Message Sequence Charts.

2.2. Graphical notation

A Basic Message Sequence Chart contains a (partial) description of the communication behavior of a number of instances. An instance is an abstract entity of which one can observe (part of) the interaction with other instances or with the environment. The first Basic Message Sequence Chart in Figure 1 defines the communication behavior between instances $i1$, $i2$, $i3$ and $i4$. An instance is denoted by a vertical axis. The time along each axis runs from top to bottom.

A communication between two instances is represented by an arrow which starts at the sending instance and ends at the receiving instance. In Figure 1 we consider the messages $m1$, $m2$, $m3$ and $m4$. Message $m0$ is sent to the environment. The behavior of the environment is not specified. For instance $i2$ we also define a local action a.

Although the activities along one single instance axis are completely ordered, we will not assume a notion of global time. The only dependencies between the timing of the instances come from the restriction that a message must have been sent before it is received. In Figure 1 this implies for example that message $m3$ is received by $i4$ only after it has been sent by $i3$, and, consequently, after the reception of $m2$ by $i3$. Thus $m1$ and $m3$ are ordered in time, while for $m4$ and $m3$ no order is specified. The execution of a local action is only restricted by the ordering of events on its own instance. The second Basic Message Sequence Chart in Figure 1 defines the same Basic Message Sequence Chart, but in an alternative drawing.

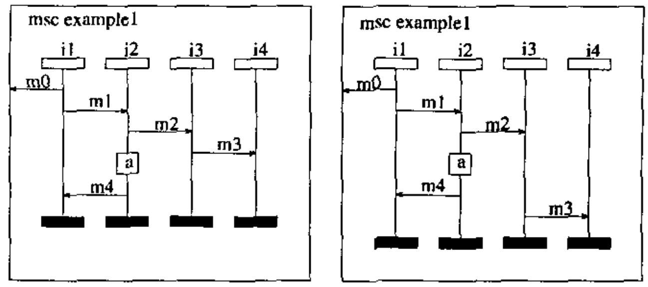

FIGURE 1. Example Basic Message Sequence Charts

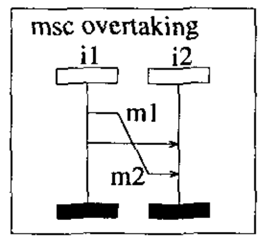

FIGURE 2. Basic Message Sequence Chart with overtaking

Since we have asynchronous communication, it would even be possible to first send $m3$, then send and receive $m4$, and finally receive $m3$. Another consequence of this mode of communication is that we allow overtaking of messages, as expressed in Figure 2.

2.3. Textual notation

Although the application of Message Sequence Charts is mainly focussed on the graphical notation, they have a concrete textual syntax. This representation was originally intended for exchanging Message Sequence Charts between computer tools only, but in this paper we will use it for the definition of the semantics.

The textual representation of a Basic Message Sequence Chart is instance oriented. This means that a Basic Message Sequence Chart is defined by specifying the behavior of all instances. A message output is denoted by "`out m1 to i2;`" and a message input by "`in m1 from i1;`". The Basic Message Sequence Charts of Figure 1 have the following textual representation.

```
msc example1;
  instance i1;
    out m0 to env;
    out m1 to i2;
    in m4 from i2;
  endinstance;
  instance i2;
    in m1 from i1;
```

```
      out m2 to i3;
      action a;
      out m4 to i1;
    endinstance;
    instance i3;
      in m2 from i2;
      out m3 to i4;
    endinstance;
    instance i4;
      in m3 from i3;
    endinstance;
  endmsc;
```

The grammar defining the syntax of textual Basic Message Sequence Charts is given in Table 1. The nonterminals `<mscid>`, `<iid>`, `<mid>` and `<aid>` represent identifiers. The symbol `<>` denotes the empty string. The following identifiers are reserved keywords: `action`, `endinstance`, `endmsc`, `env`, `from`, `in`, `instance`, `msc`, `out` and `to`.

<msc>	::=	msc <mscid>;
		<msc body> endmsc;
<msc body>	::=	<> |
		<inst def> <msc body>
<inst def>	::=	instance <iid>;
		<inst body> endinstance;
<inst body>	::=	<> |
		<event> <inst body>
<event>	::=	in <mid> from <iid>; |
		in <mid> from env; |
		out <mid> to <iid>; |
		out <mid> to env; |
		action <aid>;

TABLE 1. The concrete textual syntax of Basic Message Sequence Charts

The language generated by a nonterminal `X` in the grammar of Table 1 will be denoted by $\mathcal{L}(\mathtt{X})$.

We formulate two static requirements for Basic Message Sequence Charts. The first is that an instance may be declared only once. The second is that every message identifier occurs exactly once in an output action and once in a matching input action, or in case of a communication with the environment a message identifier occurs only once.

3. PROCESS ALGEBRA PA_ε

3.1. Introduction

The process algebra PA_ε is an algebraic theory for the description of process behavior (BK84, BW90). Such an algebraic theory is given by a signature defining the processes and a set of equations defining the equality relation on these processes. In Subsection 3.2. we will give the signature Σ_{PA_ε} and the set of equations E_{PA_ε} will be given in Subsection 3.3.

PA_ε is parameterized with the set of atomic actions. In the following section we will instantiate this set of atomic actions and extend the theory.

The signature of PA_ε specifies the constant and function symbols that may be used in describing processes. Also variables from some set V may be used in process descriptions.

3.2. The signature of PA_ε

Before we turn to the signature of PA_ε we will define the terms associated to a signature Σ and a set of variables V. A signature Σ is a set of constant and function symbols. For every function symbol in the signature its arity is specified.

DEFINITION 3.1. *Let Σ be a signature and let V be a set of variables. Terms over signature Σ with variables from V are defined inductively by*

1. *$v \in V$ is a term*
2. *if $c \in \Sigma$ is a constant symbol, then c is a term*
3. *if $f \in \Sigma$ is an n-ary ($n \geq 1$) function symbol and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term*

The set of all terms over a signature Σ with variables from V is denoted by $T(\Sigma, V)$. A term $t \in T(\Sigma, V)$ is called a closed term if t does not contain variables. The set of all closed terms over a signature Σ is denoted by $T(\Sigma)$.

Now we are ready to turn to the signature Σ_{PA_ε} of PA_ε. The signature Σ_{PA_ε} consists of

1. the special constants δ and ε
2. the set of unspecified constants A
3. the unary operator $\surd$
4. the binary operators $+$, $\cdot$, $\|$ and $\lfloor\!\lfloor$

The special constant δ denotes the process that has stopped executing actions and cannot proceed. This constant is called *deadlock*. The special constant ε denotes the process that is only capable of terminating successfully. It is called the *empty process*.

The elements of the set of unspecified constants A are called *atomic actions*. These are the smallest processes in the description. This set is considered a parameter of the theory. We will specify this set as soon as we consider an application of the theory.

The binary operators $+$ and $\cdot$ are called the *alternative* and *sequential composition*. The alternative composition of the processes x and y is the process that either executes process x or y but not both. The sequential composition of the processes x and y is the process that first executes process x, and upon completion thereof starts with the execution of process y.

The binary operator $\|$ is called the *free merge*. The free merge of the processes x and y is the process that executes the processes x and y in parallel. For a finite set $D = \{d_1, \cdots, d_n\}$, the notation $\|_{d \in D} P(d)$ is an abbreviation for $P(d_1) \| \cdots \| P(d_n)$. If $D = \emptyset$ then $\|_{d \in D} P(d) = \varepsilon$. For the definition of the merge we use two auxiliary operators. The *termination operator* $\surd$ applied to a process x signals whether or not the process x has an option to terminate immediately. The binary operator $⌊\!⌊$ is called the *left merge*. The left merge of the processes x and y is the process that first has to execute an atomic action from process x, and upon completion thereof executes the remainder of process x and process y in parallel.

3.3. The equations of PA_ε

The set of equations E_{PA_ε} of PA_ε specifies which processes are considered equal. An equation is of the form $t_1 = t_2$, where $t_1, t_2 \in T(\Sigma_{PA_\varepsilon}, V)$. For $a \in A \cup \{\delta\}$ and $x, y, z \in V$, the equations of PA_ε are given in the Table 2.

$x + y \ = \ y + x$	A1
$(x+y)+z \ = \ x+(y+z)$	A2
$x + x \ = \ x$	A3
$(x+y) \cdot z \ = \ x \cdot z + y \cdot z$	A4
$(x \cdot y) \cdot z \ = \ x \cdot (y \cdot z)$	A5
$x + \delta \ = \ x$	A6
$\delta \cdot x \ = \ \delta$	A7
$x \cdot \varepsilon \ = \ x$	A8
$\varepsilon \cdot x \ = \ x$	A9
$x \| y \ = \ x ⌊\!⌊ y + y ⌊\!⌊ x + \surd(x) \cdot \surd(y)$	TM1
$\varepsilon ⌊\!⌊ x \ = \ \delta$	TM2
$a \cdot x ⌊\!⌊ y \ = \ a \cdot (x \| y)$	TM3
$(x+y) ⌊\!⌊ z \ = \ x ⌊\!⌊ z + y ⌊\!⌊ z$	TM4
$\surd(\varepsilon) \ = \ \varepsilon$	TE1
$\surd(a \cdot x) \ = \ \delta$	TE2
$\surd(x+y) \ = \ \surd(x) + \surd(y)$	TE3

TABLE 2. Axioms of PA_ε

Axioms A1–A9 are well known. The axioms TE1–TE3 express that a process x has an option to terminate immediately if $\surd(x) = \varepsilon$, and that $\surd(x) = \delta$ otherwise. In itself the termination operator is not very interesting, but in defining the free merge we need this operator to express the case in which both processes x and y are incapable of executing an atomic action. Axiom TM1 expresses that the free merge of the two processes x and y is their interleaving. This is expressed in the three summands. The first two state that x and y may start executing. The third summand expresses that if both x and y have an option to terminate, their merge has this option too.

LEMMA 3.1. *For $x, y, z \in T(\Sigma_{PA_\varepsilon})$ and $a \in A \cup \{\delta\}$*

1. $x \| \varepsilon = x$
2. $x \| y = y \| x$
3. $(x \| y) \| z = x \| (y \| z)$
4. $a ⌊\!⌊ x = ax$

Proof. See (BW90). ∎

We can use this lemma to derive the following example.

$$
\begin{aligned}
& a \| (b+\varepsilon) = \\
& a ⌊\!⌊ (b+\varepsilon) + (b+\varepsilon) ⌊\!⌊ a + \surd(a)\surd(b+\varepsilon) = \\
& a(b+\varepsilon) + b ⌊\!⌊ a + \varepsilon ⌊\!⌊ a + \delta(\delta+\varepsilon) = \\
& a(b+\varepsilon) + ba + \delta + \delta = \\
& a(b+\varepsilon) + ba
\end{aligned}
$$

4. A PROCESS ALGEBRA FOR BASIC MESSAGE SEQUENCE CHARTS

In this section we will extend the process algebra PA_ε to a process algebra PA_{BMSC}. We do this by specifying the set of atomic actions A and by introducing the auxiliary operator λ_M.

4.1. Specifying the atomic actions

In dealing with Basic Message Sequence Charts we encounter a number of significantly different atomic actions. These are, with their representations in PA_{BMSC}:

1. the execution of an action *aid* by instance *i*: $action(i, aid)$
2. the sending of a message m by instance s to instance r: $out(s, r, m)$
3. the sending of a message m by instance s to the environment: $out(s, env, m)$
4. the receiving of a message m by instance r from instance s: $in(s, r, m)$
5. the receiving of a message m by instance r from the environment: $in(env, r, m)$

In Table 3 the sets of atomic actions are given. We use IID for $\mathcal{L}$(`<iid>`), AID for $\mathcal{L}$(`<aid>`) and MID for $\mathcal{L}$(`<mid>`).

A_a	$=$	$\{action(i, aid) \mid i \in IID, aid \in AID\}$
A_o	$=$	$\{out(s, r, m) \mid s, r \in IID, m \in MID\}$
A_i	$=$	$\{in(s, r, m) \mid s, r \in IID, m \in MID\}$
A_e	$=$	$\{out(s, env, m) \mid s \in IID, m \in MID\}$ $\cup \{in(env, r, m) \mid r \in IID, m \in MID\}$
A	$=$	$A_a \cup A_o \cup A_i \cup A_e$

TABLE 3. The atomic actions of PA_{BMSC}

4.2. The state operator λ_M

A Basic Message Sequence Chart specifies a (finite) number of instances that communicate by sending and receiving messages. A message is divided into two parts: a message output and a message input. The correspondence between message outputs and message inputs has to be defined uniquely by message name identification.

A message input may not be executed before the corresponding message output has been executed. We introduce an operator λ_M that enables only those execution paths that respect the above constraint. The operator λ_M is an instance of the state operator as can be found in (BW90). This operator remembers all message outputs that have been executed in a set M and only allows a message input if its corresponding message output is in that set.

For all $M \subseteq A_o$, $x, y \in V$, $a \in A$, $i, j \in \mathcal{L}(\texttt{<iid>})$, and $m \in \mathcal{L}(\texttt{<mid>})$, we define the state operator λ_M in Table 4.

$\lambda_M(\varepsilon) = \varepsilon$	if $M = \emptyset$
$\lambda_M(\varepsilon) = \delta$	if $M \neq \emptyset$
$\lambda_M(\delta) = \delta$	
$\lambda_M(a \cdot x) = a \cdot \lambda_M(x)$	if $a \notin A_o \cup A_i$
$\lambda_M(out(i,j,m) \cdot x) =$ $out(i,j,m) \cdot \lambda_{M \cup \{out(i,j,m)\}}(x)$	
$\lambda_M(in(i,j,m) \cdot x) =$ $in(i,j,m) \cdot \lambda_{M \setminus \{out(i,j,m)\}}(x)$	if $out(i,j,m) \in M$
$\lambda_M(in(i,j,m) \cdot x) = \delta$	if $out(i,j,m) \notin M$
$\lambda_M(x + y) = \lambda_M(x) + \lambda_M(y)$	

TABLE 4. Axioms for the state operator λ_M

Note that the state operator λ_M can be eliminated from every closed PA_{BMSC} term. This means that we have not introduced new processes. Furthermore we have not introduced new identities between existing processes, thus PA_{BMSC} is a conservative extension of PA_ε.

5. THE SEMANTICS OF BASIC MESSAGE SEQUENCE CHARTS

5.1. Introduction

In this section we will define a semantic function S that associates to every Basic Message Sequence Chart in textual format a closed PA_{BMSC} term. An example of this construction is given in subsection 5.3. Before we give the definition of this semantic function we need to explain some auxiliary functions. The powerset of a set S is denoted by $\mathbb{P}(S)$.

The function

$$Instances : \mathcal{L}(\texttt{<msc>}) \rightarrow \mathbb{P}(\mathcal{L}(\texttt{<inst def>}))$$

that associates to a Basic Message Sequence Chart the set containing all instance definitions of the instances defined in the chart, is defined by

$$\begin{aligned} &Instances(\texttt{msc <mscid>; <msc body> endmsc;}) = \\ &\quad Instances_{body}(\texttt{<msc body>}) \end{aligned}$$

where the function

$$Instances_{body} : \mathcal{L}(\texttt{<msc body>}) \rightarrow \mathbb{P}(\mathcal{L}(\texttt{<inst def>}))$$

is defined by

$$\begin{aligned} &Instances_{body}(\texttt{<>}) = \emptyset \\ &Instances_{body}(\texttt{<inst def><msc body>}) = \\ &\quad \{\texttt{<inst def>}\} \cup Instances_{body}(\texttt{<msc body>}) \end{aligned}$$

Next we define the following two functions

$$\begin{aligned} &Name : \mathcal{L}(\texttt{<inst def>}) \rightarrow \mathcal{L}(\texttt{<iid>}) \\ &Body : \mathcal{L}(\texttt{<inst def>}) \rightarrow \mathcal{L}(\texttt{<inst body>}) \end{aligned}$$

These functions associate to an instance definition its name and body.

$$\begin{aligned} &Name(\texttt{instance <iid>;} \\ &\quad \texttt{<inst body> endinstance;}) = \texttt{<iid>} \\ &Body(\texttt{instance <iid>;} \\ &\quad \texttt{<inst body> endinstance;}) = \texttt{<inst body>} \end{aligned}$$

5.2. The semantic function

The general idea is that the semantics of a Basic Message Sequence Chart is the free merge of the semantics of its constituent instances. By this construction we enable all interleavings of the message outputs and message inputs. However, a message input can only be performed after its corresponding message output. In order to rule out all interleavings where a message output is preceded by the corresponding message input we use the state operator λ_M. We define the function $S : \mathcal{L}(\texttt{<msc>}) \rightarrow T(\Sigma_{PA_{BMSC}})$ by

$$S[\![msc]\!] = \lambda_\emptyset \left(\|_{idef \in Instances(msc)} S_{inst}[\![idef]\!] \right)$$

The semantic function $S_{inst} : \mathcal{L}(\texttt{<inst def>}) \rightarrow T(\Sigma_{PA_{BMSC}})$ is defined to express the semantics of one instance in separation. In the textual representation of an instance the atomic actions are specified in the order they are to be executed, thus the semantics of an instance definition is the sequential composition of its actions.

$$S_{inst}[\![idef]\!] = S_{body}^{Name(idef)}[\![Body(idef)]\!]$$

where for $i \in \mathcal{L}(\texttt{<iid>})$ the function

$$S_{body}^i : \mathcal{L}(\texttt{<inst body>}) \rightarrow T(\Sigma_{PA_{BMSC}})$$

is defined by

$$\begin{aligned} &S_{body}^i[\![\texttt{<>}]\!] = \varepsilon \\ &S_{body}^i[\![\texttt{<event><inst body>}]\!] = \\ &\quad S_{event}^i[\![\texttt{<event>}]\!] \cdot S_{body}^i[\![\texttt{<inst body>}]\!] \end{aligned}$$

and for every $i \in \mathcal{L}(\texttt{<iid>})$ the function

$$S^i_{event} : \mathcal{L}(\texttt{<event>}) \to T(\Sigma_{PA_{BMSC}})$$

is defined by

$$\begin{array}{l}
S^i_{event}[\![\texttt{in <mid> from <iid>;}]\!] = \\
\quad in(\texttt{<iid>}, i, \texttt{<mid>}) \\
S^i_{event}[\![\texttt{in <mid> from env;}]\!] = in(env, i, \texttt{<mid>}) \\
S^i_{event}[\![\texttt{out <mid> to <iid>;}]\!] = \\
\quad out(i, \texttt{<iid>}, \texttt{<mid>}) \\
S^i_{event}[\![\texttt{out <mid> to env;}]\!] = out(i, env, \texttt{<mid>}) \\
S^i_{event}[\![\texttt{action <aid>;}]\!] = action(i, \texttt{<aid>})
\end{array}$$

Note that application of the state operator gives the possibility that the semantics of a Basic Message Sequence Chart contains a deadlock. This can be interpreted as the fact that every execution trace contains an input before the corresponding output.

5.3. An example

We consider the Basic Message Sequence Chart from Figure 3. It consists of three instances which exchange two messages.

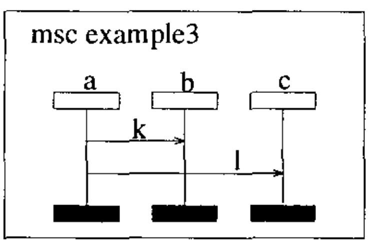

FIGURE 3. Example Basic Message Sequence Chart

```
msc example3;
  instance a;
    out k to b;
    out l to c;
  endinstance'
  instance b;
    in k from a;
  endinstance;
  instance c;
    in l from a;
  endinstance;
endmsc;
```

The interpretation of this Basic Message Sequence Chart is that along instance a the ordering of the output of messages k and l is fixed and furthermore that the output of message k comes before the input of message k and, likewise, that the output of message l comes before the input of message l. These are the only restrictions that apply.

When using the textual syntax, the Basic Message Sequence Chart is represented by describing the behavior of every instance in separation. After applying the semantic function S_{inst} to these instances we obtain

$$\begin{array}{ll}
S_{inst}[\![a]\!] = & out(a,b,k) \cdot out(a,c,l) \\
S_{inst}[\![b]\!] = & in(a,b,k) \\
S_{inst}[\![c]\!] = & in(a,c,l)
\end{array}$$

The first step in deriving the expression which we aim at is putting the instances a, b and c in parallel.

$$S_{inst}[\![a]\!] \,||\, S_{inst}[\![b]\!] \,||\, S_{inst}[\![c]\!]$$

After some calculations, we arrive at the following normalized expression.

$$\begin{array}{llll}
 & out(a,b,k) \cdot (in(a,b,k) \cdot & (out(a,c,l) \cdot in(a,c,l) & \\
 & & +in(a,c,l) \cdot out(a,c,l) & \\
 & &) & \\
 & \quad +out(a,c,l) \cdot (in(a,b,k) \cdot in(a,c,l) & & \\
 & & +in(a,c,l) \cdot in(a,b,k) & \\
 & &) & \\
 & \quad +in(a,c,l) \cdot (in(a,b,k) \cdot out(a,c,l) & & \\
 & & +out(a,c,l) \cdot in(a,b,k) & \\
 & &) & \\
 &) & & \\
+in(a,b,k) \cdot & (out(a,b,k) \cdot (in(a,c,l) \cdot out(a,c,l) & & \\
 & & +out(a,c,l) \cdot in(a,c,l) & \\
 & &) & \\
 & \quad +in(a,c,l) \cdot out(a,b,k) \cdot out(a,c,l) & & \\
 &) & & \\
+in(a,c,l) \cdot & (out(a,b,k) \cdot (in(a,b,k) \cdot out(a,c,l) & & \\
 & & +out(a,c,l) \cdot in(a,b,k) & \\
 & &) & \\
 & \quad +in(a,b,k) \cdot out(a,b,k) \cdot out(a,c,l) & & \\
 &) & &
\end{array}$$

This expression clearly shows execution traces which are not desirable, such as $in(a,b,k) \cdot out(a,b,k) \cdot in(a,c,l) \cdot out(a,c,l)$. These traces can be removed by applying the state operator $\lambda_\emptyset$ to this expression. This results in

$$\begin{array}{l}
out(a,b,k) \cdot (in(a,b,k) \cdot out(a,c,l) \cdot in(a,c,l) \\
\qquad +out(a,c,l) \cdot (in(a,b,k) \cdot in(a,c,l) \\
\qquad\qquad +in(a,c,l) \cdot in(a,b,k) \\
\qquad\qquad) \\
\qquad)
\end{array}$$

6. STRUCTURAL OPERATIONAL SEMANTICS

In this section we define a structural operational semantics of Basic Message Sequence Charts in the style of Plotkin (Plo83). For this purpose we define action relations on closed PA_{BMSC} terms. Then we give a graph model for the theory PA_{BMSC}.

6.1. Action relations for PA_{BMSC}

On the set of PA_{BMSC} terms we define a predicate $\downarrow \subseteq T(\Sigma_{PA_{BMSC}})$ and binary relations $\xrightarrow{a} \subseteq T(\Sigma_{PA_{BMSC}}) \times T(\Sigma_{PA_{BMSC}})$ for every $a \in A$. These predicates are defined by means of inference rules, which have the following form.

$$\frac{p_1, \ldots, p_n}{q}$$

This expression means that for every instantiation of variables in $p_1, \ldots, p_n, q$ we can conclude q from $p_1, \ldots, p_n$. If q is a tautology, we omit $p_1, \ldots, p_n$ and the horizontal bar.

The intuitive idea of the predicate $\downarrow$ is as follows: $t\downarrow$ denotes that t has an option to terminate immediately, i.e. ε is a summand of t. For $x, y \in T(\Sigma_{PA_{BMSC}})$, and $M \subseteq A_o$, the predicate $\downarrow$ is defined in Table 5.

$$\varepsilon\downarrow$$

$$\frac{x\downarrow}{(x+y)\downarrow} \quad \frac{x\downarrow,\ y\downarrow}{(x\cdot y)\downarrow} \quad \frac{y\downarrow}{(x+y)\downarrow}$$

$$\frac{x\downarrow}{(\surd(x))\downarrow} \quad \frac{x\downarrow,\ y\downarrow}{(x\,\|\,y)\downarrow} \quad \frac{x\downarrow}{(\lambda_M(x))\downarrow}$$

TABLE 5. The predicate $\downarrow$

The intuitive idea of the binary operator $\xrightarrow{a}$ is as follows: $t \xrightarrow{a} s$ denotes that the process t can execute the atomic action a and after this execution step the resulting process is s. For $x, x', y, y' \in T(\Sigma_{PA_{BMSC}})$, $a \in A$, $M \subseteq A_o$, $i, j \in \mathcal{L}(\text{<iid>})$, and $m \in \mathcal{L}(\text{<mid>})$, the binary relations $\xrightarrow{a}$ are defined in Table 6.

We will illustrate the use of these action relations with an example. Consider the following expression.

$$\lambda_\emptyset(out(a,b,k)\,\|\,in(a,b,k))$$

We have $out(a,b,k) \xrightarrow{out(a,b,k)} \varepsilon$, so we can derive $out(a,b,k)\,\|\,in(a,b,k) \xrightarrow{out(a,b,k)} \varepsilon\,\|\,in(a,b,k)$. From this we can conclude

$$\lambda_\emptyset(out(a,b,k)\,\|\,in(a,b,k)) \xrightarrow{out(a,b,k)} \lambda_{\{out(a,b,k)\}}(\varepsilon\,\|\,in(a,b,k))$$

Next we have $in(a,b,k) \xrightarrow{in(a,b,k)} \varepsilon$, and we can derive $\varepsilon\,\|\,in(a,b,k) \xrightarrow{in(a,b,k)} \varepsilon\,\|\,\varepsilon$. Thus we have

$$\lambda_{\{out(a,b,k)\}}(\varepsilon\,\|\,in(a,b,k)) \xrightarrow{in(a,b,k)} \lambda_\emptyset(\varepsilon\,\|\,\varepsilon)$$

In order to see that this expression has the possibility to terminate, we derive $\varepsilon\downarrow$ and thus $(\varepsilon\,\|\,\varepsilon)\downarrow$, so

$$\lambda_\emptyset(\varepsilon\,\|\,\varepsilon)\downarrow$$

Finally we conclude that the given process $\lambda_\emptyset(out(a,b,k)\,\|\,in(a,b,k))$ can first execute $out(a,b,k)$, then execute $in(a,b,k)$ and finally terminate. Note that this is the only execution sequence that can be derived from the inference rules.

6.2. Graph model for PA_{BMSC}

We will present a model for the theory PA_{BMSC}. This model is a graph model, a set of process graphs modulo bisimulation, that provides a visualization of the action relations from the previous subsection.

A process graph is a finite acyclic graph in which the edges are labeled with an atomic action, and in which every node may have a label $\downarrow$. This label $\downarrow$ indicates whether or not the state represented by the node has an option to terminate immediately. In every process graph there is one special node, the root node.

Two process graphs will be identified if they are *bisimilar*. Two graphs are bisimilar if there is a bisimulation which relates the root nodes. A bisimulation is a binary relation R, satisfying:

- if $R(p,q)$ and $p \xrightarrow{a} p'$, then there is a q' such that $q \xrightarrow{a} q'$ and $R(p',q')$
- if $R(p,q)$ and $q \xrightarrow{a} q'$, then there is a p' such that $p \xrightarrow{a} p'$ and $R(p',q')$
- if $R(p,q)$ then $p\downarrow$ if and only if $q\downarrow$.

THEOREM 6.1. *Bisimulation is a congruence for the signature of PA_{BMSC}.*

Proof. The action rules fit into the syntactical format that is called the *path* format. As a consequence bisimulation is a congruence for the function symbols for which the action rules are defined. We refer to (BV93, GV92) for both the syntactical format and the congruence theorem. ■

Every operator in the signature of PA_{BMSC} can be interpreted in the graph model. Without proof we state that PA_{BMSC} is a complete axiomatization of the graph model.

To every closed process expression we can associate a process graph using the action relations for PA_{BMSC}.

We will give the process graph for the example of the semantics in Figure 4.

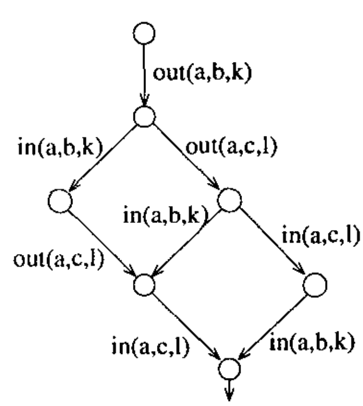

FIGURE 4. Process graph

$$a \xrightarrow{a} \varepsilon$$

$$\frac{x \xrightarrow{a} x'}{x+y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x+y \xrightarrow{a} y'} \qquad \frac{x \xrightarrow{a} x'}{x \cdot y \xrightarrow{a} x' \cdot y} \qquad \frac{x \downarrow,\ y \xrightarrow{a} y'}{x \cdot y \xrightarrow{a} y'}$$

$$\frac{x \xrightarrow{a} x'}{x \,\|\, y \xrightarrow{a} x' \,\|\, y} \qquad \frac{y \xrightarrow{a} y'}{x \,\|\, y \xrightarrow{a} x \,\|\, y'} \qquad \frac{x \xrightarrow{a} x'}{x \,⌊\!\!⌊\, y \xrightarrow{a} x' \,\|\, y}$$

$$\frac{a \notin A_o \cup A_i\ ,\ x \xrightarrow{a} x'}{\lambda_M(x) \xrightarrow{a} \lambda_M(x')} \qquad \frac{x \xrightarrow{out(i,j,m)} x'}{\lambda_M(x) \xrightarrow{out(i,j,m)} \lambda_{M \cup \{out(i,j,m)\}}(x')} \qquad \frac{out(i,j,m) \in M\ ,\ x \xrightarrow{in(i,j,m)} x'}{\lambda_M(x) \xrightarrow{in(i,j,m)} \lambda_{M \setminus \{out(i,j,m)\}}(x')}$$

TABLE 6. The action relations $\xrightarrow{a}$

7. A CHARACTERIZATION THEOREM

In this section we will relate our semantics for instance oriented Message Sequence Charts to the event oriented semantics from (dM93, MvWW93). To this end we will show that a Basic Message Sequence Chart can be represented by a single trace.

First we will define three functions and a predicate on processes. These are the alphabet function α, which determines the atomic actions involved in a process, the function ε_I (for $I \subseteq A$) which renames the atomic actions that are in the set I into ε and the function tr which determines the collection of completed traces of a process. The predicate df determines whether a process is free of deadlocks. For x and y arbitrary processes and $a \in A$, we give the axioms for those functions in Table 7. Note that the predicate $x \neq \delta$ can be defined easily.

$\alpha(\varepsilon) = \emptyset$	
$\alpha(\delta) = \emptyset$	
$\alpha(a \cdot x) = \{a\} \cup \alpha(x)$	
$\alpha(x + y) = \alpha(x) \cup \alpha(y)$	
$\varepsilon_I(\varepsilon) = \varepsilon$	
$\varepsilon_I(\delta) = \delta$	
$\varepsilon_I(a \cdot x) = a \cdot \varepsilon_I(x)$	if $a \notin I$
$\varepsilon_I(a \cdot x) = \varepsilon_I(x)$	if $a \in I$
$\varepsilon_I(x + y) = \varepsilon_I(x) + \varepsilon_I(y)$	
$tr(\varepsilon) = \{\varepsilon\}$	
$tr(\delta) = \{\delta\}$	
$tr(a \cdot x) = \{a \cdot t \mid t \in tr(x)\}$	
$tr(x + y) = tr(x) \cup tr(y)$	if $x \neq \delta \wedge y \neq \delta$
$df(\varepsilon)$	
$\neg df(\delta)$	
$df(a \cdot x) = df(x)$	
$df(x + y) = df(x) \wedge df(y)$	if $x \neq \delta \wedge y \neq \delta$

TABLE 7. Axioms for α, ε_I, tr, and df

First observe the following general properties.

LEMMA 7.1. *For $x, y \in T(\Sigma_{PA_{BMSC}})$, $M \subseteq A_o$ and $I \subseteq A$*

1. $df(y) \wedge \alpha(y) \subseteq I \ \Rightarrow\ \varepsilon_I(x \,\|\, y) = \varepsilon_I(x)$
2. $\alpha(x) \cap I = \emptyset \ \Rightarrow\ \varepsilon_I(x) = x$
3. $\forall_{t \in tr(x)}\ \varepsilon_I(t) \in tr(\varepsilon_I(x))$
4. $df(\lambda_M(x)) \ \Rightarrow\ tr(\lambda_M(x)) \subseteq tr(x)$

Proof. For 2, 3 and 4 we use induction on the structure ε, $a \cdot x$, $x + y$, whereas for 1 we use induction on the structure ε, $\Sigma_{k \in K}\ a_k \cdot x_k$, $\Sigma_{k \in K}\ a_k \cdot x_k + \varepsilon$. ∎

LEMMA 7.2. *For $i \in \mathcal{L}($`<inst def>`$)$*

$$tr(S_{inst}[\![i]\!]) = \{S_{inst}[\![i]\!]\}$$

Proof. This follows immediately from the construction of the semantic function. ∎

In the following lemmas and theorems we will use, for $i \in \mathcal{L}($`<inst def>`$)$, $\alpha(i)$ as an abbreviation of $\alpha(S_{inst}[\![i]\!])$ and *Inst* for *Instances*(*msc*) where *msc* is clear from the context. First we consider traces from $\|_{j \in Inst} S_{inst}[\![j]\!]$ which do not meet the restriction on the order of inputs and corresponding outputs. Using such a trace we can reconstruct the behavior of every single instance and, therefore, we can reconstruct the complete Basic Message Sequence Chart as described in Theorem 7.4. Theorem 7.5 states that this also holds for the restricted traces from $S[\![msc]\!]$. So a Basic Message Sequence Chart can be represented either by a collection of instances (the instance oriented approach) or by a single trace (the event oriented approach).

LEMMA 7.3. *For $msc \in \mathcal{L}($`<msc>`$)$ and $i \in Inst$*

$$\forall_{t \in tr(\|_{j \in Inst} S_{inst}[\![j]\!])}\ \varepsilon_{A \setminus \alpha(i)}(t) = S_{inst}[\![i]\!]$$

Proof. Let $t \in tr\left(\|_{j \in Inst}\ S_{inst}[\![j]\!]\right)$.
Then by applying Lemma 7.1.3 we have: $\varepsilon_{A \setminus \alpha(i)}(t) \in tr\left(\varepsilon_{A \setminus \alpha(i)}\left(\|_{j \in Inst}\ S_{inst}[\![j]\!]\right)\right)$.
We calculate

$$
\begin{array}{ll}
 & \varepsilon_{A\setminus\alpha(i)}\left(\|_{j\in Inst}\ S_{inst}[\![j]\!]\right) \\
= & \{ \text{ Lemma 7.1.1 } \} \\
 & \varepsilon_{A\setminus\alpha(i)}(S_{inst}[\![i]\!]) \\
= & \{ \text{ Lemma 7.1.2 } \} \\
 & S_{inst}[\![i]\!]
\end{array}
$$

So, from Lemma 7.2, we may conclude that $\varepsilon_{A\setminus\alpha(i)}(t) = S_{inst}[\![i]\!]$. ∎

THEOREM 7.4. *For msc* $\in \mathcal{L}(\texttt{<msc>})$

$$\forall_{t\in tr(\|_{i\in Inst} S_{inst}[\![i]\!])}\ S[\![msc]\!] = \lambda_\emptyset\left(\|_{i\in Inst}\ \varepsilon_{A\setminus\alpha(i)}(t)\right)$$

Proof. This follows from Lemma 7.3 and the definition of the semantic function S. ∎

THEOREM 7.5. *For msc* $\in \mathcal{L}(\texttt{<msc>})$ *such that* $df(S[\![msc]\!])$

$$\forall_{t\in tr(S[\![msc]\!])}\ S[\![msc]\!] = \lambda_\emptyset\left(\|_{i\in Inst}\ \varepsilon_{A\setminus\alpha(i)}(t)\right)$$

Proof. This theorem follows immediately from Lemma 7.1.4 and Theorem 7.4. ∎

Theorem 7.5 expresses that, in principle, one could choose an event oriented textual representation for Basic Message Sequence Charts. The Basic Message Sequence Chart from Figure 3 may look like

```
msc example3;
  out k from a to b;
  out l from a to c;
  in l from a to c;
  in k from a to b;
endmsc;
```

8. CONCLUSION

The definition of a formal semantics of Basic Message Sequence Charts based on process algebra as presented in this paper has turned out to be a very natural and successful method. We used the instance oriented syntax to derive a compositional semantics and indicated that this yields a semantics which is equivalent to the approach based on sequencing for an event oriented syntax (dM93, MvWW93).

The development of the semantics for the complete Message Sequence Charts language follows the same line, applying more elaborate constructs from process algebra for features such as sub Message Sequence Charts and process creation.

The algebraic approach towards the definition of the formal semantics of Message Sequence Charts enables the use of term-rewriting systems for the rapid prototyping of specifications (MW93).

Acknowledgements

We would like to thank Jos Baeten, Jan Bergstra, Ekkart Rudolph and Chris Verhoef for their useful comments and suggestions for improvements.

REFERENCES

BK84: J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information & Control*, 60:109–137, 1984.

BV93: J.C.M. Baeten and C. Verhoef. A congruence theorem for structured operational semantics with predicates. In E. Best, editor, *CONCUR'93*, Lecture Notes in Computer Science 715, 1993.

BW90: J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.

CCI88: CCITT. *CCITT Recommendation Z.100: Specification and Description Language (SDL)*. CCITT, Geneva, 1988.

CCI92: CCITT. *CCITT Recommendation Z.120: Message Sequence Chart (MSC)*. CCITT, Geneva, 1992.

dM93: J. de Man. Towards a formal semantics of Message Sequence Charts. In O. Færgemand and A. Sarma, editors, *SDL'93 Using Objects*, Proceedings of the Sixth SDL Forum, Darmstadt, 1993. Elsevier Science Publishers B.V.

GGR93: J. Grabowski, P. Graubmann, and E. Rudolph. Towards a petri net based semantics definition for Message Sequence Charts. In O. Færgemand and A. Sarma, editors, *SDL'93 Using Objects*, Proceedings of the Sixth SDL Forum, Darmstadt, 1993. Elsevier Science Publishers B.V.

GV92: J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100:202–260, 1992.

Hoa85: C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

LL94: P.B. Ladkin and S. Leue. What do Message Sequence Charts mean? In R.L. Tenney, P.D. Amer, and M. Uyar, editors, *Formal Description Techniques, VI*, IFIP Transactions C, Proceedings of the Sixth International Conference on Formal Description Techniques. North-Holland, 1994.

Mil80: R. Milner. *A Calculus of Communicating Systems*. Lecture Notes in Computer Science 92. Springer-Verlag, 1980.

MvWW93: S. Mauw, M. van Wijk, and T. Winter. A formal semantics of synchronous Interworkings. In O. Færgemand and A. Sarma, editors, *SDL'93 Using Objects*, Proceedings of the Sixth SDL Forum, Darmstadt, 1993. Elsevier Science Publishers B.V.

MW93: S. Mauw and T. Winter. A prototype toolset for Interworkings. *Philips Telecommunication Review*, 51(3):41–45, December 1993.

Plo83: G.D. Plotkin. An operational semantics for CSP. In *Proceedings of the Conference on the Formal Description of Programming Concepts*, volume 2, Garmisch, 1983.

Computing Science Notes | **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. Komatzky	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. Doornbos	Mathematical Induction Made Calculational, p. 36.