ROSA Analyser: An automatized approach to analyse processes of ROSA

Raúl Pardo and Fernando L. Pelayo
Dept. de Sistemas Informáticos
E. Superior de Ingeniería Informática de Albacete
Universidad de Castilla - La Mancha
Campus Universitario. 02071-Albacete, Spain
`Raul.Pardo@alu.uclm.es, FernandoL.Pelayo@uclm.es`

In this work we present the first version of ROSA Analyser, a tool in order to get closer to a fully automatic process of analysing the behaviour of a system specified as a process of the Markovian Process Algebra ROSA. In this first development stage, ROSA Analyser is able to generate the Labelled Transition System, according to ROSA Operational Semantics.

ROSA Analyser performance starts with the Syntactic Analysis so generating a layered structure, suitable to then, apply the Operational Semantics Transition rules in the easier way. ROSA Analyser is able to recognize some states identities deeper than the Syntactic ones. This is the very first step in the way to reduce the size of the LTS and then to avoid the explosion of states problem, so making this task more tractable.

For the sake of better illustrating the usefulness of ROSA Analyser, a case study is also provided within this work.

1 Introduction

Formal methods are more used as Computer Science becomes a more mature science; This happens due to the fact that formal methods provides software designers with a way to guarantee high security and reliability levels for their computer science systems, and what is more, formal methods would allow to find software errors in the earliest stage of the software development life cycle so making it significatively cheaper. The main problem is that for systems with a size or complexity level, the analysis could be difficult to be developed, mainly because of two reasons, the first comes from the size of the graphical model usually generated, the second is because the most times these analyses are made by hand; We begin with the latter so presenting a tool for automatically apply the operational semantics of the Process Algebra, ROSA. In the next future we will continue dealing with the problem of the size of the LTS generated by ROSA.

During the past 20 years many efforts have been done in order to develop support tools for all kinds of formal methods. For instance for timed automatons UPPAAL tool [7] gives to designers a good environment to design by using timed automatons, as well as, it supports analyzing skills. In the Petri Nets field, TINA is a frequently used tool [1]. Regarding Process Algebras, several tools have been developed also, the most known could be PEPA Workbench [2] which was developed based on PEPA process algebra [3]. The main problem of process algebra based tools is to deal with the *explosion of states* arisen in the Labelled Transition System (LTS), because of this, some of the process algebra research lines are searching the way to reduce the size of the LTS to be generated, and so to avoid this problem as much as possible [8].

In this paper we present a tool based on ROSA process algebra [4], so this tool has to deal with non-deterministic, probabilistic and temporal behaviours for a given process, which means that it has to

Submitted to:
WS-FMDS 2012

provide a proper syntax for capturing all these features. Moreover, an easy and clear user interface has been designed for it which has been no very common in previously developed tools.

In its first development stage ROSA Analyser is able to build the LTS from a given ROSA process, by means of applying all of the possible operational Semantics rules which must be used for a given input process. In addition, for the sake of avoiding the *explosion of states* problem, we are working on defining an heuristic which allows to reach from each state/process the most probably path to a given (final) state/process, by changing its exponential computational cost to a lineal computational cost. Therefore, it involves an initial attempt to solve this problem and as a consequence of this, get a more practical way to analyse concurrent and real time systems by means of process algebra based tools.

The paper is structured as follows, section 2 provides a detailed description of ROSA Analyser, pointing out the type of used data structures and the analyzing process which makes possible the LTS building; in section 3 a case of use of it is presented, specifically it has been analyzed a cognitive memorizing process. To finish with, section 4 states both conclusions and future work.

2 ROSA Analyser tool

ROSA Analyser has been developed in JAVA, this choice was taken due to the facilities to handle with complex data structures, by which is characterized this programming language. In addition, ROSA Analyser is implemented over GPL v2 license, in order to allow everybody the possibility to work on the defined data structures. The source code and the executable version can be found in `http://kenai.com/projects/semantictreerosa`.

In order to show the way in which ROSA Analyser works, we will describe all of the main parts involved in the LTS generator. Specifically, in forwards paragraphs are detailed the data structures over ROSA analyser works, the syntax analyser and the semantics analyser in which is located the main work load due to this process have to build the whole LTS starting from the layered syntax expression of the input ROSA process.

2.1 Data structures

The theoretical process expressions with which ROSA Analyser works are not optimized for the analysis in which the tool has to be involved, due to this fact, it arises the need of parsing the data reaching a better way in its analysis processes. Specifically, in this section we show the two main data structure with which the tool works Syntax and Semantics trees.

2.1.1 Syntax Tree

As a result of the syntax analysis is built a binary tree with which can be made an easy and efficient Semantics analysis. In this Syntax analysis the higher syntactical priority the elements have, the closer to the top they are located, as previously said this structure is optimized to check the conditions of the upper side of the rules and to do the necessary modifications to the process which is being analysed.

In order to show in a clear way this data structure it can be seen an example of the syntax tree of the ROSA process $< a, 0.3 > .0 ||_{\{a,c\}} < b, \infty > .0$ in figure 1.

In the example above, we can see the root node corresponded with the highest priority element of the process, also this structure allows the semantics analysis to get directly the element which chooses the rule to be applied, and by means of defined methods also make possible change the Syntax tree so as to apply the selected rule. Detailed description about the building of the Syntrax tree will be provided.

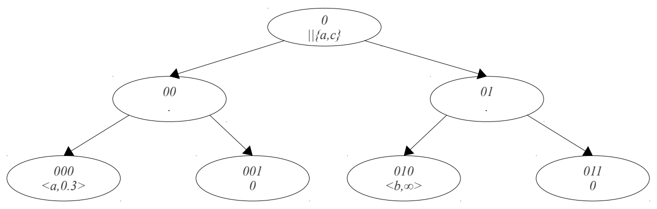

Figure 1: Syntax Tree Example

2.1.2 Semantics Tree

Once we have defined a proper structure for the process' syntax we are ready to show the Semantics tree data structure which represents the LTS for a ROSA process. The LTS is built with nodes which are themselves ROSA processes, therefore in each Semantics tree node will appear a Syntax tree, this can show the high complexity by which is characterized this data structure. Also, unlike the Syntax tree, the transitions between nodes must get more information than those of the Syntax tree which further increase the amount of data to be handle.

In this case the defined tree structure is chosen with the aim to support the natural structure of the Semantics tree, which is a n-ario tree, unlike the previous data structure which was chosen looking for making easier the Semantics analysis. A Semantics tree example can be seen in figure 2.

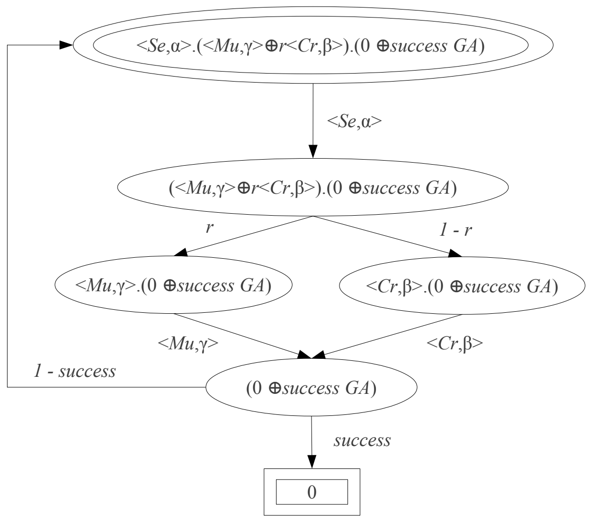

Figure 2: Semantics Tree Example

It is important to point out that this data structure also has several methods in which has been implemented some functionalities which helps to build it, for instance, joining new nodes to the Semantics tree, where in addition a search for Syntax equivalent nodes is made, everytime that a tentative “new” node is generated.

2.2 Syntax Analysis

2.2.1 Tool input Syntax

The Syntax analysis consists of the part of the tool which takes as input the plain expression of a ROSA process and returns the Syntax tree associated with that process. It was necessary to define some constrains for the input syntax because the ASCII characters do not provide the wide variety of elements by which is characterized ROSA Syntax. In the following table 1 could be seen the equivalences between ROSA Syntax and the tool input Syntax:

ROSA Syntax	**Tool Syntax**
a	a
$< a, \alpha >$	$< a, \alpha >$
$a.P$	$a.P$
$< a, \alpha > .P$	$< a, \alpha > .P$
P	P
$P;Q$	$P;Q$
$P \oplus Q$	$P - Q$
$P + Q$	$P + Q$
$P \oplus_r Q$	$P * \{r\}Q$
$P\|\|_A Q$	$P\|\|A\ Q$

Table 1: Equivalence between ROSA and tool Syntax

Moreover, it is important to point out that the parameter α capturing the temporal behaviour of actions must be a real number, and the parameter r, a probability, must be a real number within the interval $[0,1]$. The set A represents the synchronization set for a parallel operator, so that this set is $A = \{a,b,c,\ldots\}$.

2.2.2 Syntax tree building

Once the ROSA Analyser input method has been defined we are ready to describe the Syntax analysis, the first point to take into account is the priority of the operators, by default the priority is defined as follows:

1. Sequential processes operator.
2. Parallel operator.
3. External, internal and probabilistic choices.
4. Prefix.
5. Actions and process variables.

This priorities can be changed according to our needs by means of using parenthesis, according to their common behaviour, becoming a more flexible Syntax for the input method.

Taking into account the priorities above defined, the Syntax analysis is mainly a recursive method in which the expression element with the highest priority must be found, then a node with the element is created or added to the Syntax tree, once reached this point if the element is an operator, the method split the expression into two parts taking as split point the position in which was located the operator found, then the same method is called using both expressions, and the new elements found in this recursive calls are added to the Syntax tree which is being built, on the other hand, if not, i.e. if the element with the highest priority is an action or a process variable this element is added as a node to the Syntax tree but the recursive calls finish.

2.3 Semantics Analysis

Through this Semantics analysis is built the LTS for a ROSA process, the complexity of this analysis is very big since it has to determine which for all of the ROSA Semantics rules canmust be used in order to show all of the possible behaviours that a given process can perform.

Firstly, a LTS constituted with a unique Semantics node which contains the Syntax tree build for the input process is created, this node will be the root node in the Semantics tree. In the next step it has to be determined which rule for all of the ROSA operational Semantics rules can be applied. As it can be seen in ROSA operational Semantics there are three rule sets (non-deterministic, probabilistic and action transition rules). Then the first checking in which is involved ROSA Analysis is to determine whether the process is non-deterministic by means of *deterministic stability* function, if the process is non-deterministic it will be checked which of the non-deterministic transition rules must be used, that will be checked by means of the root node in the Syntax tree.

On the other hand if the process is deterministically stable, the tool must determine trough the *action* function if the process can be evolved by probabilistic or action transition rules. Once the transition rules set has been chosen, ROSA Analyser determines, by means of the upper condition, which rule must be applied. In order to show a clearly way of this process, figure 3 shows its activity diagram.

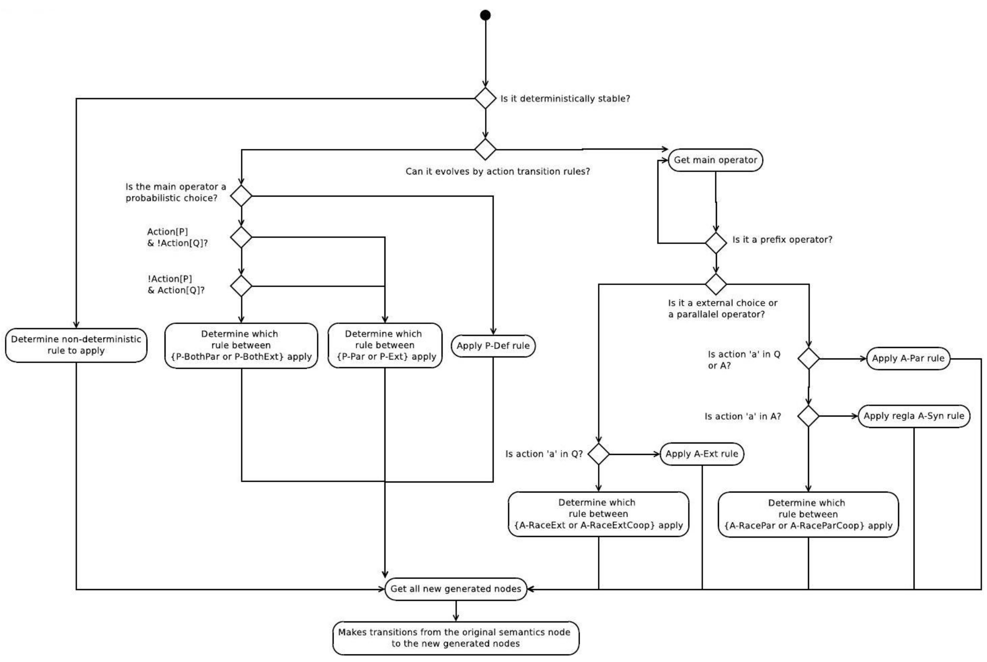
Is it deterministically stable?
Can it evolves by action transition rules?
Get main operator
Is the main operator a probabilistic choice?
Is it a prefix operator?
Action[P] & !Action[Q]?
!Action[P] & Action[Q]?
Is it a external choice or a parallalel operator?
Determine non-deterministic rule to apply
Determine which rule between {P-BothPar or P-BothExt} apply
Determine which rule between {P-Par or P-Ext} apply
Apply P-Def rule
Is action 'a' in Q or A?
Apply A-Par rule
Is action 'a' in A?
Apply regla A-Syn rule
Is action 'a' in Q?
Apply A-Ext rule
Determine which rule between {A-RaceExt or A-RaceExtCoop} apply
Determine which rule between {A-RacePar or A-RaceParCoop} apply
Get all new generated nodes
Makes transitions from the original semantics node to the new generated nodes

Figure 3: Activity diagram for Semantics nodes analysis process

Once all these nodes have been created, they must be joined to the LTS. As it is well-known the *explosion of states* problem arises when the LTS is being built, but this tool, in order to reach a lighter solution for this problem, is able to detect Syntax identities between nodes in order to avoid a node to appear more than once. When the new nodes have been added to the LTS, again all leaf nodes will be analysed, in order to search more nodes that could be generated. This loop is repeated until two consecutive iterations do not provide any new node, this will mean that the LTS is completed.

3 Analysing a memorizing process

In order to show the usefulness of this tool, in this section we describe the way in which a memorizing process is analysed. This process was specified using ROSA in [6].

Now we are going to describe the input process and the equivalences with the original specification. However, to get a clearer description we roughly describe each process involved and its corresponding ROSA specification.

Encoding process

This is the process with which the memorizing process starts, firstly elements identification is computed, once this initial identification is complete then at the same time it tries to identify objects and relations with previous data. The ROSA specification for this process can be seen in (1):

$$Encoding \equiv < Perception, \beta > . < Start\_Identification, \infty > .(< Find\_Attributes, \alpha_1 > . < stm, \infty > ||_{\{stm,abm\}}$$

$$||_{\{stm,abm\}} < Identify\_Object, \alpha_2 > . < stm, \infty > ||_{\{stm,abm\}} < Find\_Relations, \alpha_3 > . < stm, \infty >) \quad (1)$$

The specification used as input for the tool is:

$$E \equiv < a, 0.1 > .b.(< c, 0.2 > .f||\{f,i\} < d, 0.3 > .f||\{f,i\} < e, 0.4 > .f) \quad (2)$$

Consolidation process

The memorizing process specified has three stages for memory storage: *Sensory store*, *Short term memory* and *Long term memory*. The first two kinds of memory aren't definitive, i.e. not consolidated yet, and the information perceived is stored several seconds there, but when the information is longer stored, we are consolidating this information. Because of this, once the encoding process has finished with a certain amount of probability (according to parameter *r* in full memorizing process specification) the perceived information can be stored in long term memory or what is the same, it is definitively consolidated. The ROSA specification for this process is:

$$Consolidation \equiv < Soon\_Rehearsal\_or\_Finding\_Cues, \alpha_4 > . < ltm, \infty > . < abm, \gamma > \quad (3)$$

In the same way that the previous process was used, the specification for ROSA Analyser is:

$$C \equiv < g, 0.5 > .h. < i, 0.6 > \quad (4)$$

Retrieval process

Retrieval process represents the way in which the memorizing process access to information which has been previously stored, ROSA specification for this process is:

$$Retrieval \equiv < Request, \infty > . < abm, \sigma > \tag{5}$$

Therefore, as ROSA Analyser input we will use:

$$R \equiv j. < i, 0.7 > \tag{6}$$

Losing process

With a certain probability it is possible to lose the perceived information, this event could happens in the case of the information does not reach the long term memory. Obviously, this process only represents a single action execution, so in the specification there is only one action involved:

$$Losing \equiv < Fading, \alpha_5 > \tag{7}$$

For tool Syntax we take:

$$L \equiv < k, 0.8 > \tag{8}$$

LTS for the memorizing process

Finally, joining all of the defined process the expression associated for the memorizing process is:

$$Mem \equiv Encoding; (Consolidation \oplus_{0.25} Losing)||_{\{abm\}} Retrieval \tag{9}$$

And for ROSA Analyser this process must be written as follows:

$$M \equiv E; (C * \{0.25\}L)||\{i\}R \tag{10}$$

Once the Semantics analysis has been performed, ROSA Analyser provides two outputs, one in which the nodes an transitions can be textually seen, and other graphical output that by means of graphviz the LTS, can be exported to pdf, jpg, svg and eps. In figure 4 we can see the latter.

As a matter of proof of usefulness of ROSA Analyser, this LTS has two distinguished states, one red coloured representing a deadlock state, and a green coloured states representing a success.

4 Conclusions and future work

To sump up we can say that a tool to automatize the LTS building process of a ROSA process has been developed. This tool parses the process Syntax expression into a layered data structure quite suitable to better develop the Semantics analysis, such Operational Semantics provides the rules among which a proper subset of them, for each process should be applied. As a matter of fact, we have shown a real example by which the practical use of ROSA Analyser can be seen. In addition, it is important to mention that this tool provides two kinds of outputs, a textual output in which all nodes and transition can be seen; and a graphical view in which the Semantics tree is drawn in a portable format.

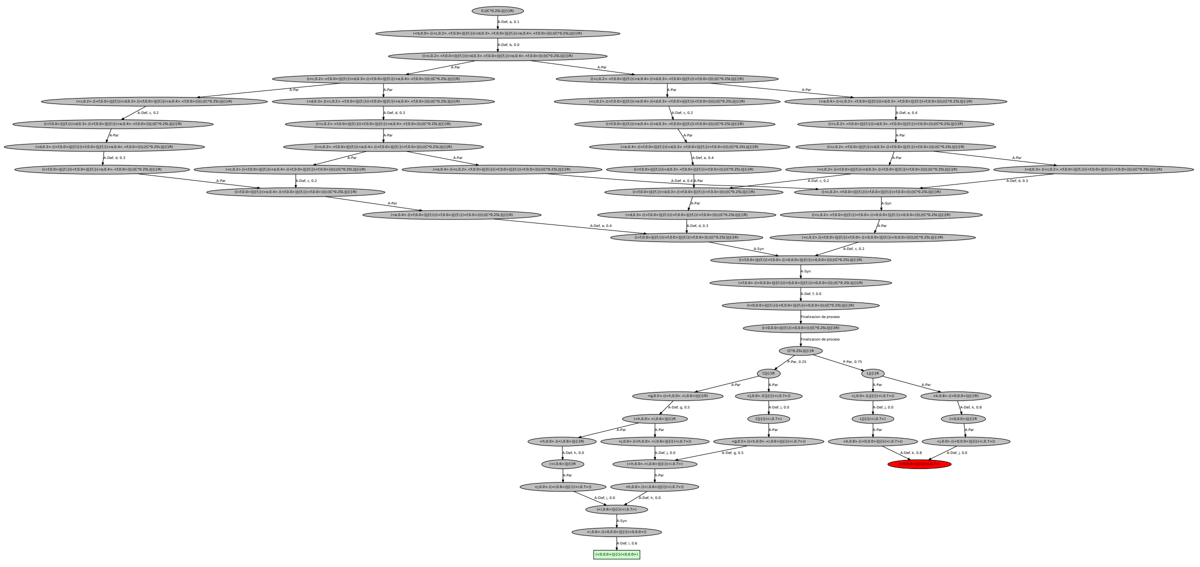
Figure 4: LTS for memorizing process

Although we reach a more practical use of ROSA process algebra, the *explosion of states* problem has not been resolved yet. At this moment, ROSA Analyser is able to “see” more identities of processes than the Syntactical one so reaching a little reduction in the states number. However, as we previously stated some first steps in this line has been done, specifically, a metric over the set of ROSA processes and a sort of normal form has been defined in [5], through this distance and normal forms can be defined several heuristics which provide the tentative more promising branch from any state to a given final one, so decreasing the number of states and making this task more practically tractable.

References

[1] B. Berthomieu, P.-O. Ribet & F. Vernadat (2004): *The tool TINA Construction of Abstract State Spaces for Petri Nets and Time Petri Nets*. Int. Journal of Production Research.

[2] Stephen Gilmore & Jane Hillston: *The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling*. Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation.

[3] Jane Hillston (1996): *A Compositional Approach to Performance Modelling*. Cambridge University Press.

[4] F. L. Pelayo (2004): *Application of formal methods to performance evaluation*. Ph.D. thesis, Universidad de Castilla - La Mancha.

[5] F. L. Pelayo, F. Cuartero & D. Cazorla (2011): *Looking for a Cheaper ROSA*. The International Workshop on Artificial Neural Networks: IWANN2011.

[6] Maria L. Pelayo, Fernando L. Pelayo, Fernando Cuartero, Valentin Valero, Gregorio Diaz & Elena Nieto (2007): *Does ROSA provide a good view of the Memorizing Process?* 6th IEEE International Conference on Cognitive Informatics (ICCI'07).

[7] F. Larsson P. Pettersson S. J. Bengtsson, K. G. Larsen & W. Yi (1996): *UPPAAL: a Tool-Suite for Automatic Verification of Real-Time Systems*. Technical Report RS-96-58, BRICS, Department of Computer Science, University of Aarhus.

[8] Anton Stefanek, Richard A. Hayden & Jeremy T. Bradley (2011): *GPA a tool for fluid scalability analysis of massively parallel systems*. 2011 Eighth International Conference on Quantitative Evaluation of SysTems.