Delayed choice: an operator for joining Message Sequence Charts

J.C.M. Baeten and S. Mauw

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

We study the extension of a simple process algebra with the delayed choice operator. It differs from the normal non-deterministic choice in that the moment of choice is delayed until all alternatives can be distinguished by their first action. An application is in joining Message Sequence Charts.

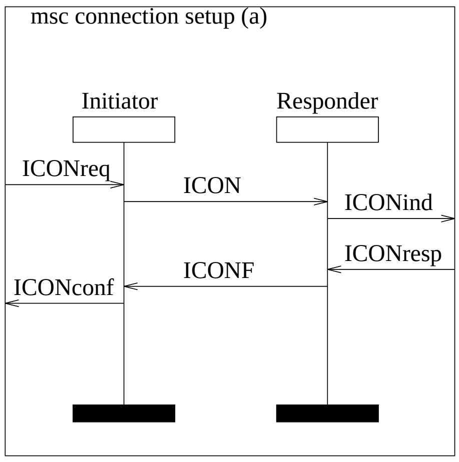
1. INTRODUCTION

Message Sequence Charts provide a graphical method for the description of the interactions between system components. The ITU-TS (the Telecommunication Standardization Section of the International Telecommunication Union, the former CCITT) maintains recommendation Z.120 [13] which contains the syntax and an informal explanation of the semantics of Message Sequence Charts. Current developments are the definition of a formal semantics based on branching time process algebra [16, 17] and the extension of the MSC language with operators [12].

The current proposal of the formal semantics only considers single Message Sequence Charts, while the application of Message Sequence Charts often consists of a collection of Message Sequence Charts. Sometimes the intended meaning is the sequential composition of the charts contained, while in other cases they are viewed as alternatives. The semantics of such a collection thus can only be defined if the relation between the contained Message Sequence Charts is explicitly stated. It is proposed to use operators for this purpose.

In [10] several operators are proposed: alternative composition ($+$), sequential composition ($\cdot$), weak sequential composition ($\circ$) and parallel composition ($\|$). We think however that in a branching time setting the non-deterministic alternative composition operator is not always the intention of the user. In branching time the moment of choice is taken into consideration, so, for example, a distinction is made between the processes $a(b+c)$ and $ab+ac$. In the first expression the choice between b and c is made after executing action a, while in the second expression the choice is made before a. Now consider the following typical use of Message Sequence Charts (see Figure 1).

The semantics of the first drawing is that first a message *ICONreq* is received from the environment by the *Initiator*. Thereafter the message *ICON* is being sent to the *Responder*, which sends an *ICONind* to the environment and receives an *ICONresp* from the environment. The *Responder* sends *ICONF* back to the *Initiator*, who sends an *ICONconf* to the environment. The second drawing displays an alternative scenario in which due to a timeout (which was left out of the picture), a failure *IDISind* is reported to the environment.



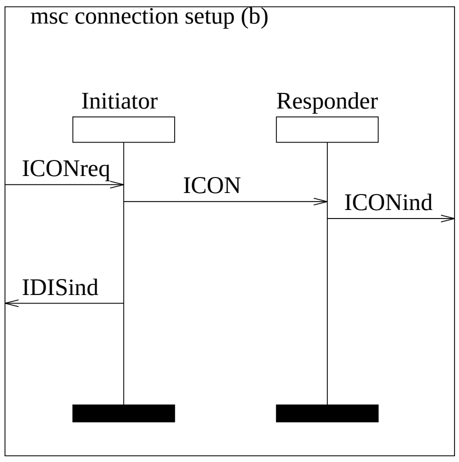


Figure 1. Two Message Sequence Charts

If we combine these two Message Sequence Charts by means of the alternative composition operator in a branching time setting, it will mean that the choice between correct operation and faulty operation has been made before reception of the *ICONreq*. This was obviously not the intention.

In order to be able to combine alternative Message Sequence Charts while sharing their initial behavior, we introduce the *delayed choice* operator. Since it behaves as the choice operator in the setting of trace theory, we will denote it by $\mp$ (Trace-+). The most important property is expressed in the following equation (a, b and c are atomic actions, $b \neq c$).

$$ab \mp ac = a(b + c)$$

while, if the two operands do not share an initial action (i.e. $a \neq c$), we have

$$ab \mp cd = ab + cd$$

In this paper we will define this operator within the framework of BPA_ε. This is the concurrency theory *BPA* (Basic Process Algebra) extended with the empty process ε, which is the neutral element for sequential composition. Introduction of this special process is motivated by the following example. In the expression $a \mp ab$ both operands start with action a. The first operand terminates after execution of a, while the second operand executes b after execution of a. This can only be expressed if we have a special symbol for immediate termination. Using the empty process we obtain $a(\varepsilon + b)$.

Several variants of the alternative composition operator have been defined before, none of which satisfies our needs.

In *TCSP* [8] (Theoretical Communicating Sequential Processes) we have the deterministic choice operator $\Box$ and the non-deterministic choice $\sqcap$. The operator $\Box$ also removes possible non-determinism, but only non-determinism due to internal steps. This operator was defined in a branching time setting in [9], there called τ-angelic choice. The operator we need should remove all non-determinism, also non-determinism due to external, visible actions. Note that TCSP's non-deterministic choice $\sqcap$ was studied in a branching time setting in [2]; there, also a form of choice intermediate between $\Box$ and $\sqcap$ is studied.

The operational rules we will present for our delayed choice operator also appear in [18] for an operator they call angelic non-determinism. In [18], only linear time semantics is considered, and no axiomatization is presented. We present our delayed choice operator in a more general branching time framework, and moreover provide a complete axiomatization for finite processes.

We prefer not to use the term angelic choice. This is because this term stems from a dualistic world, where specified behavior is distinguished from intended behavior. Transferring this notion to the monistic world of process theory will only cause confusion and differences of opinion.

The remainder of this paper is structured as follows. In Section 2 we will give a short introduction to the theory BPA_ε. Section 3 contains an axiomatization and an operational semantics of the delayed choice operator. An example of the use of this operator in the field of Message Sequence Charts is shown in Section 4.

We thank Michel Reniers for proof reading this document and Wim Hesselink for a conversation on angelic choice.

2. BASIC PROCESS ALGEBRA WITH THE EMPTY PROCESS

The process algebra BPA_ε is an algebraic theory for the description of process behavior [5, 6]. This theory is parameterized by a set of unspecified constants A, which are called *atomic actions*. A process is built up from atomic actions, the special constants δ and ε and the operators $\cdot$ and $+$.

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

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 equations of BPA_ε are given in Table 1. Axiom A1, A2 and A3 express that the $+$ is commutative, associative and idempotent. Axiom A4 defines right distributivity of the sequential composition over the alternative composition. Since we consider a branching time theory, we do not have left distributivity. Axiom A5 expresses associativity of the sequential composition. Deadlock is defined by axioms A6 and A7 and the empty process is defined by axioms A8 and A9.

The precedence of the operators is as follows: $\cdot$ binds stronger than all other operators to be introduced, and $+$ binds weaker. The other operators have the same binding power. Brackets are associated to the left. We sometimes use xy to denote $x \cdot y$.

Table 2 defines a structured operational semantics (S.O.S.) in the style of [19]. Predicate $\downarrow$ expresses that a process has an option to terminate. For every atomic action a, predicate $\xrightarrow{a}$ expresses that the first argument can execute action a and become the second argument.

Table 1
Axioms of BPA_ε

$x + y \;=\; y + x$	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

Thus the first rule states that ε has a termination option. The second rule states that whenever x has a termination option then so do $x + y$ and $y + x$, and the third rule states that whenever both x and y have a termination option, then so does $x \cdot y$.

Table 2
Operational semantics for BPA_ε

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

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

Using these predicates it is easy to associate an element of the so-called graph model to every process expression. The domain G of this model consists of finitely branching rooted graphs. Edges are labeled with an atomic action and nodes may be attributed with the symbol $\downarrow$ to indicate the option to terminate successfully.

The model $G/\leftrightarrows$ of BPA_ε is the graph domain divided out by bisimulation. Two graphs are bisimilar if there is a bisimulation relating their 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 $q{\downarrow}$.

3. THE DELAYED CHOICE

3.1. Axioms

The delayed choice between processes x and y, is the process obtained by joining the common initial parts of x and y and continuing with a normal choice between the remaining parts. However, we do not want to remove nondeterministic choices which are internal to x or y. This is expressed in the definition of the delayed choice operator in Table 3. For this definition we need two auxiliary operators. The first is the join operator ($\bowtie$). In $x \bowtie y$ exactly those summands of x and y are selected which have a common initial action. The corresponding summands of x and y are joined with respect to their first common action. The unless operator ($\lhd$) has the opposite behavior. In $x \lhd y$ only those summands from x are selected which share no initial action with y. Now the definition of the delayed choice can be interpreted as follows. There are three options. The first is that a summand of x and a summand of y have a common initial action. In this case the result is a summand where this action is shared ($x \bowtie y$). The second case is that a summand from x has no initial action with any summand of y in common ($x \lhd y$) and the third case is that a summand of y has no initial action with any summand of x in common ($y \lhd x$).

Table 3
Axioms for delayed choice

$x \mp y = x \bowtie y + x \lhd y + y \lhd x$	DC1		
$\varepsilon \bowtie x = \delta$	DC2	$\varepsilon \lhd x = \varepsilon$	DC10
$x \bowtie \varepsilon = \delta$	DC3	$x \lhd \varepsilon = x$	DC11
$\delta \bowtie x = \delta$	DC4	$\delta \lhd x = \delta$	DC12
$x \bowtie \delta = \delta$	DC5	$x \lhd \delta = x$	DC13
$ax \bowtie ay = a(x \mp y)$	DC6	$ax \lhd ay = \delta$	DC14
$a \neq b \Rightarrow ax \bowtie by = \delta$	DC7	$a \neq b \Rightarrow ax \lhd by = ax$	DC15
$(x+y) \bowtie z = x \bowtie z + y \bowtie z$	DC8	$(x+y) \lhd z = (x \lhd z) + (y \lhd z)$	DC16
$x \bowtie (y+z) = x \bowtie y + x \bowtie z$	DC9	$x \lhd (y+z) = (x \lhd y) \lhd z$	DC17

The unless operator $\lhd$ that we have here has a filtering behavior very much like the unless operator used in the axiomatization of the priority operator in [3]; this is the reason we use the same name and notation.

Examples (assuming that a, b, c, d, e and f are distinct atomic actions):

$$(ab + ac) \mp ad = a(b+d) + a(c+d)$$

$$(ab + ac) \mp (ad + ae + f) = a(b+d) + a(c+d) + a(b+e) + a(c+e) + f$$

The theory BPA_ε extended with the delayed choice operator is denoted by $BPA_\varepsilon + DC$.

Table 4
Operational semantics for Delayed Choice

$$\frac{x \xrightarrow{a} x',\; y \xrightarrow{a} y'}{x \mp y \xrightarrow{a} x' \mp y',\; x \bowtie y \xrightarrow{a} x' \mp y'}$$

$$\frac{x \xrightarrow{a} x',\; y \not\xrightarrow{a}}{x \mp y \xrightarrow{a} x',\; y \mp x \xrightarrow{a} x',\; x \triangleleft y \xrightarrow{a} x'}$$

$$\frac{x{\downarrow}}{(x \mp y){\downarrow},\; (y \mp x){\downarrow},\; (x \triangleleft y){\downarrow}}$$

3.2. Structured Operational Semantics

The rules in Table 4 define an operational semantics for the delayed choice operator. In the definition of the $\xrightarrow{a}$ predicate, we make use of so-called negative premises (see [20]). This means that a negation occurs in the condition of an S.O.S. rule. Expression $y \not\xrightarrow{a}$ means that process y cannot execute action a. In Lemma 3.3.4 we prove correctness of this definition. The construction of the graph associated to a process is not as straightforward as in the case without negative premises. Using a layering technique (see [7]) we can derive the so-called *deduction graph*.

We started by formulating the axiomatic laws and afterwards came up with operational rules. This can also be done in reverse order: as [1] shows, there are heuristics on deriving a set of axiomatic laws on the basis of the operational rules. The auxiliary operators $\bowtie$ and $\triangleleft$, needed for a finite axiomatization, arise by splitting up the operational rules into different cases.

3.3. Soundness and Completeness

In this section we prove soundness and completeness of the axioms with respect to the S.O.S. rules. In order to prove that the extension of BPA_ε with the delayed choice is conservative, we use term rewrite techniques. Consider the term rewrite system consisting of the rules A3–A9 from Table 1, all axioms of Table 3 and the additional axioms from Table 5 ($a \neq b$), oriented from left to right. The additional rules are derivable from the axioms for the delayed choice operator. They are needed because in some derivations axiom A8 ($x \cdot \varepsilon = x$) is used from right to left.

Table 5
Additional rules

$a \bowtie a = a$	$a \bowtie b = \delta$	$a \triangleleft a = \delta$	$a \triangleleft b = a$
$a \bowtie ax = a(\varepsilon + x)$	$a \bowtie bx = \delta$	$a \triangleleft ax = \delta$	$a \triangleleft bx = a$
$ax \bowtie a = a(\varepsilon + x)$	$ax \bowtie b = \delta$	$ax \triangleleft a = \delta$	$ax \triangleleft b = ax$

Lemma 3.3.1 *The term rewrite system defined above is strongly normalizing*

Proof We do this by applying the method of the lexicographical path ordering [14, 15]. There is a complication with this, as in the reduction of $\mp$, the operator $\bowtie$ appears, and in the reduction of $ax \bowtie ay$, it is the other way around. The solution is to weigh these operators with the size of their arguments, as is done in [6]. Thus, we get operators $\mp_n$, $\bowtie_n$, and the ordering is as in Figure 2 (a any atomic action).

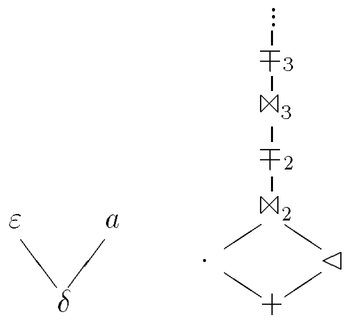


Figure 2. Ordering of operators

Furthermore, we give $\cdot$ the lexicographical status for the first argument, and $\lhd$ the lexicographical status for the second argument. For more information on lexicographical path ordering, we refer to [15]. $\square$

Definition 3.3.2 *Define the class B of basic terms over $BPA_\varepsilon + DC$ as the smallest class satisfying*

1. $\varepsilon, \delta \in B$, $A \subset B$
2. $a \in A, t \in B \Rightarrow a \cdot t \in B$
3. $s, t \in B \Rightarrow s + t \in B$

Theorem 3.3.3 *Let t be a closed $BPA_\varepsilon + DC$-term. Then there is a basic term s with $BPA_\varepsilon + DC \vdash t = s$.*

Proof Using 3.3.1 rewrite t to normal form s. We claim that s must be basic. For, s cannot contain $\mp$ because DC1 can be applied. If s contains $\bowtie$ or $\lhd$, take a smallest sub-term containing one of them, say $s_1 \bowtie s_2$, then we can apply one of DC2–DC9 or the additional rules for $\bowtie$ from Table 5. If this subterm is $s_1 \lhd s_2$ then we can apply one of DC10–DC17 or the additional rules for $\lhd$. It is now easy to finish the proof. $\square$

This is the so-called elimination theorem, saying that the operators newly introduced can be eliminated from closed terms.

Lemma 3.3.4 *Bisimulation is a congruence on the set of deduction graphs generated by the S.O.S. rules.*

Proof The S.O.S. rules for BPA_ε and delayed choice satisfy the *panth* format of [20]. Thus, all we need to do is to provide a stratification for this term deduction system. This is easy: the rank of a step $t \xrightarrow{a} t'$ or a termination option $t\downarrow$ is the number of $\mp$ symbols plus the number of $\lhd$ symbols in t. It is now easy to check that the conditions are met. □

Thus we have that all operators are defined on $G/\leftrightarroweq$, the set of deduction graphs modulo $\leftrightarroweq$.

Theorem 3.3.5 *Soundness:* $G/\leftrightarroweq \models BPA_\varepsilon + DC$.

Proof By [4] we have $G/\leftrightarroweq \models BPA_\varepsilon$. For DC1, consider the relation that relates every closed term of the form $x \mp y$ to the term $x \bowtie y + x \lhd y + y \lhd x$ (and vice versa) and moreover relates every term to itself. On the basis of the definition of a deduction graph, it is easy to verify that this is a bisimulation. All other axioms are equally simple. □

Theorem 3.3.6 $BPA_\varepsilon + DC$ *is a conservative extension of* BPA_ε.

Proof The operational conservativity follows since our operational rules are in panth format, and pure and well-founded (see [21]). Then we get equational conservativity since the axiomatization of BPA_ε is sound and complete (see [4]) and the axiomatization of DC is sound (Theorem 3.3.5). □

Theorem 3.3.7 $BPA_\varepsilon + DC$ *is a complete axiomatization for* $G/\leftrightarroweq$.

Proof See [21]. In addition to the ingredients of the previous proof, all we need is the elimination theorem (Theorem 3.3.3). □

3.4. Properties

The main property proven in this section is the commutativity and associativity of $\mp$. This is not derivable from the axioms, but it holds for all closed terms. We need several lemmas to prove this.

Lemma 3.4.1 *The following identities are derivable from the equations.*

1. $ax \mp ay = a(x \mp y)$
2. $a \neq b \Rightarrow ax \mp by = ax + by$
3. $x \mp \delta = x$
4. $x \mp \varepsilon = x + \varepsilon$
5. $(x \lhd y) \lhd z = (x \lhd z) \lhd y$

Proof

1. $ax \mp ay = ax \bowtie ay + ax \lhd ay + ay \lhd ax = a(x \mp y) + \delta + \delta = a(x \mp y)$.
2. If $a \neq b$ then $ax \mp by = ax \bowtie by + ax \lhd by + by \lhd ax = \delta + ax + by = ax + by$.
3. $x \mp \delta = x \bowtie \delta + x \lhd \delta + \delta \lhd x = \delta + x + \delta = x$.
4. $x \mp \varepsilon = x \bowtie \varepsilon + x \lhd \varepsilon + \varepsilon \lhd x = \delta + x + \varepsilon = x + \varepsilon$.
5. $(x \lhd y) \lhd z = x \lhd (y + z) = x \lhd (z + y) = (x \lhd z) \lhd y$. □

In the following we use that every closed term can be written in the form $\sum a_i x_i (+\varepsilon)$, where $(+\varepsilon)$ denotes an optional summand ε. This follows from the elimination theorem (Theorem 3.3.3).

Lemma 3.4.2 *Let a_i and b_j ($i \in I$, $j \in J$, I and J finite) be atomic actions and let x_i and y_j be processes.*

1. $\sum_i a_i x_i(+\varepsilon) \bowtie \sum_j b_j y_j(+\varepsilon) = \sum_{i,j(a_i=b_j)} a_i(x_i \mp y_j)$

2. $\sum_i a_i x_i \lhd \sum_j b_j y_j(+\varepsilon) = \sum_{i(\forall_j a_i \neq b_j)} a_i x_i$

3. $\left(\sum_i a_i x_i + \varepsilon\right) \lhd \sum_j b_j y_j(+\varepsilon) = \sum_{i(\forall_j a_i \neq b_j)} a_i x_i + \varepsilon$

Proof

1. $\sum_i a_i x_i(+\varepsilon) \bowtie \sum_j b_j y_j(+\varepsilon) = \sum_{i,j}(a_i x_i \bowtie b_j y_j) = \sum_{i,j(a_i=b_j)} a_i(x_i \mp y_j)$
2. $\sum_i a_i x_i \lhd \sum_j b_j y_j(+\varepsilon) = \sum_i (a_i x_i \lhd \sum_j b_j y_j(+\varepsilon)) = \sum_{i(\forall_j a_i \neq b_j)} a_i x_i$

   This last equality can be proven with induction on the number of elements in J.
3. Analogously. □

Define the set of initial actions of a process as in Table 6.

Table 6
Initial actions of a process

$I(\delta) = \emptyset$
$I(\varepsilon) = \emptyset$
$I(ax) = \{a\}$
$I(x + y) = I(x) \cup I(y)$

Lemma 3.4.3 *For closed terms x and y we have*

1. $I(x \bowtie y) = I(x) \cap I(y)$
2. $I(x \lhd y) = I(x) - I(y)$
3. $I(x \mp y) = I(x) \cup I(y)$

Proof This follows directly from the definitions and Lemma 3.4.2. □

The next lemma expresses the fact that in an expression of the form $x \lhd y$, only the initial actions of y matter.

Lemma 3.4.4 *For closed terms x, y and z*

$I(y) = I(z) \Rightarrow x \lhd y = x \lhd z$

Proof Write $x = \sum a_i x_i$, $y = \sum b_j y_j(+\varepsilon)$ and $z = \sum c_k z_k(+\varepsilon)$.
We use the following property: $I(\sum_{j \in J} b_j y_j(+\varepsilon)) = \{b_j : j \in J\}$.

$$\sum_i a_i x_i \lhd \sum_j b_j y_j(+\varepsilon) = \sum_{i(\forall_j a_i \neq b_j)} a_i x_i = \sum_{i(\forall_k a_i \neq c_k)} a_i x_i = \sum_i a_i x_i \lhd \sum_k c_k z_k(+\varepsilon)$$

If x has a summand ε an analogous proof applies. □

Theorem 3.4.5 *For closed terms x, y and z we have*

1. $x \bowtie y = y \bowtie x$
2. $x \mp y = y \mp x$
3. $x \lhd (y \mp z) = x \lhd (y + z) = (x \lhd y) \lhd z$
4. $x \bowtie (y \lhd z) = (x \bowtie y) \lhd z = (x \lhd z) \bowtie y$
5. $x \bowtie (y \bowtie z) = (x \bowtie y) \bowtie z$
6. $x \mp (y \mp z) = (x \mp y) \mp z$

Proof

- (1) and (2) are proven by mutual induction. Again write $x = \sum a_i x_i(+\varepsilon)$, $y = \sum b_j y_j(+\varepsilon)$ and $z = \sum c_k z_k(+\varepsilon)$. First consider (1).

$$\begin{aligned} & \sum_i a_i x_i(+\varepsilon) \bowtie \sum_j b_j y_j(+\varepsilon) \\ = & \sum_{i,j(a_i = b_j)} a_i(x_i \mp y_j) \\ = & \sum_{i,j(a_i = b_j)} a_i(y_j \mp x_i) \\ = & \sum_{i,j(a_i = b_j)} b_j(y_j \mp x_i) \\ = & \sum_j b_j y_j(+\varepsilon) \bowtie \sum_i a_i x_i(+\varepsilon) \end{aligned}$$

For (2) we calculate:
$x \mp y = x \bowtie y + x \lhd y + y \lhd x = y \bowtie x + y \lhd x + x \lhd y = y \mp x$.

- (3) follows from Lemma 3.4.3.3, Lemma 3.4.4 and the definitions.

- For (4) write $x = \sum a_i x_i$, $y = \sum b_j y_j(+\varepsilon)$ and $z = \sum c_k z_k(+\varepsilon)$. Then we have:

$$\begin{aligned}
& (x \bowtie y) \lhd z \\
= & \left(\sum_i a_i x_i \bowtie \sum_j b_j y_j(+\varepsilon)\right) \lhd \sum_k c_k z_k(+\varepsilon) \\
= & \left(\sum_{i,j(a_i=b_j)} a_i(x_i \mp y_j)\right) \lhd \sum_k c_k z_k(+\varepsilon) \\
= & \sum_{i,j(a_i=b_j,\forall_k a_i \neq c_k)} a_i(x_i \mp y_j) \\
= & \sum_{i(\forall_k a_i \neq c_k)} a_i x_i \bowtie \sum_j b_j y_j(+\varepsilon) \\
= & (x \lhd z) \bowtie y
\end{aligned}$$

If x has a summand ε, the proof is analogous. The other equation goes similarly.

- For (5) and (6) we use mutual induction. Write $x = \sum a_i x_i(+\varepsilon)$, $y = \sum b_j y_j(+\varepsilon)$, $z = \sum c_k z_k(+\varepsilon)$. Then for (5) we calculate:

$$\begin{aligned}
& x \bowtie (y \bowtie z) \\
= & \sum_i a_i x_i(+\varepsilon) \bowtie \left(\sum_j b_j y_j(+\varepsilon) \bowtie \sum_k c_k z_k(+\varepsilon)\right) \\
= & \sum_i a_i x_i(+\varepsilon) \bowtie \sum_{j,k(b_j=c_k)} b_j(y_j \mp z_k) \\
= & \sum_{i,j,k(a_i=b_j=c_k)} a_i(x_i \mp (y_j \mp z_k)) \\
= & \sum_{i,j,k(a_i=b_j=c_k)} a_i((x_i \mp y_j) \mp z_k) \\
= & \sum_{i,j(a_i=b_j)} a_i(x_i \mp y_j) \bowtie \sum_k c_k z_k(+\varepsilon) \\
= & (x \bowtie y) \bowtie z
\end{aligned}$$

For (6) we have

$$\begin{aligned}
& x \mp (y \mp z) \\
= & x \bowtie (y \mp z) + x \lhd (y \mp z) + (y \mp z) \lhd x \\
& x \bowtie (y \bowtie z) + x \bowtie (y \lhd z) + x \bowtie (z \lhd y) + \\
& x \lhd (y \mp z) + \\
& (y \bowtie z) \lhd x + (y \lhd z) \lhd x + (z \lhd y) \lhd x \\
= & (x \bowtie y) \bowtie z + (x \bowtie y) \lhd z + (x \lhd y) \bowtie z +
\end{aligned}$$

$$\begin{aligned}
& (x \lhd y) \lhd z + \\
& (y \lhd x) \bowtie z + (y \lhd x) \lhd z + z \lhd (x \mp y) \\
= \; & (x \mp y) \bowtie z + (x \mp y) \lhd z + z \lhd (x \mp y) \\
= \; & (x \mp y) \mp z
\end{aligned}$$ □

We have established associativity and commutativity for $\mp$. However, the delayed choice operator is not idempotent and it does not satisfy several laws of distributivity.

Proposition 3.4.6 *The following equations are NOT valid in the initial algebra.*

1. $x \mp x = x$
2. $(x + y) \mp z = (x \mp z) + (y \mp z)$
3. $(x \mp y) + z = (x + z) \mp (y + z)$
4. $(x \mp y)z = xz \mp yz$
5. $z(x \mp y) = zx \mp zy$

Proof Let a, b, c, d and e be distinct atomic actions.

1. $(ab + ac) \mp (ab + ac) = ab + a(b + c) + ac$
2. $(ab + cd) \mp ae = a(b + e) + cd$, while
   $(ab \mp ae) + (cd \mp ae) = a(b + e) + cd + ae$.
3. $(ab \mp c) + ad = ab + c + ad$, while
   $(ab + ad) \mp (c + ad) = a(b + d) + ad + c$.
4. $(\varepsilon \mp a)a = a + aa$, while
   $\varepsilon a \mp aa = a(\varepsilon + a)$.
5. $(ab + ac)(d \mp e) = ab(d + e) + ac(d + e)$, while
   $(ab + ac)d \mp (ab + ac)e = (abd + acd) \mp (abe + ace) = ab(d + e) + a(bd + ce) + a(cd + be) + ac(d + e)$. □

4. EXAMPLE

Figure 3 shows an example of the use of the delayed choice operator for combining Message Sequence Charts. The complete MSC document consists of four Message Sequence Charts. The first MSC describes the *normal operation* of a system which consists of two instances a and b. Instance a sends a *start* message to instance b. Next a test has to be performed by b, which checks if b operates correctly. This is indicated by a condition named *testing*. Such a condition can be used to denote that there are two possible continuations. The first possibility is expressed in MSC *continuation 1*. It describes the situation where b actually performs a test (indicated by the local action *test*), followed by an acknowledgement to a. The alternative continuation is described in MSC *continuation 2*. In this case, a test is performed and followed by a negative acknowledgment. These two continuations are alternatives, thus expressing that the choice between *ok* and *fail* is non-deterministic. Furthermore, MSC *cancel* expresses yet another scenario. It describes the option that a, immediately after starting b, can *cancel* operation of b. It is clearly not the intention to have a non-deterministic choice between testing and cancellation. So these alternative scenarios are combined with the delayed choice operator.

Using the techniques from [16] we calculate the semantics of these four Message Sequence Charts separately in a simplified notation. Since a Message Sequence Chart describes asynchronous communication, a distinction is made between the output and the input of a message.

$$\begin{aligned}
normaloperation &= out(start) \cdot in(start)\\
continuation1 &= test \cdot out(ok) \cdot in(ok)\\
continuation2 &= test \cdot out(fail) \cdot in(fail)\\
cancel &= out(start) \cdot\ (in(start) \cdot out(cancel) \cdot in(cancel)\\
&\qquad\qquad\qquad +out(cancel) \cdot in(start) \cdot in(cancel)\\
&\qquad\qquad\qquad)
\end{aligned}$$

Thus the intended semantics of the complete MSC document is

$$normaloperation \cdot (continuation1 + continuation2) \mp cancel$$

This is equal to

$$\begin{aligned}
&\ out(start) \cdot in(start) \cdot (test \cdot out(ok) \cdot in(ok) + test \cdot out(fail) \cdot in(fail))\\
&\mp\ out(start) \cdot\ (in(start) \cdot out(cancel) \cdot in(cancel)\\
&\qquad\qquad\qquad +out(cancel) \cdot in(start) \cdot in(cancel)\\
&\qquad\qquad\qquad)\\
&= out(start) \cdot\ (in(start) \cdot\ (test \cdot out(ok) \cdot in(ok)\\
&\qquad\qquad\qquad\qquad\qquad +test \cdot out(fail) \cdot in(fail)\\
&\qquad\qquad\qquad\qquad\qquad +out(cancel) \cdot in(cancel)\\
&\qquad\qquad\qquad\qquad\qquad)\\
&\qquad\qquad\qquad +out(cancel) \cdot in(start) \cdot in(cancel)\\
&\qquad\qquad\qquad)
\end{aligned}$$

5. CONCLUSION

We defined the semantics of the delayed choice operator, without proposing a concrete textual or graphical syntax for it in the MSC language. The reason is that the integration of compositional operators and Message Sequence Charts must be uniform for all operators proposed.

We defined operational semantics for the delayed choice operator and gave a sound and complete set of equational laws. We also proved some additional properties of this operator.

If we replace in a process all choices by delayed choices we throw away all branching time information. Thus, on processes that contain only delayed choice, bisimulation semantics

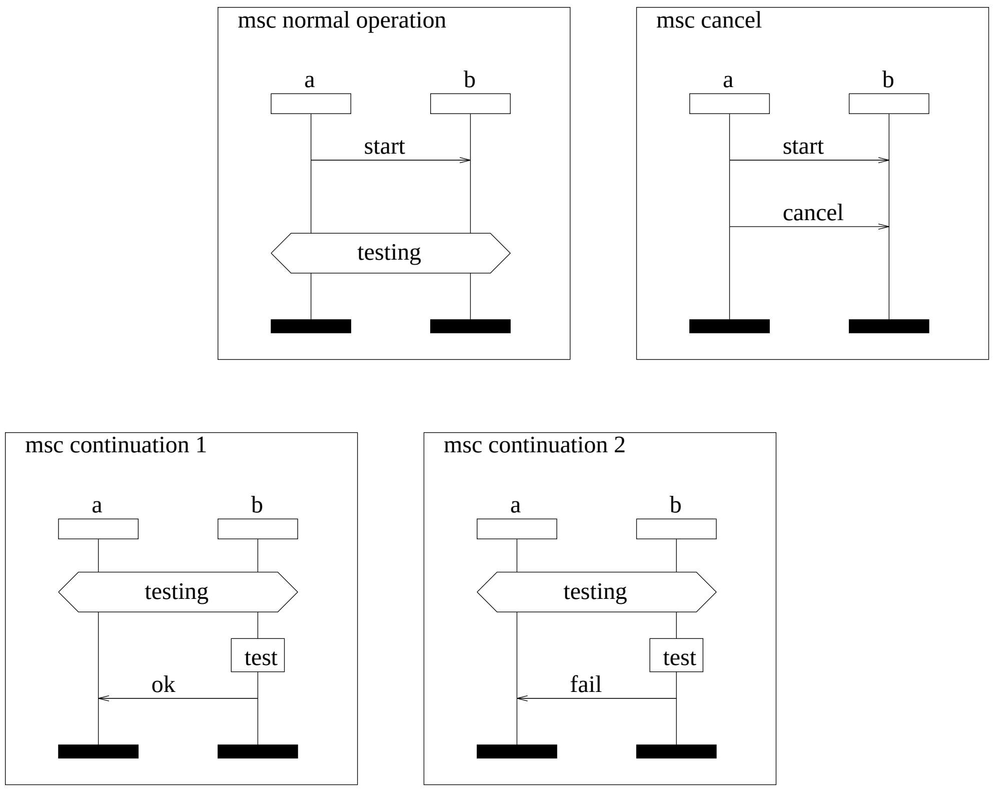


Figure 3. Example Message Sequence Charts

and trace semantics coincide [11]. We leave the formalization of this statement as future work.

The modular approach to process algebra in the style of [5, 6] makes it easy to extend the theory presented here with additional operators. Thus, it is straightforward to achieve extensions with parallel composition, synchronous communication, asynchronous communication, abstraction and other features.

Of course the use of the delayed choice operator is not restricted to the domain of Message Sequence Charts. It is applicable in all cases where we need to express trace-like alternatives in a branching time setting. In general, we prefer the introduction of an extra operator to a total reconsideration of a theory in a different semantics.

REFERENCES

1. L. Aceto, B. Bloom, and F.W. Vaandrager. Turning sos rules into equations. In *Proc. LICS92*, pages 113–124. Santa Cruz, IEEE Computer Society Press, 1992.
2. J.C.M. Baeten and J.A. Bergstra. Process algebra with partial choice. In J. Parrow, editor, *Proc. CONCUR'94*. Uppsala, LNCS, 1994. (to appear).
3. J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fund. Inf.*, IX(2):127–168, 1986.
4. J.C.M. Baeten and R.J. van Glabbeek. Merge and termination in process algebra. In

K.V. Nori, editor, *Proc. FST&TCS 7*, pages 153–172. Pune, Springer Verlag, 1987. LNCS 287.

5. J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.
6. J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *I&C*, 60(1/3):109–137, 1984.
7. B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced: preliminary report. In *Proc. 15th ACM symposium on Principles of Programming Languages*, pages 229–239. San Diego, California, 1988.
8. S.D. Brookes, C.A.R. Hoare, and W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31:560–599, 1984.
9. P. D'Argenio. τ-angelic choice for process algebra. Technical report, LIFIA, Dpto. de Informàtica, Fac. Cs. Exactas, UNLP, 1994.
10. 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.
11. J. Engelfriet. Determinacy $\rightarrow$ (observation equivalence $=$ trace equivalence. *TCS*, 36(1):21–25, 1985.
12. O. Haugen. MSC Structural concepts. Experts Meeting SG10, Turin, TD9006, ITU-TS, 1994.
13. ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC).* ITU-TS, Geneva, 1994.
14. S. Kamin and J.-J. Lévy. Two generalizations of the recursive path ordering. Unpublished manuscript, 1980.
15. J.W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume II, pages 1–116. Oxford University Press, 1992.
16. S. Mauw and M.A. Reniers. An algebraic semantics of Basic Message Sequence Charts. *The computer journal*, 37(4), 1994. (to appear).
17. S. Mauw and M.A. Reniers. An algebraic semantics of Message Sequence Charts. Experts Meeting SG10, Turin, TD9009, ITU-TS, 1994. Report CSN94/23, Eindhoven University of Technology, 1994.
18. M.W. Mislove and F.J. Oles. A simple language supporting angelic nondeterminism and parallel composition. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Semantics, 7th international conference*, pages 77–101. Springer Verlag, 1991.
19. G.D. Plotkin. An operational semantics for CSP. In *Proceedings of the Conference on the Formal Description of Programming Concepts*, volume 2, Garmisch, 1983.
20. C. Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. In J. Parrow, editor, *Proc. CONCUR'94*. Uppsala, LNCS, 1994. (to appear).
21. C. Verhoef. A general conservative extension theorem in process algebra. In *Proc. PROCOMET'94, IFIP 2 Working Conference.* San Miniato, North-Holland, 1994. (to appear), report CSN 93/38, Eindhoven University of Technology 1993.