Pomsets for Message Sequence Charts[*]

Joost-Pieter Katoen and Lennard Lambert[†]
Lehrstuhl für Informatik VII. Friedrich-Alexander University of Erlangen-Nürnberg.
Martensstrasse 3. D-91058 Erlangen. Germany.
{*katoen, ldlamber*}*@informatik.uni-erlangen.de*

Abstract

Message sequence charts (MSCs) are a standardised formalism for the specification of the system's communication behaviour that is widely used by industry. Various extensions to MSCs have recently been standardised by the ITU. This paper treats the extension of MSCs with structural operators that allow the hierarchical composition of MSCs. In particular, we propose a linear-time partial-order semantics for this extension that is based on the notion of partial-order multi-sets (pomsets, for short). We provide a compositional denotational semantics for hierarchical MSCs and show that well-known operations on pomsets correspond to the new composition operators in MSCs. The main benefit of our approach is that the semantics is conceptually rather straightforward, as opposed to the operational semantics for MSCs that is currently in the process of standardisation.

Keywords
Denotational semantics, partial-order multi-set, hierarchical MSC

1 INTRODUCTION

In the telecommunication sector, MSC (Message Sequence Chart) is a standardised formal specification language for describing an overview of the system's communication behaviour. This overview is intended to be incomplete, that is, an MSC-specification describes only sample behaviours. Typically, an MSC-specification is complemented by a more complete specification in SDL (Specification and Description Language) [3] or a similar language. MSC is employed across a wide spectrum of activities in the design of distributed systems, ranging from requirements specification, over simulation, down to test-case specification of system implementations.

MSC has been standardised in two versions, namely MSC'92 [7] and MSC'96 [8]. In MSC'92 only basic MSCs with asynchronous message exchange between parallel instances are specified. The main improvement from MSC'92 to MSC'96 is the incorporation of structural operators that allow the hierarchical composition of MSCs. Prominent examples of these operators are weak sequential composition, parallel composition, delayed choice, and constructs to specify infinite behaviour (iteration). Whereas the formal semantics of MSC'92 is based on a translation into process algebra, the semantics of MSC'96 [8] is defined in an operational way [12, 13]. The operational semantics—which is currently under consideration for standardisation—does not explicitly model the independence of subsystems (called instances). Instead, independence is modelled by the non-deterministic alternation of independent parallel activities. It is a so-called interleaving semantics. The operational semantics maps an MSC-specification onto a labelled transition system, a model that keeps track of the moments of choice, i.e. it is a branching-time model.

Due to the nature of delayed choice—an intrinsically linear-time operator—and weak sequential composition, the current operational semantics is significantly more complex than that of MSC'92. We believe that a more natural and simpler semantics is obtained when considering a *partial-order semantics* in a linear-time domain as opposed to an interleaved semantics. In this paper we propose such semantics for the hierarchical composition operators for MSC'96. This denotational semantics is based on partial-order multi-sets (pomsets, for short) [14, 5], a well-established model in the class of linear-time non-interleaving models. The semantics is compositional, which means that the interpretation of a composed MSC is defined in terms of the interpretation of its constituent MSCs. In particular, we show that the new MSC composition operators correspond to 'standard' operations on pomsets.

[*]Reprinted with permission from *Formale Beschreibungstechniken für verteilte Systeme, 8. GI/ITG-Fachgespräch*, Cottbus, 1998.

[†]Supported by the *Deutsche Forschungsgemeinschaft* under grant SFB 182, B3

Structure of the paper. Section 2 introduces the notion of pomsets. Section 3 presents a partial-order semantics for basic MSCs, that is, MSCs that do not contain composition operators. This is, in fact, an extension of the partial-order semantics by Alur, Holzmann and Peled [1] with co-regions in a pomset setting. Section 4 extends this pomset semantics with the operators that allow hierarchical composition of MSCs. Finally, Section 5 discusses related work and summarises our main conclusions.

2 PARTIAL-ORDER MULTI-SETS

Partial-order multi-sets (pomsets), mainly developed by Pratt [14] and Gischer [5], play a prominent role in the branch of non-interleaving models. A pomset is basically a set of labelled events enriched with a partial-order relation that represents causality. Events model occurrences of actions, like sending a message or setting a timer.[1] Before defining pomsets, we consider the auxiliary notion of lposet:

Definition 1. Let **L** be a set of labels. A labelled partially ordered set (*lposet*) is a triple $(E, \leqslant, l)$ with E, a set of events, $\leqslant \subseteq E \times E$, a reflexive, anti-symmetric, and transitive order on E, and $l : E \longrightarrow \mathbf{L}$ a labelling function. □

$\leqslant$ is called a partial order that represents causality. For $e \neq e'$, $e \leqslant e'$ denotes that if e and e' both occur then e' is caused by e. The empty lposet $(\varnothing, \varnothing, \varnothing)$ is denoted by ε. Non-empty lposets are often graphically denoted: e.g., $(\{\, e, e' \,\}, \leqslant, \{\, (e, a), (e', b) \,\})$ is denoted by $\boxed{\begin{matrix} e \\ e' \end{matrix}}$ if e and e' are unrelated under $\leqslant$, and by $\boxed{e \longrightarrow e'}$ if $e \leqslant e'$. Note that l is not required to be injective. If for e and e' we have $l(e) = l(e')$ it means that e and e' model different occurrences of the same action. Two lposets are isomorphic if they are equal up to renaming of events.

Definition 2. $(E, \leqslant, l)$ and $(E', \leqslant', l')$ are *isomorphic* iff there exists a bijection $\phi : E \longrightarrow E'$ such that $e \leqslant \hat{e}$ iff $\phi(e) \leqslant' \phi(\hat{e})$ and $l = l' \circ \phi$, for all $e, \hat{e} \in E$. □

Definition 3. (**Partial-order multi-set [14]**)
A partial-order multi-set (*pomset*) is an isomorphism class of lposets. □

The isomorphism class of $(E, \leqslant, l)$ is denoted by $[E, \leqslant, l]$. The representation of pomsets is similar to that of lposets, except that events are replaced by their labels. For instance, $\boxed{\begin{matrix} a \\ b \end{matrix}}$ rather than $\boxed{\begin{matrix} e \\ e' \end{matrix}}$. The idea of our semantics for MSCs will be to map an MSC onto a set of pomsets that is closed under the following relation.

Definition 4. $[E, \leqslant, l]$ is a *prefix* of $[E', \leqslant', l']$ iff $E \subseteq E'$, $\leqslant = \leqslant' \cap (E' \times E)$ and $l = l' \restriction E$. □

Here, $\restriction$ denotes restriction. The second constraint says that no event in $E' \setminus E$ may precede under $\leqslant'$ an event in E. For instance, $\boxed{a}$ is a prefix of $\boxed{\begin{matrix} a \\ b \end{matrix}}$ and $\boxed{a \longrightarrow b}$, whereas $\boxed{b}$ is a prefix of the first, but not of the latter. The empty pomset $[\varepsilon]$ is a prefix of each pomset. Evidently, the relation 'is a prefix of' is a partial order on pomsets. In the next sections an MSC will be mapped onto a set of pomsets that is downwards closed under 'is a prefix of'. Rather than writing this entire set we concentrate on maximal pomsets under this notion. The complete semantics is thus the set of all prefixes of this maximal pomset. The maximal pomset can be considered as the maximal 'run' of the system, whereas its prefixes are partial 'runs'. A pomset is similar to a trace, a sequence of events, with the difference that a pomset is partially ordered whereas a trace is totally ordered.

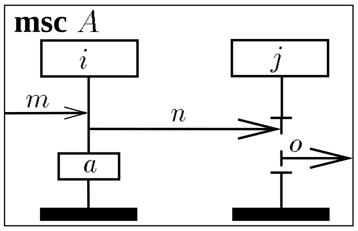
3 BASIC MESSAGE SEQUENCE CHARTS

An MSC specification is an MSC document which consists of many MSCs. Basic MSC (BMSC) describe the functional interaction between parallel working instances which exchange messages asynchronously. They do not contain composition operators.[2]

Syntax of basic MSC. MSC has a textual and a graphical representation. Usually the intuitive graphical representation is used, but the definition of our formal semantics will base on an abbreviated subset of the textual syntax as given (in Backus-Naur-Form) in Table 1. This syntax is instance-oriented, i.e. an MSC is given by describing

[1] Events in pomsets should not be confused with MSC-events, since events (in our sense) can only occur at most once, while MSC-events may occur multiple times, e.g. in case of iteration.

[2] In literature, various notions of 'basic' MSCs exist; these should not be confused with our notion that includes co-regions.



```
msc A;
    inst i;                      inst j;
        in m from env;               co  in n from i;
        out n to j;                      out o to env;
        local a;                     endco;
    endinst;                     endinst;
                             endmsc;
```

Figure 1: Example of a basic MSC

all its instances. Alternatively, an event-oriented syntax exists [8]. All non-terminal symbols with ⟨*name*⟩ (e.g. ⟨*msc name*⟩) are normal alpha-numerical names.

⟨*Msc*⟩	::=	**msc** ⟨*msc name*⟩ **;** ⟨*Body*⟩ **endmsc;**
⟨*Body*⟩	::=	⟨ ⟩ | **inst** ⟨*inst name*⟩ **;** ⟨*Ev-s*⟩ **endinst;** ⟨*Body*⟩
⟨*Ev-s*⟩	::=	⟨ ⟩ | ⟨*Ev*⟩ **;** ⟨*Ev-s*⟩ | **co** ⟨*CoR*⟩ **endco;** ⟨*Ev-s*⟩
⟨*CoR*⟩	::=	⟨ ⟩ | ⟨*Ev*⟩ **;** ⟨*CoR*⟩
⟨*Ev*⟩	::=	**in** ⟨*msg name*⟩ **from** {⟨*inst name*⟩ | **env**} |
		out ⟨*msg name*⟩ **to** {⟨*inst name*⟩ | **env**} |
		local ⟨*action name*⟩

Table 1: Syntax of basic MSC

Example 5. Figure 1 presents the graphical and textual representation of BMSC A. Instances i and j communicate with the environment that is represented by the outer frame in the graphical and by **env** in the textual representation. The events on a solid instance axis are totally ordered. Thus, instance i has to receive message m from the environment before it can send message n to instance j and before the local action a executes. The dotted line on instance j specifies a co-region. It describes the absence of ordering along the instance axis, i.e. the events of sending a message o to the environment and the reception of the message m are unordered. Of course, a message must first be sent before it can be received. So, instance i must send message m before instance j can receive it. *(End of example.)*

Operations on pomsets. The idea is to map a BMSC onto a (maximal) pomset. To start with, let the set **L** equal **A** × **I**, where **A** is the set of actions[3] and **I** the set of instances. Send, receive, and local actions belong to **A**. We denote by $!(m,j)@i$ the sending of message m from instance i to j, $?(m,i)@j$ for the receipt of this message, and $a@i$ for local action a at instance i. If no confusion arises we write $!m$, $?m$ and a as shorthand. We let i, j and k range over **I**. The distinguishing element $\mathbf{env} \in \mathbf{I}$ denotes the environment.

Let $p = [E_p, \leqslant_p, l_p]$ and $q = [E_q, \leqslant_q, l_q]$ be two pomsets such that $E_p \cap E_q = \varnothing$. (Since pomsets are isomorphism classes, this can be required w.l.o.g..)

Definition 6. (**Concatenation [14]**)
$p \cdot q \triangleq [E_p \cup E_q, \leqslant_p \ \cup \ \leqslant_q \ \cup (E_p \times E_q), l_p \cup l_q]$. □

In $p{\cdot}q$, every event of p is forced to precede every event of q. It is straightforward to check that $\leqslant_p \cup \leqslant_q \cup (E_p \times E_q)$ is a partial order, so $p \cdot q$ is indeed a pomset. Concatenation is intended to correspond to strong sequencing in MSCs. As an example of concatenation, consider:

$$\begin{tikzcd}[row sep=small]
 & b@j & & d@k & & & & b@j \arrow[r] \arrow[dr] & d@k & \\
a@i \arrow[ur] \arrow[r] & c@k & \cdot & e@j \arrow[r] & f@i & = & a@i \arrow[ur] \arrow[r] & c@k \arrow[r] \arrow[ur] & e@j \arrow[r] & f@i
\end{tikzcd}$$

[3] MSC-events together with local actions are called actions.

Joining of pomsets is used to compose pomsets that model behaviour at (possibly several) different instances. It is basically an element-wise union of pomsets, except that send and corresponding receive events are causally related. To let joining be defined properly we consider *consistent* pomsets, a notion which we explain below.

Definition 7. (**Joining**)
For p and q consistent pomsets: $p \bowtie q \triangleq [E_p \cup E_q, (\leqslant_p \cup \leqslant_q \cup \leqslant_q^p \cup \leqslant_p^q)^+, l_p \cup l_q]$ where

$$\leqslant_q^p = \{ (e, e') \in E_p \times E_q \mid l_p(e) = !(m, i)@j \ \wedge \ l_q(e') = ?(m, j)@i \} .$$

□

For arbitrary pomsets p and q it is not guaranteed that $p \bowtie q$ is a pomset. For instance, let $?m \leqslant_p !n$ and $?n \leqslant_q !m$ and suppose m and n are messages sent to instances in q and p, respectively. Then, we would obtain $?m \leqslant_p !n \leqslant ?n \leqslant_q !m$ and $!m \leqslant' ?m$ which violates the anti-symmetry constraint of Definition 1. Therefore we require p and q to be consistent. Pomsets p and q are consistent if by the addition of new dependencies between send and corresponding receive events (as defined for $\bowtie$) no cycles are introduced. As an example of joining consider:

$$\begin{tikzcd} !(m,i)@j \arrow[r] & a@j \\ ?(n,k)@j \end{tikzcd} \bowtie \begin{tikzcd} ?(m,j)@i \\ b@k \arrow[r] & !(n,j)@k \end{tikzcd} = \begin{tikzcd} & a@j \\ !(m,i)@j \arrow[ur] \arrow[r] & ?(m,j)@i \\ b@k \arrow[r] & !(n,j)@k \arrow[r] & ?(n,k)@j \end{tikzcd}$$

Semantics of basic MSC. The function $\mathcal{M}_{msc}[\![\ ]\!]$ assigns to a basic MSC (a singleton set consisting of) a pomset:

$$\mathcal{M}_{msc}[\![\ \textbf{msc}\ \textit{msc name}\ \textbf{;}\ B\ \textbf{endmsc}\ ]\!] \ \triangleq\ \{ \mathcal{M}_{bmsc}[\![\ B\ ]\!] \} .$$

(When we consider the hierarchical composition operators the use of a set will become clear.) The function $\mathcal{M}_{bmsc}[\![\ ]\!]$ assigns to an msc-body a pomset in the following way.

$$\begin{aligned} \mathcal{M}_{bmsc}[\![\ \langle\ \rangle\ ]\!] &\ \triangleq\ [\varepsilon] \\ \mathcal{M}_{bmsc}[\![\ \textbf{inst}\ i\ ;\ S\ \textbf{endinst}\ ;\ B\ ]\!] &\ \triangleq\ \mathcal{M}_{inst}[\![\ S\ ]\!](i) \bowtie \mathcal{M}_{bmsc}[\![\ B\ ]\!] \end{aligned} \qquad (1)$$

Function $\mathcal{M}_{inst}[\![\ S\ ]\!](i)$ assigns a meaning to the instance body S of instance i. This function is defined by induction on the structure of the syntax of S. Here, we use the singleton pomset $[e_{a@i}] \triangleq [\{ e \}, \{ (e, e) \}, \{ (e, a@i) \}]$ for $a \in \mathbf{A}$. Let

$$\begin{aligned} \mathcal{M}_{inst}[\![\ \langle\ \rangle\ ]\!](i) &\ \triangleq\ [\varepsilon] \\ \mathcal{M}_{inst}[\![\ a\ ;\ S\ ]\!](i) &\ \triangleq\ \mathcal{M}_{inst}[\![\ a\ ]\!](i) \cdot \mathcal{M}_{inst}[\![\ S\ ]\!](i) \\ \mathcal{M}_{inst}[\![\ \textbf{in}\ m\ \textbf{from}\ j\ ]\!](i) &\ \triangleq\ [e_{?(m,j)@i}] \\ \mathcal{M}_{inst}[\![\ \textbf{out}\ m\ \textbf{to}\ j\ ]\!](i) &\ \triangleq\ [e_{!(m,j)@i}] \\ \mathcal{M}_{inst}[\![\ \textbf{local}\ b\ ]\!](i) &\ \triangleq\ [e_{b@i}] \\ \mathcal{M}_{inst}[\![\ \textbf{co}\ \langle\ \rangle\ \textbf{endco}\ ]\!](i) &\ \triangleq\ [\varepsilon] \\ \mathcal{M}_{inst}[\![\ \textbf{co}\ a\ ;\ C\ \textbf{endco}\ ]\!](i) &\ \triangleq\ \mathcal{M}_{inst}[\![\ a\ ]\!](i) \bowtie \mathcal{M}_{inst}[\![\ \textbf{co}\ C\ \textbf{endco}\ ]\!](i) \end{aligned} \qquad (2)$$

The reader might have expected, instead of (2):

$$\mathcal{M}_{inst}[\![\ \textbf{co}\ a\ ;\ C\ \textbf{endco}\ ]\!](i) \triangleq \mathcal{M}_{inst}[\![\ a\ ]\!](i) \,||\, \mathcal{M}_{inst}[\![\ \textbf{co}\ C\ \textbf{endco}\ ]\!](i).$$

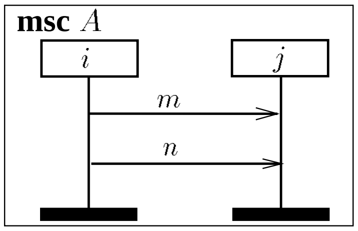
This would correspond with the intuition that a co-region specifies events that are causally independent. (Here, $||$ denotes the element-wise union of pomsets.) However, in case

$$\textbf{co}\ \ldots\ ;\ \textbf{in}\ m\ \textbf{from}\ i\ ;\ \ldots\ ;\ \textbf{out}\ m\ \textbf{to}\ i\ ;\ \ldots\ \textbf{endco}$$

this would lead to two unrelated send and receive events (of m) which contradicts that each message can only be received if it has sent before. Therefore, we use $\bowtie$ rather than $||$.

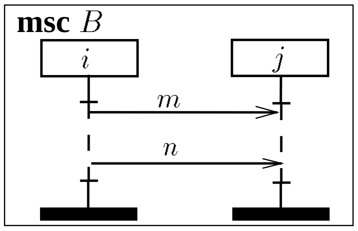
Theorem 8. For BMSC A, $\mathcal{M}_{bmsc}[\![\ A\ ]\!]$ is a pomset.

Example 9. Let us illustrate the above formal definitions by some examples. The first example shows a simple application of the join operator. It concerns the transmission of two messages between the same instances:



$$\begin{tikzcd}
 & ?(m,i)@j \arrow[dr] & \\
!(m,j)@i \arrow[ur] \arrow[dr] & & ?(n,i)@j \\
 & !(n,j)@i \arrow[ur] &
\end{tikzcd}$$

If we replace the vertical solid lines by dashed lines, indicating co-regions, at both instances, then the orderings at these instances are eliminated and the only resulting dependencies are the (always present) relations between transmissions and receptions:



$$\boxed{\begin{array}{l} !(m,j)@i \rightarrow ?(m,i)@j \\ !(n,j)@i \rightarrow ?(n,i)@j \end{array}}$$

(End of example.)

Consistency with MSC'92 semantics. The relationship between our pomset semantics for BMSC and the standard process algebra semantics for MSC'92, as published in [11], is as follows. Let the process algebra term $\mathcal{P}[\![A]\!]$ denote the MSC'92 semantics of BMSC A. As an auxiliary notion we define the traces described by a pomset:

Definition 10. Let $p = [E_p, \leqslant_p, l_p]$ be a pomset with $\{ e_1, \ldots, e_n \} \subseteq E_p$. The sequence $e_1 \ldots e_n$ is a *trace* of p iff $e_i \leqslant_p e_j \;\Rightarrow\; i \leqslant j$ for all $0 < i, j \leqslant n$. $\square$

Stated in words, a sequence of distinct events in E_p is a trace if the ordering in the sequence contains the ordering $\leqslant_p$. For instance, ab is a trace of pomsets $\boxed{\begin{array}{c} a \\ b \end{array}}$ and $\boxed{a \rightarrow b}$, whereas ba is a trace of the first, but not of the latter.

Theorem 11. For BMSC A, $\mathcal{M}_{msc}[\![A]\!]$ and $\mathcal{P}[\![A]\!]$ are trace-equivalent.

This result says that an interleaving 'view' of our partial-order semantics is equal to the standard interleaving semantics for the fragment BMSC.

Extension with other MSC'96 constructs. For the sake of simplicity, we only consider send, receive and local actions. Our semantics can be extended with the use of timer actions (like timer set, reset, and timeout), dynamic creation of instances, halting of instances (stop), general orderings and lost and found messages without great difficulty. For instance, the setting of a timer (at instance i) can be considered as the transmission of a self-message (by i to i) and the timeout (or reset) as the receipt of a self-message (from i). The pomset semantics constructs a causal relation between the setting and expiration of the timer, as required by recommendation Z.120 [8]. (What we have not considered are multiple set (reset) events after each other without the corresponding reset or timeout (set) event in between.) General orderings can be considered as the sending and receipt of empty messages (no content), dynamic creation as specific messages, stop is treated as a local action, and lost and found messages as messages like $!(m,-)@i$ and $?(m,-)@i$ where $-$ denotes absence of originator and recipient, respectively. The use of conditions is not considered, since their role seems to be superfluous by the introduction of composition operators. The decomposition of instances by substructures is a refinement of instances to a certain level of abstraction. We consider instance decomposition as a syntactical transformation. Under the assumption that each instance is transformed to exactly one level of abstraction our semantics can be applied.

4 HIERARCHICAL MESSAGE SEQUENCE CHARTS

To model larger systems MSCs can be composed by hierarchical composition operators. The resulting language is called hierarchical MSC (HMSC), referred to as 'high-level' MSC in [8].

Syntax of hierarchical MSC. Table 2 presents the extensions to the syntax of Table 1 that include the composition operators for HMSC. Here, the non-terminal $\langle k \rangle$ stands for any natural number (including zero) or ∞. **empty** denotes an MSC without any action. The binding order of the operators is in descending order: **loop**, **seq**, **par**, **alt**.

$\langle$*Msc*$\rangle$	::=	**msc** $\langle$*msc name*$\rangle$ **;** $\{\langle$*Body*$\rangle$ | **expr** $\langle$*Ex*$\rangle\}$ **endmsc;**
$\langle$*Ex*$\rangle$	::=	**empty** | $\langle$*msc name*$\rangle$ | $\langle$*Ex*$\rangle$ **seq** $\langle$*Ex*$\rangle$ | $\langle$*Ex*$\rangle$ **alt** $\langle$*Ex*$\rangle$ |
		$\langle$*Ex*$\rangle$ **par** $\langle$*Ex*$\rangle$ | $\mathbf{loop}_{\langle k\rangle,\langle k\rangle}$**(**$\langle$*Ex*$\rangle$**)**

Table 2: Extending the syntax of basic MSC with composition operators

The intuitive interpretation of the composition operators is as follows. The **seq** operator denotes weak sequential composition. This means, that an instance may start with its first action, only if it has completed all its actions in the predecessor MSC. Consider, for instance, B **seq** A, where MSCs A and B are depicted in Figure 2. Here, instance j can receive message n and send message o only after it has sent message p, but it need not wait for instance k to receive message p.

The **alt** operator denotes delayed choice composition. This means that only one alternative is chosen. For instance, B **alt (**B **seq** A**)** specifies that the instances j and k communicate only via message p or that the instances i, j, k communicate. "In the case where alternative MSCs have a common preamble the choice of which MSC will be executed is performed after the execution of the common preamble"[8]. Due to this delayed interpretation of the choice, B **alt (**B **seq** A**)** and B **seq (empty alt** A**)** have the same interpretation.

The **par** operator denotes parallel composition. All actions from the parallel composed MSCs will occur, but only the precedences of the actions defined by the MSCs must be fulfilled. This implies in B **par** B that two messages p are sent from instance j to k. Since only the receive actions must follow the send actions it is possible that the second message p can be received first. Thus, messages can overtake each other, but need not.

Finally, the **loop** operator denotes iterative composition. The iteration can be unfolded by applying the weak sequential composition. The lower and upper bound of the iteration are parameters for the loop operator. The specified system can choose an arbitrary number of iterations between the lower and upper bound. In $\mathbf{loop}_{0,\infty}(B)$ either no message is sent or an arbitrary (finite or infinite) number of messages p are sent without overtaking.

Operations on pomsets. The basic idea to provide a semantics to hierarchical MSCs (HMSCs) is to use *sets of* pomsets. We use sets, since with delayed choice different alternatives can be described, and thus different system runs can appear. Again, we focus on maximal pomsets and leave their prefixes implicit. In fact, the semantics of an HMSC is a set of a set of pomsets (downwards closed under prefixing). Let for pomset p the set E_p^i denote the set of events in p that occur at instance i.

Definition 12. (**Concurrence and local concatenation [14]**)
Let $p = [E_p, \leqslant_p, l_p]$ and $q = [E_q, \leqslant_q, l_q]$ be two pomsets such that $E_p \cap E_q = \varnothing$.

$$p \,||\, q \quad \triangleq \quad [E_p \cup E_q, \leqslant_p \cup \leqslant_q, l_p \cup l_q]$$

$$p \circ q \quad \triangleq \quad \left[E_p \cup E_q, \left(\leqslant_p \cup \leqslant_q \bigcup_i (E_p^i \times E_q^i)\right)^+, l_p \cup l_q\right]$$

□

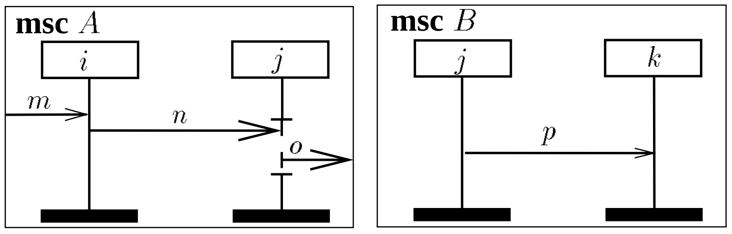


Figure 2: Two basic MSCs

It is easy to verify that $\leqslant_p \cup \leqslant_q$ is indeed a partial order, so $p \,||\, q$ is indeed a pomset. In local concatenation events in p should precede those events in q that appear at the same instance (i.e., these events are 'co-located' [14]). Notice that for $p \circ q$ we need to take the transitive closure to guarantee that the resulting relation is indeed a partial order. Concurrence is intended to correspond to **par**, whereas local concatenation is linked to weak sequencing **seq**. As an example of local concatenation, consider:

$$\boxed{\begin{tikzcd} & b@j \\ a@i \arrow[ur] \arrow[r] & c@k \end{tikzcd}} \circ \boxed{\begin{tikzcd} d@k \\ e@j \arrow[r] & f@i \end{tikzcd}} = \boxed{\begin{tikzcd} & b@j \arrow[dr] & d@k \\ a@i \arrow[ur] \arrow[r] & c@k \arrow[ur] & e@j \arrow[r] & f@i \end{tikzcd}}$$

Evidently, if p and q only contain events that appear at one and the same instance, local concatenation boils down to (global) concatenation, as defined before. This means that we could safely replace the use of $\cdot$ in the semantics of BMSC by $\circ$.

Semantics of hierarchical MSC. Since **empty** denotes the MSC without any actions, it is mapped onto the singleton set containing $[\varepsilon]$, the empty pomset. For A **par** B and A **seq** B we take the component-wise combinations according to $||$ and $\circ$, respectively, of the pomsets of A and B. The pomsets of A **alt** B are simply the pomsets of A and those of B.

Example 13. The semantics of **alt** can best be understood by means of a simple example. Consider the BMSCs A and B depicted in Figure 3. According to our idea, the pomsets of A **alt** B are $\boxed{!m \rightarrow ?m \rightarrow !n \rightarrow ?n}$ and $\boxed{!m \rightarrow ?m \rightarrow b}$,

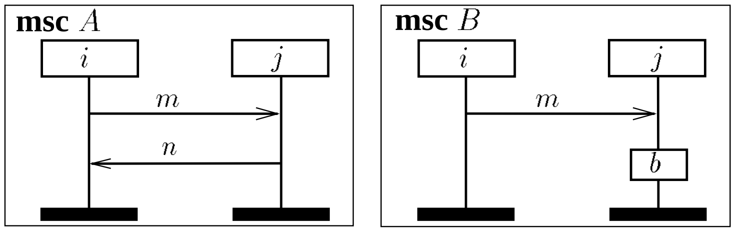


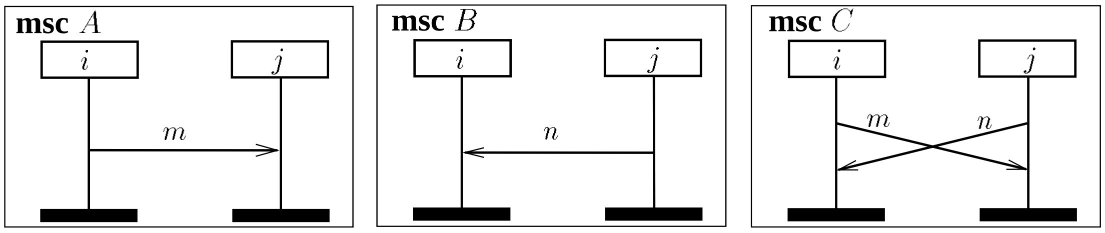
Figure 3: Two basic MSCs with a common preamble

respectively. Observe that these two maximal pomsets have a few prefixes in common. The maximal pomset that both have as a prefix is $\boxed{!m \rightarrow ?m}$. Since this is a common prefix, it means that the choice between the two possible behaviours of A **alt** B is made after the reception of m (by instance j). That is, the choice is indeed delayed until after the common preamble. *(End of example.)*

In summary, for HMSCs A and B we have:

$$\begin{array}{rcl}
\mathcal{M}_{\mathit{hmsc}}[\![\, \textbf{empty} \,]\!] & \triangleq & \{\, [\varepsilon] \,\} \\
\mathcal{M}_{\mathit{hmsc}}[\![\, A \;\textbf{par}\; B \,]\!] & \triangleq & \{\, p \,||\, q \mid p \in \mathcal{M}_{\mathit{hmsc}}[\![\, A \,]\!], q \in \mathcal{M}_{\mathit{hmsc}}[\![\, B \,]\!] \,\} \\
\mathcal{M}_{\mathit{hmsc}}[\![\, A \;\textbf{seq}\; B \,]\!] & \triangleq & \{\, p \circ q \mid p \in \mathcal{M}_{\mathit{hmsc}}[\![\, A \,]\!], q \in \mathcal{M}_{\mathit{hmsc}}[\![\, B \,]\!] \,\} \\
\mathcal{M}_{\mathit{hmsc}}[\![\, A \;\textbf{alt}\; B \,]\!] & \triangleq & \mathcal{M}_{\mathit{hmsc}}[\![\, A \,]\!] \cup \mathcal{M}_{\mathit{hmsc}}[\![\, B \,]\!].
\end{array}$$

Example 14. As an example consider the three basic MSCs depicted below.



By using the constructions for basic MSCs (see previous section), we obtain that

$$\mathcal{M}_{\mathit{hmsc}}[\![\, A \,]\!] = \boxed{!m \rightarrow ?m} \;\; \mathcal{M}_{\mathit{hmsc}}[\![\, B \,]\!] = \boxed{!n \rightarrow ?n} \;\; \mathcal{M}_{\mathit{hmsc}}[\![\, C \,]\!] = \boxed{\begin{tikzcd} !m \arrow[r] \arrow[dr] & ?m \\ !n \arrow[r] \arrow[ur] & ?n \end{tikzcd}}.$$

Then, using the operators defined above, the reader is invited to check that:

$$\begin{array}{rcl}
\mathcal{M}_{\mathit{hmsc}}[\![\, A \ \mathbf{par}\ B \,]\!] & = & \boxed{\begin{array}{l} !m \rightarrow ?m \\ !n \rightarrow ?n \end{array}} \\
\mathcal{M}_{\mathit{hmsc}}[\![\, A \ \mathbf{seq}\ B \,]\!] & = & \boxed{!m \rightarrow ?m \rightarrow !n \rightarrow ?n} \\
\mathcal{M}_{\mathit{hmsc}}[\![\, A \ \mathbf{alt}\ B \ \mathbf{alt}\ C \,]\!] & = & \left\{ \boxed{!m \rightarrow ?m},\ \boxed{!n \rightarrow ?n},\ \boxed{\begin{tikzcd} !m \arrow[r] \arrow[dr] & ?m \\ !n \arrow[r] \arrow[ur] & ?n \end{tikzcd}} \right\}
\end{array}$$

The reader may check that $\mathcal{M}_{\mathit{hmsc}}[\![\, A \ \mathbf{seq}\ (B \ \mathbf{alt}\ C) \,]\!] = \mathcal{M}_{\mathit{hmsc}}[\![\, (A \ \mathbf{seq}\ B) \ \mathbf{alt}\ (A \ \mathbf{seq}\ C) \,]\!]$.

(End of example.)

Loops. Finite loops can be rewritten into finite terms consisting of alternative and weak sequential composition. This is defined as follows.

Definition 15. (**Rewriting of finite loop**)
For HMSC A and n, k natural numbers we have

$$\mathbf{loop}_{n,k}(A) \triangleq \begin{cases} \mathbf{empty} & \text{if } n > k \\ A^n & \text{if } n = k \\ A^n \ \mathbf{alt}\, \mathbf{loop}_{n+1,k}(A) & \text{if } n < k \end{cases}$$

where A^n is defined by $A^0 \triangleq$ **empty** and $A^{n+1} \triangleq A \ \mathbf{seq}\ A^n$ for $n \geqslant 0$. □

$\mathbf{loop}_{n,k}(A)$ for $n < k$ can choose to execute A a finite number of times in between n and k. Note that the choice is delayed and that sequencing is weak. For instance, if

$$\mathcal{M}_{\mathit{hmsc}}[\![\, A \,]\!] = \left\{ \boxed{\begin{array}{l} a@i \\ b@i \end{array}} \right\} \text{ then } \mathcal{M}_{\mathit{hmsc}}[\![\, \mathbf{loop}_{1,3}(A) \,]\!] = \left\{ \boxed{\begin{array}{l} a@i \\ b@i \end{array}}, \boxed{\begin{tikzcd} a@i \arrow[r] \arrow[dr] & a@i \\ b@i \arrow[r] \arrow[ur] & b@i \end{tikzcd}}, \boxed{\begin{tikzcd} a@i \arrow[r] \arrow[dr] & a@i \arrow[r] \arrow[dr] & a@i \\ b@i \arrow[r] \arrow[ur] & b@i \arrow[r] \arrow[ur] & b@i \end{tikzcd}} \right\}$$

Inspired by [14] we define the operator $\mathbf{loop}_{*}(A)$ that is able to execute A a finite, but arbitrary many (including zero) times. This auxiliary operator is defined by

$$\mathcal{M}_{\mathit{hmsc}}[\![\, \mathbf{loop}_{*}(A) \,]\!] \triangleq \bigcup_{n \geqslant 0} \mathcal{M}_{\mathit{hmsc}}[\![\, A^n \,]\!].$$

It remains to consider infinite loops. For $\mathcal{P}$ a set of pomsets let $\mathcal{P}^{\omega}$ denote the set of infinite sequences of elements in $\mathcal{P}$, where sequencing denotes weak sequential composition. For instance, if

$$\mathcal{P} = \left\{ \boxed{\begin{array}{l} a@j \\ b@i \rightarrow b@i \rightarrow b@i \rightarrow \ldots \end{array}} \right\} \text{ then } \mathcal{P}^{\omega} = \left\{ \boxed{\begin{array}{l} a@j \rightarrow a@j \rightarrow a@j \rightarrow \ldots \\ b@i \rightarrow b@i \rightarrow b@i \rightarrow \ldots \end{array}} \right\}$$

Note that in $\mathcal{P}^{\omega}$ it is possible to execute infinitely many local actions a (at instance j) without ever executing action b (at instance i). For HMSC A we define

$$\mathcal{M}_{\mathit{hmsc}}[\![\, \mathbf{loop}_{\infty,\infty}(A) \,]\!] \triangleq (\mathcal{M}_{\mathit{hmsc}}[\![\, A \,]\!])^{\omega}$$

That is, $\mathbf{loop}_{\infty,\infty}(A)$ executes A infinitely many times. The other variants of **loop** can be rewritten in the following way.

Definition 16. (**Rewriting of infinite loop**)
For HMSC A and natural number n

$$\mathbf{loop}_{n,\infty}(A) \triangleq \begin{cases} \mathbf{loop}_{*}(A) \ \mathbf{alt}\ \mathbf{loop}_{\infty,\infty}(A) & \text{if } n = 0 \\ A^n \ \mathbf{seq}\ \mathbf{loop}_{0,\infty}(A) & \text{if } n > 0 \end{cases}$$

□

Extensions with other MSC'96 constructs. The operators option **opt** and exception **exc** have not been considered since they can be rewritten with the given MSC operators. Substitutions on MSC names, messages and instances can be applied on MSCs used in expressions. Like instance decomposition we consider this as a transformation on syntax that has to take place before our semantics can be applied. Till now we have not considered the alternative representation of HMSC in an automaton like notation where states are MSC expressions and state transitions represent weak sequential composition. Our conjecture is that all HMSC in the alternative representation can be transformed in a MSC expression without loss of semantical information. Embedded expressions, like inline expressions, can be integrated without difficulties when special consistency rules apply between the embedded expression and the outer MSC. These rules include unambiguous communication between the embedded expression and the outer MSC and validity[4] of all possible alternatives in the embedded MSC. Closely connected to embedded expressions are gates which are not considered by our approach since they do not influence the dynamic behaviour.

5 CONCLUDING REMARKS

Related work. Different models for concurrency have been proposed as semantical domain for MSCs, such as Petri nets [6], Büchi automata [10], partial orders [1] and process algebra [11]. Only the recent operational semantics of Mauw and Reniers [12] covers all composition operators. We have shown that for BMSCs, uncomposed MSCs, our semantics is just a partial-order view of the standard MSC'92 denotational process algebra semantics [11]. Our partial-order semantics can be considered as an extension of the partial-order semantics [1] with co-regions and composition operators. Currently, the semantics for MSC'96 is under development [12, 13]. [13] consider HMSC without loops. We conjecture that for this class of HMSCs our semantics is trace-equivalent to the operational semantics of [13].

Main conclusions. This paper presented a pomset semantics for a major fragment of the MSC'96 [8] language. We focussed on basic MSCs including co-regions, and showed how the hierarchical composition operators can be modelled. It is interesting to observe that for finite behaviours we only needed to introduce one new operator on pomsets (joining). All other operators, in particular the composition operators, are 'standard' for pomsets [14]. To our opinion, this indicates that the presented fragment of MSCs are, in fact, graphical representations of pomsets.

Further work. The denotational pomset semantics induces an equivalence on MSCs in the following simple sense: two HMSCs are pomset-equivalent (say) iff they denote the same pomset. This equivalence notion could be axiomatised. Some example axioms are associativity of **par**, **seq** and **alt**, the fact that **empty** is a neutral element for these operators, and, more interestingly $A\,\textbf{par}\,(B\,\textbf{alt}\,C) \;=\; (A\,\textbf{par}\,B)\,\textbf{alt}\,(A\,\textbf{par}\,C)$, $(A\,\textbf{alt}\,B)\,\textbf{seq}\,C \;=\; (A\,\textbf{seq}\,C)\,\textbf{alt}\,(B\,\textbf{seq}\,C)$ and $\textbf{loop}_{\infty,\infty}(A)\,\textbf{seq}\,\textbf{loop}_{\infty,\infty}(A) = \textbf{loop}_{\infty,\infty}(A)$. A complete axiomatisation for the language with loops is infeasible [5], but it would be interesting to consider such axiomatisation for HMSCs without iteration, for instance in the line of work of Janssen [9]. Other topics of interest are the extension of MSC with quantitative information such as real-time constructs and probabilities.

Acknowledgement. The authors like to thank Sjouke Mauw for fruitful discussions and Michel Reniers for his responses on our questions on the MSC-semantics.

6 REFERENCES

[1] R. Alur, G. J. Holzmann, and D. Peled. An analyzer for Message Sequence Charts. *Software - Concepts and Tools*, 17(2):70–77, 1996.

[2] A. Cavalli and A. Sarma, editors. *SDL'97: Time for Testing - SDL, MSC and Trends*, Evry, France, September 1997. Eighth SDL Forum, Elsevier Science Publishers B.V.

[3] CCITT. *Recommendation Z.100: Specification and Description Language SDL, Blue Book*. ITU, Geneva, 1992.

[4] O. Færgemand and A. Sarma, editors. *SDL'93: Using Objects*, Darmstadt, Germany, 1993. Sixth SDL Forum, North-Holland.

[5] J. Gischer. The equational theory of pomsets. *Theoretical Computer Science*, 61:199–224, 1989.

[6] J. Grabowski, P. Graubmann, and E. Rudolph. Towards a Petri net based semantics definition for Message Sequence Charts. In Færgemand and Sarma [4], pages 179–190.

[4] Validity can be ensured in different ways, e.g. implicit lost and found messages when an alternative of the embedded expression cannot serve all necessary communications from the outer MSC.

[7] ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Charts (MSC)*. ITU, Geneva, September 1993.

[8] ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Charts (MSC)*. ITU, Geneva, 1996.

[9] W. Janssen. *Layered Design of Parallel Systems*. PhD thesis, University of Twente, 1994.

[10] P. B. Ladkin and S. Leue. Interpreting message flow graphs. *Formal Aspects of Computing*, 7(5):473–509, 1995.

[11] S. Mauw and M. A. Reniers. An algebraic semantics of Basic Message Sequence Charts. *The Computer Journal*, 37(4):269–277, 1994.

[12] S. Mauw and M.A. Reniers. High-Level Message Sequence Charts. In Cavalli and Sarma [2], pages 291–306.

[13] S. Mauw and M.A. Reniers. Operational semantics for MSC'96. *Computer Networks & ISDN Systems*, 1998. (to appear).

[14] V. Pratt. Modeling concurrency with partial orders. *International Journal of Parallel Programming*, 15:33–71, 1986.