Available online at www.sciencedirect.com
ScienceDirect
Journal of Computational and Applied Mathematics 204 (2007) 38–47
JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS
www.elsevier.com/locate/cam

Generating the syntactic and semantics graphs for a Markovian process algebra

Fernando L. Pelayo[a,*], Maria L. Pelayo[a], Juan L.G. Guirao[b]

[a] *Computing Systems Department, Polytechnic Engineering High School of Albacete, University of Castilla-La Mancha, Spain*
[b] *Applied Mathematics & Statistics Department, Polytechnic Engineering High School of Cartagena, Spain*

Received 15 July 2005; received in revised form 10 November 2005

Abstract

We present a couple of programs (over C and *Mathematica*) which allow to generate the syntactic and the semantics representations of a ROSA (**R**easoning **O**n **S**tochastic **A**lgebras) process.

ROSA is a Markovian process algebra for the description and analysis of probabilistic and non-deterministic concurrent processes. ROSA allows us to evaluate performance indexes as well as to check some temporal requirements. As application, we analyse the alternating bit protocol obtaining the average time to send a message, considering that channels may fail with a known probability.
© 2006 Published by Elsevier B.V.

Keywords: Theoretical computer science; Formal methods; Non-deterministic and probabilistic aspects; Stochastic process algebra; Performance analysis; Real-time requirements; Design specifications; Fault tolerance; Case studies

1. Introduction

Process algebras are mathematical theories which model concurrent systems by their algebra and provide apparatus for reasoning about the structure and behaviour of the model. Examples include the calculus of communicating systems (CCS) [8,9], communicating sequential processes (CSP) [5,6], algebra of communicating processes (ACP) [1] and example process language (EPL) [4]. A system is characterized by its active components and their *interactions* or *communications*, between each other. Unlike Queuing Networks or Petri Nets there is no notion of *entity* or *flow* within a model. However, in recompense compositional reasoning, it is an intrinsic part of the language.

Both CCS and CSP could be two of the most representative examples of classical process algebras, thus, in the eighties fundamentals in process algebras were firmly established, but the proliferation of distributed, real-time and fault-tolerant systems has generated a great interest in the definition of timed and probabilistic extensions of these models.

Due to this fact, we find that currently it is not enough to consider qualitative aspects of concurrent systems. Thus, several researchers have studied possible extensions of these models by including some quantitative factors, like time restrictions, priorities and probabilities.

* Corresponding author.
E-mail addresses: fpelayo@dsi.uclm.es (F.L. Pelayo), mpelayo@dsi.uclm.es (M.L. Pelayo), juan.garcia@upct.es (J.L.G. Guirao).

0377-0427/$ - see front matter © 2006 Published by Elsevier B.V.
doi:10.1016/j.cam.2006.04.054

Probabilistic extensions of process algebras were introduced in the nineties in order to consider not only qualitative aspects of concurrent systems but also quantitative ones. Among several models, we can find PNAL, an algebraic language which extends Hennessy's EPL by means of a probabilistic choice; This language combines both kinds of choice, non-deterministic and probabilistic ones, it is presented in [2] and fully described in [3], where an operational, a denotational and a testing semantics have been defined by that language. The language used in this paper, **ROSA**(**R**easoning **O**n **S**tochastic **A**lgebras), has some functional similarities with PNAL.

Once an enough number of probabilistic models have been presented, the need to take into account performance aspects in the design of concurrent systems arises. It is true that often efficiency considerations are forgotten until the system works properly, i.e., once it has been fully tested. But, it becomes obvious that there are some systems (real-time systems) in which we need to know a priori if the system will fulfil with its restrictions. Then, a model for describing concurrent systems should also include some capabilities to determine efficiency considerations.

The Markovian process algebra **ROSA**, has a great expressive power, as it is able to describe and analyse processes with non-deterministic, probabilistic and temporal behaviours. Indeed, time considered is exponentially distributed because of its good properties (memory-less), specially when defining both the interleaving semantics for **ROSA** and the performance evaluation algorithm. This language captures the non-determinism in the choices which cannot be quantified, and, its interpretation of probability is generative. **ROSA** assumes the cooperation of processes by adding the involved rates and assumes the synchronization by choosing the minimum rate (the slowest process will impose its time).

ROSA [11,12] follows some ideas presented in [10]; it does not impose any syntactical restrictions on the components of a parallel operator. Thus, the parameter λ associated to any action does not limit its capabilities for a synchronization, and thus, the specification labour becomes easier than in some other models.

This language has been already fully described by means of an operational and a denotational semantics (which provides its operational and equivalent behaviour) in order to provide the support for the various phases in a functional study of **ROSA** processes. Over this issue a couple of programs has been developed for allowing to pass from a **ROSA** process to its corresponding semantic graph, and so for knowing how does the process evolve when time elapses:

- Syntactic Parser (C program):
  - INPUT: **ROSA** process
  - OUTPUT: Syntactic tree
- Semantics Analyser (*Mathematica* program):
  - INPUT: Syntactic tree
  - OUTPUT: Transition graph

An evaluation algorithm has been also presented for **ROSA** and it is planned to implement it (this evaluation algorithm makes possible to obtain probabilistic and timing performance measures over concurrent processes).

This paper is structured as follows:

- Section 2 describes the syntax of **ROSA**.
- Section 3 describes the operational semantics of **ROSA**.
- Section 4 describes the Alternating Bit Protocol and its "specification" in this process algebra.
- Section 5 shows how the software presented works from the ABP specification to its corresponding transition graph.
- Section 6 compiles the conclusions and future work in this field.

2. Syntax of ROSA

Let $\Delta = \{a, b, c, \ldots\}$ be a finite set of action types. Let $Id = \{X, Y, Z, \ldots\}$ be a finite set of variables of process. We will denote by the latest letters of the latin alphabet $r, s, t, \ldots$ probabilities. We will denote by greek letters $\alpha, \beta, \gamma, \ldots$ time parameters for actions. Terms of **ROSA** are defined by the following BNF expression:

$$P ::= \mathbf{0} | X | a.P | \langle a, \lambda \rangle.P | P \oplus P | P + P | P \oplus_r P | P \|_A P | recX.P,$$

where $r \in [0, 1]$, $\lambda \in \mathbb{R}^+ - \{0\}$, $A \subseteq \Delta$, $a \in \Delta$, $X \in Id$ and P is a process of **ROSA**.

The Algebra induced by this expression makes up the set of **ROSA** processes.

ROSA $= \{P, Q, P', Q', \ldots\}$.

The informal interpretation of the **ROSA** operators follows:

0: The process that cannot execute any action.

The appearance of **0** as the last term of any syntactic term will be understood.

X: The variable of process.

$a.P$: The classical prefix operator. Given a process P and an action a, $a.P$ executes the action a and then behaves like the process P.

$\langle a, \lambda\rangle.P$: The timed prefix operator, where the action labelled with a has a duration following a negative exponential random distribution with parameter λ and prefixes the process P.

Let us observe that $a.P$ is now a particular case of this operator, when we take $\lambda{=}\infty$, which means that $\mathbf{E}[\mathrm{Exp}[\infty]]{=}0$, i.e., this action has an average duration of 0, it is an immediate action with type a.

$P \oplus Q$: The internal choice operator. Given the processes P and Q, $P \oplus Q$ is the process which can behave like P or like Q as an internal decision of the system.

$P + Q$: The external choice operator. Given the processes P and Q, $P + Q$ is the process which can behave like P or like Q as asked from the environment.

$P \oplus_r Q$: The probabilistic choice operator with the classical generative meaning. Given the processes P and Q and the probability r, $P \oplus_r Q$ is the process which behaves like P with probability r and behaves like Q with probability $1 - r$.

The external environment has no influence on this choice.

$P \|_A Q$: The parallel operator synchronizing actions whose types belong to A. Given the processes P and Q and $A \subseteq \Lambda$, $P \|_A Q$ is the process which asynchronously executes actions form P or Q whose types do not belong to A, but when the actions have their types in A they must be executed at the same time by both processes.

$recX.P$: The classical recursion operator in which occurrences of X are substituted by $recX.P$.

3. Operational semantics of ROSA

Operational semantics gives to a process its meaning represented by a labelled transition system which shows how the process is transformed into other(s) process(es), thus a operational semantics is made up by every possible transformation of all the language terms.

On the other hand, the operational semantics gives a precise interpretation of the operators presented when defining the syntax of the language.

The operational semantics is defined in the Plotkin [13] and Milner [9] style by using a labelled transition system with three kinds of transitions:

- A non-deterministic transition is a tuple $\langle P, Q\rangle$ where P and Q are processes. This transition will be represented by

  $$P \longrightarrow Q.$$

  It means that P can evolve immediately (they do not take any time at all) and behaves like Q. We will also say that process P can evolve internally to process Q.

  This transition captures the internal decisions that the system makes for resolving the non-deterministic choices which can come either from internal choice operators or from competitions between actions with similar rates.
- A probabilistic transition is a tuple $\langle P, Q, r\rangle$ where P and Q are processes and $r \in [0, 1]$. This transition will be represented by

  $$P \longrightarrow_r Q.$$

  This means that P can evolve immediately and behaves like Q with probability r. We will also call the process P can evolve probabilistically to process Q with probability r or that process P can evolve with probability r to process Q.

- An action transition is a tuple $\langle P, Q, a, \lambda\rangle$ where P and Q are processes $a \in \Delta$ and $\lambda \in \mathbb{R}^+ - \{0\}$. This transition will be represented by

$$P \xrightarrow{a,\lambda} Q.$$

It means that P can evolve by executing the action labelled by a, taking a time described by an exponential random distribution with parameter λ, and then behaves like Q. We will also say that process P can evolve to process Q by executing the action $\langle a, \lambda\rangle$.

Before giving the rules of the operational semantics four auxiliary functions are needed:

- *Action* is a boolean function defined on the set of **ROSA** processes which gives TRUE for those processes that can only evolve by means of *action transitions* and FALSE otherwise:

$$\textbf{Action} : \{\textbf{ROSA}\ \textit{processes}\} \longrightarrow \{\textit{TRUE}, \ \textit{FALSE}\}.$$

This function has been defined by Gavin Lowe in [7] and it is used in order to establish the precedence between probabilistic transitions and action transitions.

- *Available* is a function defined on the set of **ROSA** processes which associates to every process its multiset of available actions, it is defined in a structural way on the set of **ROSA** processes:

$$\begin{array}{lllll}
\textbf{Available}: & \{\textbf{ROSA}\textit{procs.}\} & \longrightarrow & \mathscr{M}[\Delta \times \mathbb{R}^+ - \{0\}] \\
& \textbf{0} & \mapsto & \varnothing \\
& X & \mapsto & \varnothing \\
& a.P & \mapsto & \{a\} \\
& \langle a, \lambda\rangle.P & \mapsto & \{\langle a, \lambda\rangle\} \\
& P \oplus Q & \mapsto & \mathbf{A}[P] \cup \mathbf{A}[Q] \\
& P + Q & \mapsto & \mathbf{A}[P] \cup \mathbf{A}[Q] \\
& P \oplus_r Q & \mapsto & \mathbf{A}[P] \cup \mathbf{A}[Q] \\
& P \|_A Q & \mapsto & ((\mathbf{A}[P] \cup \mathbf{A}[Q]) \setminus A) \cup (\mathbf{A}[P] \cap \mathbf{A}[Q]) \\
& recX : P & \mapsto & \mathbf{A}[P].
\end{array}$$

- *Deterministic Stability*, **DS** is a boolean function defined on the set of **ROSA** processes which gives TRUE for those processes that cannot evolve by means of *non-deterministic transitions* and FALSE otherwise:

$$\begin{array}{lllll}
\textbf{DS}: & \{\textbf{ROSA}\textit{procs.}\} & \longrightarrow & \{\textit{TRUE}, \textit{FALSE}\} \\
& \textbf{0} & \mapsto & \textit{TRUE} \\
& X & \mapsto & \textit{TRUE} \\
& a.P & \mapsto & \textit{TRUE} \\
& \langle a, \lambda\rangle.P & \mapsto & \textit{TRUE} \\
& P \oplus Q & \mapsto & \textit{FALSE} \\
& P + Q & \mapsto & \mathbf{DS}[P] \wedge \mathbf{DS}[Q] \\
& P \oplus_r Q & \mapsto & \mathbf{DS}[P] \wedge \mathbf{DS}[Q] \\
& P \|_A Q & \mapsto & \mathbf{DS}[P] \wedge \mathbf{DS}[Q] \\
& recX : P & \mapsto & \mathbf{DS}[P].
\end{array}$$

- *Type* is a function defined on the set of multisets of actions which associates to every multiset of actions, the set consisting of their action types:

$$\textbf{Type} : \mathscr{M}[\Delta \times \mathbb{R}^+ - \{0\}] \longrightarrow \mathscr{P}[\Delta].$$

Once defined all these functions, the rules defining the behaviour of **ROSA** processes are presented in a detailed way. They will be divided into three tables according to the transition type concerned.

Table 1
Non-deterministic transition rules

(ND-Def)	$\frac{}{P \oplus Q \to P}$	**(ND-Def)**	$\frac{}{P \oplus Q \to Q}$
(ND-Ext)	$\frac{P \to P'}{P+Q \to P'+Q}$	**(ND-Ext)**	$\frac{Q \to Q'}{P+Q \to P+Q'}$
(ND-Pro)	$\frac{P \to P'}{P \oplus_r Q \to P' \oplus_r Q}$	**(ND-Pro)**	$\frac{Q \to Q'}{P \oplus_r Q \to P \oplus_r Q'}$
(ND-Par)	$\frac{P \to P'}{P \parallel_A Q \to P' \parallel_A Q}$	**(ND-Par)**	$\frac{Q \to Q'}{P \parallel_A Q \to P \parallel_A Q'}$
(ND-Rec)	$\frac{}{recX.P \to P[recX.P/X]}$		

Table 2
Probabilistic transition rules assuming $\mathbf{DS}[P] \wedge \mathbf{DS}[Q]$

(P-Def)	$\frac{}{P \oplus_r Q \to_r P}$	**(P-Def)**	$\frac{}{P \oplus_r Q \to_{1-r} Q}$
(P-Ext)	$\frac{P \to_r P' \wedge \mathbf{Action}[Q]}{P+Q \to_r P'+Q}$	**(P-Ext)**	$\frac{Q \to_t Q' \wedge \mathbf{Action}[P]}{P+Q \to_t P+Q'}$
(P-Par)	$\frac{P \to_r P' \wedge \mathbf{Action}[Q]}{P \parallel_A Q \to_r P' \parallel_A Q}$	**(P-Par)**	$\frac{Q \to_t Q' \wedge \mathbf{Action}[P]}{P \parallel_A Q \to_t P \parallel_A Q'}$
(P-BothExt)	$\frac{P \to_r P' \wedge Q \to_t Q'}{P+Q \to_{r \cdot t} P'+Q'}$	**(P-BothPar)**	$\frac{P \to_r P' \wedge Q \to_t Q'}{P \parallel_A Q \to_{r \cdot t} P' \parallel_A Q'}$

Table 3
Action transition rules assuming $\mathbf{DS}[P] \wedge \mathbf{DS}[Q]$

(A-Def)	$\frac{}{a.P \overset{a,\infty}{\to} P}$
(A-Def)	$\frac{}{\langle a,\lambda \rangle.P \overset{a,\lambda}{\to} P}$
(A-Ext)	$\frac{P \overset{a,\lambda}{\to} P' \wedge a \notin \mathbf{Type}[\mathbf{Available}[Q]]}{P+Q \overset{a,\lambda}{\to} P'}$
(A-Ext)	$\frac{Q \overset{a,\lambda}{\to} Q' \wedge a \notin \mathbf{Type}[\mathbf{Available}[P]]}{P+Q \overset{a,\lambda}{\to} Q'}$
(A-Par)	$\frac{P \overset{a,\lambda}{\to} P' \wedge a \notin \mathbf{Type}[\mathbf{Available}[Q]] \cup A}{P \parallel_A Q \overset{a,\lambda}{\to} P' \parallel_A Q}$
(A-Par)	$\frac{Q \overset{a,\lambda}{\to} Q' \wedge a \notin \mathbf{Type}[\mathbf{Available}[P]] \cup A}{P \parallel_A Q \overset{a,\lambda}{\to} P \parallel_A Q'}$
(A-RaceExt)	$\frac{P \overset{a,\infty}{\to} P' \wedge Q \overset{a,\lambda}{\to} Q' \wedge \lambda \neq \infty}{P+Q \overset{a,\infty}{\to} P'}$
(A-RaceExt)	$\frac{P \overset{a,\lambda}{\to} P' \wedge Q \overset{a,\infty}{\to} Q' \wedge \lambda \neq \infty}{P+Q \overset{a,\infty}{\to} Q'}$
(A-RaceExtCoop)	$\frac{P \overset{a,\lambda_1}{\to} P' \wedge Q \overset{a,\lambda_2}{\to} Q' \wedge (\lambda_1 = \infty = \lambda_2 \vee \lambda_1 \neq \infty \neq \lambda_2)}{P+Q \overset{a,\lambda_1+\lambda_2}{\to} P' \oplus Q'}$
(A-RacePar)	$\frac{P \overset{a,\infty}{\to} P' \wedge Q \overset{a,\lambda}{\to} Q' \wedge \lambda \neq \infty \wedge a \notin A}{P \parallel_A Q \overset{a,\infty}{\to} P' \parallel_A Q}$
(A-RacePar)	$\frac{P \overset{a,\lambda}{\to} P' \wedge Q \overset{a,\infty}{\to} Q' \wedge \lambda \neq \infty \wedge a \notin A}{P \parallel_A Q \overset{a,\infty}{\to} P \parallel_A Q'}$
(A-RaceParCoop)	$\frac{P \overset{a,\lambda_1}{\to} P' \wedge Q \overset{a,\lambda_2}{\to} Q' \wedge (\lambda_1 = \infty = \lambda_2 \vee \lambda_1 \neq \infty \neq \lambda_2) \wedge a \notin A}{P \parallel_A Q \overset{a,\lambda_1+\lambda_2}{\to} P' \parallel_A Q \oplus P \parallel_A Q'}$
(A-Syn)	$\frac{P \overset{a,\lambda_1}{\to} P' \wedge Q \overset{a,\lambda_2}{\to} Q' \wedge a \in A}{P \parallel_A Q \overset{a,\mathbf{min}[\lambda_1,\lambda_2)]}{\to} P' \parallel_A Q'}$

Table 1 contains the inference rules for non-deterministic transitions.

These *non-deterministic* or *internal rules* are quite obvious, just note the precedence of the internal choice operator over the external choice, the probabilistic choice, the parallel and the recursion operators.

Therefore, for all the remainders rules, probabilistic and action ones require the involved processes both P and Q to be *deterministically stable*, i.e., $\mathbf{DS}[P] \wedge \mathbf{DS}[Q]$.

The rules for probabilistic transitions are presented in Table 2.

The operational semantics of **ROSA** is defined as the minimum multiset of transitions (non-deterministic, probabilistic and action transitions) we can derive by using the rules in Tables 1, 2 and 3, allowing a transition to appear as many times as different ways exist in order to obtain it.

4. Specification and analysis of the alternating bit protocol

The alternating bit protocol is a communication protocol that guarantees a secure communication between two stations: a sender and a receiver, through a channel which can fail. The protocol works by adding a bit to the messages, which allows us to control the possible loss/corruption of them.

We study the most general scenario for this protocol consisting of one sender, one receiver and one channel, which can fail with a known probability, p.

Here, we must consider the process, activator, which controls the alternation of the bit added to the messages. This activator and the sender must synchronize at in_j (which corresponds to add the bit j to the message, $j \in \{0, 1\}$). The sender and the channel must synchronize at tm_j (transmission of message), at da_j (deliver of acknowledgment) and at λ'', and the channel and the receiver must synchronize at dm_j (deliver of message), at λ, at ta_j (transmission of acknowledgment) and at λ'.

For the sake of clarity we have preferred to divide the channel into two processes, (Sender$\longrightarrow$Receiver) Go and (in the other direction) $Ba(ck)$ as well as, the different error messages which appear after some delay if the channel has failed in any direction and will be denoted by *lambdas*.

The **ROSA** process modelling the ABP is the parallel composition of the following processes:

$$
\begin{array}{ll}
\text{Activator :} & A = in_0.in_1.A. \\
\text{Sender :} & S = in_0.S_0 + in_1.S_1 \\
& S_0 = \langle tm_0, \gamma\rangle.(da_0.S + \lambda''.S_0) \\
& S_1 = \langle tm_1, \gamma\rangle.(da_1.S + \lambda''.S_1). \\
\text{Channels :} & Go = \sum_{j=1}^{2} (tm_j.(\langle\lambda, \mu\rangle.Go \oplus_p \langle dm_j, \mu\rangle.Go)) \\
& Ba = \lambda'.\langle\lambda'', \mu\rangle.Ba \sum_{j=1}^{2} (ta_j.(\langle\lambda'', \mu\rangle.Ba \oplus_p \langle da_j, \mu\rangle.Ba)). \\
\text{Receiver :} & R = \lambda.\lambda'.R + dm_0.out.ta_0.RR + dm_1.ta_1.R \\
& RR = \lambda.\lambda'.RR + dm_0.ta_0.RR + dm_1.out.ta_1.R.
\end{array}
$$

Alternating bit protocol 2:

$$
\begin{aligned}
&ABP \equiv A\|_{A_2} S\|_{A_2} Go\|_{A_2} Ba\|_{A_2} R, \\
&A_2 = \{in_0, in_1, tm_0, tm_1, dm_0, dm_1, ta_0, ta_1, da_0, da_1, \lambda, \lambda', \lambda''\}.
\end{aligned}
$$

From the **ROSA** specification of the ABP the operational semantics allows us to construct the corresponding *transition graph*, which describes the behaviour of this protocol as time elapses.

Finally, the average time T to complete the communication of a message can be estimated from a graph 2, which is generated (our immediate future work in this field) from that generated by the operational semantics 1 and that will be computed by the *Mathematica* program here presented. In this final graph we have enumerated the states upside down and left to right, being S_0 the initial one, and S_F the final one:

$$
\begin{aligned}
&T_{S_0} = 1/\gamma + T_{S_1}, \\
&T_{S_1} = p(1/\mu + T_{S_2}) + (1-p)(1/\mu + T_{S_3}), \\
&T_{S_2} = 1/\mu + T_{S_0} = 1/\mu + 1/\gamma + T_{S_1}, \\
&T_{S_3} = (1-p)(1/\mu + T_{S_F}) + p(1/\mu + T_{S_4}) = (1-p)/\mu + p(1/\mu + T_{S_4}), \\
&T_{S_4} = 1/\gamma + T_{S_5}, \\
&T_{S_5} = p(1/\mu + T_{S_6}) + (1-p)(1/\mu + T_{S_3}),
\end{aligned}
$$

$$T_{S_6} = 1/\mu + T_{S_4},$$

$$\vdots$$

Hence

$$\mathrm{T} = \frac{1/\gamma + 2/\mu}{(1-p)^2}.$$

This average time to complete the transmission of a message is equal to the quotient between the sum of the necessary time to put the message into the channel, $1/\gamma$, plus the necessary time for completing the transmission by the channel in the two directions, $2 \cdot 1/\mu$, and, the probability with which the channel works properly (no failure in either direction), $(1-p)^2$.

5. Syntactic and semantics analysers

It has been developed a syntactic parser implemented over *C language* which takes as input the **ROSA** word which represents the process to study. In our case for the alternating bit protocol is

$$ABP \equiv A\|_{A_2} S\|_{A_2} Go\|_{A_2} Ba\|_{A_2} R,$$
$$A_2 = \{in_0, in_1, tm_0, tm_1, dm_0, dm_1, ta_0, ta_1, da_0, da_1, \lambda, \lambda', \lambda''\},$$

where

$$\begin{aligned}
&\text{Activator :} && A = in_0.in_1.A.\\
&\text{Sender :} && S = in_0.S_0 + in_1.S_1\\
&&& S_0 = \langle tm_0, \gamma\rangle.(da_0.S + \lambda''.S_0)\\
&&& S_1 = \langle tm_1, \gamma\rangle.(da_1.S + \lambda''.S_1).\\
&\text{Channels :} && Go = \sum_{j=1}^{2} (tm_j.(\langle\lambda, \mu\rangle.Go \oplus_p \langle dm_j, \mu\rangle.Go))\\
&&& Ba = \lambda'.\langle\lambda'', \mu\rangle.Ba \sum_{j=1}^{2} (ta_j.(\langle\lambda'', \mu\rangle.Ba \oplus_p \langle da_j, \mu\rangle.Ba)).\\
&\text{Receiver :} && R = \lambda.\lambda'.R + dm_0.out.ta_0.RR + dm_1.ta_1.R\\
&&& RR = \lambda.\lambda'.RR + dm_0.ta_0.RR + dm_1.out.ta_1.R.
\end{aligned}$$

To begin with, we have to rename some terms of the original expression, due to some restrictions on the possible range of names for actions or using capital letters for name of processes or, etc.

- $A = in_0.in_1.A \Rightarrow A = \mu Act(a.b.Act)$.
- $S = in_0.S_0 + in_1.S_1 \cdots \Rightarrow S = \mu Send..(((a.\mu Send_0.(\langle c, 0.5\rangle.(i + m.Send_0))) + (b.\mu Send_1.(\langle d, 0.5\rangle.(j + m.Send_1))))).Send)$.
- Similarly with processes Go, Ba, R and RR.

The OUTPUT generated by our C program is the following plain text sequence:

0 Synchronization 0 0 abcdefghijklm 0 00 Recursion Act 0 0 0 001 Prefix 0 0 0 0 0010 Action 0 0 a 0 0011 Prefix 0 0 0 0 00110 Action 0 0 b 0 00111 Variable Recursion Act 0 0 0 01 Synchronization 0 0 abcdefghijklm 0 010 Recursion Send 0 0 0 0101 Prefix 0 0 0 0 01010 External Choice 0 0 0 0 010100 Prefix 0 0 0 0 0101000 Action 0 0 a 0 0101001 Recursion Send0 0 0 0 01010011 Prefix 0 0 0 0 010100110 Action 0 0 c 0.5 010100111 External Choice 0 0 0 0 0101001110 Action 0 0 i 0 0101001111 Prefix 0 0 0 0 01010011110 Action 0 0 m 0 01010011111 Recursion Variable Send0 0 0 0 010101 Prefix 0 0 0 0 0101010 Action 0 0 b 0 0101011 Recursion Send1 0 0 0 01010111 Prefix 0 0 0 0 010101110 Action 0 0 d 0.5 010101111 External Choice 0 0 0 0 0101011110 Action 0 0 j 0 0101011111 Prefix 0 0 0 0 01010111110 Action 0 0 m 0 01010111111 Recursion Variable Send1 0 0 0 01011 Recursion Variable Send 0 0 0 011 Synchronization 0 0 abcdefghijklm 0 0110 Recursion Go 0 0 0 01101 Prefix 0 0 0 0 011010 External Choice 0 0 0 0 0110100 Prefix 0 0 0 0 01101000 Action 0 0 c 0 01101001 Probabilistic Choice 0 r 0 0 011010010 Action 0 0 k 0.5 011010011 Action 0

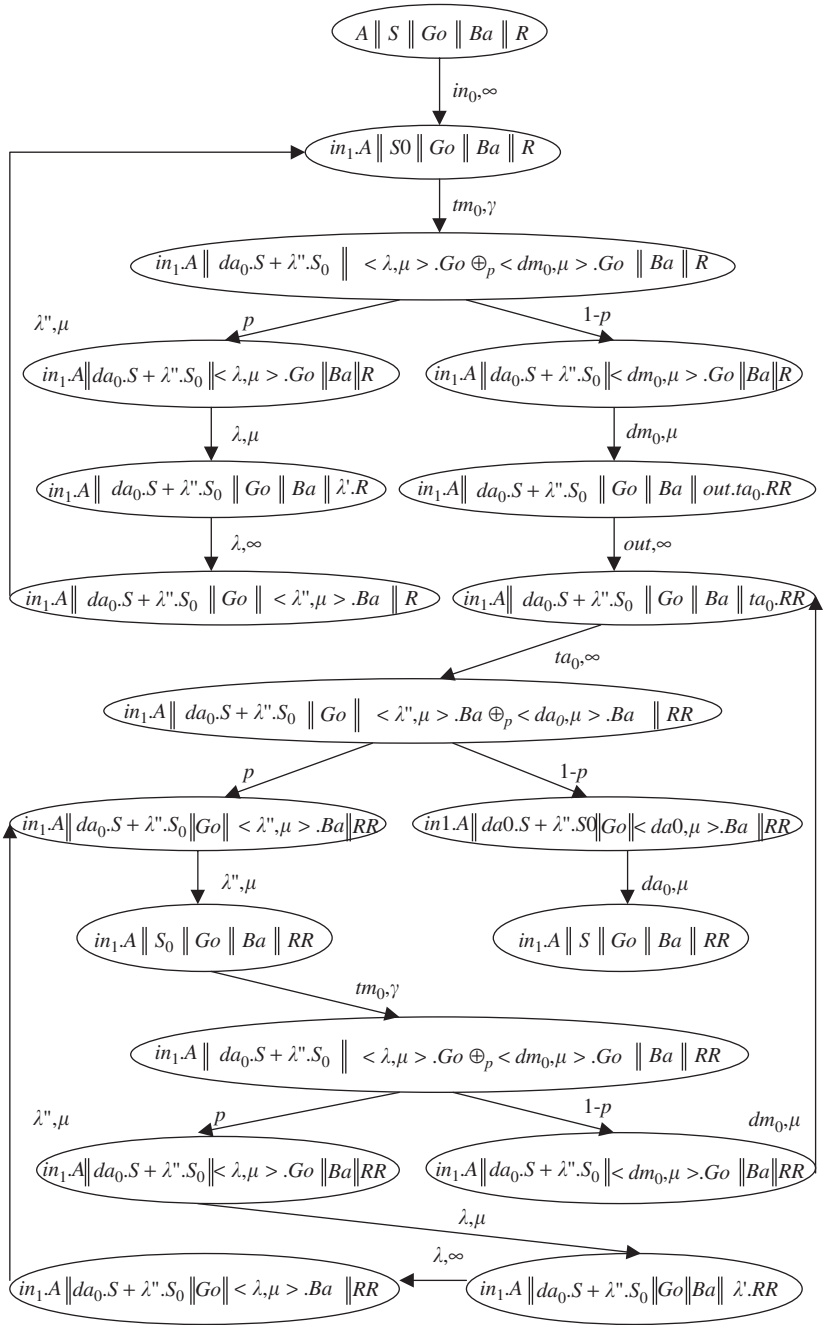


Fig. 1. ABP—*transition graph*.

0 e 0.5 0110101 Prefix 0 0 0 0 01101010 Action 0 0 d 0 01101011 Probabilistic Choice 0 r 0 0 011010110 Action 0 0 k 0.5 011010111 Action 0 0 f 0.5 011011 Probabilistic Choice Go 0 0 0 0111 Synchronization 0 0 0abcdefghijklm 0 01110 Recursion Back 0 0 0 011101 Prefix 0 0 0 0 0111010 External Choice 0 0 0 0 01110100 External Choice 0 0 0 0 011101000 Prefix 0 0 0 0 0111010000 Action 0 0 g 0 0111010001 Probabilistic Choice 0 r 0 0 01110100010 Action 0 0 m 0.5 01110100011 Action 0 0 i 0.5 011101001 Prefix 0 0 0 0 0111010010 Action 0 0 l 0 0111010011 Action 0 0 m 0.5 01110101 External Choice 0 0 0 0 011101010 Prefix 0 0 0 0 0111010100 Action 0 0 h 0 0111010101 Probabilistic Choice 0 r 0 0 01110101010 Action 0 0 m 0.5 01110101011 Action 0 0 j 0.5 011101011 Prefix 0 0 0 0

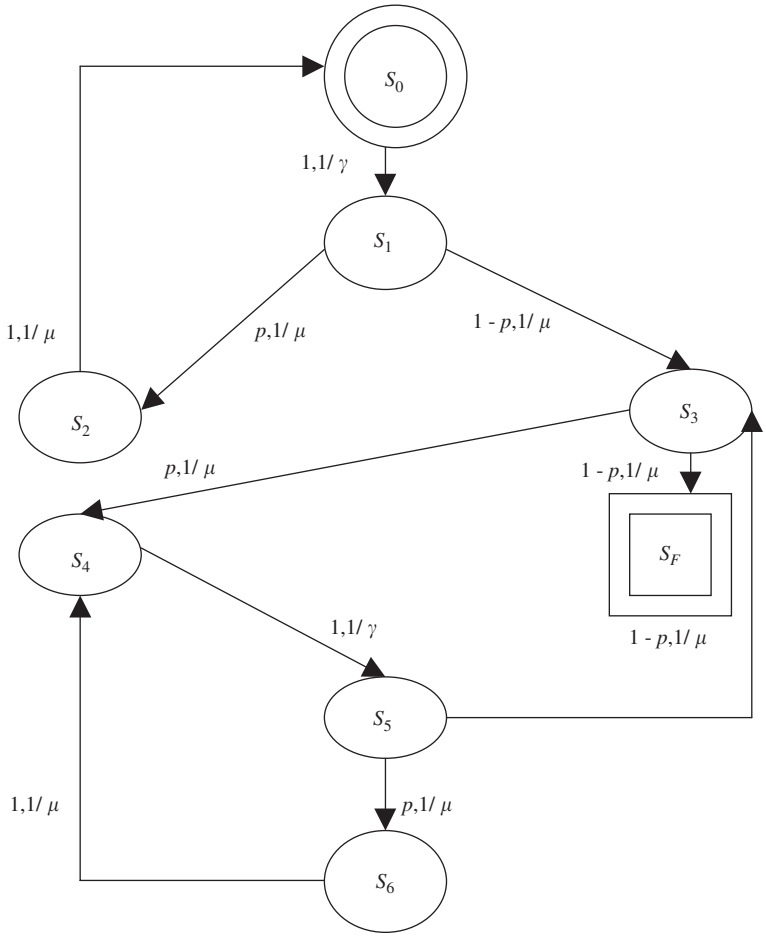


Fig. 2. ABP—final graph.

0111010110 Action 0 0 1 0 0111010111 Action 0 0 m 0.5 0111011 Recursion Variable Back 0 0 0 01111 Recursion R 0 0 0 011111 Prefix 0 0 0 0 0111110 External Choice 0 0 0 0 01111100 Prefix 0 0 0 0 011111000 Action 0 0 k 0 011111001 Action 0 0 l 0 01111101 External Choice 0 0 0 0 011111010 Prefix 0 0 0 0 0111110100 Action 0 0 e 0 0111110101 Prefix 0 0 0 0 01111101010 Action 0 0 h 0 01111101011 Prefix 0 0 0 0 011111010110 Action 0 0 g 0 011111010111 Recursion RR 0 0 0 0111110101111 External Choice 0 0 0 0 01111101011110 Prefix 0 0 0 0 011111010111100 External Choice 0 0 0 0 0111110101111000 Prefix 0 0 0 0 01111101011110000 Action 0 0 k 0 01111101011110001 Action 0 0 l 0 0111110101111001 Prefix 0 0 0 0 01111101011110010 Action 0 0 e 0 01111101011110011 Action 0 0 g 0 011111010111101 Recursion Variable RR 0 0 0 01111101011111 Prefix 0 0 0 0 011111010111110 Action 0 0 f 0 011111010111111 Prefix 0 0 0 0 0111110101111110 Action 0 0 n 0 0111110101111111 Action 0 0 h 0 011111011 Prefix 0 0 0 0 0111110110 Action 0 0 f 0 0111110111 Action 0 0 h 0 0111111 Recursion Variable R 0 0 0

This sequence can be easily read by *Mathematica*, by considering it as a list formed of 6-tuple "NODE-id Operator Recursion Prob ACTION-id Time", therefore the sequence will be understood as:

0 Synchronization 0 0 abcdefghijklm 0

The node 0 (TOP, first level), is a synchronization, with 0 recursion (non-recursive), with 0 probability (non-probabilistic, e.g., deterministic), with the set of synchronization actions formed of $\{a, b, c, d, e, f, g, h, i, j, k, l, m\}$, which represents $\{in_0, in_1, tm_0, \ldots\}$, and with 0 associated temporal cost.

00 Recursion Act 0 0 0

The node 00 (Second level, left hand side), is a recursion one, with the recursion over Act, with 0 probability (non-probabilistic, e.g. deterministic), with the empty set of synchronization, and with 0 associated temporal cost.

. . . And so on.

Taking this "ugly" OUTPUT, as the INPUT for *Mathematica*, our program applies the operational semantics rules of **ROSA** and generates a plain representation of the transition graph associated to the ABP, from which (after renaming the actions $a, b, \ldots$ with the original ones and using the appropriate *package*) produces the graph of Fig.1 (see also Fig. 2).

6. Conclusions and future work

We have presented a couple of programs for generating the semantics graph associated to any **ROSA** process.

We plan to implement software able to reduce the graphs so generated, and then to produce as final output an interval for the time necessary to evolve from one state to another.

Acknowledgments

Research partially supported by the Spanish MCYT project TIC2003-07848-C02 and the Junta de Castilla-La Mancha project PAC-03-001.

References

[1] J. Bergstra, J.W. Klop, Process algebra for synchronous communication, Inform. Control 31 (3) (1984) 109–137.
[2] D. Cazorla, F. Cuartero, V. Valero, F.L. Pelayo, A process algebra for probabilistic and nondeterministic processes, Inform. Process. Lett. 80 (1) (2001) 15–23.
[3] D. Cazorla, F. Cuartero, V. Valero, F.L. Pelayo, J.J. Pardo, Algebriac theory of probabilistic and nondeterministic processes, J. Logic Algebraic Programming 55 (1–2) (2003) 57–103.
[4] M. Hennessy, Algebraic Theory of Processes, MIT Press, Cambridge, MA, 1988.
[5] C.A.R. Hoare, Communicating sequential processes, Comm. ACM 21 (8) (1978) 666–677.
[6] C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall, Englewood Cliffs, NJ, 1985.
[7] G. Lowe, Representing non-deterministic and probabilistic behaviour in reactive processes, Technical Report PRG-TR-11-93, Oxford University Computing Laboratory, 1993.
[8] R. Milner, A Calculus of Communicating Systems, Lecture Notes in Computer Science, vol. 92, Springer, Berlin, 1980.
[9] R. Milner, Communication and Concurrency, Prentice-Hall, Englewood Cliffs, NJ, 1989.
[10] F.L. Pelayo, F. Cuartero, V. Valero, D. Cazorla, PPNAL: performance evaluation in an algebraic model for probabilistic an non-deterministic processes, in: Proceedings of 15th Annual UK Performance Engineering Workshop (UKPEW'99), Bristol, UK, 1999, pp. 1–12.
[11] F.L. Pelayo, F. Cuartero, V. Valero, D. Cazorla, An example of performance evaluation by using the stochastic process algebra: ROSA, in: Proceedings of the Seventh IEEE International Conference on Real-Time Computing Systems and Applications (RTCSA'2000), IEEE Computer Society Press, Cheju Island, South Korea, 2000, pp. 271–278.
[12] F.L. Pelayo, F. Cuartero, V. Valero, D. Cazorla, Analysis of the MPEG-2 encoding algorithm with ROSA, Electronic Notes on Theoretical Computer Science 80 (1) (2003) 185–202 ⟨http://www.elsevier.nl/locate/entcs/volume80.html⟩.
[13] G.D. Plotkin, A structural approach to operational semantics, Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.