A Non–Interleaving Semantics for MSC

Stefan Heymer
University of Lübeck, Institute for Telematics
Ratzeburger Allee 160, D–23538 Lübeck, Germany
Tel. +49 451 500 3724, Fax +49 451 500 3722
E–mail `heymer@itm.mu-luebeck.de`

Abstract
In this paper, we develop a non–interleaving semantics and an interleaving semantics for MSC'96 based on the model of families of partially ordered sets. These semantics are no denotational semantics, but are defined via translations of the textual syntax for MSC'96 into a process algebra. We show that the interleaving semantics agrees on BMSC with the standardized semantics for MSC'92, while the non–interleaving semantics can be given an interleaving interpretation.

Keywords
MSC'96, process algebra, families of posets, semantics

1 MOTIVATION

With the recent additions given in [3], Message Sequence Charts have become a very sophisticated formal description technique for the specification of distributed and reactive systems. Thus, one has a fully developed specification language for a constraint oriented specification of systems.

Yet, there is only a tentative semantics for full MSC'96 given in [6]. This semantics is an interleaving semantics, i. e. the behavior of a system is modeled by sequences of transitions. The intuition behind this approach is that actions are atomic and consume no time, and furthermore that at each moment in time only *one* action can be executed.

Such an interleaving semantics does not model distributed and communicating systems faithfully, because as long as there is no direct communication or synchronization between events at different locations in a distributed system, there are no causal relations between these events. With respect to this, a non–interleaving semantics would do a better job: Causally not related events in a system trace would be modeled by unrelated elements in a partially ordered trace of a system.

This especially holds if one is about to take the transition from untimed systems to timed systems, i. e. systems where actions consume time. Here, using transition systems as a system model usually leads to complicated and unintuitive results. Mostly, instantaneous actions are used together with (time consuming) delays in such models. In reality, we have actions consuming time, providing for interesting situations such as "event a happens while event b is still happening". We are convinced that this transition from untimed to timed systems can be modeled more naturally with non–interleaving models, such as Petri nets or event structures.

In the scope of this paper, we give a non–interleaving semantics to MSC. This semantics will be based on families of labeled partially ordered sets (posets), a model introduced by Rensink in [8] as a quite general event–based model. We have chosen this model over other event–based models (for example prime event structures [10]) for its direct representation of system runs as partially ordered traces.

This paper proceeds as follows: First, we define a process algebraic language which we use in our construction of a semantics for MSC by translating MSC to terms in the process algebraic language. Next, we define two different semantics for the process algebraic language, an interleaving one and a non–interleaving one. By composing these the translation function and the denotation function we will get an alternative interleaving semantics for MSC as well as an non–interleaving semantics. We will compare these new semantics to the one given for MSC'92 in [2]. Finally, we close the paper with some conclusions and perspectives on future work.

2 TRANSLATING MSC TO A PROCESS ALGEBRA

To give a semantics to MSC, we will translate it into a process algebraic language instead of giving a denotational semantics directly to the MSC syntax. This has the advantage of making it possible to use different models for the semantics for this language. We will use two event oriented models in the next sections.

We start by developing a process algebra for describing the behavior of MSCs. As a basis, we choose an extension of the process algebra given by Winskel and Nielsen in [11]. Its syntax is given by

$$p ::= \mathsf{nil} \mid a \cdot p \mid p_0 \oplus p_1 \mid p_0 \mp p_1 \mid p_0 \times p_1 \mid p \upharpoonright \Lambda \mid p\{\Xi\} \mid x \mid \mathit{rec}\ x.p,$$

where a is a label, Λ is a subset of labels and Ξ is a partial function from labels to labels. We will denote the class of processes generated with this syntax with P. In this language, nil denotes a process that terminates instantly; $a \cdot p$ denotes a process that performs the action a and then behaves like p; $p_0 \oplus p_1$ denotes the choice between the behaviors of the processes p_0 and p_1; $p_0 \times p_1$ denotes the behavior of two processes p_0 and p_1 observed in parallel, the observations being pairs of labels or $*$, with $*$ being an idling action of a process; $p \upharpoonright \Lambda$ denotes the restriction of the behavior of p to just those actions labelled with symbols in Λ; $p\{\Xi\}$ denotes a relabelling of the actions in p according to the labelling function Ξ; and $\mathit{rec}\ x.p$ denotes a recursive behavior with x being a process variable.

Our definition of P deviates from the process algebra presented in [11] in two aspects: We introduce another operator $\mp$ denoting the *delayed choice* operator from [1], and we allow the renaming function Ξ in $p\{\Xi\}$ to be partial. The relaxation with respect to the relabelling operator allows us to hide actions from the behavior of a process by "forgetting about them", instead of restricting the behavior of it to behaviors not doing specific actions at all (like the restriction operator $\upharpoonright$ does). The first change is a real extension. Delayed choice cannot be expressed by a combination of the other operators, but is no universal construction in the categorical setting of [11], as it can be derived from the sum operation in the models. The same holds for our relaxation of relabelling. Furthermore, we allow processes to be named inside *process environments*.

To give a semantics to MSC, we start with a fragment of the language describing Basic MSCs (BMSCs). The textual syntax of this fragment is shown in Figure 1 and is a simplified form of the textual syntax for MSC'96 presented in [3]. To encode BMSCs into process algebraic terms, we first have to fix the set of actions underlying the terms. Letting *InstId* be the set of instance identifiers with $\mathit{env} \in \mathit{InstId}$ being the identifier for the environment, *MscId* be the set of Message Sequence Chart identifiers, *ActId* be the set of action identifiers and *MsgId* be the set of message identifiers, we define the set of actions to be

$$\begin{array}{rcl} \mathit{Act} & = & \{\mathit{in}(i,m)@j \mid i,j \in \mathit{InstId}, m \in \mathit{MsgId}\} \\ & & \cup\, \{\mathit{out}(i,m)@j \mid i,j \in \mathit{InstId}, m \in \mathit{MsgId}\} \\ & & \cup\, \{\mathit{act}(a)@i \mid i \in \mathit{InstId}, a \in \mathit{ActId}\} \\ & & \cup\, \{\mathit{start}@i \mid i \in \mathit{InstId}\} \\ & & \cup\, \{\mathit{end}@i \mid i \in \mathit{InstId}\}. \end{array}$$

In this alphabet, $\mathit{in}(i,m)@j$ denotes the receiving of message m from instance i by instance j, $\mathit{out}(i,m)@j$ denotes the sending of message m from instance j to instance i, and $\mathit{act}(a)@i$ denotes the execution of action a by instance i. The two special actions $\mathit{start}@i$ and $\mathit{end}@i$ denote the start and end of instance i, respectively. We use these special actions in modeling the weak sequential composition of MSCs.

We proceed in a bottom–up manner, defining translation $\mathbb{T}_{\mathtt{i}}$ for each nonterminal `<i>` of the syntax in Figure 1. Starting out with inputs, outputs and actions, we define translation functions

$$\begin{array}{rcl} \mathbb{T}_{\text{in}}(i)(\mathtt{in}\ m\ \mathtt{from}\ \mathtt{<address>}) & = & \mathit{start}@i \cdot \mathit{in}(\mathbb{T}_{\mathit{address}}(\mathtt{<address>}), m)@i \cdot \mathit{end}@i \cdot \mathsf{nil} \\ \mathbb{T}_{\text{out}}(i)(\mathtt{out}\ m\ \mathtt{to}\ \mathtt{<address>}) & = & \mathit{start}@i \cdot \mathit{out}(\mathbb{T}_{\mathit{address}}(\mathtt{<address>}), m)@i \cdot \mathit{end}@i \cdot \mathsf{nil} \\ \mathbb{T}_{\text{action}}(i)(\mathtt{action}\ a) & = & \mathit{start}@i \cdot \mathit{act}(a)@i \cdot \mathit{end}@i \cdot \mathsf{nil} \end{array}$$

parameterized with the instance identifier, prefixing the input, output and execution actions of the process algebra with start and termination actions for the instance. This may seem odd at first sight, but will become clear as soon as we define a translation function for the instance body. But first we define the translation $\mathbb{T}_{\mathtt{address}}$ used in the equations above:

$$\begin{array}{rcll} \mathbb{T}_{\text{action}}(i)(j) & = & j, & \quad j \in \mathit{InstId} \\ \mathbb{T}_{\text{action}}(i)(\mathtt{env}) & = & \mathit{env} & \end{array}$$

```
<msc>          ::=  msc <msc name>; <msc body> endmsc;
<msc body>     ::=  <> | <inst def> <msc body>
<inst def>     ::=  instance <inst name>; <inst body> endinstance;
<inst body>    ::=  <> | <event> <inst body>
<event>        ::=  <out> | <in> | <action> | <coregion>
<out>          ::=  out <msg name> to <address>
<in>           ::=  in <msg name> from <address>
<action>       ::=  action <at>;
<address>      ::=  <inst name> | env
<coregion>     ::=  concurrent <coevents> endconcurrent;
coevents       ::=  <> | <in> <coevents> | <out> <coevents>
```

Figure 1: Simplified syntax for BMSC

The special events $\mathit{start}@i$ and $\mathit{end}@i$ are also used in the definition of the translation of coregions. Here, these actions are used to gather the events in the coregion by synchronizing on the start and end actions. This is done with the synchronization operator $||_A$ defined for $A \subseteq \mathit{Act}$ as

$$p_1 \,||_A\, p_2 = ((p_1 \times p_2) \restriction \Lambda_A)\{\Xi_A\},$$

restricting the actions of the product $p_1 \times p_2$ to the label set

$$\Lambda_A = (\mathit{Act} - A) \times \{*\} \cup \{(a, a) \mid a \in A\} \cup \{*\} \times (\mathit{Act} - A)$$

and relabelling the labels with the function

$$\Xi_A : \begin{cases} (a, *) & \mapsto & a \\ (a, a) & \mapsto & a \\ (*, a) & \mapsto & a. \end{cases}$$

With this operation, we are able to define the translation for coregions as

$$\mathbb{T}_{\text{coregion}}(i)(\texttt{concurrent <coevents> endconcurrent;}) = \mathbb{T}_{\text{coevents}}(i)(\texttt{<coevents>}).$$

The translation for coevents is done using the equations

$$\begin{array}{rcl} \mathbb{T}_{\text{coevents}}(i)(\texttt{<>}) & = & \mathit{start}@i \cdot \mathit{end}@i \cdot \mathsf{nil} \\ \mathbb{T}_{\text{coevents}}(i)(\texttt{<in> <coevents>}) & = & \mathbb{T}_{\text{in}}(i)(\texttt{<in>} \,||_{\{\mathit{start}@i, \mathit{end}@i\}}\, \mathbb{T}_{\text{coevents}}(i)(\texttt{<coevents>}) \\ \mathbb{T}_{\text{coevents}}(i)(\texttt{<out> <coevents>}) & = & \mathbb{T}_{\text{out}}(i)(\texttt{<out>} \,||_{\{\mathit{start}@i, \mathit{end}@i\}}\, \mathbb{T}_{\text{coevents}}(i)(\texttt{<coevents>}), \end{array}$$

the sequence $\mathit{start}@i \cdot \mathit{end}@i \cdot \mathsf{nil}$ as the translation for the empty sequence of coevents serving as a "connector" for the remaining coevents or events in the instance.

How do we translate the bodies of instances? For this, we use an operator for weak sequential composition, which we define as a shorthand notation in our process algebra P as

$$p_1 \circ p_2 = ((p_1 \times p_2) \restriction \Lambda_\circ)\{\Xi_\circ\}$$

with p_1 and p_2 being process terms. The unrestricted product $p_1 \times p_2$ is restricted to the label set

$$\begin{array}{rcl} \Lambda_\circ & = & (\mathit{Act} - \{\mathit{end}@i \mid i \in \mathit{InstId}\}) \times \{*\} \\ & & \cup \{(\mathit{end}@i, \mathit{start}@i) \mid i \in \mathit{InstId}\} \\ & & \cup \{*\} \times (\mathit{Act} - \{\mathit{start}@i \mid i \in \mathit{InstId}\}), \end{array}$$

thus forcing the events in p_2 to happen strictly after the events in p_1. Hence, the instances are "pasted" together, synchronizing the $\mathit{end}@i$ action of p_1 with the $\mathit{start}@i$ action of p_2. With the relabelling function Ξ defined as

$$\Xi_\circ : \begin{cases} (a, *) & \mapsto & a, & a \in \mathit{Act} - \{\mathit{end}@i \mid i \in \mathit{InstId}\} \\ (*, a) & \mapsto & a, & a \in \mathit{Act} - \{\mathit{start}@i \mid i \in \mathit{InstId}\}, \end{cases}$$

we drop the distinction between actions from p_1 and p_2, while also dropping the now superfluous composite action $(\mathit{end}@i, \mathit{start}@i)$ at the point of connection between p_1 and p_2 leaving it out of the domain of $\Xi_\circ$.

With this operation, we are able to define the translation of an instance body as

$$\begin{aligned}
&\mathbb{T}_{\text{inst body}}(i)(\texttt{<>}) = \mathsf{nil}\\
&\mathbb{T}_{\text{inst body}}(i)(\texttt{<event> <inst body>})\\
&\quad = \mathbb{T}_{\text{event}}(i)(\texttt{<event>}) \circ \mathbb{T}_{\text{inst body}}(i)(\texttt{<inst body>}),
\end{aligned}$$

with $\mathbb{T}_{\text{event}}$ being defined as

$$\begin{aligned}
\mathbb{T}_{\text{event}}(i)(\texttt{<in>}) &= \mathbb{T}_{\text{in}}(i)(\texttt{<in>})\\
\mathbb{T}_{\text{event}}(i)(\texttt{<out>}) &= \mathbb{T}_{\text{out}}(i)(\texttt{<out>})\\
\mathbb{T}_{\text{event}}(i)(\texttt{<action>}) &= \mathbb{T}_{\text{action}}(i)(\texttt{<action>})\\
\mathbb{T}_{\text{event}}(i)(\texttt{<coregion>}) &= \mathbb{T}_{\text{coregion}}(i)(\texttt{<coregion>}).
\end{aligned}$$

For the definition of instances, the instance identifier in the definition is used as an additional parameter for the translation function for instance bodies, leading to the definition

$$\begin{aligned}
&\mathbb{T}_{\text{inst def}}(\texttt{instance}\,i\texttt{; <inst body> endinstance;})\\
&\quad = \mathbb{T}_{\text{inst body}}(i)(\texttt{<inst body>}), \qquad\qquad i \in \mathit{InstId}.
\end{aligned}$$

Now the translation of Message Sequence Chart poses no problem: the instances in the body are translated, and the translation of the complete MSC simply is the unsynchronized parallel composition of the translations of the instances. Thus, the definition for $\mathbb{T}_{\text{mscbody}}$ reads

$$\begin{aligned}
&\mathbb{T}_{\text{msc body}}(\texttt{<>}) = \mathsf{nil}\\
&\mathbb{T}_{\text{msc body}}(\texttt{<inst def> <msc body>})\\
&\quad = \mathbb{T}_{\text{inst def}}(\texttt{<inst def>}) \,||_{\emptyset}\, \mathbb{T}_{\text{msc body}}(\texttt{<msc body>}),
\end{aligned}$$

using the synchronization operator $||_A$ with an empty synchronization set for the parallel composition.

For the translation of an MSC definition, we define the translation function to return an process environment by binding a process algebraic term (the translation of the MSC body) to an identifier (the name of the MSC). But furthermore we have to take care of the communications between the instances. Here, we have to define simple buffers guaranteeing the right order of *in* and *out* events. This can be done with the definition

$$\mathit{CHANS} = ||_{i,j \in \mathit{InstId}, m \in \mathit{MsgId}}\, \mathit{rec}\; x.\mathit{out}(j,m)@i \cdot (x \,||_{\emptyset}\, \mathit{in}(i,m)@j \cdot \mathsf{nil}),$$

building a matrix of unsynchronized simple channels that take care of the delivery of specific messages. In this definition, the operator $||_{i,j \in \mathit{InstId}, m \in \mathit{MsgId}}$ denotes the unsynchronized parallel composition of its argument parameterized with all combinations of i, j and m. As we are using recursive process definitions for the translation of High–Level MSCs (HMSCs), we are not able to take the simpler solution

$$\mathit{CHANS}' = ||_{i,j \in \mathit{InstId}, m \in \mathit{MsgId}}\, \mathit{out}(j,m)@i \cdot \mathit{in}(i,m)@j \cdot \mathsf{nil}.$$

Even though we rely on the uniqueness of message identifiers in MSCs, possible loops in HMSCs and the weak sequential composition of nodes in HMSCs require the single transmission processes in the channel collection to spawn copies of themselves after the sending of the message.

Defining the synchronization set

$$\mathit{INOUT} = \{\mathit{in}(i,m)@j \mid i,j \in \mathit{InstId}, m \in \mathit{MsgId}\} \cup \{\mathit{out}(i,m)@j \mid i,j \in \mathit{InstId}, m \in \mathit{MsgId}\},$$

we can define the translation of a MSC to be

$$\mathbb{T}_{\text{msc}}(\texttt{msc}\;m\;\texttt{; <msc body> endmsc;}) = \{m \mapsto (\mathbb{T}_{\text{msc body}}(\texttt{<msc body>}) \,||_{\mathit{INOUT}}\, \mathit{CHANS})\}.$$

The need of returning an environment of process definitions instead of just one tuple consisting of a MSC identifier and the translation of the body will become clearer when we take a look at the translation of High–Level MSCs.

With the definitions made above we are able to describe Basic MSCs as environments of process definitions. Next, we are going to extend our syntax by allowing to specify High–Level MSCs. The additional syntax rules are shown in Figure 2. These rules can also be found in [3].

```
<msc>            ::=  msc <msc name>; expr <msc expr> endmsc;
<msc expr>       ::=  <start>; <node exprs>
<start>          ::=  <label> <alternatives>
<alternatives>   ::=  <> | alt <label> <alternatives>
<node exprs>     ::=  <> | <node expr> <node exprs>);
<node expr>      ::=  <label>:  <node> seq (<label> <alternatives>);
                       | <label>:  end
<node>           ::=  empty | <msc name> | <par expr> | connect
<par expr>       ::=  expr <msc expr> endexpr <par exprs>
<par exrs>       ::=  <> | par expr <msc expr> endexpr <par exprs>
```

Figure 2: Simplified syntax for HMSC

For defining the translations of HMSCs, we again proceed in a bottom–up manner, giving the definition of `par` expressions later in this section. Hence, we start out with the definition of the translation function for HMSC nodes:

$$\begin{aligned} \mathbb{T}_{\text{node}}(\texttt{empty}) &= ||_{i \in \mathit{InstId}} \mathit{start}@i \cdot \mathit{end}@i \cdot \mathsf{nil} \\ \mathbb{T}_{\text{node}}(m) &= m, \qquad m \in \mathit{MscId} \\ \mathbb{T}_{\text{node}}(\texttt{connect}) &= ||_{i \in \mathit{InstId}} \mathit{start}@i \cdot \mathit{end}@i \cdot \mathsf{nil} \end{aligned}$$

In this translation, `empty` as well as `connect` are mapped to a collection of process denoting just the start and termination of instances. The occurrence of the MSC identifier m on the right hand side of the second equation means the interpretation of m as a process name.

Alternatives in HMSCs are meant to be accumulated with delayed choice according to [3]. This is captured in the translation function for alternatives reading

$$\begin{aligned} \mathbb{T}_{\text{alternatives}}(\texttt{<>}) &= \mathsf{nil} \\ \mathbb{T}_{\text{alternatives}}(\texttt{alt}\, l \,\texttt{<alternatives>}) &= l \mp \mathbb{T}_{\text{alternatives}}(\texttt{<alternatives>}). \end{aligned}$$

With this definition we are able to give a translation for node expressions, mapping each node expression to an association in an environment, hence

$$\begin{aligned} &\mathbb{T}_{\text{node expr}}(l_1\texttt{:}\quad \texttt{<node> seq (}l_2\, \texttt{<alternatives>);}) \\ &\quad = \{l_1 \mapsto \mathbb{T}_{\text{node}}(\texttt{<node>}) \circ (l_2 \mp \mathbb{T}_{\text{alternatives}}(\texttt{<alternatives>}))\} \\ &\mathbb{T}_{\text{node expr}}(l\texttt{:}\quad \texttt{end}) = \{l_1 \mapsto ||_{i \in \mathit{InstId}} \mathit{start}@i \cdot \mathit{end}@i \cdot \mathsf{nil}\}. \end{aligned}$$

These environments are then accumulated in the translation $\mathbb{T}_{\text{node exprs}}$ of a list of node expressions:

$$\begin{aligned} &\mathbb{T}_{\text{node exprs}}(<>) = \{\} \\ &\mathbb{T}_{\text{node exprs}}(\texttt{<node expr> <node exprs>}) \\ &\qquad = \mathbb{T}_{\text{node expr}}(\texttt{<node expr>}) \cup \mathbb{T}_{\text{node exprs}}(\texttt{<node exprs>}) \end{aligned}$$

As a translation of a starting node of an HMSC we choose a process term gained from a delayed choice between the set of labels occurring in the `start` expression, as they denote the processes starting from that labels:

$$\mathbb{T}_{\text{start}}(l\, \texttt{<alternatives>}) = l \mp \mathbb{T}_{\text{alternatives}}(\texttt{<alternatives>})$$

An MSC expression contains a start node description as well as a set of node expressions. Hence, we translate a MSC expression into a tuple consisting of a process term and an environment:

$$\mathbb{T}_{\text{msc expr}}(\texttt{<start>; <node exprs>}) = (\mathbb{T}_{\text{start}}(\texttt{<start>}), \mathbb{T}_{\text{node exprs}}(\texttt{<node exprs>}))$$

These tuples are then separated by the extended translation function for MSCs. The environment found in the translation of the node expressions has to be extended with an association of the MSC identifier with the start expression. Thus we define

$$\begin{aligned} &\mathbb{T}_{\text{msc}}(\texttt{msc}\, m\texttt{; expr <msc expr> endmsc;}) \\ &\quad = \{m \mapsto \sigma_1(\mathbb{T}_{\text{msc expr}}(\texttt{<msc expr>}))\} \cup \sigma_2(\mathbb{T}_{\text{msc expr}}(\texttt{<msc expr>})), \end{aligned}$$

with the selection functions σ_1 and σ_2 being defined as follows:

$$\begin{array}{rcl} \sigma_1(a,b) & = & a \\ \sigma_2(a,b) & = & b \end{array}$$

So far, we have presented not much more than a change in syntax — going over from the concrete textual syntax of HMSC to the more abstract syntax of our process algebra P. In the next section, we consider two different semantics for this process algebra.

3 BUILDING A FAMILIES OF POSETS SEMANTICS FOR MSC

For the semantics for our process algebra P, we need a suitable class of models. We have decided to use families of labelled partial ordered sets (posets) as this class of models, a model introduced by Rensink in [8] as a quite general event–based model. In this model, the behavior of a system is modeled by the evolution of traces, starting with an empty trace.

To define families of posets, we first have to introduce labelled posets as an auxiliary notion.

Definition Let Λ be a set of labels. A partially ordered set labelled over Λ (lposet) is a triple $P = \langle E_P, \leq_P, \ell_P \rangle$ with E_P a set of events, $\leq_P \subseteq E_P \times E_P$ a reflexive, anti–symmetric and transitive order (i. e. a partial order) on E_P, and $\ell_P : E_P \to \Lambda$ a labelling function. The class of all labelled partial orders is denoted LPO.

In the setting presented here, we will label the events in an lposet with actions, hence an event denotes the occurrence of the action it is labelled with.

For comparing labelled posets, we define a number of ordering relations on them. The first one focuses on the causality information conveyed by the orders inside the posets.

Definition Let P and Q be lposets. P is called *smoother* than Q, denoted $P \sqsubseteq Q$, if and only if

$$E_P = E_Q \ \wedge \ \ell_P = \ell_Q \ \wedge \ \leq_Q \subseteq \leq_P$$

holds.

That is, a lposet P is smoother than an lposet Q, if more events in it are causally dependent on each other.

The next relation defined on lposets is the prefix relation:

Definition Let P and Q be lposets. P is a prefix of Q, denoted $P \preceq Q$, if and only if

$$E_P \subseteq E_Q \ \wedge \ (\leq_P = \leq_Q \cap (E_P \times E_P)) \ \wedge \ \ell_P = \ell_Q \restriction E_P$$

holds. P is an *immediate predecessor* of Q, denoted $P \prec\!\!\cdot\, Q$, if and only if $|E_Q - E_P| = 1$.

With these definitions, we are able to define families of labelled posets.

Definition A *family of labelled posets* is a non–empty, $\preceq$–left–closed set of labelled partial orders, which are compatibly labelled, i. e. for a family of lposets $\mathcal{P}$ the condition

$$\forall P, Q \in \mathcal{P}.\ e \in E_P \cap E_Q \Rightarrow \ell_P(e) = \ell_Q(e)$$

holds. The class of all families of labelled partial orders is denoted $\mathbf{f}_{\mathrm{LPO}}$. Families of lposets will be ranged over by the letters $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{R}$.

In this definition, the condition of prefix left–closedness ensures that every possible lposet leading up to a given system run, which is itself a partial system run, is included in the family. The condition on being compatibly labelled ensures that the same event used in different members of the family cannot have different labels. This is only partially ensured by the prefix closure, as lposets with a common prefix do not have to be prefixes of another lposet themselves.

As special cases of the definitions of labelled partial orders and families of these we also consider labelled total orders and families of labelled total orders, denoting them with LTO and $\mathbf{f}_{\mathrm{LTO}}$ without elaborating the (obvious) definitions.

As we are going to develop a compositional semantics for the process algebra P, we now have to define operations on families of posets that mirror the intended effects of the operators in the process algebra. We begin with the prefixing operator $a \cdot p$ of P. On the model level, this operator should prefix every trace in a family of posets with a new event e labelled a. For this, we define the prefixing of lposets as

$$ {}_e a \cdot P = \langle E_P \cup \{e\}, \leq_P \cup (\{e\} \times E_P), \ell_P \cup \{e \mapsto a\} \rangle \,, \qquad e \neq E_P $$

and use it in the definition of the prefixing of a family of posets in the obvious way:

$$ {}_e a \cdot \mathcal{P} = \{\epsilon\} \cup \{ {}_e a \cdot P \mid P \in \mathcal{P}\} $$

The choice operator $\oplus$ on P could easily be modeled as the union of two families of lposets. Yet, to stay inside the categorical framework of [11], we define injection functions ι_0 and ι_1 on lposets as

$$ \iota_0(P) = \langle \{0\} \times E_P, \{((0,e_1),(0,e_2)) \mid (e_1,e_2) \in \leq_P\}, \{(0,e) \mapsto a \mid e \mapsto a \in \ell_P \rangle $$

and

$$ \iota_1(P) = \langle \{1\} \times E_P, \{((1,e_1),(1,e_2)) \mid (e_1,e_2) \in \leq_P\}, \{(1,e) \mapsto a \mid e \mapsto a \in \ell_P \rangle $$

marking every lposet in sum of two families with the argument position of the family of lposets it came from. Hence, we define the sum of two families of lposets as

$$ \mathcal{P} \oplus \mathcal{Q} = \{\iota_0(P) \mid P \in \mathcal{P}\} \cup \{\iota_1(Q) \mid Q \in \mathcal{Q}\}. $$

Similarly we handle the product of families of lposets. The standard technique in event–based models is to construct the events of a product by taking pairs of events from the operands. Such pairs are assumed to be the result of synchronization of the two original events. A "pseudo–event" $*$ is inserted in either position of these pairs to denote that the corresponding operand does not partake in synchronization, i. e. that in the combined event there is no real synchronization involved. We define two projection functions π_1 and π_2 on events as

$$ \pi_i : (e_1, e_2) \mapsto \begin{cases} \text{undefined} & \text{if } e_i = * \\ e_i & \text{otherwise.} \end{cases} $$

We extend these functions to lposets by defining

$$ \pi_i^*(P) = \langle E, \leq, \ell \rangle $$

with the event set

$$ E = \{\pi_i((e,e')) \mid (e,e') \in E_P\}, $$

the partial ordering relation

$$ \leq \, = \{(\pi_i((e_1,e_1')), \pi_i((e_2,e_2')) \mid ((e_1,e_1'),(e_2,e_2')) \in \leq_P \wedge \pi_i((e_1,e_1')) \neq * \wedge \pi_i((e_2,e_2')) \neq *\} $$

and the labelling function

$$ \ell = \{\pi_i((e,e')) \mapsto a \mid (e,e') \mapsto a \in \ell_P \wedge \pi_i((e,e')) \neq *\}. $$

The product of two families of lposets then is defined as

$$ \mathcal{P} \times \mathcal{Q} = \max\nolimits_{\sqsubseteq}\{R \in \mathrm{LPO} \mid \exists P \in \mathcal{P}, Q \in \mathcal{Q}.\, \pi_1^*(R) \sqsubseteq P \wedge \pi_2^*(R) \sqsubseteq \mathcal{Q}\}, $$

i. e. we take the maxima with respect to "unorderedness" of those lposets that project on something smoother than lposets in $\mathcal{P}$ and $\mathcal{Q}$, respectively.

For the relabelling operator we again define first a relabelling on lposets as

$$ P\{\Xi\} = \langle E, \leq_P \cap (E \times E), \Xi \circ \ell_P \rangle \,, $$

where

$$ E = \mathrm{dom}\,(\Xi \circ \ell_P). $$

The restriction to the new event set E in this definition has its causes in Ξ being possibly partial. Events that are not assigned any label by the new labelling function $\Xi \circ \ell_P$ have to be removed from the event set as well as the ordering relation of the resulting lposet. Thus, we define the relabelling of a family of lposets by

$$\mathcal{P}\{\Xi\} = \{P\{\Xi\} \mid P \in \mathcal{P}\}.$$

For the restriction operator $\cdot \upharpoonright \cdot$ of P, we filter the family of lposets and discard all those posets containing elements outside of the label set Λ:

$$\mathcal{P} \upharpoonright \Lambda = \{P \in \mathcal{P} \mid \operatorname{ran} \ell_P \sqsubseteq \Lambda\}$$

We do not present the semantics for the delayed choice operator and the recursion operator in the scope of this paper. These operators are the more complicated ones, the delayed choice operator requiring the selection of specific members of a product family of posets, and the recursion operator requiring an introduction into fixed point theory. These two operators will be denoted by $(\cdot \mp \cdot)$ and $(\mathrm{rec}\ \cdot.\cdot)$.

With the operators presented so far, we are able to define a non–interleaving semantics for P. For this, we define a denotation function $[\![\cdot]\!]_1$ taking as its arguments a process term and an environment $\Theta : \mathit{Var} \to \mathbf{f}_{\mathrm{LPO}}$, Var being a set of variables. The environment Θ is deduced from a process environment ρ obtained from the translation process described in the previous chapter. Hence, we are able to define our compositional semantics as

$$\begin{array}{rcl}
[\![\mathsf{nil}]\!]_1\Theta & = & \{\varepsilon\} \\
[\![a \cdot p]\!]_1\Theta & = & {}_e a \cdot ([\![p]\!]_1\Theta) \\
[\![p_1 \oplus p_2]\!]_1\Theta & = & ([\![p_1]\!]_1\Theta) \oplus ([\![p_2]\!]_1\Theta) \\
[\![p_1 \mp p_2]\!]_1\Theta & = & ([\![p_1]\!]_1\Theta) \mp ([\![p_2]\!]_1\Theta) \\
[\![p_1 \times p_2]\!]_1\Theta & = & ([\![p_1]\!]_1\Theta) \times ([\![p_2]\!]_1\Theta) \\
[\![p \upharpoonright \Lambda]\!]_1\Theta & = & ([\![p]\!]_1\Theta) \upharpoonright \Lambda \\
[\![p\{\Xi\}]\!]_1\Theta & = & ([\![p]\!]_1\Theta)\{\Xi\} \\
[\![x]\!]_1\Theta & = & \Theta(x) \\
[\![\mathit{rec}\ x.p]\!]_1\Theta & = & \mathrm{fix}(F), \qquad F(P) = [\![p]\!]_1(\{y \mapsto P \mid y \neq x\} \cup \{x \mapsto T\})
\end{array}$$

To define an interleaving semantics for P, we simply restrict our attention to families of ltosets instead of families of lposets. Nearly all the operators we defined on families of lposets also work as expected on families of ltosets, the notable exception being the product operator. Here, we need a different definition, as the maximal set with respect to “roughness” of the posets will contain mainly partial orders instead of total ones. This can easily be fixed by requiring the projections used in the definition of the product on families of lposets to be equal to the respective members of the argument. Hence, we define

$$\mathcal{P} \times_2 \mathcal{Q} = \{R \in \mathrm{LTO} \mid \exists P \in \mathcal{P}, Q \in \mathcal{Q}. \pi_1^*(R) = P \wedge \pi_2^*(R) = Q\},$$

This way, we are able to define the interleaving semantics mirroring the equations for the denotation function given above, with the equation for the product operator in P adapted.

$$\begin{array}{rcl}
[\![\mathsf{nil}]\!]_2\Theta & = & \{\varepsilon\} \\
[\![a \cdot p]\!]_2\Theta & = & {}_e a \cdot ([\![p]\!]_2\Theta) \\
[\![p_1 \oplus p_2]\!]_2\Theta & = & ([\![p_1]\!]_2\Theta) \oplus ([\![p_2]\!]_2\Theta) \\
[\![p_1 \mp p_2]\!]_2\Theta & = & ([\![p_1]\!]_2\Theta) \mp ([\![p_2]\!]_2\Theta) \\
[\![p_1 \times p_2]\!]_2\Theta & = & ([\![p_1]\!]_2\Theta) \times_2 ([\![p_2]\!]_2\Theta) \\
[\![p \upharpoonright \Lambda]\!]_2\Theta & = & ([\![p]\!]_2\Theta) \upharpoonright \Lambda \\
[\![p\{\Xi\}]\!]_2\Theta & = & ([\![p]\!]_2\Theta)\{\Xi\} \\
[\![x]\!]_2\Theta & = & \Theta(x) \\
[\![\mathit{rec}\ x.p]\!]_2\Theta & = & \mathrm{fix}(F), \qquad F(P) = [\![p]\!]_2(\{y \mapsto P \mid y \neq x\} \cup \{x \mapsto T\})
\end{array}$$

How do these semantics relate to the one standardized in [2]? To show the consistency of our semantics with the MSC’92 semantics, we first look at our interleaving semantics. After building a labelled transition system from a family of ltosets, we check for strong bisimilarity between this transition system and the process graph associated to the process algebraic semantics of [2].

So let $\mathcal{P}$ be a family of lposets. We define the transition system associated to $\mathcal{P}$ as

$$\mathit{Trans}(\mathcal{P}) = (\mathcal{P}, \varepsilon, \to)$$

with the transition relation $\rightarrow$ defined as

$$\rightarrow\, = \{(P, a, Q) \mid P, Q \in \mathcal{P} \wedge P \prec\!\!\cdot\, Q \wedge a \in \operatorname{ran} \ell_{Q \setminus P}\}.$$

That is, we take the members of the family of lposets as the nodes in the transition systems, allowing transitions only from direct predecessors of a lposet. Abstracting from *start* and *end* actions, we get the following theorem:

Theorem For a BMSC M, the transition system $\mathit{Trans}(\llbracket M\{\Xi\}\rrbracket_2\{\})$ and the synchronization tree obtained from unfolding the process graph for $\mathcal{P}\llbracket M\rrbracket$, with Ξ being the relabelling function

$$\Xi : a \mapsto \begin{cases} \text{undefined} & \text{if } a = \mathit{start}@i \text{ or } a = \mathit{end}@i \text{ for some } i \in \mathit{InstId} \\ a & \text{otherwise.} \end{cases}$$

To relate our $\mathbf{f}_{\mathrm{LPO}}$–semantics to the standardized one, we show a property of the diagram in Figure 3. Assuming M to be a MSC with MSC identifier i, we observe that

$$(\mathit{seq} \circ (\llbracket\cdot\rrbracket_1\emptyset) \circ \mathbb{T}_{\mathrm{msc}}(M))(i) = ((\llbracket\cdot\rrbracket_2\emptyset) \circ \mathbb{T}_{\mathrm{msc}}(M))(i)$$

holds, where *seq* is a function defined as

$$\begin{aligned} &\mathit{seq} : \mathbf{f}_{\mathrm{LPO}} \rightarrow \mathbf{f}_{\mathrm{LTO}} \\ &\mathit{seq}(\mathcal{P}) = \textstyle\bigcup_{P \in \mathcal{P}} \{T \in \mathrm{LTO} \mid T \sqsubseteq P\} \end{aligned}$$

which sequentializes each member of its argument. This we, we inherit the property of the interleaving semantics stated above.

$$\begin{tikzcd}
 & \mathrm{MSC} \arrow[d, "\mathbb{T}_{\mathrm{msc}}(\cdot)"'] & \\
 & \mathsf{P} \arrow[dl, "\llbracket\cdot\rrbracket_2\emptyset"'] \arrow[dr, "\llbracket\cdot\rrbracket_1\emptyset"] & \\
\mathbf{f}_{\mathrm{LTO}} & & \mathbf{f}_{\mathrm{LPO}} \arrow[ll, "seq"']
\end{tikzcd}$$

Figure 3: Giving two different semantics to MSC

4 CONCLUSIONS AND FUTURE WORK

What have we achieved so far? We have shown how to construct a non–interleaving semantics for MSC. This semantics follows the idea presented in [2] and [5]. We have also constructed an interleaving semantics for MSC that agrees with the standardized one for Basic MSCs.

But why again an translational approach? There already have been presented operational semantics for MSC [7], as well as an non–interleaving semantics based on partially ordered multisets [4]. What is the difference between these semantics and the one presented in this paper?

The operational semantics from [7] is an interleaving semantics, which is not bad in itself. Yet, if we are about to take the transition to timed systems (and there are already extensions of MSC with the incorporation of time aspects, e. g. [9]), this interleaving behavior is in our opinion more of a hindrance than really helpful. Simultaneously occurring events that consume time cannot be faithfully modeled in an interleaving setting. They mostly are modeled by using instantaneous events together with delay events. In this aspect, non–interleaving semantics are advantageous, as two unordered events e_1 and e_2 are not artificially ordered, hence the assignment of timing information to the events is easier.

The denotational semantics from [4] is conceptually similar to the semantics in terms of families of lposets we are presenting in this paper. The main differences lie in the two–step construction process we are applying here, and in the usage of a model not abstracting from event identities (as pomsets are isomorphism classes of labelled posets). Yet, the two–step approach has its own advantage: it is easily possible to give a different semantics for P, using as the class of models for instance Petri nets, event structures or asynchronous transition systems. Except for our relaxation on the relabelling operator and the delayed choice operator introduced into P, such semantics are readily available, for instance in [11].

For our future work, we will concentrate on the introduction of timing aspects into the semantics shown here, working towards a semantics for a timed variant of MSC'96.

REFERENCES

[1] J. C. M. Baeten; S. Mauw: "Delayed choice: an operator for joining Message Sequence Charts", in D. Hogrefe; S. Leue (eds.): "Formal Description Techniques VII", IFIP Transactions C, Proceedings Seventh International Conference on Formal Description Techniques, pages 340–354, Chapman & Hall, 1995.

[2] ITU-TS: "ITU-TS Recommendation Z.120 Annex B: Algebraic semantics of Message Sequence Charts", ITU-TS, Geneva, April 1995.

[3] ITU-TS: "ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)", ITU-TS, Geneva, September 1996.

[4] J.-P. Katoen; L. Lambert: "Pomsets for message sequence charts", in H. König *et. al.* (eds.): "Formale Beschreibungstechniken für verteilte Systeme, 8. GI/ITG-Fachgespräch", June 1998.

[5] S. Mauw; M. A. Reniers: "An Algebraic Semantics of Message Sequence Charts", Technical Report CSN 94/23, Eindhoven University of Technology, Department of Computing Science, Eindhoven, 1994.

[6] S. Mauw; M. A. Reniers: "High–Level Message Sequence Charts", in A. Cavalli and D. Vincent, editors, "SDL'97: Time for Testing — SDL, MSC and Trends, Proceedings of the Eighth SDL Forum", North–Holland, 1997.

[7] S. Mauw; M. A. Reniers: "Operational semantics for MSC'96", in A. Cavalli and D. Vincent, editors, "SDL'97: Time for Testing — SDL, MSC and Trends, Tutorials of the Eighth SDL Forum", pages 135–152, North–Holland, 1997.

[8] A. Rensink: "Models and Methods for Action Refinement", PhD thesis, University of Twente, Enschede, Netherlands, August 1993.

[9] I. Schiefferdecker; A. Rennoch; O. Mertens: "Timed MSCs – an extension to MSC'96", in A. Wolisz; I. Schiefferdecker; A. Rennoch (eds.): "Formale Beschreibungstechniken für verteilte Systeme, 7. GI/ITG-Fachgespräch". GMD, June 1997.

[10] G. Winskel: "An Introduction to Event Structures", volume 354 of *Lecture Notes in Computer Science*, pages 364–397, Springer-Verlag, Bratislava, 1989.

[11] G. Winskel; M. Nielsen: "The Handbook of Logic in Computer Science", chapter "Models for Concurrency", Springer–Verlag, 1992.