An M-Net Semantics for a Real-Time Extension of μSDL

Hans Fleischhack and Josef Tapken
Fachbereich Informatik
Carl-von-Ossietzky-Universität Oldenburg
D-26111 Oldenburg
{fleischhack, tapken}@informatik.uni-oldenburg.de

Abstract

In this paper an extension of SDL by real-time assertions is suggested. A denotational compositional semantics in terms of timed high level Petri nets is given. This semantics allows to reason about timed SDL-specifications by simulation of the underlying high level nets as well as by unfolding these into timed low level nets and then applying analysis methods for timed Petri nets or temporal model checking. The approach is embedded in the MOBY-tool.

1 Introduction

In the MOBY-project (Modelling of distributed systems; [1]), a design environment for SDL-specifications (Specification and Description Language; [8]) has been developed. Based on a high level Petri net semantics and a temporal logic for SDL, the MOBY-tool allows to reason about functional and qualitative temporal properties of such specifications. Reasoning is done by simulation or by checking SDL-specifications against properties expressed by temporal logic formulae, using the model checker of the the PEP project (Programming Environment Based on Petri Nets; cf. [4, 11]).

In general, the behaviour of a distributed system does depend on its real-time as on its functional and qualitative temporal properties. Therefore, a formal description method should offer features to express real-time requirements. In standard SDL, this is done by the timer concept. But, since – with the exception of the timer part – there are no assumptions about the timed behaviour of (the other parts of) an SDL-specification, the timer concept leads to certain anomalies. E.g., as Leue [13] points out, in some cases no finite bound of the delay between the point of time when a timer expires, and the moment at which the SDL-specification reacts to the expiry can be found.

As a solution in this paper an extension of SDL-specifications by real-time assertions is suggested. The assertions are given by time intervals and allow the specification of lower and upper time bounds for all events which may be observed in an SDL-system.

In accordance with the time model for timers in standard SDL (cf. Annex F3 of Z.100 [8]), our notion of time is discrete. We have chosen time intervals rather than time points, to be able to describe minimal as well as maximal reaction times of (parts of) SDL systems.

A compositional denotational semantics for the extended language SDL/R by means of high level Petri nets is developed. For this purpose the algebra of M-nets [6] is augmented by additional inscriptions which, for each transition, denote the earliest and latest possible firing time, yielding the algebra of TM-nets. A semantic function is then defined which specifies a TM-net for each (fragment of an) SDL/R-program.

There have been other proposals for real time extensions of SDL: In [2], information about the (average) amount of time the system will remain in a certain state and/or probabilities of transitions to other states, may be added to SDL specifications. This approach enables probabilistic validation and performance evaluation of SDL-specifications based on stochastic processes. Hence, the domain of properties to be analysed using this approach is rather different from the one presented here.

In [13], quantitative aspects of SDL-systems are modelled by additional specifications in terms of real-time extended temporal logic (MTL-) formulae. The allowed behaviour of an SDL system is given via a global state transition system, and is constrained by the MTL-formulae. In this approach, SDL-system and MTL-formula are viewed as complementary parts of the specification, while in ours the SDL-system rather establishes an abstract implementation which is checked against a temporal logic specification.

The paper is organized as follows: The real-time extension SDL/R of μSDL (cf. [1]) is presented in section 2. Section 3 contains the definition of the TM-net algebra. The core of the paper consists of the TM-net semantics of SDL/R in section 4. In the concluding section 5 we discuss verification of real-time properties of SDL/R-specifications and the embedding in the MOBY-tool.

2 SDL/R-Specifications

Over many years SDL has emerged as a standard language for the specification of distributed systems, especially in the area of telecommunication. An SDL-specification describes a *system* consisting of *processes*. Processes are extended communicating finite state machines. For processes timing constraints may be modelled by *timers*. Processes may be *created* and *terminated dynamically* and may also contain *procedures*. There is a notion of *hierarchy* which supports structured development of specifications. For the purpose of communication and for local variables *objects* are definable in SDL. Among others the fact that SDL-specifications may also be *represented graphically* has contributed to the wide acceptance of SDL.

The prototype implemented in the MOBY-project concentrates on that subset of SDL that deals with concurrency and communication [1]. In this setting a system appears as a set of non-deterministic sequential finite automata with

τ-transitions, extended by local variables and communicating asynchronously with one another. In this paper, we extend the language under consideration by timers. To avoid the anomalies mentioned in the introduction section, we also augment input queues, channels and atomic actions with time expressions, denoting the minimal and maximal delay time for an input queue or a channel or, respectively, the minimal and maximal execution time for an action. Our notion of time is *discrete*; hence we use natural numbers to denote time points. Still, we do not consider procedures, dynamic handling of processes, hierarchy, and abstract data types/objects.

$$
\begin{array}{lll}
< timerdef > & ::= \texttt{timer}\ t\ \texttt{;} & (1) \\
< set > & ::= \texttt{set}\ (t, \texttt{now} + < expr >) & (2) \\
< reset > & ::= \texttt{reset}\ (t) & (3) \\
< timeint > & ::= [< expr >, < expr >\ ] & (4)
\end{array}
$$

Table 1. Extension of the SDL-syntax pertaining to timers.

The syntax pertaining to timers is shown in Table 2. Timers are used to handle time supervision and have to be declared inside a process like local variables (1). It is assumed that a global time is available through the global variable `now`. A timer is set with an expiry time, which usually depends on the variable `now` (2). When the expiry time is reached, a signal with the name of the timer is inserted in the input queue of the process. A timer is cancelled by a `reset` statement (3). Setting a timer, which is already set, implies overwriting the original expiry time. Setting a timer to an earlier expiry time than the actual time `now`, causes an immediate insertion of the timer signal in the input queue. Resetting a timer that is not set has no effect. When the timer is set and neither consumed from the input queue nor reset, it is called *active*. Timer parameters are not considered in this paper. They may be treated like signal parameters, which are handled by our semantics.

Like in [1], as parameter types of signals and as types of local variables we allow the set of Boolean values and the set of integers (both with the usual operations). The resulting part of the language is called *SDL/R*.

The formal syntax of SDL/R is based on the *textual representation* of SDL and is given in appendix A. To increase readability, all examples in this text will be given using the *graphical representation*.

In the remaining part of this section, an example of an SDL/R-specification is given which describes a simple communication protocol, the ARQ (automatic repeat request) protocol with alternating acknowledgement.

The top level of the specification consists of a system ARQ (cf. Figure 1) comprising two processes, a process *sender* and a process *receiver*.

Via the channel d the *sender* sends data packages which are labelled alternately by 0 and 1 to the *receiver*. The data packages are modelled by a signal type *data* with parameter type $\{0, 1\}$. The channel d causes a delay of at least 5 time units and at most 10 time units.

The *receiver* either receives a data package correctly and answers, via the a channel, the label of the next data package. Otherwise, an error occurs. There are two types of error. First, the *receiver* may receive a faulty package. In this case it answers by repeating the label of the last package. Second, a package may get lost. In this case, the *receiver* does nothing. The possibility of erroneous transmission of data is modelled in the *receiver* process by the non-deterministic choice between the answers 0, 1, and no answer, respectively. For the answers there is a signal type *ack*, also with parameter type $\{0, 1\}$. Also channel a delays signals for at least 5 and at most 10 time units.

Before sending a data package, the *sender* sets a timer t, which will be reset, if the *receiver*'s acknowledge arrives in time and which otherwise forces the sender to send the last data package again.

The processes *sender* and *receiver* are modelled separately (cf. Figure 2). We assume that internal actions of a process consume 1 time unit and that access to a channel needs 2 time units. Timing constraints are given in additional square brackets; intervals of the form [*eft*, *eft*] are abbreviated by (*eft*).

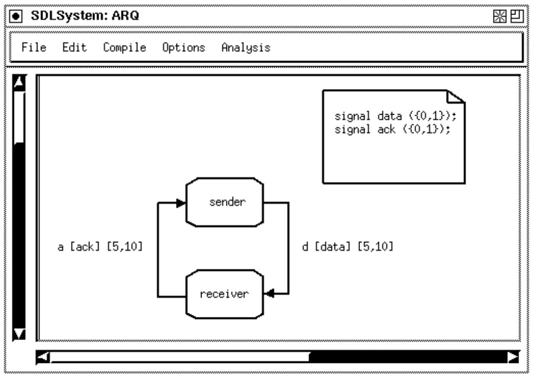

Fig. 1. The ARQ-System with time constraints

3 An Algebra of Timed Multi Labelled Petri Nets

M-nets (multi labelled nets) form an algebra of high level Petri nets. They were introduced in [6] as an abstract and flexible semantic model for concurrent programs and used in [7] for the definition of a high level net semantics of the experimental programming language $B(PN)^2$.

The most distinguishing feature of the algebra of M-nets is given by the rich set of composition operators they provide. These allow the compositional construction of complex nets from simple ones, thereby satisfying various algebraic properties.

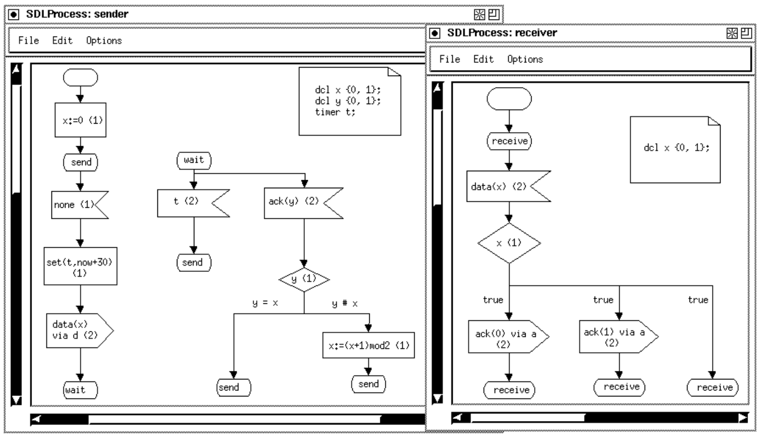

Fig. 2. The *sender* and *receiver* Process

Annotations of places (sets of allowed tokens), arcs (multisets of variables or values or tuples of variables and values), and transitions (occurrence conditions – called value terms) support the unfolding of an M-net into an elementary net. Communication capabilities are denoted by labels of transitions (action terms), while labels of places (called status) denote their interface capabilities. A status can be 'entry', 'exit' or 'internal'.

In this section we extend the notion of M-nets by adding time restrictions to transitions. *Discrete time* is assumed and therefore the domain for time points is restricted to the set $\mathbf{N}$ of the natural numbers. To each transition t, a time interval $[\mathit{eft}, \mathit{lft}]$ is attached. The informal meaning is that t has to occur at least *eft* and at most *lft* time units after being enabled, unless it becomes disabled by the occurrence of another transition. The resulting TM-net (time multi labelled Petri net) algebra is closely related to the notion of *time Petri net* (cf. e.g. [3, 14, 15]). We restrict the class of nets under consideration to *safe* nets. A (T)M-net N is called safe, if each marking m reachable in N from the standard initial marking m_0 (cf. section 3.2) is a set[1].

3.1 Definition of TM-Nets

The building blocks (places, transitions, arcs) of a TM-net N are equipped with inscriptions which are important for the behaviour as well as for the compositional structure of N.

There are three kinds of *places* in TM-nets: *entry* places (without incoming arcs; labelled by e), *exit* places (without outgoing arcs; labelled by x), and *internal* places (labelled by i). Moreover, each place s has a *type* which comprises the set of possible values for any tokens on that place. A type is a nonempty finite subset of the set VAL^*, where $VAL = \mathbf{N} \cup \mathbf{B} \cup \{\bullet\}$, with the set of Boolean

[1] I.e., $(m(s))(v) \in \{0,1\}$ for all $s \in S$ and $v \in \alpha_s$.

values **B** and the usual token in Petri nets $\bullet$. Hence the inscription ι_s of a place s consists of a pair (λ_s, α_s), where λ_s and α_s denote the label and the type of s, respectively. All entry places as well as all exit places have type $\{\bullet\}$.

We also assume a set VAR of typed variables called *value variables* and ranging over the types.

Each *transition* t has a triple $\iota_t = (\lambda_t, \alpha_t, \chi_t)$ of inscriptions. The first component λ_t expresses the synchronisation capabilities of t and consists of a finite multiset of *simple action terms*. A simple action term is an expression of the form $A(a_1, ..., a_n)$ where A is an *action symbol* with arity $S_1 \times ... \times S_n$ and a_i is either a value variable of type S_i or an element of S_i, $1 \leq i \leq n$.

On the set ACT of action symbols we assume a *conjugation function* $con: ACT \to ACT$ such that, for all $A \in ACT, con(A) \neq A, con(con(A)) = A$, and $arity(A) = arity(con(A))$. We use $\overline{A}$ as abbreviation for $con(A)$.

The second inscription α_t of a transition t consists of a set of value expressions. *Value expressions*[2] are built up by elements of VAL, value variables, and the usual relational, arithmetic, and boolean operators. Value expressions denote elements of VAL.

The third inscription χ_t of a transition restricts the set of time points, at which t may occur. It consists of a pair (eft, lft) of value terms which may be evaluated to a natural number. The intended meaning is that eft (lft, resp.) denotes the minimal (maximal, resp.) number of time units, which may pass between enabling of t and occurrence of t. The earliest firing time eft may be 0, the latest firing time lft may be ∞. Usually, χ_t is denoted by $[eft(t), lft(t)]$.

Finally, each *arc* (s,t) ((t,s), resp.) of an TM-net is inscribed by a finite multiset $\iota_{s,t}$ ($\iota_{t,s}$, resp.) of value variables of type α_s[3].

Hence, altogether, an TM-net is a triple $N = (S, T, \iota)$ such that

1. S is a set (of places).
2. T is a set (of transitions) satisfying $S \cap T = \emptyset$.
3. ι is a mapping with domain $S \cup T \cup (S \times T) \cup (T \times S)$ as described above,

satisfying the additional requirements that N has at least one entry place and one exit place and that each transition t has at least one incoming and one outgoing arc.

In part (i), Figure 3 shows a simple TM-net N_1.

3.2 The Transition Rule

First, the transition rule for M-nets is described. Let N be an TM-net and M its underlying M-net, which is given by forgetting about N's time restrictions.

A *marking* m of an M-net M maps each place s to a finite multiset over α_s. Under the *standard initial marking* there is exactly one token on each entry place and no token on any other place.

[2] In examples, this set is given as the conjunction of its elements; in particular, the empty set of value expressions is represented by the value expression `true`.

[3] Since we only consider safe nets, arc inscriptions are in fact sets.

An *assignment* is a mapping $\beta : VAR \to VAL$ which associates a value $v \in \tau$ to each variable $a \in VAR$ of type τ. A transition t is called *enabled* under the marking m w.r.t. the assignment β iff $\beta(\alpha_t)$ evaluates to `true` and $\beta(\iota_{s,t}) \leq m(s)$ for each arc (s,t) entering t.

A pair (t,β), is called *firing event* of M. Firing events (t,β) and (t',β') are called *in conflict* ($(t,\beta) \bowtie (t',\beta')$) iff $\beta(\iota(s,t)) \cap \beta'(\iota(s,t')) \neq \emptyset$ for some place s. The set of all firing events which are enabled under the marking m, is denoted by $act(m)$.

If the *firing event* (t,β) is enabled in a marking m, then (t,β) may occur, yielding a new marking m' such that $m'(s) = m(s) - \beta(\iota_{s,t}) + \beta(\iota_{t,s})$ for all places s.

Now the transition rule for TM-nets is introduced. A *configuration* of N is a pair (m,n), consisting of a marking m and a mapping n which maps each firing event (t,β) of N to an element of $(\mathbf{N} \cup \{\$\})$. For each firing event (t,β), $n(t,\beta)$ may be seen as a clock showing the number of time units that passed since the enabling of (t,β), \$ indicating that the event is not enabled. The *initial configuration* (m_0,n_0) for N is given by the standard initial marking m_0 and by n_0, mapping each firing event which is enabled under m_0 to 0 and any other firing event to \$. Now we consider two types of steps, namely steps where time passes (which we call *time-steps*) and steps where a nonempty set of firing events occurs (which we call *occur-steps*):

1. *Time-steps*: A time-step may occur in a configuration (m,n), if no firing event is forced to occur, i.e. if, for any firing event (t,β), $n(t,\beta) < \beta(lft(t))$, whenever $n(t,\beta) \neq \$$. In this case, the successor configuration (m',n') is given by $m' = m$ and
$$n'(t,\beta) = \begin{cases} n(t,\beta)+1 & \text{if } n(t,\beta) \neq \$ \\ \$ & \text{otherwise} \end{cases}$$
2. *Occur-steps*: An occur-step is possible in a configuration (m,n), if some firing event may occur, i.e. if there is a firing event (t,β) such that $n(t,\beta) \in \mathbf{N}$ and $\beta(eft(t)) \leq n(t,\beta) \leq \beta(lft(t))$. In this case, the successor configuration (m',n') w.r.t. (m,n) and (t,β) is given by $m'(s) = m(s) - \beta(\iota_{s,t}) + \beta(\iota_{t,s})$ for all places s and
$$n'(t',\beta') = \begin{cases} \$ & \text{if } (t',\beta') \notin act(m') \\ 0 & \text{if } (t',\beta') \in act(m') \\ & \text{and } ((t',\beta') \bowtie (t,\beta) \text{ } or \text{ } (t',\beta') \notin act(m)) \\ n(t',\beta') & \text{otherwise} \end{cases}$$

E.g., in the TM-net N_1 of part (i) of Figure 3, transition t is enabled under the current marking w.r.t. the assignments β_1 given by $\beta_1(a) = 1$ and $\beta_1(b) = 2$ and β_2 given by $\beta_2(a) = 2$ and $\beta_2(b) = 1$, respectively. Event (t,β_1) may occur at time 1, event (t,β_2) may occur at time 1 or time 2. The possible behaviour of N_1 is described by its occurrence graph which is displayed in Figure 4. Here, the configurations are represented as tuples, whose components denote, respectively, $m(s)$, $m(s')$, $n(t,\beta_1)$, and $n(t,\beta_2)$.

Remark A TM-net N may be unfolded to a low level time Petri net.

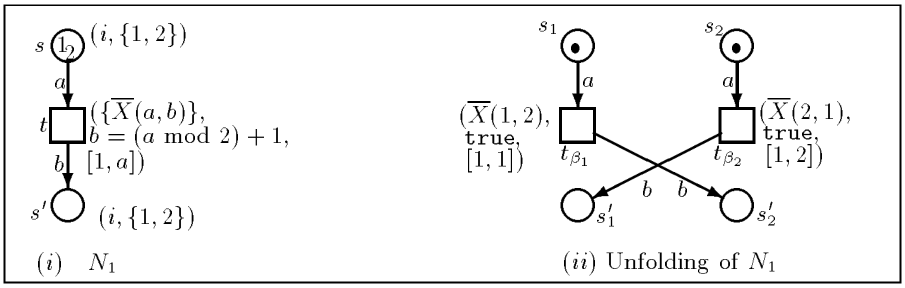

Fig. 3. A simple TM-net N_1 and its unfolding

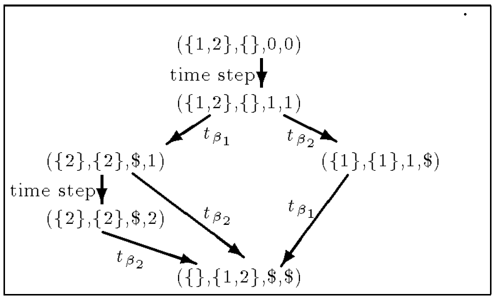

Fig. 4. Occurrence graph of N_1

3.3 Operations on TM-Nets

In this section we define composition operations on TM-nets corresponding to the operators of the PBC [5]. These create a new TM-net out of one, two, or three given TM-nets.

We consider the following *basic nets*:

1. $M(\iota) = \left(\overset{e}{\bigcirc} \xrightarrow{\{c\}} \overset{\iota}{\square} \xrightarrow{\{c\}} \overset{x}{\bigcirc} \right)$ with $\iota = (\lambda, \alpha, \chi)$, in particular
2. $M_{silent} = M(\emptyset, \mathtt{true}, [0, 0])$
3. $M_{stop} = \left(\overset{e}{\bigcirc} \;\; \overset{x}{\bigcirc} \right)$

TM-nets may be combined *vertically* by means of

- *parallel composition* $N_2 || N_3$: The nets are executed independently (cf. Figure 5, (iii)).
- *sequential composition* $N_2;N_3$: The nets are executed one after the other (cf. Figure 5, (iv)).
- *choice composition* $N_2 \,\square\, N_3$: One of the nets is nondeterministically chosen for execution (cf. Figure 5, (v)).
- *iteration* $[M_1 * M_2 * M_3]$: An (initial) net M_1 is executed once, followed by zero or more executions of the (iterated) net M_2, followed by one execution

of the (final) net M_3. The repetition of the iterated part is enabled by $M_4 = M_{silent}$ (cf. Figure 5, (vi)).

and also *horizontally* by means of

- *synchronisation*: The synchronisation N **sy** A of an M-net N with respect to an action symbol A is constructed by exhaustively applying basic synchronisation w.r.t. A. A transition t'' arises through a *basic synchronisation* w.r.t. A of two transitions t and t' iff
  1. $A(\tau_1, ..., \tau_n) \in \lambda_t$ and $\overline{A}(\tau'_1, ..., \tau'_n) \in \lambda_{t'}$.
  2. There exists a common most general unifier β of the pairs (τ_i, τ'_i), $1 \leq i \leq n$, such that
     (a) $\lambda_{t''} = \beta(\lambda_t) + \beta(\lambda_{t'}) \setminus \beta(\{A(\tau_1, ..., \tau_n), \overline{A}(\tau'_1, ..., \tau'_n)\})$,
     (b) $\alpha_{t''} = \beta(\alpha_t) \wedge \beta(\alpha_{t'})$,
     (c) $\chi_{t''} = [min(\beta(eft(t)), \beta(eft(t'))), max(\beta(lft(t)), \beta(lft(t')))]^4$,
     (d) $\iota(s, t'') = \beta(\iota(s,t)) + \beta(\iota(s,t'))$ and $\iota(t'', s) = \beta(\iota(t,s)) + \beta(\iota(t',s))$ for all $s \in S$.

  An example of a basic synchronisation step (which in this case already covers the **sy** -operator) is shown in Figure 5, (vii)).
- *restriction*: The restriction N **rs** A of an M-net N with respect to an action symbol A is N with all transitions t removed such that A or $\overline{A}$ occurred in λ_t (cf. Figure 5, $(viii)$).
- *scoping*: Scoping is synchronisation followed by restriction:
  $[A : N] = N$ **sy** A **rs** A.

Note that synchronisation and restriction – and, hence, scoping – are associative and commutative. Therefore, for an TM-net N, and for $ACT' \subseteq ACT$, N **sy** ACT', N **rs** ACT', and $[ACT' : N]$ are all well defined.

4 Translation of SDL/R-specifications into TM-Nets

An SDL/R-specification describes a system consisting of a set of sequential processes which may communicate asynchronously with each other. In this section, we define a semantic function which associates to each such system S a

TM-net $M(S)$, the *high level semantics* of S. $M(S)$ may then be unfolded to a (safe) time Petri net $U(M(S))$, the *low level semantics* of S.

The standard semantic model of SDL-specifications contains two types of infinite objects, namely channels and input queues of processes, which both are unbounded. This fact causes no problem as far as simulation is concerned, but, for the purpose of verification of properties of SDL-systems, one has to assume a bound for the capacity of channels and input queues. Often, an upper bound for these queues is explicit part of the problem specification. Otherwise, if such a bound exists at all, it may be determined using a model checker by checking

[4] For intervals $[eft(t), lft(t)]$ and $[eft(t'), lft(t')]$ with constant bounds this term denotes intersection.

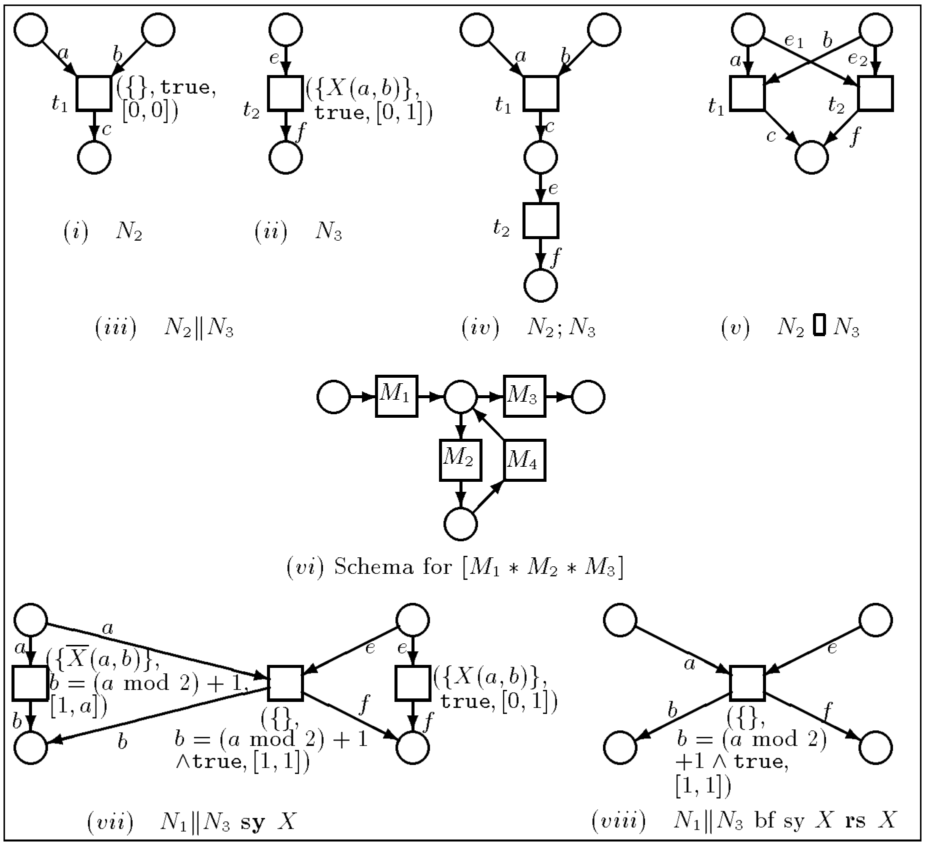

Fig. 5. Operations on TM-nets

with an increasing upper bound until the queue cannot be filled completely. In this paper, we assume a finite bound $k \in \mathbf{N}$ for channels and input queues. Unbounded versions of channels and input queues may be constructed similarly.

In the following, we illustrate the semantic function using the ARQ example. The formal definition is given in appendix A.

The high level semantics of a system S is given by the parallel composition of TM-nets $M_1, ..., M_n$ for the different processes and channels. To model the communication between processes, according to the semantic model of SDL (cf. [8]), each process has an implicit input queue. Like channel variables in $B(PN)^2$, these input queues are represented by TM-nets $I_1, ..., I_k$ (cf. [7]). The TM-net representing the whole system is then given by the parallel composition of $M_1, ..., M_n$ and $I_1, ..., I_k$, respectively, followed by synchronising and restricting the resulting net according to the action symbols occurring in the input queues and the channels.

The relevant parts of the semantic model of the ARQ system (cf. Figure 6)

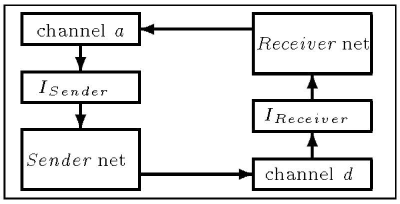

Fig. 6. Structure of the standard semantics of the ARQ-system

consist of the process *sender* together with its input queue I_{sender}, the process *receiver* together with its input queue $I_{receiver}$, and the channels a and d. These parts have to be synchronised and restricted w.r.t. the corresponding read and write actions of the channels ($A^!, A^?, D^!, D^?$) and of the input queues of the processes ($I^!_{sender}, I^?_{sender}, I^!_{receiver}, I^?_{receiver}$). Hence, the semantics for the whole ARQ system is described by the following equation:

$$M(ARQ) = [\{I^!_{sender}, I^?_{sender}, I^!_{receiver}, I^?_{receiver}, A^!, A^?, D^!, D^?\} : \\ M(sender)||M(receiver)||M(a)||M(d)]$$

The semantics of a channel is given by the net $M_{channel}(c, k, Set, [eft, lft])$ (also called *channel box*; cf. section 4.2), where c, k, Set, $[eft, lft]$ denote the name, the capacity, the type, and the time interval of the channel. This net is connected to the target process of the channel by a transition which in turn is embedded in an infinite loop, thus synchronising each read operation of the channel with a corresponding write operation to the input queue of the target process. E.g., for channel d this yields:

$$\begin{aligned} M(d) = {} & M_{channel}(d, Capacity(d), \{\mathsf{data.0}, \mathsf{data.1}\}, [5, 10])|| \\ & [M(\{D^?(s_1), I^!_{receiver}(s_2)\}, \{s_1 = s_2\}) \\ & *M(\{D^?(s_1), I^!_{receiver}(s_2)\}, \{s_1 = s_2\}) \\ & *M_{stop}] \end{aligned}$$

The TM-net of a single process P consists of

1. a part for the input queue of the process. This is given by the net $M_{input}(P)$, cf. section 4.3.
2. a part for each declaration `dcl` $v : Set$; of a local variable, given by a special TM-net $M_{data}(v, Set)$, called *data box*, cf. section 4.1.
3. a part for the set $States$ of states of the process. These are treated as a local variable q and are represented by the data box $M_{data}(q, States \cup \{\bullet\})$.
4. a part for each declaration of a timer t, which is given by a *timer box* $M_{timer}(P, t)$, cf. section 4.4.
5. a part for the control flow of the process, cf. section 4.5.

These parts are put in parallel and then synchronised and restricted according to the (action symbols for the) local variables, the timers, and the states of the process. Hence e.g. for the *sender* process we get the following:

$$
\begin{array}{lcr}
M(sender) = [\{X, Y, Q, T_{set}, T_{reset}, I^{rem}_{sender}\} & : & \\
M_{input}(sender, Capacity(I_{sender}), \{\mathsf{data.0}, \mathsf{data.1}\}) & \| & M_{data}(q, \{\mathsf{send}, \mathsf{wait}, \bullet\}) \\
 & \| & M_{data}(x, \{0, 1\}) \\
 & \| & M_{data}(y, \{0, 1\}) \\
 & \| & M_{timer}(sender, t) \\
 & \| & M(body(sender))]
\end{array}
$$

The semantics for the control part consists of the choice composition of the action sequences for each pair (q, in) of state (q) and input symbol (in) such that an SDL- transition is defined for (q, in). This is embedded in an infinite loop, e.g. for the body of the *sender* process, which consists of a start transition *start* and two transitions $transition_1$ and $transition_2$ we have:

$$
\begin{aligned}
M(body(sender)) = [&M(start) \\
&*(M(transition_1) \Box M(transition_2)) \\
&*M_{stop}]
\end{aligned}
$$

4.1 Semantics of Data Declarations

The semantics of the declaration of a local variable v with value set Set inside a process P is given by the *data box* $M_{data}(v, Set)$ as shown in Figure 7. The first access to v is done by firing of transition T_1, which has to synchronise with an appropriate transition within the control part of process P, as is indicated by the action label of T_1. All further access to v is handled by transition T_2 in the same manner. The actual value of the variable is held in the central place S_2. Place S_3 is only needed for compositions of the data box.

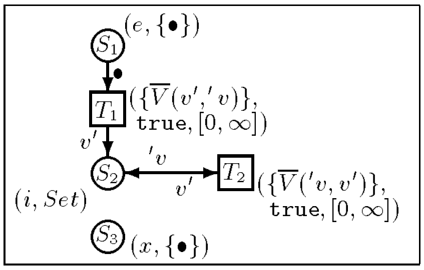

Fig. 7. TM-net $M_{data}(v, Set)$ for the declaration of variable v with value set Set

4.2 Semantics of Channel Declarations

In SDL/R, a channel c is used as a unidirectional connection from a process P to a process Q. Signals of a certain type Set, which are written to c by P, are delivered after a time delay which ranges from the minimum *eft* to the maximum *lft*, to the input queue of Q. Thereby, the arrival order of the signals is retained.

The TM-net semantics of a channel c of capacity k is depicted in Figure 8[5].

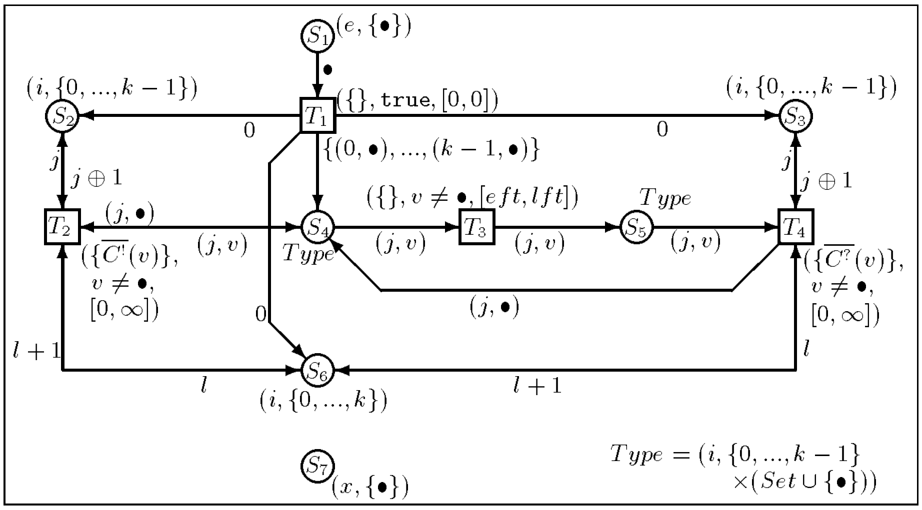

Fig. 8. TM-net $M_{channel}(c,k,Set,[eft,lft])$ for the delayed channel

By firing of transition T_2, a signal v may be written to place S_4 – which contains the undelayed signals – if the current write position j (contents of place S_2) is empty (i.e., the pair $(j,\bullet)$ is on S_4) and the channel contains at most $(k-1)$ entries (i.e., the contents l of place S_6 satisfies $l<k$.

Upon firing of T_2, the current write position (S_2) and the current number of entries (S_6) are updated. T_2 has to synchronise with an appropriate transition of the control flow as is indicated by its action label. The time behaviour of T_2 is not restricted, but T_2 will 'inherit' some time constraints through synchronisation.

For each entry (j,v) of the channel transition T_3 nondeterministically chooses a delay time from the interval $[eft,lft]$ and puts the delayed entry on place S_5[6].

Signals may be read from the channel by firing of transition T_4 which is enabled if at the current read position j (contents of place S_3) there is an entry $v\neq\bullet$. As is indicated by its action label, also T_4 has to synchronise with an appropriate transition of the control part. Upon firing, T_4 removes the entry

[5] To enhance readability, here and in the following pictures, as abbreviation expressions are allowed as arc inscriptions. Also, arcs pointing in both directions are used. Inscriptions of these are placed next to the source node of that direction of the arc to which they refer.

[6] The inscription of transition T_3 is somewhat simplified. In fact the action part should contain the action terms for all variables occurring in *eft* and *lft*, respectively. Using the denotations of appendix A, the inscription would read $\iota_{T_3}=(AS,v\neq\bullet,[E',E''])$, where $AS=AS'\cup AS''$, $\zeta(eft)=(AS'/E')$, and $\zeta(lft)=(AS''/E'')$, respectively.

(j, v) from P_5, adds $(j, \bullet)$ to S_4, and also updates the read index S_5 and the current number of entries S_6.

Transition T_1 initialises write position (S_2), read position (S_3), and current number of entries (S_6) with 0. Place S_7 is only needed for compositions of the channel box.

4.3 Semantics of Input Queues

As pointed out above, the standard semantic model of SDL assigns an implicit input queue to each process P, which takes up all signals sent to P. The semantics of an input queue is given by the TM-net $M_{input}(P, k, Set)$, as shown in figure 9, where Set denotes the union of all types of signals which may be sent to P by any other process. The net for the input queue is very similar to the net for a channel. It may be generated by first collapsing S_4, T_3, and S_5 of the channel, yielding the place S_8 and then adding a new transition, T_5. The purpose of T_5 is to remove timer signals, that have been invalidated, from the input queue. To do so, it has to synchronise with an appropriate transition in a timer box.

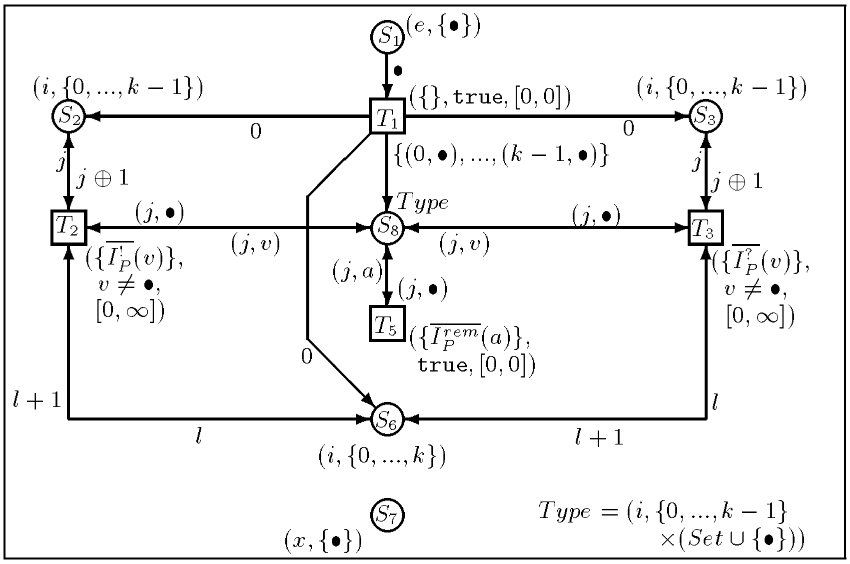

Fig. 9. TM-net $M_{input}(P, k, Set)$ for the input queue of process P

4.4 Semantics of Timers

The semantics for the declaration of timer t within process P is given by the TM-net $M_{timer}(P, t)$. It contains two central places: S_2 to hold the expiry time for t when a `set`-command was executed and S_3 to remember whether a timer signal for t was inserted in the input queue of the process P. Depending on the contents of S_3, there are two transitions for `set` (T_3, T_4) and for `reset` (T_5, T_6), respectively. If t is set with expiry time a, a is put on place S_2, thereby enabling transition T_2, which - after a time units, inserts a t-signal in the input queue of P, unless it looses concession in the meantime. If there was already a t-signal in the input queue (i.e., S_3 was marked with a 1), it has to be removed first. This is done through the additional action label $I_P^{rem}(t)$ which causes a synchronisation with transition T_5 of the input box. If t is reset, the expiry time is removed from S_2 (if there is any). In addition, a timer signal is removed from the input queue (if there is any).

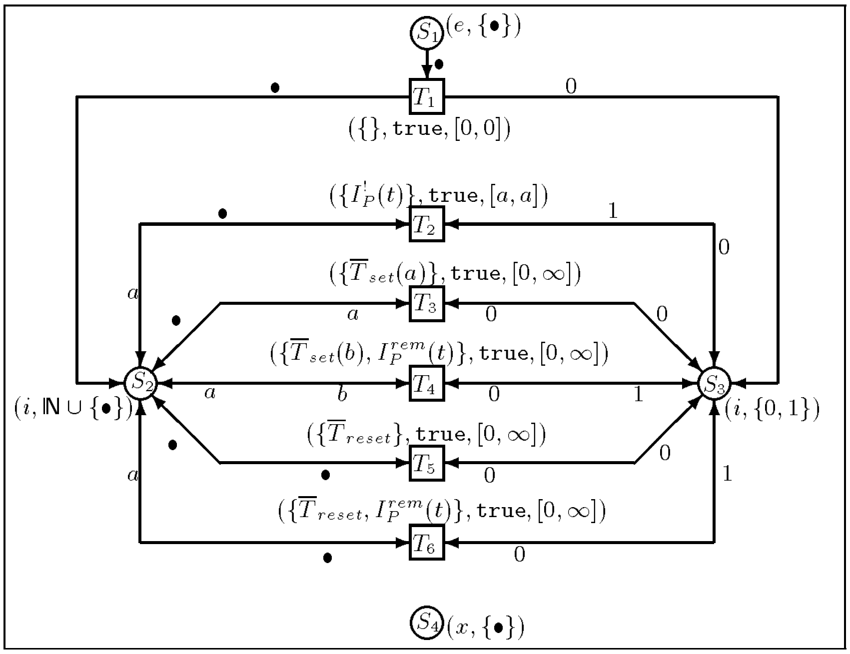

Fig. 10. TM-net $M_{timer}(P, t)$ for declaration of timer t in process P

4.5 Semantics of State Transitions

We will not describe the compositional derivation of the semantics of a state transition here, but merely show the semantics of one transition of the *sender* process as an example in Figure 11. The state transition consists of four parts which are composed sequentially: Consuming of input `none` in state *send* (which consumes 1 time unit), setting timer t to react after 30 time units (which also consumes 1 time unit), output of $data(x)$ via channel d (which consumes 2 time units), and entering of state *wait* (without consuming time). These parts are represented in the semantic model by four transitions which are also connected sequentially. The first makes the current state be undefined (represented by

$q' = \bullet$) and has to synchronise with the data box for the states. The second transition, which has to synchronise with a "`set` -transition" of the timer box for t causes delivery of the delay time on place S_2 of the timer box. By synchronising with the data box for the local variable x and with the write transition of the channel box for channel d, the third transition inserts the pair $m = (data, x)$ into the channel d. Finally, the last transition, which has to synchronise with the data box for the states, changes the state to *wait*. All timing requirements of the SDL-actions are just passed to the according transitions.

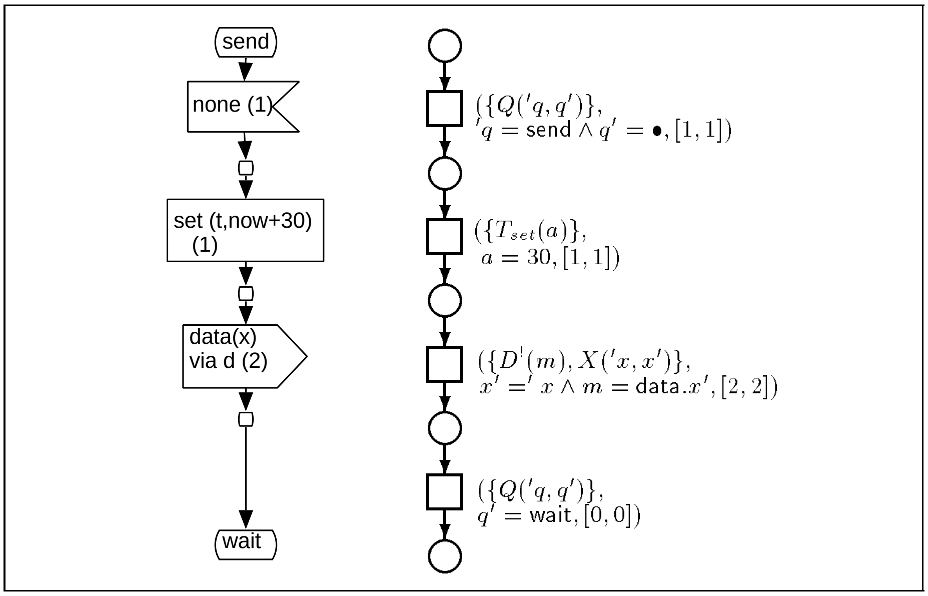

Fig. 11. Semantics of an SDL-transition

5 Conclusion

In this paper, we have extended SDL by real-time requirements to be able to deal also with quantitative temporal aspects of distributed systems. A compositional denotational semantics for the resulting language SDL/R has been defined, which allows to analyse SDL/R-specifications using methods from time Petri nets.

Currently, the real-time extension is being implemented in the MOBY-tool. The next step will consist in expanding the approach to the verification of qualitative and quantitative temporal properties.

For the verification of qualitative temporal properties of SDL/R-systems, a translation of the low level semantics into equivalent (untimed) Petri boxes will be implemented. This will enable us to use the model checker for μSDL-specifications of the MOBY-tool also for SDL/R-specifications.

To deal with quantitative temporal properties, first a real-time extension of the temporal logic for SDL has to be developed. We will then extend Esparza's partial order model checking algorithm to cope also with this extension. As an alternative, we will apply other verification methods. E.g., we will translate the low level semantics for SDL/R-systems as well as formulae of the (extended) temporal logic for SDL to formulae of Monadic Second Order Logic (M2L) and then use a proof-tool for M2L like the MONA-tool (cf. [12]).

We also plan to extend our approach to complete SDL. E.g., we have defined an M-net semantics for procedures and dynamic process creation and termination in SDL [10].

Acknowledgement

The authors would like to thank the members of the Oldenburg semantics group for helpful comments on the paper.

References

1. P. Amthor, H. Fleischhack, and J. Tapken: *MOBY – more than a Tool for the Verification of SDL-Specifications*, Research Paper, Universität Oldenburg, 1996.
2. F. Bause and P. Buchholz, Protocol Analysis Using a Timed Version of SDL, in J. Quemada, J. Mañas, and E. Vazquez (Eds.), *Formal Description Techniques, III*, pp. 269-285, North-Holland, 1991.
3. B. Berthomieu and M. Diaz, Modelling and Verification of Time Dependent Systems Using Time Petri Nets, *IEEE Transactions on Software Engineering*, Volume 17/3, pages 259 – 273 (1991).
4. E. Best, *Partial Order Verification with PEP*, Report HIB 26/96, Universität Hildesheim, 1996.
5. E. Best, R. Devillers, and J. G. Hall. The box calculus: a New Causal Algebra With Multi-Label Communication. In G. Rozenberg, (Ed.), *Advances in Petri Nets 92*, Volume 609 of *Lecture Notes in Computer Science*, pages 21 – 69. Springer, 1992.
6. E. Best, H. Fleischhack, W. Fraczak, R. P. Hopkins, H. Klaudel, and E. Pelz. A Class of Composable High Level Petri Nets. In G. De Michelis and M. Diaz, (Eds.), *Application and Theory of Petri Nets 1995*, Volume 935 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 1995.
7. E. Best, H. Fleischhack, W. Fraczak, R. P. , H. Klaudel, and E. Pelz. An M-net Semantics of $B(PN)^2$. In J. Desel, (Ed.), *Structures in Concurrency Theory*, Workshops in Computing, Springer, 1995.
8. CCITT. *Specification and Description Language*, CCITT Z.100, International Consultative Committee on Telegraphy and Telephony, Geneva, 1992.
9. J. Esparza. *Model Checking Using Net Unfoldings*, Science of Computer Programming, Volume 23, pages 151–195, Elsevier, 1994.
10. H. Fleischhack, and B. Grahlmann, *Towards Compositional Verification of SDL Systems*, Research Paper, 1997. (submitted)

11. B. Grahlmann and E. Best: PEP – More Than a Petri Net Tool. In T. Margaria and B. Steffen, (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems*, Volume 1055 of *Lecture Notes in Computer Science*, pages 397–401, Springer, 1996.
12. J. Henriksen e.a., MONA: Monadic Second-Order Logic in Practice, in Brinksma, Cleaveland, Larsen, Margaria, and Steffen, (Eds.) *TACAS '95, Tools and Algorithms for The Construction and Analysis of Systems*, Volume 1019 of *Lecture Notes in Computer Science*, pages 89-110, Springer, 1995.
13. S. Leue, *Specifying Real-Time Requirements for SDL Specifications – a Temporal Logic-Based Approach*, Research Paper, University of Waterloo, Canada, 1996.
14. P. Merlin and D. Farber, Recoverability of Communication Protocols – Implication of a Theoretical Study, *IEEE Transactions on Software Communications* Vol. 24, 1036 – 1043 (1976).
15. L. Popova: On Time Petri Nets. *Journal of Information Processing and Cybernetics*, Volume 1055 of *Lecture Notes in Computer Science*, Springer, 1991.

A Appendix

Syntax of SDL/R:

$$
\begin{array}{ll}
< systemdef > & ::= \textbf{system}\ syst\ \textbf{;}\ < signaldef >^* < channeldef >^* \\
 & \quad < processdef >^+ \ \textbf{endsystem;} \\
< signaldef > & ::= \textbf{signal}\ sign\ [\ (\ < sortlist >\ )\ ]\ \textbf{;} \\
< sortlist > & ::= < sort >\ \textbf{,}\ < sortlist > \mid < sort > \\
< sort > & ::= Set \\
< channeldef > & ::= \textbf{channel} < timeint >\ chan\ \textbf{from}\ proc1\ \textbf{to}\ proc2 \\
 & \quad \textbf{with} < signallist >\ \textbf{;} \\
< signallist > & ::= sign\ \textbf{,}\ < signallist > \mid sign \\
< processdef > & ::= \textbf{process}\ proc\ \textbf{;}\ < vardef >^* < timerdef >^* < processbody > \\
 & \quad \textbf{endprocess;} \\
< vardef > & ::= \textbf{dcl}\ w < sort >\ \textbf{;} \\
< timerdef > & ::= \textbf{timer}\ t\ \textbf{;} \\
< processbody > & ::= \textbf{start}\ \textbf{;}\ < transition > < state >^+ \\
< state > & ::= \textbf{state}\ st\ \textbf{;}\ < input >^+ \\
< input > & ::= \textbf{input}\ \{\ sign\ [\ (\ < varlist >\ )\ ] < timeint > \mid \textbf{none}\ \}\ \textbf{;} \\
 & \quad < transition > \\
< varlist > & ::= w\ \textbf{,}\ < varlist > \mid w \\
< transition > & ::= < transaction > < timeint >\ \textbf{;}\ < transition > \\
 & \quad \mid < decision > \mid \textbf{nextstate}\ st\ \textbf{;} \\
< transaction > & ::= \textbf{task}\ v\ \textbf{:=}\ < expr > \mid < set > \mid < reset > \\
 & \quad \mid \textbf{output}\ sign\ [\ (\ < exprlist >\ )\ ]\ \textbf{via}\ chan \\
< set > & ::= \textbf{set}\ (t, \mathtt{now} + < expr >) \\
< reset > & ::= \textbf{reset}\ (t) \\
< exprlist > & ::= < expr >\ \textbf{,}\ < exprlist > \mid < expr > \\
< expr > & ::= w \mid const \mid < expr >\ op < expr > \mid op < expr > \\
< decision > & ::= \textbf{decision}\ w < timeint >\ \textbf{;}\ < answer >^+ \ \textbf{enddecision;} \\
< answer > & ::= (\ < expr >\ )\ \textbf{:}\ < transition > \\
< timeint > & ::= [< expr >, < expr >]
\end{array}
$$

Auxiliary Semantic Functions:

$Proc$	: set of all process names in the system.
$Chan$	: set of all channel names in the system.
$InputChan(p)$	: set of all names of channels with receiver p.
$State(p)$	: set of all state names in p, without '**input none**'.
$StateNone(p)$	: set of all state names in p, with '**input none**'.
$InputSignal(p, s)$	: set of all input signals (without parameter), which p may consume in state s.
$Signal(c)$	: set of all (parametrised) signals of channel c.
$Capacity(B)$	: capacity for the buffer B (channel or input queue).

Semantic Rules:

$$\begin{aligned}&\mathcal{M}(\textbf{system}\, syst;\, < signaldef >^{*} < channeldef >^{*} < processdef >^{+}\ \textbf{endsystem};) = \\ &\qquad [\bigcup_{p \in Proc} \{I_p^{!}, I_p^{?}\} \cup \bigcup_{c \in Chan} \{C^{!}, C^{?}\} : \mathcal{M}(< channeldef >^{*} < processdef >^{+})]\end{aligned} \tag{1}$$

$$\mathcal{M}(< channeldef >^{+} < processdef >^{+}) = \mathcal{M}(< channeldef >^{+}) \,||\, \mathcal{M}(< processdef >^{+}) \tag{2}$$

$$\mathcal{M}(< channeldef_1 >< channeldef >^{+}) = \mathcal{M}(< channeldef_1 >) \,||\, \mathcal{M}(< channeldef >^{+}) \tag{3}$$

$$\mathcal{M}(< processdef_1 >< processdef >^{+}) = \mathcal{M}(< processdef_1 >) \,||\, \mathcal{M}(< processdef >^{+}) \tag{4}$$

$$\begin{aligned}&\mathcal{M}(\textbf{channel}\ < timeint >\ c\ \textbf{from}\ p1\ \textbf{to}\ p2\ \textbf{with} < signallist >) = \\ &\qquad M_{channel}(c, Capacity(c), Signal(c), < timeint >) \\ &\qquad || \,[M(\{C^{?}(s1), I_{p2}^{!}(s2)\}, \{s1 = s2\}) * M(\{C^{?}(s1), I_{p2}^{!}(s2)\}, \{s1 = s2\}) * M_{stop}]\end{aligned} \tag{5}$$

$$\begin{aligned}&\mathcal{M}(\textbf{process}\ p\ \textbf{;} < vardef >^{+} < timerdef >^{*} < processbody >\ \textbf{endprocess;}) = \\ &\qquad [\,\delta(< vardef >^{+}) : \mathcal{M}(< vardef >^{+}) \\ &\qquad || \,\mathcal{M}(\textbf{process}\ p\ \textbf{;} < timerdef >^{*} < processbody >\ \textbf{endprocess;})\,]\end{aligned} \tag{6}$$

with $\delta(< vardef_1 >< vardef >^{+}) = \delta(< vardef_1 >) \cup \delta(< vardef >^{+})$

and $\delta(\textbf{dcl}\ w\ Set\textbf{;}) = \{W\}$

$$\begin{aligned}&\mathcal{M}(\textbf{process}\ p\ \textbf{;} < timerdef >^{+} < processbody >\ \textbf{endprocess;}) = \\ &\qquad [\,\delta(< timerdef >^{+}) : \mathcal{M}^{p}(< timerdef >^{+}) \\ &\qquad || \,\mathcal{M}(\textbf{process}\ p\ \textbf{;} < processbody >\ \textbf{endprocess;})\,]\end{aligned} \tag{7}$$

with $\delta(< timerdef_1 >< timerdef >^{+}) = \delta(< timerdef_1 >) \cup \delta(< timerdef >^{+})$

and $\delta(\textbf{timer}\ t\textbf{;}) = \{T_{set}, T_{reset}\}$

$$\mathcal{M}(< vardef_1 >< vardef >^{+}) = \mathcal{M}(< vardef_1 >) \,||\, \mathcal{M}(< vardef >^{+}) \tag{8}$$

$$\mathcal{M}(\textbf{dcl}\ w\ Set\textbf{;}) = M_{data}(w, Set) \tag{9}$$

$$\mathcal{M}^{p}(< timerdef_1 >< timerdef >^{+}) = \mathcal{M}^{p}(< timerdef_1 >) \,||\, \mathcal{M}^{p}(< timerdef >^{+}) \tag{10}$$

$$\mathcal{M}^{p}(\textbf{timer}\ t\textbf{;}) = M_{timer}(p, t) \tag{11}$$

$$\begin{aligned}&\mathcal{M}(\textbf{process}\ p\ \textbf{;} < processbody >\ \textbf{endprocess;}) = \\ &\qquad [\{Q, I_p^{rem}\} : M_{input}(\,I_p,\ Capacity(I_p), \bigcup_{c \in InputChan(p)} Signal(c)\,) \\ &\qquad\qquad || \,\mathcal{M}^{p}(< processbody >) \,||\, M_{data}(q, State(p) \cup \{\bullet\})\,]\end{aligned} \tag{12}$$

$$\begin{aligned}&\mathcal{M}^{p}(\textbf{start;} < transition >< state >^{+}) = [\mathcal{M}^{p}(< transition >) * \\ &\qquad \mathcal{M}^{p}(< state >^{+}) \,\square\, M(\,\{Q('q, q'), I_p^{?}(m)\}, \{'q = q' \wedge \bigwedge_{s \in StateNone(p)} ('q \neq s) \\ &\qquad\quad \wedge \bigwedge_{s \in State(p)} ('q = s \Rightarrow \bigwedge_{i \in InputSignal(p,s)} m[0] \neq i)\}, [0, 0]) \,*\, M_{stop}]\end{aligned} \tag{13}$$

$$\mathcal{M}^p(< state_1 >< state >^+) = \mathcal{M}^p(< state_1 >) \,\square\, \mathcal{M}^p(< state >^+) \tag{14}$$

$$\mathcal{M}^p(\textbf{state } st\textbf{;} < input_1 >< input >^+) = \mathcal{M}^p(\textbf{state } st\textbf{;} < input_1 > \ \textbf{state } st\textbf{;} < input >^+) \tag{15}$$

$$\begin{aligned}&\mathcal{M}^p(\textbf{state } st\textbf{; input none; [} < expr_1 >, < expr_2 > \textbf{];} < transition >) = \\ &\qquad M(\{Q('q, q')\} \cup AS_1 \cup AS_2, \{'q = st \wedge q' = \bullet\}, [E_1, E_2]); \mathcal{M}^p(< transition >)\end{aligned} \tag{16}$$

$$\begin{aligned}&\mathcal{M}^p(\textbf{state } st\textbf{; input } s\textbf{(}v_1, \ldots, v_n\textbf{)} \textbf{ [} < expr_1 >, < expr_2 > \textbf{];} < transition >) = \\ &\qquad M(\{Q('q, q'), I_p^?(m), V_1('v_1, v_1'), \ldots, V_n('v_n, v_n')\} \cup AS_1 \cup AS_2, \\ &\qquad\quad \{'q = st \wedge q' = \bullet \wedge m[0] = s \wedge len(m) = n \wedge \bigwedge_{i=1}^{n} m[i] = v_i'\}, [E_1, E_2]); \\ &\qquad \mathcal{M}^p(< transition >) \qquad , \text{for } n \geq 0\end{aligned} \tag{17}$$

and $\zeta(< expr_j >) = (AS_j/E_j), j = 1, 2$ with

$$\begin{aligned}\zeta(w) &= (\{W('w, w')\}/'w) \\ \zeta(const) &= (\emptyset/const) \\ \zeta(< expr_1 > op < expr_2 >) &= (AS_1 \cup AS_2/E_1 \; op \; E_2) \\ &\qquad \text{if } \zeta(< expr_i >) = (AS_i/E_i) \text{ , } i\text{=1,2} \\ \zeta(op < expr >) &= (AS/op\, E) \qquad \text{if } \zeta(< expr >) = (AS/E)\end{aligned}$$

$$\mathcal{M}^p(< transaction > \textbf{;} < transition >) = \mathcal{M}^p(< transaction >); \mathcal{M}^p(< transition >) \tag{18}$$

$$\begin{aligned}&\mathcal{M}^p(\textbf{task } v := < expr_1 > \textbf{[} < expr_2 >, < expr_3 > \textbf{]}) = M(AS \cup \{V('v, v')\}, \\ &\qquad \{v' = E_1\} \cup \{'w = w' \mid w \neq v \text{ and } w \text{ in } < expr_i > i = 1, 2 \text{ or } 3\}, [E_2, E_3])\end{aligned} \tag{19}$$

if $\zeta(< expr_j >) = (AS_j/E_j), 1 \leq j \leq 3$, and $AS = \bigcup_{j=1}^{3} AS_j$

$$\begin{aligned}&\mathcal{M}^p(\textbf{output } s\textbf{(} < expr_1 >, \ldots, < expr_n > \textbf{) via } c \textbf{ [} < expr_{n+1} >, < expr_{n+2} > \textbf{]}) = \\ &\qquad M(\{C^!(m)\} \cup \bigcup_{i=1}^{n+2} AS_i, \{m[0] = s \wedge len(m) = n \wedge \bigwedge_{i=1}^{n} m[i] = E_i\}, [E_{n+1}, E_{n+2}])\end{aligned} \tag{20}$$

with $n \geq 0$ and $\zeta(< expr_i >) = (AS_i/E_i)$ for all $i = 1, \ldots, n+2$.

$$\begin{aligned}&\mathcal{M}^p(\textbf{set } (t, \texttt{now} + < expr_1 >), \textbf{[} < expr_2 >, < expr_3 > \textbf{]}) = \\ &\qquad M(\{T_{set}(b)\} \cup AS, \{b = E_1\}, [E_2, E_3])\end{aligned} \tag{21}$$

if $\zeta(< expr_i >) = (AS_i/E_i), 1 \leq i \leq 3$, and $AS = \bigcup_{i=1}^{3} AS_i$.

$$\mathcal{M}^p(\textbf{reset } (t), \textbf{[} < expr_1 >, < expr_2 > \textbf{]}) = M(\{T_{reset}\} \cup AS, \{\}, [E_1, E_2]) \tag{22}$$

if $\zeta(< expr_i >) = (AS_i/E_i), 1 \leq i \leq 2$, and $AS = AS_1 \cup AS_2$.

$$\begin{aligned}&\mathcal{M}^p(\textbf{decision } w < timeint > \textbf{;} < answer >^+ \textbf{ enddecision;}) = \\ &\qquad \mathcal{M}^p_{< timeint >}(< answer >^+)\end{aligned} \tag{23}$$

$$\begin{aligned}&\mathcal{M}^p_{< timeint >}(< answer_1 >< answer >^+) = \\ &\qquad \mathcal{M}^p_{< timeint >}(< answer_1 >) \,\square\, \mathcal{M}^p_{< timeint >}(< answer >^+)\end{aligned} \tag{24}$$

$$\begin{aligned}&\mathcal{M}^p_{< timeint >}((< expr >) :< transition >) = \\ &\qquad \mathcal{M}^p(< expr >< timeint >); \mathcal{M}^p(< transition >)\end{aligned} \tag{25}$$

$$\begin{aligned}&\mathcal{M}^p(< expr_1 > \textbf{[} < expr_2 >, < expr_3 > \textbf{]}) = \\ &\qquad M(AS, E_1 \cup \{'w = w' \mid w \text{ in } < expr_i > i = 1, 2 \text{ or } 3\}, [E_2, E_3])\end{aligned} \tag{26}$$

if $\zeta(< expr_j >) = (AS_j/E_j), 1 \leq j \leq 3$, and $AS = \bigcup_{j=1}^{3} AS_j$

$$\mathcal{M}^p(\textbf{nextstate } st\textbf{;}) = M(\{Q('q, q')\}, \{q' = st\}, [0, 0]) \tag{27}$$

This article was processed using the LaTeX macro package with LLNCS style