MPA: a Stochastic Process Algebra

Marco Bernardo  Lorenzo Donatiello  Roberto Gorrieri

Technical Report UBLCS-94-10
May 1994

Laboratory for Computer Science
University of Bologna
Piazza di Porta S. Donato, 5
40127 Bologna (Italy)
The University of Bologna Laboratory for Computer Science Research Technical Reports are available via anonymous FTP from the area ftp.cs.unibo.it:/pub/TR/UBLCS in compressed PostScript format. Abstracts are available from the same host in the directory /pub/TR/UBLCS/ABSTRACTS in plain text format. All local authors can be reached via e-mail at the address last-name@cs.unibo.it.

UBLCS Technical Report Series

93-7 Distributed Programming with Logic Tuple Spaces, P. Ciancarini, April 1993.
93-8 Coordinating Rule-Based Software Processes with ESP, P. Ciancarini, April 1993.
93-14 Guard Compilation in Logic Shared Dataspace Languages, M. Gaspari, June 1993.
94-6 Exploring the Coordination Space with LO, S. Castellani, P. Ciancarini, April 1994.
94-7 Comparative Semantics of LO, S. Castellani, P. Ciancarini, April 1994.
MPA: a Stochastic Process Algebra

Marco Bernardo 1  Lorenzo Donatiello 1  Roberto Gorrieri 1

Technical Report UBLCS-94-10

May 1994

Abstract

Process algebras constitute one of the main tools for modeling and analyzing concurrent systems. Unfortunately, with classical process algebras only the functional aspect of the behavior of concurrent systems can be modeled and analyzed. Since the relevance of integrating the performance modeling and evaluation into the process of specification, design and implementation of concurrent systems has been widely recognized, an effort has been made in the field of the process algebras in order to handle also the temporal aspect of the behavior of concurrent systems.

In this technical report the stochastic process algebra MPA (Markovian Process Algebra) is defined together with its operational interleaving semantics and its markovian semantics. The operational interleaving semantics is defined by following the structured operational semantics approach augmented with two transformations. The markovian semantics is defined through an algorithm which transforms the automata representing the operational interleaving semantics of MPA terms into state transition rate diagrams of homogeneous continuous time Markov chains. Some examples are shown which demonstrate the expressiveness and the compositionality of MPA and the correctness of its semantics. Finally, MPA is compared with other stochastic process algebras already appeared in the literature.

1. Dipartimento di Matematica, Università di Bologna, Piazza di Porta S. Donato 5, 40127 Bologna, Italy.
1 Introduction

Since the 60's a lot of work has been done in order to devise modeling techniques which are adequate to represent concurrent systems, i.e. systems which exhibit parallelism and synchronization between their components. Process algebras are one of the most important results of this work and represent a very useful tool for modeling and analyzing concurrent systems. As a matter of fact,

- Languages are the main tool for designing hardware and software components, and process algebras are nothing else but abstract languages conceived for defining formally concurrent systems.
- With process algebras concurrent systems can be described in a compositional way, i.e. they provide the designer with a restricted but powerful set of operators by means of which constructing more and more complex systems from simpler ones; as a consequence, the designer is led to use a hierarchical and modular design style.
- With process algebras it is possible to formally express the intended notion of equivalence among concurrent systems by means of adequate equivalence relations defined over the process algebra terms. Sometimes it is moreover possible to axiomatize in a sound and complete way these relations when they are congruences w.r.t. all (or some) of the operators. Hence, it becomes feasible to mechanically reason about the terms in order to analyze the concurrent systems they describe.
- With process algebras concurrent systems can be given formal descriptions which are more easily understandable and more easily modifiable than the ones obtained by many other models.

Classical process algebras, like CCS (Calculus of Communicating Systems) proposed by R. Milner [16] and CSP (Communicating Sequential Processes) proposed by C. A. R. Hoare [13], cannot be used to evaluate the performance of concurrent systems because they allow to describe only the functional aspect of the behavior of concurrent systems. Neglecting the temporal aspect of the behavior of concurrent systems is a remarkable drawback for the expressiveness of the classical process algebras because:

- Firstly, it is not possible to adequately model concurrent systems which require the specification of temporal parameters (e.g. real time systems and time-dependent communication protocols).
- Secondly, it is not possible to evaluate the performance of the concurrent systems being modeled. It often happens that a concurrent system is firstly fully designed and tested for functionality and afterwards tested for efficiency; so, if the performance is detected to be poor, the concurrent system has to be redesigned with negative consequences for both the design costs and the delivery at a fixed deadline.

The relevance of integrating the analysis of the performance of a concurrent system into the design process of the system itself has recently stimulated many researchers. In the framework of process algebras, this need is satisfied by introducing the concept of time in order to express the durations of the activities performed by the concurrent systems being modeled; when such durations are expressed in a deterministic way, the process algebra is said to be a temporal process algebra, whereas when such durations are expressed through random variables, the process algebra is said to be a stochastic process algebra.

Among the temporal process algebras present in the literature we quote Temporal CCS [22][4][17][18], Timed CSP [20][21][5] and other process algebras in which the cost of executions is formally defined after associating a deterministic duration with each action [6][8][9].
In this work we shall be more concerned with stochastic process algebras; among the stochastic process algebras present in the literature we quote:

- **Timed Processes and Performance Evaluation (TIPP)** proposed by N. Götz-U. Herzog-M. Rettelbach [10][11]. It is an extension of a mixture of CCS and CSP in which actions are classified as passive, immediate or timed where the durations of timed actions are expressed through random variables, and in which there is a probabilistic alternative composition operator. In particular, **Markovian TIPP (MTIPP)** is an interesting sublanguage of TIPP in which all the timed actions of MTIPP have exponentially distributed durations so that it is possible to associate a *homogeneous continuous time Markov chain* (HCTMC) with each MTIPP term.

- **Performance Evaluation Process Algebra (PEPA)** proposed by J. Hillston [12][7]. It is an extension of a mixture of CCS and CSP in which actions are classified as passive or timed where the durations of timed actions are expressed through exponentially distributed random variables so that it is possible to associate a HCTMC with each PEPA term.

The stochastic process algebra we shall introduce in the following is called **Markovian Process Algebra (MPA)**. Its name stems from the fact that the random variables expressing the durations of timed actions are exponentially distributed, hence it is possible to resort to the study of HCTMCs in order to evaluate the performance of concurrent systems modeled with MPA. Obviously, the above mentioned stochastic process algebras MTIPP and PEPA have acted as a starting point during the development of MPA. In this technical report, which is the first document in which MPA is presented, we shall define the syntax, the operational interleaving semantics and the markovian semantics of MPA terms; we shall also present several examples and finally we shall compare MPA with both MTIPP and PEPA.

This technical report is organized as follows. In Section 2 the syntax and the informal semantics of MPA are introduced. In Section 3 the structured operational semantics rules for MPA are defined. In Section 4 the operational interleaving semantics of MPA terms is defined by showing two transformations to be applied to the automata directly obtainable by the structured operational semantics rules. In Section 5 the markovian semantics of MPA terms is defined by showing an algorithm which associates a HCTMC with each MPA term. In Section 6 some concluding remarks are reported.

2 Syntax and informal semantics of MPA

In this section we shall introduce the syntax of MPA terms and we shall informally describe the semantics of MPA operators.

This section is organized as follows. In Section 2.1 actions are introduced and classified together with relabeling functions. In Section 2.2 constants are introduced. In Section 2.3 the syntax of MPA terms is defined. In Section 2.4 the informal semantics of MPA operators is presented. In Section 2.5 the property of guarded closure is defined.

2. For a more detailed presentation of MPA, the reader is referred to the master thesis of M. Bernardo [1]. Furthermore, the authors have written a technical report in which the markovian semantics of MPA is checked for correctness by using a benchmark involving the description of queueing systems [2] and a technical report in which an operational net semantics is defined for MPA and an integrated modeling technique for concurrent systems is considered [3].
2.1 Actions and relabeling functions

The first step in the definition of MPA consists of introducing a set of actions which model the activities performed by concurrent systems.

To better understand the meaning of the actions, we make the following basic assumptions concerning MPA:

- A system is a network of processes which execute actions.
- Processes run in a message passing scenario by means of a synchronous or handshake communication mechanism.
- Each action performed by a process is either an interaction with other processes (communication) or it takes place independently of the actions being executed by the other processes (parallelism); because of the adoption of a synchronous communication mechanism, interactions between processes are nothing else but synchronizations between processes.

Each MPA action is described by a pair $<a, \lambda>$ consisting of the type of the action and the rate of the action.

Depending on the type, MPA actions are subdivided into:

- **External or observable actions**, i.e. actions whose execution can be seen by an external observer; we shall denote with $Com$ the set of external action types.
- **Internal or invisible actions**, i.e. actions whose execution cannot be seen by an external observer; we shall denote with $\tau$ the only internal action type we shall use.

Depending on the rate, MPA actions are subdivided into:

- **Passive actions**, i.e. actions whose execution rate is zero. The duration of a passive action is undefined and is fixed only upon a synchronization with a nonpassive action of the same type. Passive actions are used for modeling waitings.
- **Active actions**, i.e. actions whose execution rate is nonzero. Active actions are themselves subdivided into:
  - **Timed actions**, i.e. actions whose execution rate is finite. The duration of a timed action is expressed by an exponentially distributed random variable with parameter given by the action rate.
  - **Immediate actions**, i.e. actions whose execution rate is infinite. The duration of an immediate action is zero hence immediate actions are used for modeling both logical conditions and activities much faster than the activities which are relevant from the performance evaluation point of view. Each immediate action has a priority level $l$ and a weight $p$ associated with it.

We shall denote with

$$Act = Com \cup \{\tau\}$$

the set of action types and we shall denote with

$$Rate = \{0\} \cup \mathbb{R}^+ \cup Inf, Inf = \{\alpha l, p | l \in \mathbb{N}^+ \land p \in \mathbb{R}^+\},$$

the set of action rates.

A synchronization between two terms may take place whenever the two terms execute observable actions of the same type. Sometimes, in order to obtain more concise models of concurrent systems, we would like to permit the synchronization between observable actions.
actions of different types. To achieve this, we introduce a set \( \Phi \) of relabeling functions which transform action types into other action types preserving the observability of the action types themselves; more accurately, let

\[
\Phi = \{ \varphi : \text{Act} \rightarrow \text{Act} \mid \varphi(\tau) = \tau \wedge \forall a \in \text{Com}. \varphi(a) \in \text{Com} \}.
\]

We shall see that such relabeling functions, together with the relabeling operator, avoid having several terms representing components of a given system which differ only for the type of some actions, because they make feasible to design only a term with the appropriate structure and then to make for each component a copy of this term to which an appropriate relabeling function is applied. \(^4\)

2.2 Constants

The second step in the definition of MPA consists of introducing a set \( \text{Con} \) of constants which are used as shorthands for MPA terms and can be used to express recursion. \(^5\)

Each constant \( A \) is specified by means of a defining equation of the form \( A \overset{\Delta}{=} E \) where \( E \) is a MPA term possibly containing constants inside it (if \( A \) is one of such constants then we have a recursive definition). Such a definition states that the behavior of \( A \) is the same as that of \( E \).

It is possible to use a set of variables and a recursion operator to define recursive terms, but when modeling real systems it turns out to be easier and more natural to work with constants.

2.3 Syntax of terms

The set \( \mathcal{L} \) of MPA terms is defined as the set of terms \( E \) generated by the following production:

\[
E ::= \emptyset \mid <a, \tilde{\lambda}> \cdot E \mid E / L \mid E \setminus H \mid E[\varphi] \mid E + E \mid E \parallel E \mid A
\]

where:

- \( a \in \text{Act}, \tilde{\lambda} \in \text{Rate}, L \subseteq \text{Com}, H \subseteq \text{Act}, \varphi \in \Phi, S \subseteq \text{Com}, A \in \text{Con}; \)
- “\( \emptyset \)” is said to be the null term;
- “\( \cdot \)” is said to be the prefix operator;
- “\( / \)” is said to be the functional abstraction operator;
- “\( \setminus \)” is said to be the temporal restriction operator;
- “\( \parallel \)” is said to be the relabeling operator;
- “\( + \)” is said to be the alternative composition operator;
- “\( \parallel \)” is said to be the parallel composition operator;
- we shall assume that “\( / \)”,”\( \setminus \)” and “\( \parallel \)” take precedence over “\( \cdot \)”,”\( \cdot \)” takes precedence over “\( \parallel \)”,”\( \parallel \)” takes precedence over “\( + \)” and “\( \parallel \)” are left associative (it is possible to use parentheses in order to enforce priorities and associativities other than the previous ones).

Observing the above production we can immediately realize that with MPA it is possible modeling concurrent systems in a compositional way, i.e. by iteratively composing terms describing subsystems with appropriate operators to obtain a final term describing the whole system; this fact forces MPA users to adopt a hierarchical and modular design style.

\(^4\) We shall use \( \varphi, \varphi', \psi, \ldots \) as metavariables for \( \Phi \).

\(^5\) We shall use \( A, B, C, \ldots \) as metavariables for \( \text{Con} \).
2.4 Informal semantics of operators
In this section we shall informally explain the semantics of MPA operators.

2.4.1 Null term
The null term is a zero-ary operator which models a deadlocked system, i.e. a system which cannot execute actions any more.

2.4.2 Prefix operator
The prefix operator expresses the sequential composition of an action and a term. Hence, the system described by the term \(<a, \lambda> . E\) executes the action \(<a, \lambda>\) and then behaves as the system described by the term \(E\).

2.4.3 Functional abstraction operator
The functional abstraction operator expresses the abstraction from the type of an action being executed provided that its type is in a given set. Hence, the system described by the term \(E /\tau\) behaves as the system described by the term \(E\), but the actions whose type appears in \(\tau\) are denoted with the type \(\tau\).

Denoting the type of an action with \(\tau\) means that the type of that action is unknown or irrelevant or to be hidden.

2.4.4 Temporal restriction operator
The temporal restriction operator prevents passive actions whose type is in a given set from being executed. Hence, the system described by the term \(E \setminus \tau\) behaves as the system described by the term \(E\), but the passive actions whose type appears in \(\tau\) cannot be executed.

We shall return to this operator in Section 4.3 when we shall introduce the property of temporal closure.

2.4.5 Relabeling operator
The relabeling operator changes the type of the actions being executed. Hence, the system described by the term \(E[\varphi]\) behaves as the system described by the term \(E\), up to transformations of action types as indicated by the relabeling function \(\varphi\).

As the relabeling operator is available, it could seem that the functional abstraction operator is redundant, but this is not the case because the two operators have clearly distinct roles; in fact:

- The functional abstraction operator transforms the type of certain actions into \(\tau\) so that, from the point of view of an external observer, these actions are not visible. As a consequence, the role of the functional abstraction operator is that of implementing a form of information hiding.
- The relabeling operator transforms the type of the actions into another type, provided that observable actions remain observable and invisible actions remain invisible. As a consequence, the role of the relabeling operator is not concerned with the information hiding but is that of allowing visible actions with different types to synchronize. This is sometimes useful for modeling systems with analogous components which differ only for the types of some actions, because it becomes possible to construct only a term for modeling the structure of such components and then to apply to each occurrence of this term a suitable relabeling function.
2.4.6 Alternative composition operator

The alternative composition operator expresses a choice between two terms (the stochastic nature of such a choice will result clear in Section 3.2 where we shall introduce the race policy). Hence, the system described by the term $E_1 + E_2$ behaves as either the system described by the term $E_1$ or the system described by the term $E_2$.

2.4.7 Parallel composition operator

The parallel composition operator expresses the parallel composition of two terms which sometimes synchronize by executing actions whose type is in a given set. Hence, the system described by the term $E_1 \parallel S E_2$, where $S$ is called the synchronization set, can execute asynchronously actions whose type does not appear in $S$ from term $E_1$ or term $E_2$, and synchronously actions whose type appears in $S$ from term $E_1$ and term $E_2$.

Depending on the values of the rates of the involved actions, synchronizations can be classified as follows:

- $<a, \lambda>$ or $<a, \infty, p>$ synchronized with $<a, 0>$. This case is admissible and allows to model the temporal closure of passive actions (see Section 4.3 for the property of temporal closure).
- $<a, 0>$ synchronized with $<a, 0>$. This case is admissible and allows to model $n$-way synchronizations, $n > 2$, in which $n - 1$ passive actions and one active action are involved.
- $<a, \bar{\lambda}>$ synchronized with $<a, \bar{\mu}>$ where $\bar{\lambda} \neq 0 \neq \bar{\mu}$. This case is not admissible because we require that in a synchronization at most one active action is involved so that the rate of the global action resulting from the synchronization of the single actions is uniquely determined by the rate of such an active action.

The choice mentioned in the last point avoids the problem of determining the rate of a synchronization involving two active actions which have different rates, and leads to the adoption of a clearer modular design style. Suppose for instance that a system is made out of two components which have to be modeled by two different designers, and assume that such components run in parallel and that the timing of some actions of the first component are determined by synchronization with the second component; thanks to the above choice, the designer of the first component can take the liberty of ignoring the timing of the second component by introducing in the first one some passive actions.

2.5 Guarded closure

An important property for recursive terms is the property of guarded closure.

**Definition 2.1** A program is a tuple

$$(E, A_1 \triangleq E_1, \ldots, A_n \triangleq E_n)$$

such that $E, E_1, \ldots, E_n \in \mathcal{L}$ and $A_1, \ldots, A_n \in \text{Con}$.

**Definition 2.2** A program $E, A_1 \triangleq E_1, \ldots, A_n \triangleq E_n$ is said to be closed if each constant occurring in $E, E_1, \ldots, E_n$ is in $\{A_1, \ldots, A_n\}$.
Definition 2.3 A closed program \((E, A_1 \triangleq E_1, \ldots, A_n \triangleq E_n)\) is said to be guarded iff for each term \(E' \in \{E, E_1, \ldots, E_n\}\) it turns out that each constant occurring in \(E'\) can be replaced (after finitely many applications of the defining equations) by a term \(E''\) in which each constant appears in a subterm of the form \(<a, \hat{x}>.E''\).

Definition 2.4 A program \((E, A_1 \triangleq E_1, \ldots, A_n \triangleq E_n)\) is said to be guardedly closed iff it is closed and guarded.

In the following we shall not deal with programs but terms because we shall implicitly assume as defined the constants occurring in the terms. We thus extend the notions of closure, guardedness and guarded closure to terms and we shall denote with \(G\) the set of guardedly closed terms of \(L\).

The property of guarded closure concerns the correct definition of terms, in particular recursive terms, from the functional point of view because:

- A guardedly closed term is closed so it does not contain constants no more instantiable due to the lack of the corresponding defining equations.
- A guardedly closed term is guarded so it cannot result in an infinite sequence of substitutions of a constant with another constant without executing actions.

3 Structured operational semantics of MPA

In this section we shall present the semantics of MPA in a formal way by resorting to the structured operational semantics approach.

This section is organized as follows. In Section 3.1 some notions concerning automata are recalled. In Section 3.2 the execution policy named race policy is defined and adopted. In Section 3.3 auxiliary labels are introduced. In Section 3.4 structured operational semantics rules are shown and briefly commented.

3.1 Automata

Since the results of the application of the structured operational semantics rules are automata, we recall the following definitions concerning automata taken from the textbook of E. - R. Olderog [19].

Definition 3.1 An automaton or labeled transition system with initial state is a quadruple 
\[(S, L, \longrightarrow, s_0)\]
such that:

- \(S\) is a set whose elements are called states;
- \(L\) is a set whose elements are called labels;
- \(\longrightarrow \subseteq S \times L \times S\) is called transition relation;
- \(s_0 \in S\) is called initial state.

The automaton \(A\) is said to be finite iff \(S\) is finite.
**Definition 3.2** The automaton $A_1 = (S_1, L, s_0, s_0)$ is said to be isomorphic to the automaton $A_2 = (S_2, L, s_0, s_0)$ iff there exists a bijection

$$\beta : S_1 \rightarrow S_2$$

such that:

- $\beta (s_0) = s_0$;
- $\forall s, s' \in S_1, \forall l \in L : s \xrightarrow{l_1} s' \iff \beta(s) \xrightarrow{l_2} \beta(s')$.

**Remark 3.3** In Section 5 we shall sometimes be concerned with automata for which the initial state is not uniquely defined; in the description of such automata the initial state will therefore be replaced by a function defined over the set of states which determines for each state the probability that the automaton will initially be in that state.

Since the states of the automata obtained by applying the structured operational semantics rules for MPA correspond to MPA terms, we take the liberty of using the words ‘term’ and ‘state’ as synonyms.

### 3.2 Race policy

Since several active actions can be simultaneously executable, it is necessary to choose a policy determining which of them is to be executed first.

In this framework we adopt the race policy, which chooses the active action having the least duration; since by definition immediate actions have duration zero whereas timed actions cannot sample durations zero from their exponential distributions, immediate actions take precedence over timed actions and this will be taken into account in Section 4.1 when we shall define the operational interleaving semantics of MPA terms.

It is worth noting that each immediate action is equipped with a priority level and a weight; therefore, when several immediate actions are simultaneously executable, only those having the highest priority level are actually executable and the choice among them is probabilistically made by giving each of them an execution probability proportional to its weight.

Before continuing, we recall the following property of the exponentially distributed random variables.

**Proposition 3.4** If $X_1, X_2, \ldots, X_n$ are $n \geq 2$ exponentially distributed random variables with parameters $\lambda_1, \lambda_2, \ldots, \lambda_n$, respectively, and a random variable $Y$ is defined by

$$Y = \min_{1 \leq i \leq n} X_i,$$

then $Y$ is an exponentially distributed random variable with parameter

$$\lambda = \sum_{i=1}^{n} \lambda_i.$$
As a consequence of the previous remarks and the above proposition, we have that:

- If in a given state \( n \) timed actions of the form
  \[ \langle a_1, \lambda_1 \rangle, \langle a_2, \lambda_2 \rangle, \ldots, \langle a_n, \lambda_n \rangle \]
  are executable and no immediate actions are executable, then the execution probability of the timed action \( \langle a_k, \lambda_k \rangle \) is given by
  \[ \frac{\lambda_k}{\sum_{i=1}^{n} \lambda_i} \]

- If in a given state \( n \) immediate actions of the form
  \[ \langle a_1, \ominus l, p_1 \rangle, \langle a_2, \ominus l, p_2 \rangle, \ldots, \langle a_n, \ominus l, p_n \rangle \]
  are executable and no immediate actions with higher priority are executable, then the execution probability of the immediate action \( \langle a_k, \ominus l, p_k \rangle \) is given by
  \[ \frac{p_k}{\sum_{i=1}^{n} p_i} \]

We shall also assume that the passive actions executable in a given state have the same nonzero execution probability.

The possibility of having several simultaneously executable actions in a given state is due to the two binary operators of MPA, i.e. the alternative composition operator and the parallel composition operator. Concerning the alternative composition operator, in Section 2.4.6 it is stated that such an operator expresses a choice between two terms and now we can understand that such a choice is of a probabilistic nature, as based on the race policy. To be more precise, such a choice takes:

- An **implicit** form when it concerns the choice between two timed actions, because in this case the execution probability of each of the two timed actions is implicitly determined by the race policy.
- An **explicit** form when it concerns the choice between two immediate actions at the same priority level, because in this case the execution probability of each of the two immediate actions is explicitly determined by its weight.

### 3.3 Auxiliary labels

Consider a term composed of two identical subterms whose topmost operator is the prefix operator. There are two cases:

- In the case of a classical process algebra the state \( a.E + a.E \) has only an outgoing transition to the state \( E \) so it is equivalent to the state \( a.E \).
- In the case of a stochastic process algebra (such as MPA) the state \( \langle a, \lambda \rangle.E \langle a, \lambda \rangle.E \) must have two different outgoing transitions to the state \( E \) because the race policy has been enforced and therefore the global transition rate from the first state to the second state is not \( \lambda \) but \( 2\lambda \) (by virtue of Proposition 3.4). Also the immediate transitions labeled by the same action type and the same temporal parameter and lying between the same ordered pair of states have to be kept disjoint in order to correctly construct the HCTMCs to be associated with MPA terms.
In order to enforce the race policy by keeping disjoint some transitions exiting from the same state (see above) and to correctly define the operational interleaving semantics of MPA terms (see Section 4.2), we introduce a set $Aux$ of auxiliary labels as the set of strings $w$ generated by the following production:

$$w ::= p \mid t \mid i \mid l(w) \mid r(w) \mid s(w, w)$$

where the symbol $p$ stands for passive action, the symbol $t$ stands for timed action, the symbol $i$ stands for immediate action, the symbol $l$ stands for action executed by the left term of an alternative or parallel composition, the symbol $r$ stands for action executed by the right term of an alternative or parallel composition and the symbol $s$ stands for a synchronization action; although these symbols can be considered as functors and consequently auxiliary labels can be considered as functional expressions, in what follows auxiliary labels will be considered as strings. \footnote{We shall use $u, v, w, \ldots$ as metavariables for $Aux$.}

We have taken from MTIPP the idea of using auxiliary labels in order to solve the problem considered at the beginning of this section. We have however fitted them in order to solve also another problem which will be considered in Section 4.2.

### 3.4 Structured operational semantics rules

The semantics of MPA is defined by following the structured operational semantics approach through the labeled transition system

$$(\mathcal{L}, \text{Act} \times \text{Rate} \times Aux, \rightarrow)$$

where $\rightarrow \subseteq \mathcal{L} \times (\text{Act} \times \text{Rate} \times Aux) \times \mathcal{L}$.

In general, in such an approach the transition relation $\rightarrow$ is defined by means of inference rules of the form

$$\begin{align*}
\{ E_i \overset{x_i}{\rightarrow} E'_i \mid 1 \leq i \leq m \} \{ F_j \overset{y_j}{\rightarrow} F'_j \mid 1 \leq j \leq n \} & \text{ if \ (condition)} \\
\{ G_k \overset{z_k}{\rightarrow} G'_k \mid 1 \leq k \leq p \in \mathbb{N}^+ \}
\end{align*}$$

The above inference rule states that if the transitions in the first set of the upper part exist, the transitions in the second set of the upper part do not exist and the condition (when present) is met by the transitions in the upper part, then the transitions in the lower part exist. Inference rules like the previous one will be used throughout the technical report. Furthermore, in the following we shall sometimes encounter inference rules whose upper part contains a sequence of transitions partly shortened with the symbol “$\ldots$”; this means that such transitions are different from each other and are all and only the transitions of that form meeting the condition possibly associated with the inference rule.

Turning to our transition relation $\rightarrow$, we define it as the least subset of $\mathcal{L} \times (\text{Act} \times \text{Rate} \times Aux) \times \mathcal{L}$ such that

$$\begin{align*}
\langle a, 0 \rangle, E & \overset{a, 0, p}{\rightarrow} E \\
\langle a, \lambda \rangle, E & \overset{a, \lambda, t}{\rightarrow} E \\
\langle a, \infty t, p \rangle, E & \overset{a, \infty t, p, i}{\rightarrow} E
\end{align*}$$

\footnote{We shall use $u, v, w, \ldots$ as metavariables for $Aux$.}
The above transition rules have already been informally explained in Section 2.4; here we only would like to point out the usage of the auxiliary labels in the transition rules for the binary operators and the clause $\min(\lambda, \mu) = 0$, appearing in the condition associated with the third transition rule for the parallel composition operator (such a clause requires that at most one of the actions involved in the synchronization is active).

We conclude this section by showing two properties of auxiliary labels that will result very useful in the following.

**Proposition 3.5** All the transitions exiting from a given state have different auxiliary labels.

**Proof**
We exploit the modus tollens tautology; so we shall prove that given two transitions

$$E \xrightarrow{a,\lambda,w} E_1' \text{ and } E \xrightarrow{a,\lambda,w} E_2'$$

exiting from the same state and having the same auxiliary label, it turns out that the two transitions are identical, i.e. $a_1 = a_2 \land \lambda_1 = \lambda_2 \land E_1 \equiv E_2$. We proceed by induction on the syntactical structure of $w$.
• If \( w \in \{p, t, i\} \) then both transitions have been derived through the application of the transition rule for the prefix operator possibly preceded by several applications of the transition rules for the functional abstraction operator, the temporal restriction operator and the relabeling operator. Since all these operators are unary, the two transitions must have been generated from the execution of the same action hence the thesis holds.

• Let \( w = l(v) \) and let the thesis hold for \( v \). By the initial hypothesis it follows that

\[
E' \xrightarrow{a'_1, \lambda'_1, v} E'_1 \quad \text{and} \quad E' \xrightarrow{a'_2, \lambda'_2, v} E'_2
\]

where either \( E \equiv E' + E'' \) or \( E \equiv E' \parallel S \ E'' \), \( a'_1 \in S \), \( a'_2 \in S \), unless functional abstraction operators, temporal restriction operators and relabeling operators are applied to \( E' \) and to the composition of \( E' \) with \( E'' \) (these operators are irrelevant for the proof, as noticed in the previous step). By the induction hypothesis we have that \( a'_1 = a'_2 \land \lambda'_1 = \lambda'_2 \land E'_1 \equiv E'_2 \) hence \( a_1 = a_2 \land \lambda_1 = \lambda_2 \land E_1 \equiv E_2 \).

• The case \( w = r(v) \) is analogous to the previous one.

• Let \( w = s(u, v) \) and let the thesis hold for \( u \) and for \( v \). By the initial hypothesis it follows that (in the most general case)

\[
E' \xrightarrow{a'_1, \lambda'_1, u} E'_1 \quad \text{and} \quad E' \xrightarrow{a'_2, \lambda'_2, u} E'_2
\]

and

\[
E'' \xrightarrow{a''_1, \lambda''_1, v} E''_1 \quad \text{and} \quad E'' \xrightarrow{a''_2, \lambda''_2, v} E''_2
\]

where \( E \equiv E' \parallel S \ E'' \), \( a'_1 = a''_1 \in S \), \( a'_2 = a''_2 \in S \), unless functional abstraction operators, temporal restriction operators and relabeling operators are applied to \( E' \), to \( E'' \) and to their composition (as we already know, these operators are irrelevant for the proof). By the induction hypothesis we have that \( a'_1 = a''_2 \land \lambda'_1 = \lambda''_2 \land E'_1 \equiv E'_2 \) hence \( a_1 = a_2 \land \lambda_1 = \lambda_2 \land E_1 \equiv E_2 \).

**Proposition 3.6** Let \( w_1, w_2 \in Aux \). If \( w_1 \neq w_2 \) then \( w_1 \) is not a prefix of \( w_2 \).

**Proof**

Given \( w_1, w_2 \in Aux \), we suppose \( |w_1| < |w_2| \) in order to avoid trivial cases. There are two possibilities:

• If \( w_1 \in \{p, t, i\} \) then, recalling that \( |w_1| < |w_2| \), we have that \( |w_2| \geq 2 \). Since the grammar for \( Aux \) does not produce strings of length \( \geq 2 \) beginning with a symbol in \( \{p, t, i\} \), it turns out that \( w_1 \) is not a prefix of \( w_2 \).

• If \( w_1 = l(v_1) \) or \( w_1 = r(v_1) \) or \( w_1 = s(u_1, v_1) \) then, recalling that \( |w_1| < |w_2| \), it turns out that \( w_1 \) is not a prefix of \( w_2 \) because if this were not true then the right parenthesis which is the last symbol of \( w_1 \) would be followed in \( w_2 \) by at least another symbol and this is not the case in that \( w_2 \in Aux \). In fact:

  - If the first symbol of \( w_2 \) following the above mentioned right parenthesis were in \( \{p, t, i, l, r, s, , , \} \) then the initial hypothesis \( w_2 \in Aux \) would be contradicted as one can immediately see from the grammar for \( Aux \).
– If the first symbol of $w_2$ following the above mentioned right parenthesis were another right parenthesis then a nonbalanced sequence of parentheses would appear in $w_2$ and this is not allowed by the grammar for $Aux$.

4 Operational interleaving semantics of MPA terms

The semantic model of a MPA term can be defined through an appropriate automaton. At first glance, the semantic model of a term $E \in G$ could be defined through the automaton

$$I_0[E] = (\uparrow_0 E, Act \times Rate \times Aux, \rightarrow_{E,0}, E)$$

where:

- $\uparrow_0 E$ is the least subset of $G$ such that

$$E \in \uparrow_0 E$$

- $\rightarrow_{E,0}$ is the restriction of $\rightarrow$ to the set $\uparrow_0 E$.

The correct semantic model of a term $E$ is not $I_0[E]$, i.e. it is not the automaton directly obtainable by applying the structured operational semantics rules, because:

- Such rules partly ignore the race policy because they take into account neither the priority of immediate actions over timed actions nor the different priority levels existing among immediate actions. As a consequence, such rules do not forbid the simultaneous presence of transitions having different priority levels which exit from the same state.
- Such rules do not assign the correct temporal parameter to identically labeled (up to auxiliary labels) transitions deriving from the synchronization of a timed action with several passive actions (of the same type) which are either independent of each other or mutually exclusive.

In the following we shall show how to transform the automaton $I_0[E]$ into the automaton $I_1[E]$ in order to solve the first problem and then how to transform the automaton $I_1[E]$ into the automaton $I[E]$ in order to solve the second problem. The automaton $I[E]$ will be considered the semantic model of the term $E$ and will be said the operational interleaving semantics of the term $E$; it is an interleaving model because it simulates concurrency through alternative sequential executions obtained by interleaving actions of subterms composed in parallel.

We shall see that the two transformations above can be performed interactively with the structured operational semantics rules, i.e. it is not necessary to generate the whole state space before applying them. This is due to the fact that we consider only terms in $G$, hence infinite branching is avoided and only a finite amount of time is needed to generate the transitions from a given state.

This section is organized as follows. In Section 4.1 the first transformation is defined. In Section 4.2 the second transformation is defined. In Section 4.3 the properties of functional
closure and temporal closure are introduced based on the operational interleaving semantics of MPA terms. In Section 4.4 some examples are reported which concern both the operational interleaving semantics of MPA terms and the modeling of real system components with MPA terms.

Remark 4.1 For the sake of simplicity, in the graphic representation of $I_0[E], T_1[E], T[E]$ auxiliary labels will always be omitted.

Example 4.2 We shall clarify the above transformations by means of a running example. The term that will be considered is $E \equiv <a, \infty_{3,1}>.E_1+<c, \infty_{2,1} \triangleright .A>+<g, 0>\emptyset$ where $E_1 \equiv <b, \lambda>.\emptyset\emptyset\emptyset\emptyset+<e, \infty_{1,1} \triangleright .E_2, A \triangleright <f, \gamma>.A, E_2 \equiv <d, \mu>.\emptyset\emptyset\emptyset\emptyset(d, 0>\emptyset\emptyset\emptyset\emptyset)$; note that $I_0[E]$ is represented by the following automaton:

![Automaton Diagram]

where $E_3 \equiv \emptyset\emptyset\emptyset\emptyset(d, 0>\emptyset\emptyset\emptyset\emptyset), E_4 \equiv \emptyset\emptyset\emptyset\emptyset(d, 0>\emptyset\emptyset\emptyset\emptyset)$.  

### 4.1 First transformation

The first transformation guarantees that all the active transitions exiting from a given state have the same priority level. It consists of modifying the automaton $T_0[E]$ obtained by applying the structured operational semantics rules in order to construct the automaton

$$T_1[E] = \langle \uparrow_1 E, Act \times Rate \times Aux, \rightarrow_1, E \rangle$$

where:

- $\uparrow_1 E$ is the least subset of $G$ such that

$$E \in \uparrow_1 E$$

$$E_1 \in \uparrow_1 E \quad E_1 \xrightarrow{a,0,w} E_2 \quad E_2 \in \uparrow_1 E$$

$$E_1 \in \uparrow_1 E \quad E_1 \xrightarrow{a,\lambda,w} E_2 \quad E_2 \in \uparrow_1 E$$

$$E_1 \in \uparrow_1 E \quad E_1 \xrightarrow{a,\infty_{l},w} E_2 \quad E_2 \in \uparrow_1 E$$

$$E_1 \in \uparrow_1 E \quad E_1 \xrightarrow{b,\infty_{l},w} E_2 \quad E_2 \in \uparrow_1 E$$

$$E_1 \in \uparrow_1 E \quad E_1 \xrightarrow{b,\infty_{l'},w} E_2 \quad E_2 \in \uparrow_1 E$$

if $l < l'$
Operational interleaving semantics of MPA terms

$E_1 \subseteq E_2$ is the least subset of $E_2$ such that

$$E_1 \in \mathcal{E}_1 E_1 \xrightarrow{a_{1,w}} E_2$$

$$E_1 \xrightarrow{a_{1,w}} E_2$$

$$E_1 \in \mathcal{E}_1 E_1 \xrightarrow{a_{1,w}} E_2 \xrightarrow{b_{1,w}} E_2'$$

$$E_1 \xrightarrow{a_{1,w}} E_2 \xrightarrow{b_{1,w}} E_2'$$

if $l < l'$

Example 4.3 Consider the term $E$ defined in Example 4.2. By applying the first transformation to $T_0[E]$ we obtain the following automaton representing $T_1[E]$:

![Diagram]

4.2 Second transformation

The second transformation assigns the correct temporal parameter to identically labeled (up to auxiliary labels) transitions deriving from the synchronization of a timed action with several passive actions (of the same type) which are either independent of each other, i.e. composed in parallel with a synchronization set not containing the action type at hand, or mutually exclusive, i.e. composed in alternative.

In order to clarify the problem to be solved with the second transformation, we shall consider for instance the term

$$E \equiv <a, \lambda, Q||_{\{a\}}((\ldots <a, 0><a, 0>Q||_0<a, 0>Q||_0<a, 0>Q||_0<a, 0>Q||_0<a, 0>Q||_0<a, 0>Q||_0<a, 0>Q||_0<a, 0>Q||_0<a, 0>Q\ldots))$$

which is constructed from the parallel composition of a term that can execute a timed action of type $a$ with a term deriving from the parallel composition of $n \geq 2$ independent terms each of which can execute a passive action of type $a$; by applying the structured operational semantics rules, we get the $n$ transitions

$$E \xrightarrow{a_{1,w}} E_1 \xrightarrow{\ldots \xrightarrow{a_{1,w}} E_n \xrightarrow{\ldots} Q||_{\{a\}}((\ldots Q||_0<a, 0>Q||_0\ldots))Q||_0<a, 0>Q||_0<a, 0>Q||_0<a, 0>Q||_0<a, 0>Q||_0<a, 0>Q||_0<a, 0>Q||_0<a, 0>Q||_0<a, 0>Q||_0<a, 0>Q||_0<a, 0>Q$$
The problem is that each of these transitions has \( \lambda \) as temporal parameter. Instead, the correct temporal parameter of each transition is \( \lambda/n \) because in \( E \) there exists only one active action whose execution rate is \( \lambda \) (hence the total rate at which the state \( E \) is left must be \( \lambda \)) and the \( n \) transitions exiting from the state \( E \) have the same execution probability; if each transition had \( \lambda \) as temporal parameter (and this is what happens above), then the total rate at which the state \( E \) is left would be \( n\lambda \) (by virtue of Proposition 3.4). The same phenomenon happens if in \( E \) each occurrence of \( "||_0" \) is replaced by an occurrence of \( "+" \).

In order to solve the above problem we can exploit the auxiliary labels in order to identify the timed actions exiting from the same state and involving the same timed action; this can be accomplished by defining the relation \( eq_t \) as the least subset of \( Aux \times Aux \) such that

\[
\frac{w_1 eq_t w_2}{l(w_1) eq l(w_2)} \quad \frac{w_1 eq_t w_2}{r(w_1) eq r(w_2)}
\]

where \( t \in w \) means that the symbol \( t \) appears in the string \( w \).

The second transformation consists of modifying the automaton \( \mathcal{T}_1[E] \) so as to construct the automaton

\[
\mathcal{T}[E] = (\uparrow E, Act \times Rate \times Aux, \rightarrow E, E)
\]

where:

- \( \uparrow E = \uparrow_1 E \);
- \( \rightarrow \) is the least subset of \( \rightarrow \) such that

\[
E' \in \uparrow E \quad E' \xrightarrow{\alpha, \lambda/w} E'' \quad \text{if} \quad \tilde{\lambda} \in \{0\} \cup \text{Inf} \\
E' \in \uparrow E \quad E' \xrightarrow{\alpha, \lambda/w} E'' \quad \text{if} \quad \forall i, j = 1, \ldots, n. w_i eq_t w_j
\]

From now on we shall denote the relation \( \bigcup_{E \in \mathcal{E}} \rightarrow \) with \( \rightarrow \).
Example 4.4 Consider the term $E$ defined in Example 4.2. By applying the second transformation to $I[E]$ we obtain the following automaton representing $I[E]$:

![Automaton Diagram]

because the two transitions entering the states $E_3$ and $E_4$ have auxiliary labels given by $l(l(r(s(t, l(p)))))$ and $l(l(r(s(t, r(p)))))$ which are equivalent according to $\epsilon q$.

4.3 Functional closure and temporal closure

In this section we shall introduce two interesting properties which can be easily defined after specifying the operational interleaving semantics of MPA terms.

The first property is the property of functional closure and is defined by means of the functional abstraction operator.

Definition 4.5 A term $E \in \mathcal{G}$ is said to be functionally closed iff $I[E]$ is isomorphic to $I[E/\text{Com}]$. We shall denote with $\mathcal{F}$ the set of terms in $\mathcal{L}$ that are functionally closed.

The property of functional closure concerns the complete definition of terms from the functional point of view, because a functionally closed term can execute only invisible actions hence it cannot be involved in a synchronization with another term.

The second property is the property of temporal closure and is defined by means of the temporal restriction operator. Its introduction stems from the fact that it is possible to define terms which can execute passive actions. In fact, priority levels concern active actions only, as one can see from the definition of $I_1[E]$ given in Section 4.1; passive actions are not involved (i.e. they must not give precedence to active actions) because if this were not the case then several admissible synchronizations would be prevented (like for instance those possible for the term $(<a, 0> \oplus <b, \infty_{1,1}> \oplus) \parallel (a,b) <a, \infty_{1,1}> \oplus <b, 0> \parallel \overline{0})$. In general, it is not acceptable that a term representing a real system can execute passive actions because passive actions have no actual timing associated with them.

Definition 4.6 A term $E \in \mathcal{G}$ is said to be temporally closed iff $I[E]$ is isomorphic to $I[E/\text{Act}]$. We shall denote with $\mathcal{T}$ the set of terms in $\mathcal{L}$ that are temporally closed; we shall also denote with $\mathcal{E}$ the set of terms in $\mathcal{L}$ that are guardedly and temporally closed, i.e. $\mathcal{E} = \mathcal{G} \cap \mathcal{T}$.

The property of temporal closure concerns the complete definition of terms from the temporal
point of view because a temporally closed term cannot execute passive actions; this does not
mean that a temporally closed term cannot have in its definition some passive actions but
that these actions, if present, are prevented by a temporal restriction or synchronized with
active actions of the same type.

4.4 Examples

We show in this section some examples concerning both the operational interleaving seman-
tics of MPA terms and the modeling of real system components with MPA terms.

Example 4.7 Consider the term $E \equiv <a, \lambda> \parallel <b, \mu>$; it turns out that $\mathcal{I}[E]$ is represented
by the following automaton:

This example highlights the concept of interleaving in that the previous automaton gives
no information about the fact that actions $<a, \lambda>$ and $<b, \mu>$ are executed in parallel, but
it only states that there exist two alternative sequential executions obtained by interleaving
the two actions $<a, \lambda>$ and $<b, \mu>$ appearing in the two subterms $<a, \lambda> \parallel$ and $<b, \mu> \parallel$ which are composed in parallel. One can thus expect that $\mathcal{I}[E]$ is isomorphic to $\mathcal{I}[E']$ where

$E' \equiv <a, \lambda> \parallel <b, \mu> + <b, \mu> \parallel <a, \lambda>$, and this is actually true:

The above isomorphism is correct also from the temporal point of view due to the memoryless
property of the exponentially distributed random variables expressing the execution rates
of the actions occurring in $E$ and in $E'$.

Example 4.8 An arrival process following a Poisson distribution with parameter $\lambda$ can be
modeled through the term

$Poisson \overset{\Delta}{=} <a, \lambda>. Poisson$

---

8. These examples are analogous to some examples present in [11].
Example 4.9 An arrival process following an hyperexponential distribution with order \( n \) and parameters \( p_1, p_2, \ldots, p_n \in \mathbb{R}^{[0,1]} \) and \( \lambda_1, \lambda_2, \ldots, \lambda_n \in \mathbb{R}^+ \) can be modeled through the terms

\[
\text{Exp}_n \overset{\Delta}{=} <a_1, \infty_{1,p_1}>, \text{Exp}_1 + <a_2, \infty_{1,p_2}>, \text{Exp}_2 + \ldots + <a_n, \infty_{1,p_n}>, \text{Exp}_n
\]

\[
\text{Exp}_i \overset{\Delta}{=} <a_i, \lambda_i>, \text{Exp}_n, \quad 1 \leq i \leq n
\]

Example 4.10 A service process following an Erlang distribution with order \( n \) and parameter \( \mu \) can be modeled through the term

\[
\text{Erlang}_n \overset{\Delta}{=} <s_1, \mu>, <s_2, \mu>, \ldots, <s_n, \mu>. \text{Erlang}_n
\]

Example 4.11 A queue of capacity \( q \) which accomodates customers arriving from the outside (action \( a \)) and delivers customers to servers (action \( d \)) can be modeled through the terms

\[
\text{Queue}_0 \overset{\Delta}{=} <a, 0>. \text{Queue}_1
\]

\[
\text{Queue}_i \overset{\Delta}{=} <a, 0>. \text{Queue}_{i+1} + <d, 0>. \text{Queue}_{i-1}, \quad 0 < i < q
\]

\[
\text{Queue}_q \overset{\Delta}{=} <d, 0>. \text{Queue}_{q-1}
\]

It is interesting noting that we have modeled the queue although we completely ignored the timing of both the arrival process and the servers.

Example 4.12 Suppose we want to model a simple communication protocol \( P \). The protocol \( P \) deals with three entities: the sender \( S \), the channel \( C \) and the receiver \( R \). The sender \( S \) generates a message (action \( gm \)) and then sends it (action \( sm \)) by means of the channel \( C \) to the receiver \( R \); when the receiver \( R \) receives a message (action \( rm \)), it processes it (action \( pm \)) and then sends an acknowledgment message (action \( sa \)) by means of the channel \( C \) to the sender \( M \); after receiving the acknowledgment message (action \( ra \)), the sender \( M \) can generate and send a new message. Assume that the sender \( M \) generates messages at rate \( \lambda \) and the receiver \( R \) processes messages at rate \( \mu \); assume also that the channel \( C \) is fully reliable, that it is composed of two independent lines \( L_m \) and \( L_a \) which are used for transmitting messages and acknowledgments, respectively, and that the transmission on these lines is infinitely faster than both the generation and the processing of messages. By virtue of the previous assumptions, the protocol \( P \) can be modeled through the terms

\[
M \overset{\Delta}{=} <gm, \lambda>. <sm, \infty_{1,1}>. <ra, 0>. M
\]

\[
R \overset{\Delta}{=} <rm, 0>. <pm, \mu>. <sa, \infty_{1,1}>. R
\]
5 Markovian semantics of MPA terms

The distinguishing feature of MPA (and obviously all stochastic process algebras) is the possibility of expressing not only the functional aspect of the behavior of concurrent systems but also the temporal aspect of their behavior and therefore the possibility of evaluating the performance of the concurrent systems to be modeled. In this section we shall see how analyzing the performance of a concurrent system represented by a MPA term.

Since in MPA the durations of the timed actions are expressed through exponentially distributed random variables, it is natural (though not immediate) to associate a HCTMC with each MPA term. Here we shall consider guardedly and temporally closed terms only; it is not possible to directly associate HCTMCs with such terms due to the presence of immediate actions.

Given a term \( E \in \mathcal{E} \) and denoted with \( \mathcal{M}[E] \) the state transition rate diagram of the HCTMC associated with \( E \), the idea is that of obtaining \( \mathcal{M}[\llbracket E \rrbracket] \) by adequately manipulating \( \mathcal{I}[E] \); at the end, \( \mathcal{M}[\llbracket E \rrbracket] \) will turn out to be defined as a variant of an automaton in which there is not an initial state but a probability mass function defined over the set of states which determines for each state the probability that the automaton will initially be in that state.

The algorithm to transform \( \mathcal{I}[E] \) into \( \mathcal{M}[\llbracket E \rrbracket] \) is subdivided into three phases:

9. For the basic notions about Markov chains the reader is referred to the textbook of L. Kleinrock [15] and the textbook of J. G. Kemeny-J. L. Snell [14].
5 Markovian semantics of MPA terms

- The first phase is concerned with the elimination of immediate transitions.
- The second phase operates in such a way that between each ordered pair of states there is at most one transition.
- The third phase is concerned with the detection and the merging of equivalent states according to the definition of lumping.

These phases generate a sequence of automata \( A_k[E] \) where the first one is a modified version of \( T[E] \) and the last one is \( \mathcal{M}[E] \); also these \( A_k[E] \) are variants of an automaton in which there is not an initial state but a probability mass function defined over the set of states which determines for each state the probability that the automaton will initially be in that state. The last automaton generated by the algorithm, i.e. \( \mathcal{M}[E] \), will be said to be the markovian semantics of the term \( E \).

This section is organized as follows. In Section 5.1 the first phase of the algorithm is defined and checked for correctness. In Section 5.2 the second phase of the algorithm is defined and checked for correctness. In Section 5.3 the third phase of the algorithm is defined and checked for correctness. In Section 5.4 some examples of application of the algorithm are reported.

5.1 First phase of the algorithm: eliminating immediate transitions

The first phase of the algorithm is concerned with the elimination of the immediate transitions; such an elimination causes some states to be removed and the temporal parameters of some transitions to be modified. This phase is subdivided into several steps.

5.1.1 First step

The first step consists of obtaining \( A_0[E] \) from \( T[E] = (\uparrow E, \text{Act} \times \text{Rate} \times \text{Aux}, \xrightarrow{E}, E) \); more accurately, let

\[
A_0[E] = (S_0, \mathbb{R}_{[0,1]} \times (\mathbb{R}^+ \cup \text{Inf}) \times \text{Aux}^*, \xrightarrow{0}, P_0)
\]

where:

- \( S_0 = \uparrow E; \)
- \( \xrightarrow{0} \) is the least subset of \( S_0 \times (\mathbb{R}_{[0,1]} \times (\mathbb{R}^+ \cup \text{Inf}) \times \text{Aux}^*) \times S_0 \) such that

\[
\begin{align*}
\forall s, s' \in S_0 & \quad \frac{s \xrightarrow{E} s'}{s \xrightarrow{1,\lambda,w} s'} \\
\forall s \in S_0 & \quad \frac{s \xrightarrow{0} s'}{s \xrightarrow{a,\lambda,s} s'}
\end{align*}
\]

- \( P_0 : S_0 \rightarrow \mathbb{R}_{[0,1]} \) such that

\[
P_0(E) = 1 \quad \forall s \in S_0 - \{E\}. P_0(s) = 0
\]
With this step we initialize the initial state probability mass function and we associate with each state having outgoing immediate transitions a probability mass function which, taking into account the weights of such outgoing immediate transitions, determines for each of such transitions the probability that it occurs.

### 5.1.2 $K$-th step

The $k$-step, $k \in \mathbb{N}^+$, consists of applying to the states $s_0, s_1, \ldots, s_n \in S_{k-1}$, where the states $s_1, \ldots, s_n$ are not necessarily distinct, the following graph reduction rule GRR1 in which the shown transitions are all and only the ones exiting from $s_0$ (the state $s_0$ is different from each of the states $s_1, \ldots, s_n$ because, due to the assumption we shall make in Section 5.1.3, the state $s_0$ cannot have an immediate transition exiting from it and entering it):

![Graph Reduction Rule GRR1](image)

More accurately, let

$$A_k[E] = (S_k, \mathbb{R}_{[0,1]} \times (\mathbb{R}^+ \cup Inf) \times Aux^*, \rightarrow_k, P_k)$$

where:

- $S_k = S_{k-1} - \{s_0\}$;
- $\rightarrow_k$ is the least subset of $S_k \times (\mathbb{R}_{[0,1]} \times (\mathbb{R}^+ \cup Inf) \times Aux^*) \times S_k$ such that
  
  $s \xrightarrow{k-1} s'$

  if $s \neq s_0 \land s' \neq s_0$

  $s \xrightarrow{k} s'$

  $s \xrightarrow{k-1} s_0 \xrightarrow{k-1} s_1 \ldots s_0 \xrightarrow{k} s_n$

- $P_k : S_k \rightarrow \mathbb{R}_{[0,1]}$ such that
  
  $\forall s \in S_k - \{s_i \mid 1 \leq i \leq n\}. \ P_k(s) = P_{k-1}(s)$

  $\forall s \in \{s_i \mid 1 \leq i \leq n\}. \ P_k(s) = P_{k-1}(s) + P_{k-1}(s_0) \cdot \sum_{j=1}^{n} \left\{p_j \mid s_0 \rightarrow_{k-1} s\right\}$

With this step we consider a fork of immediate transitions which is treated by eliminating the immediate transitions involved in the fork, multiplying the transitions entering the state upstream the fork and distributing the initial state probability associated with the state upstream the fork among the states downstream the fork.
5.1.3 Cycles of immediate transitions

We assume that the algorithm we are describing is applied only to terms $E \in \mathcal{E}$ such that $\mathcal{I}[E]$ has no immediate transition cycles; we shall denote with $\mathcal{E}^+$ the set of such terms in $\mathcal{E}$.

This assumption stems from the fact that the presence of immediate transition cycles entails the presence of sequences of repetitive actions with which no timing is associated and this is meant as the fact that the MPA term chosen for modeling a given system does not adequately represent the system itself; even if we wanted to deal with immediate transition cycles, we should decide how to treat them and state as a consequence a new graph reduction rule which will surely be more complicated than GRR1.

Now we show by means of an example a manner of treating such immediate transition cycles. Consider the term

$$E \equiv <a, \lambda>.B, \ B \overset{\lambda}{\rightarrow} <b, \infty_{1,1}>.B + <c, \infty_{1,1}>.\emptyset||_{e_1}\emptyset + \emptyset||_{e_1'}<d, \infty_{1,1}>.\emptyset$$

from which it is possible to reach the states

$$E_1 \equiv B$$
$$E_2 \equiv \emptyset||_{e_1}\emptyset$$
$$E_3 \equiv \emptyset||_{e_1'}\emptyset$$

and consider the automaton $A_0[E]$ and the automaton that we would obtain after applying the first phase of the algorithm:

In order to obtain the second automaton we can proceed in the following way:

- Firstly, we eliminate the transition labeled by $1/3, \infty_{1,1}$ entering the state $E_1$ and we introduce a new dummy state $E_1^{(1)}$ which is entered by a transition labeled by $1/3, \infty_{1,1}$ exiting from the state $E_1$ and from which three transitions labeled by $1/3, \infty_{1,1}$ entering the states $E_1^{(1)}, E_2, E_3$ exit. In other words, the immediate transition from the state $E_1$ to the state $E_1$ itself is treated by differing of a step the cycle it constitutes.
- Secondly, we apply GRR1 to the fork having the state $E_1$ upstream by eliminating the state $E_1$ and introducing three transitions labeled by $1/3, \lambda$ exiting from the state $E$ and entering the states $E_1^{(1)}, E_2, E_3$.

Repeating this proceeding we have that the transition rate from the state $E$ to the state $E_1^{(k)}$ is given by

$$\lambda \cdot \frac{1}{3^k} \xrightarrow{k \rightarrow +\infty} 0$$

UBLCS-94-10
and that the transition rate from the state $E$ to the states $E_2$ and $E_3$ is given by
\[
\lambda \cdot \sum_{i=1}^{k} \frac{1}{3^i} = \lambda \cdot \left( \frac{1 - 1/3^{k+1}}{1 - 1/3} - 1 \right) \xrightarrow{k \to +\infty} \frac{\lambda}{2}
\]
hence to the limit we obtain the desired automaton.

### 5.1.4 Correctness of the first phase of the algorithm

Now we prove the correctness of the first phase of the algorithm.

**Theorem 5.1** The application of GRR1 does not merge initially distinct transitions exiting from a given state.

**Proof**

By virtue of Proposition 3.5, in the automaton $A_0[[E]]$ all the transitions exiting from a given state have different auxiliary labels. Having denoted with $s_0, s_1, \ldots, s_n \in S_{k-1}$ the states involved in the fork considered at step $k \in \mathbb{N}^+$ (where $s_0$ is the state upstream the fork), suppose that
\[
\exists s \in S_{k-1}. \exists i \in \{1, \ldots, n\}. s \xrightarrow{p_i \cdot \lambda_{i, w_i}}_{k-1} s_0 \land s_0 \xrightarrow{p_{i_1}, \overline{\lambda_{i_1, w_{i_1}}}}_{k-1} s_i \land s \xrightarrow{p' \cdot \lambda', w'}_{k-1} s_i
\]
so that after applying GRR1 we have the two transitions
\[
s \xrightarrow{p \cdot p_i \cdot \lambda_{i, w_i}}_{k} s_i \text{ and } s \xrightarrow{p' \cdot \lambda', w'}_{k} s_i
\]
where the second transition has been simply inherited from step $k - 1$. Having supposed that $p \cdot p_i = p'$ and $\lambda = \lambda'$ in order to avoid trivial cases, we prove that $ww_i \neq w'$; there are two cases:
- If $|ww_i| \neq |w'|$ then $ww_i \neq w'$ hence the two transitions are not merged.
- If $|ww_i| = |w'|$ then both $ww_i$ and $w'$ being concatenations of $Aux$ strings, let $w_1$ be the first $Aux$ string appearing in $ww_i$ and $w'_1$ be the first $Aux$ string in $w'$; there are three cases:
  - If $|w_1| = |w'_1|$ then $w_1 \neq w'_1$ because $w_1$ and $w'_1$ are the auxiliary labels of two different transitions exiting from the same state $s_0$. Since $|ww_i| = |w'| \land ww_i = w_1 w \land w'_1 w \land |w_1| = |w'_1| \land w_1 \neq w'_1$ entails that $ww_i \neq w'$, the two transitions are not merged.
  - If $|w_1| < |w'_1|$ then $w_1 \neq w'_1$ hence $w_1$ is not a prefix of $w'_1$ by virtue of Proposition 3.6. It follows that $ww_i \neq w'$ hence the two transitions are not merged.
  - The case $|w_1| > |w'_1|$ is analogous to the previous one.

**Theorem 5.2** The order in which the immediate transition forks are considered to the end of the application of GRR1 is irrelevant, i.e. GRR1 is confluent.

**Proof**

The thesis holds due to the commutativity of the multiplication. More in detail, given two immediate transition forks, there are the following cases:
• There exists a state downstream a fork and upstream the other fork:

• There exists at least one state downstream both forks:
• There is no interaction between the two forks; in such a case, it is obvious that the order in which the two forks are considered to the end of the application of GRR1 is irrelevant.

\textbf{Theorem 5.3} Let $E \in \mathcal{E}^+$. It turns out that:

(i) $\forall k \in \mathbb{N} \forall s \in \{s' \in S_k \mid s' P_{p_i,\infty}^{j_i,0} \xrightarrow{p_i,\infty} s_i\}. \sum \{q_j \mid s \xrightarrow{q_j,\infty} s_j \neq s \} = 1$.

(ii) $\forall k \in \mathbb{N} \sum_{s \in S_k} P_k(s) = 1$.

(iii) If $I[E]$ has finitely many states, then the first phase of the algorithm terminates and the last generated automaton has finitely many states, has not immediate transitions and is unique.

\textbf{Proof}

(i) We proceed by induction on $k \in \mathbb{N}$

- If $k = 0$ then the thesis immediately follows from the definition of $P_0$.

- Let $k > 0$ and let the thesis hold for $k-1$. Having denoted with $s_0, s_1, \ldots, s_n \in S_{k-1}$ the states involved in the fork considered at step $k$ (where $s_0$ is the state upstream the fork), let us take a state $s \in S_k$.

* If $s \in S_k - \{s' \in S_{k-1} \mid s' P_{p_i,\infty}^{j_i,0} \xrightarrow{p_i,\infty} s_i\}$ then either $s$ has not outgoing immediate transitions hence the thesis is not concerned with it, or $s$ has outgoing immediate transitions none of which enters $s_0$ hence $s$ satisfy the thesis by the induction hypothesis.

* Let $s \in \{s' \in S_{k-1} \mid s' P_{p_i,\infty}^{j_i,0} \xrightarrow{p_i,\infty} s_i\}$. Since by the induction hypothesis at step $k-1$ it turns out that $\sum \{q_j \mid s \xrightarrow{q_j,\infty} s_j \neq s \} = 1$ and that $\sum \{q_j \mid s \xrightarrow{q_j,\infty} s_j \neq s \} = 1$, at step $k$ we have $\sum \{q_j \mid s \xrightarrow{q_j,\infty} s_j \neq s \} + \sum \{q_j \mid s \xrightarrow{q_j,\infty} s_j \neq s \} = 1 + 1 = 2$.

(ii) We proceed by induction on $k \in \mathbb{N}$

- If $k = 0$ then the thesis immediately follows from the definition of $P_0$.

- Let $k > 0$ and let the thesis hold for $k-1$. Having denoted with $s_0, s_1, \ldots, s_n \in S_{k-1}$ the states involved in the fork considered at step $k$ (where $s_0$ is the state upstream the fork) and noticed that by the induction hypothesis it turns out that $\sum_{s \in S_{k-1}} P_{k-1}(s) = 1$, we have that $\sum_{s \in S_k} P_k(s) = \sum_{s \in S_k - \{s_0\}} P_{k-1}(s) + \sum_{s \in \{s_0\}} P_k(s)$. 

UBLCS-94-10 27
The rate of the timed transitions by the corresponding weights. The first step consists of obtaining a HCTMC. This phase is subdivided into several steps. Suppose that the first phase of the algorithm terminates with the generation of the automaton \( A_k \{ E \}, k' \geq 1 \). The second phase of the algorithm exploits Proposition 3.4 in order to guarantee that between each ordered pair of states there is at most one transition so as to obtain a HCTMC. This phase is subdivided into several steps.

### 5.2.1 First step

The first step consists of obtaining \( A_{k'+1} \{ E \} \) from \( A_k \{ E \} = (S_k, \mathbb{R}_{[0,1]} \times (\mathbb{R}^+ \cup \inf) \times Aux^*, \xrightarrow{k'}, P_{k'}) \); more accurately, let

\[
A_{k'+1} \{ E \} = (S_{k'+1}, \mathbb{R}^+ \times Aux^*, \xrightarrow{k'+1}, P_{k'+1})
\]

where:

- \( S_{k'+1} = S_{k'} \);
- \( \xrightarrow{k'+1} \) is the least subset of \( S_{k'+1} \times (\mathbb{R}^+ \times Aux^*) \times S_{k'+1} \) such that

\[
\begin{align*}
&\xrightarrow{k'}_\mu \ x \Rightarrow x' \\
&\mu \in [\mathbb{R}^+]^{n+1}
\end{align*}
\]

- \( P_{k'+1} = P_k \).

With this step we make explicit the absence of immediate transitions and we multiply the rate of the timed transitions by the corresponding weights.

### 5.2.2 K-th step

The \( k \)-th step, \( k \in \mathbb{N}^* \), consists of applying to the two not necessarily distinct states \( s_0, s_1 \in S_{k'+(k-1)} \) the following graph reduction rule GRR2 where the \( n \geq 2 \) shown transitions are all and only those from \( s_0 \) to \( s_1 \):
More accurately, let

\[ A_{k' + k}[E] = (S_{k' + k}, \mathbb{R}^+ \times Au, \sum_{k' + k} \lambda_i, P_{k' + k}) \]

where:

- \( S_{k' + k} = S_{k' + (k-1)} \);
- \( \sum_{k' + k} \lambda_i \)

is the least subset of \( S_{k' + k} \times (\mathbb{R}^+ \times Au) \times S_{k' + k} \) such that

\[
\begin{align*}
    s \xrightarrow{\lambda_{i,w}} s' & \quad \text{if } s \neq s_0 \lor s' \neq s_1 \\
    s \xrightarrow{\lambda_{i,w}} s' & \quad \text{if } s_0 \neq s_0 \lor s_1 \neq s_1
\end{align*}
\]

- \( P_{k' + k} = P_{k' + (k-1)} \).

With this step we consider a set of timed transitions between two given states and we merge them in a single timed transition between the same two states whose rate is the sum of the rates of the transitions that have been merged.

### 5.2.3 Correctness of the second phase of the algorithm

Now we prove the correctness of the second phase of the algorithm.

**Theorem 5.4** The order in which the sets of timed transitions between two states are considered to the end of the application of GRR2 is irrelevant, i.e., GRR2 is confluent.

**Proof**

The thesis immediately stems from the fact that each application of GRR2 concerns only the transitions that are local to two states.

**Theorem 5.5** Let \( E \in E^+ \). It turns out that:

- \( \forall k \in \mathbb{N}, \sum_{s \in S_{k' + k}} P_{k' + k}(s) = 1 \).
(ii) If $\mathcal{I}[E]$ has finitely many states, then also the second phase of the algorithm terminates and the last generated automaton has finitely many states, has not ordered pairs of states with more than one transition between them and is unique.

**Proof**

(i) It immediately stems from Theorem 5.3(ii) and the fact that $\forall k \in \mathbb{N}. R_{k+1}^{+} = R_{k}^{+}$.

(ii) If $\mathcal{I}[E]$ has finitely many states then $\mathcal{I}[E]$ has finitely many transitions hence, in particular, finitely many timed transitions. Since under the hypothesis that $\mathcal{I}[E]$ is finite the first phase of the algorithm terminates by virtue of Theorem 5.3(iii) and therefore each timed transition is replaced finitely many times by finitely many timed transitions, at the end of the first phase the number of states remains unchanged) there are finitely many states thus there exist finitely many ordered pairs of states, the second phase of the algorithm terminates and the last generated automaton has not ordered pairs of states with more than one transition between them. Besides, such an automaton is unique by virtue of the confluence of GRR2.

#### 5.3 Third phase of the algorithm: lumping

Suppose that the second phase of the algorithm terminates with the generation of the automaton $A_{k}^{+}[E], k'' \geq 2$. The third phase of the algorithm is concerned with the detection and the merging of equivalent states; this phase therefore minimizes the number of states of the HCTMC obtained at the end of the second phase. The concept of equivalence we shall refer to is the one deriving from the definition of lumping of a HCTMC [14].

**Definition 5.6** Let $X$ be a HCTMC with state space $\mathcal{S}_X$ and infinitesimal generator $Q = [q_{i,j}]$. $X$ is said to be lumpable w.r.t. a partition $\mathcal{P}$ of $\mathcal{S}_X$, i.e. the partition $\mathcal{P}$ of $\mathcal{S}_X$ is said to be a lumping of $X$, iff

$$\forall L_i, L_j \in \mathcal{P}. \forall h, l \in L_i. \sum_{k \in L_j} q_{h,k} = \sum_{k' \in L_j} q_{l,k'}.$$ 

If this is the case, the lumped HCTMC obtained from $X$ has state space $\mathcal{P}$ and infinitesimal generator $Q' = [q_{i,j}']$ where

$$q_{i,j}' = \sum_{k \in L_j} q_{i,k}$$

for some $h \in L_i$.

This phase is subdivided into two steps.

#### 5.3.1 First step

The first step consists of obtaining $A_{k''+1}[E]$ from $A_{k''}[E] = (S_{k''}, \mathbb{R}^+, \Delta u)x^*, \longrightarrow_{k''}, P_{k''})$; more accurately, let

$$A_{k''+1}[E] = (S_{k''+1}, \mathbb{R}^+, \longrightarrow_{k''+1}, P_{k''+1})$$

where:
\[ S_{k''+1} = S_{k''} \]

\[ \rightarrow \text{ is the least subset of } S_{k''+1} \times \mathbb{R}^+ \times S_{k''+1} \text{ such that} \]

\[ \frac{s_{\lambda_w}}{k''+1} \rightarrow s' \]

\[ \frac{s_{\lambda}}{k''} \rightarrow s' \]

\[ P_{k''+1} = P_{k''} \]

With this step we make explicit the absence of ordered pairs of states with more than one transition between them.

### 5.3.2 Second step

The second step consists of determining first of all a lumping \( \mathcal{P} \) of \( A_{k''+1}[[E]] \) that is minimum, i.e. having as few elements as possible, by using the following procedure:

\[
\begin{align*}
\text{begin} \\
< \text{let } \mathcal{P} \text{ be the least partition of } S_{k''+1} \text{ such that } \forall L \in \mathcal{P} : \forall s_p, s_q \in L : \sum \{ \lambda_l | \frac{s_p}{k''+1} \rightarrow s_l \} = \sum \{ \lambda_l | \frac{s_q}{k''+1} \rightarrow s_l \} >; \\
\text{repeat} \\
\mathcal{P}_{old} := \mathcal{P}; \\
\mathcal{P} := \emptyset; \\
\text{for each } L \in \mathcal{P}_{old} \text{ do begin} \\
< \text{let } \mathcal{P}_L \text{ be the least partition of } L \text{ such that } \forall L' \in \mathcal{P}_L : \forall L'' \in \mathcal{P}_{odd} : \forall s_p, s_q \in L'. \sum \{ \lambda_l | \frac{s_p}{k''+1} \rightarrow s_l \land s_l \in L'' \} = \sum \{ \lambda_l | \frac{s_q}{k''+1} \rightarrow s_l \land s_l \in L'' \} >; \\
\mathcal{P} := \mathcal{P} \cup \mathcal{P}_L \\
\text{end} \\
\text{until } \mathcal{P} = \mathcal{P}_{odd} \\
\text{end}
\end{align*}
\]

Once determined the lumping \( \mathcal{P} \), let

\[ \mathcal{M}[E] = (S_M, \mathbb{R}^+, \rightarrow_M, P_M) \]

where:

- \( S_M = \mathcal{P} \);
- \( \rightarrow \text{ is the least subset of } S_M \times \mathbb{R}^+ \times S_M \text{ such that} \)

\[ \frac{s_{\lambda_i}}{k''+1} \rightarrow s_1 \]

\[ \frac{s_{\lambda_n}}{k''+1} \rightarrow s_n \]

\[ \frac{\sum \lambda_i}{M} \rightarrow L' \]

\[ \forall s \in L \land \{ s_i | 1 \leq i \leq n \} \subseteq L' \]

- \( P_M : S_M \rightarrow \mathbb{R}_{[0,1]} \text{ such that} \)

\[ \forall L \in S_M : P_M(L) = \sum_{s \in L} P_{k''+1}(s) \]

With this step we minimize the HCTMC obtained at the end of the second phase according to the notion of lumping.

UBLCS-94-10
5.3.3 Correctness of the third phase of the algorithm

Now we prove the correctness of the third phase of the algorithm.

**Theorem 5.7** The partition $\mathcal{P}$ of $S_{k^v+1}$ is a lumping of $A_{k^v+1}[E]$.

**Proof**

Each partition $\mathcal{P}$ of $S_{k^v}$ constructed in the repeat-until cycle is such that

$$\forall L' \in \mathcal{P}, \forall L'' \in \mathcal{P}_{old}, \forall s_p, s_q \in L', \sum \{ \lambda_l | s_p \xrightarrow{k^v+1} s_l \land s_q \in L'' \} = \sum \{ \lambda_l | s_q \xrightarrow{k^v+1} s_l \land s_p \in L'' \}.$$

Since after the last iteration the partition $\mathcal{P}$ is equal to the partition $\mathcal{P}_{old}$, it follows that $\mathcal{P}$ is a lumping of $A_{k^v+1}[E]$.

**Theorem 5.8** Let $E \in \mathcal{E}^+$. It turns out that:

(i) $\sum_{L \in S_M} P_M(L) = 1$.

(ii) If $I[E]$ has finitely many states, then also the third phase of the algorithm terminates and the automaton $M[E]$ has finitely many states, has not ordered pairs of states with more than one transition between them and is unique.

**Proof**

(i) It immediately stems from Theorem 5.5(i) and from the definitions of $P_{k^v+1}$ and $P_M$.

(ii) If $I[E]$ has finitely many states then, by virtue of Theorem 5.5(ii), the second phase of the algorithm terminates and the set $S_{k^v}$ is finite. Since the third phase of the algorithm consists of repeatedly partitioning the set $S_{k^v}$, since this proceeding certainly terminates when it reaches the partition whose sets contain each only one element in $S_{k^v}$ and since such a partition is certainly reached after a number of repeat-until cycles bounded by $\#S_{k^v}$, also the third phase of the algorithm terminates. The previous argument furthermore allows to deduce that $M[E]$ has finitely many states. The absence of ordered pairs of states in $M[E]$ with more than one transition between them is guaranteed by the absence of such pairs in $A_{k^v}[E]$, and thus in $A_{k^v+1}[E]$, by virtue of Theorem 5.5(ii) and the definition of both $S_M$ as lumping and $\xrightarrow{M}$ as relation determining the rate of the transitions between the elements of the lumping as sum of the rates of the transitions between the states of the elements of the lumping (by definition of lumping, such a sum is the same for all the states belonging to the same element of the lumping). The uniqueness of $M[E]$ stems from the fact that each partition must meet the minimality requirement.

5.4 Examples

We show in this section some examples in which the markovian semantics of some MPA terms is obtained. In each example there are four graphs representing $I[E]$, $A_{k^v+1}[E]$, $A_{k^v+1}[E]$, $M[E]$; in the last three graphs the numbers associated with the states specify the initial state probability mass function.
Example 5.9 Let \( E \equiv \langle a, \lambda \rangle . (\langle d, \infty_{1,1} \rangle . \langle b, \mu \rangle . \emptyset + \langle d, \infty_{1,1} \rangle . \langle b, \mu \rangle . \emptyset + \langle c, \infty_{1,1} \rangle . \emptyset ); \) then:

Example 5.10 Let \( E \equiv \langle d, \infty_{1,1} \rangle . \langle a, \lambda \rangle . \langle h, \infty_{1,1} \rangle . \emptyset + \langle d, \infty_{1,1} \rangle . \langle a, \lambda \rangle . \langle h, \infty_{1,1} \rangle . \emptyset + \langle a, \lambda \rangle . \langle g, \infty_{1,1} \rangle . \emptyset ); \) then:

Example 5.11 Let \( E \equiv \langle a, \gamma \rangle . A + \langle d, \gamma \rangle . B \) where \( A \overset{\Delta}{=} \langle b, \lambda \rangle . \langle c, \mu \rangle . A, B \overset{\Delta}{=} \langle e, \lambda \rangle . \langle f, \mu \rangle . B; \) then:

Example 5.12 Let \( E \equiv A \) where \( A \overset{\Delta}{=} \langle a, \lambda \rangle . B + \langle a, \lambda \rangle . C, \quad B \overset{\Delta}{=} \langle b, \mu \rangle . A + \langle b, \gamma \rangle . D, \quad C \overset{\Delta}{=} \langle c, \mu \rangle . A + \langle c, \gamma \rangle . D, \quad D \overset{\Delta}{=} \langle d, \mu \rangle . B + \langle d, \mu \rangle . C; \) then:
Example 5.13 Let $E \equiv \text{Poisson}$ where $\text{Poisson} \centernot= <a, \lambda>$. Then:

Having noticed that the term $E$ models a Poisson arrival process with parameter $\lambda$, we would like to point out that $M\llbracket E \rrbracket$ is a lumped version of the HCTMC for a Poisson process.

6 Conclusions

In this technical report a new stochastic process algebra named MPA has been defined. It can be viewed as an extension of the stochastic process algebras MTIPP and PEPA; now we list the main differences between MPA and these other stochastic process algebras:

- In PEPA there are passive and timed actions, whereas in MPA and in MTIPP there are also immediate actions and in MPA such immediate actions are subdivided into priority levels and have weights associated with them, unlike MTIPP. Furthermore, in MPA the value indicating the execution rate of the actions is 0 for passive actions, $\lambda \in \mathbb{R}^+$ for timed actions, $\propto_{LP}$ for immediate actions, hence in each case such a value seems consistent with the concept of execution rate of the actions, while the value indicating the execution rate of passive actions is $-\infty$ in MTIPP and $\infty$ in PEPA.

- In MPA there are two operators which appear neither in MTIPP nor in PEPA. They are the temporal restriction operator, which prevents passive actions from being executed and is used for defining the property of temporal closure, and the relabeling operator, which allows to synchronize observable actions having different types and therefore sometimes allows to model concurrent systems in a more concise way.

- In MPA it is possible to easily express explicit probabilistic choices by means of the alternative composition operator and the weights associated with immediate actions (this is possible in MTIPP too, but MTIPP has both an alternative composition operator and a probabilistic choice operator which interact in a nonnatural way). Due to this fact, in MPA activities whose durations are hyperexponentially distributed with any order can be naturally modeled.

- In MPA the transition rules for the parallel composition operator impose that in case of synchronization at least one of the involved action is passive so that in general it is possible to model $n$-way synchronizations, $n \geq 2$, in which $n-1$ passive actions and one active action are involved (it is also possible having an $n$-way synchronization, $n \geq 2$,
in which all the $n$ involved actions are passive but this case can be avoided by using the temporal restriction operator). This limitation concerning the synchronization, completely absent both in MTIPP and in PEPA, makes the distinction between active actions and passive actions clear and gives resalt to the role of passive actions.

- In MPA the operational interleaving semantics is defined by taking into account the different priority levels existing among active actions and by assigning the right temporal parameters to identically labeled (up to auxiliary labels) transitions deriving from the synchronization of a timed action with several passive actions (of the same type) which are either independent of each other or mutually exclusive.

- In MPA the markovian semantics is extensively defined (unlike MTIPP, where it is only sketched by means of examples) and also includes a phase of lumping (unlike PEPA). The work that has led to the birth of MPA is obviously open to future developments; more in detail, the future researches will be concerned with the following subjects:
  - Identifying an equivalence relation over MPA terms which takes into account both the functional aspect and the temporal aspect of the behavior of concurrent systems.
  - Investigating whether the markovian semantics is compositional, after defining some suitable operators over HCTMCs in correspondence to those of MPA.
  - Modeling timed activities whose durations are not expressible by means of random variables following an exponential, a hyperexponential or an Erlang distribution. This could lead to the development of a semi-markovian process algebra.

We conclude by noting that the work done till now, together with the future developments above and [3], can lead to the implementation of a software tool based on MPA which make the qualitative and quantitative analysis of concurrent systems computer aided and possibly automatic.

References

[1] M. Bernardo  
“Verso l’Integrazione di Modelli di Concorrenza Stocastici”  
Master Thesis  
University of Bologna, Bologna (Italy), March 1994

“Describing Queueing Systems with MPA”  
Technical Report  
University of Bologna, Bologna (Italy), 1994

“Operational GSPN Semantics of MPA”  
Technical Report  
University of Bologna, Bologna (Italy), 1994

“A Timed Calculus of Communicating Systems”  
Technical Report  
University of Edinburgh, Edinburgh (UK), 1990

“A Brief History of Timed CSP”
REFERENCES

Technical Report
University of Oxford, Oxford (UK), 1992

“Observing Time-Complexity of Concurrent Programs”
Technical Report
University of Pisa, Pisa (Italy), 1993

“The PEPA Workbench: a Tool to Support a Process Algebra-Based Approach to Performance Modelling”
Technical Report
University of Edinburgh, Edinburgh (UK), 1993

[8] R. Gorrieri, M. Roccetti
“Towards Performance Evaluation in Process Algebras”
in Proceedings of AMAST ’93, Twente (Netherlands), 1993
Springer-Verlag, Berlin (Germany), 1993

“A Theory of Processes with Durational Actions”
Technical Report
University of Bologna, Bologna (Italy), 1994

“TIPP: a Language for Timed Processes and Performance Evaluation”
Technical Report
University of Erlangen-Nürnberg, Erlangen (Germany), 1992

in Lecture Notes in Computer Science, vol. 729, pp. 121–146
Springer-Verlag, Berlin (Germany), 1993

[12] J. Hillston
“PEPA: Performance Enhanced Process Algebra”
Technical Report
University of Edinburgh, Edinburgh (UK), 1993

“Communicating Sequential Processes”
Prentice Hall, Englewood Cliffs (NJ), 1985

“Finite Markov Chains”
Springer-Verlag, Berlin (Germany), 1977

[15] L. Kleinrock
“Queuing Systems”
Wiley, New York (NY), 1975

[16] R. Milner
“Communication and Concurrency”
Prentice Hall, Englewood Cliffs (NJ), 1989

[17] F. Moller, C. Tofts
“A Temporal Calculus of Communicating Systems”
in Lecture Notes in Computer Science, vol. 458, pp. 401–415
Springer-Verlag, Berlin (Germany), 1990

[18] F. Moller, C. Tofts
“Behavioural Abstraction in TCCS”
in Lecture Notes in Computer Science, vol. 623, pp. 559–570
Springer-Verlag, Berlin (Germany), 1992

“Nets, Terms and Formulas. Three Views of Concurrent Processes and their Relationship”

[20] G. Reed, W. Roscoe
“A Timed Model for CSP”
in Lecture Notes in Computer Science, vol. 226, pp. 314–323
Springer-Verlag, Berlin (Germany), 1986

[21] S. Schneider
“An Operational Semantics for Timed CSP”
Technical Report
University of Oxford, Oxford (UK), 1991

[22] C. Tofts
“Timing Concurrent Processes”
Technical Report
University of Edinburgh, Edinburgh (UK), 1989