The HAL-online Tool

Stefania Gnesi, Gianluca Trentanni

{ stefania.gnesi, gianluca.trentanni}@isti.cnr.it

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", CNR, Pisa, Italy

Abstract. *The HD-Automata Laboratory (HAL) [1] is an integrated tool set for the specification, verification and analysis of concurrent and distributed systems. The core of HAL are the HD-automata: they are used as a common format for the various history-dependent languages. The HAL environment includes modules which implement decision procedures to calculate behavioral equivalences, and modules which support verification of behavioral properties expressed as formulae of suitable temporal logics. At this moment HAL works only with concurrent and distributed systems expressed by π-calculus formalism. The HAL environment allows π-calculus agents to be translated into ordinary automata, so that existing equivalence checkers can be used to calculate whether the π-calculus are bisimilar. The environment also supports verification of logical formulae expressing desired properties of the behavior of π-calculus agents.*

In this paper the online version of the toolkit is shown.

Keywords: Formal method, History Dependant Automata, Pi Calculus, Software/Program Verification

Introduction

HD-automata have been introduced in [33], with the name of π-automata, as a convenient structure to describe in a compact way the operational behaviours of π-calculus agents. HD-automata have been further generalized to deal with name passing process calculi, process calculi equipped with location, causality and Petri Nets [37, 34, 35].

Due to the mechanism of input, the ordinary operational semantics of the π-calculus requires an infinite number of states also for very simple agents. The creation of a new name gives rise to an infinite set of transitions: one for each choice of the new name. To handle this problems in HD-automata names appear explicitly in states, transitions and labels. Indeed, it is convenient to assume that the names which appear in a state, a transition or a label of a HD-automaton are local names and do not have a global identity. In this way, for instance, a single state of the HD-automaton can be used to represent all the states of a system that differ just for a bijective renaming.

The theory of HD-automata ensures that they provides a finite state faithful semantical representation of the behaviour of π-calculus agents. Indeed, it is possible to extract from the HD-automaton of a π-calculus agent its ordinary early operational semantics. This is done by a simple algorithm (basically visiting the HD-automaton) which maintains the global meaning of the local names of the reached states.

Clearly, we have a transition for all the possible choices of the fresh names. In other words, this procedure yields an infinite state automaton. To obtain a finite state automaton it suffices to take as fresh name the first name which has been not

already used. In this way, a finite state automaton is obtained from each finite HD-automaton.

To define an automatic verification procedure to model check whether or not a π-logic formula holds for a π-calculus specification. it is possible to derive an ordinary automaton for finitary π-calculus. Hence, if we were able to translate formulae of the π-logic into "ordinary" logic formulae, it should be possible to use existing model checking algorithms to check the satisfiability of "ordinary" logic formulae over ordinary automata. This translation is possible using Actl [16], for which an efficient model checker has been implemented [19] and for which a sound translation exists.

The HAL toolkit provides facilities to deal with π-calculus specification by exploiting HD-automata. In the following, the HAL-online toolkit architecture and functions are shown and briefly explained.

System Overview

In **Fig. 1** the HAL-online starting Web page is shown. This page provides useful links pointing to essential references for π-calculus based model checking and leads to the TOOLs page (Fig. 2).

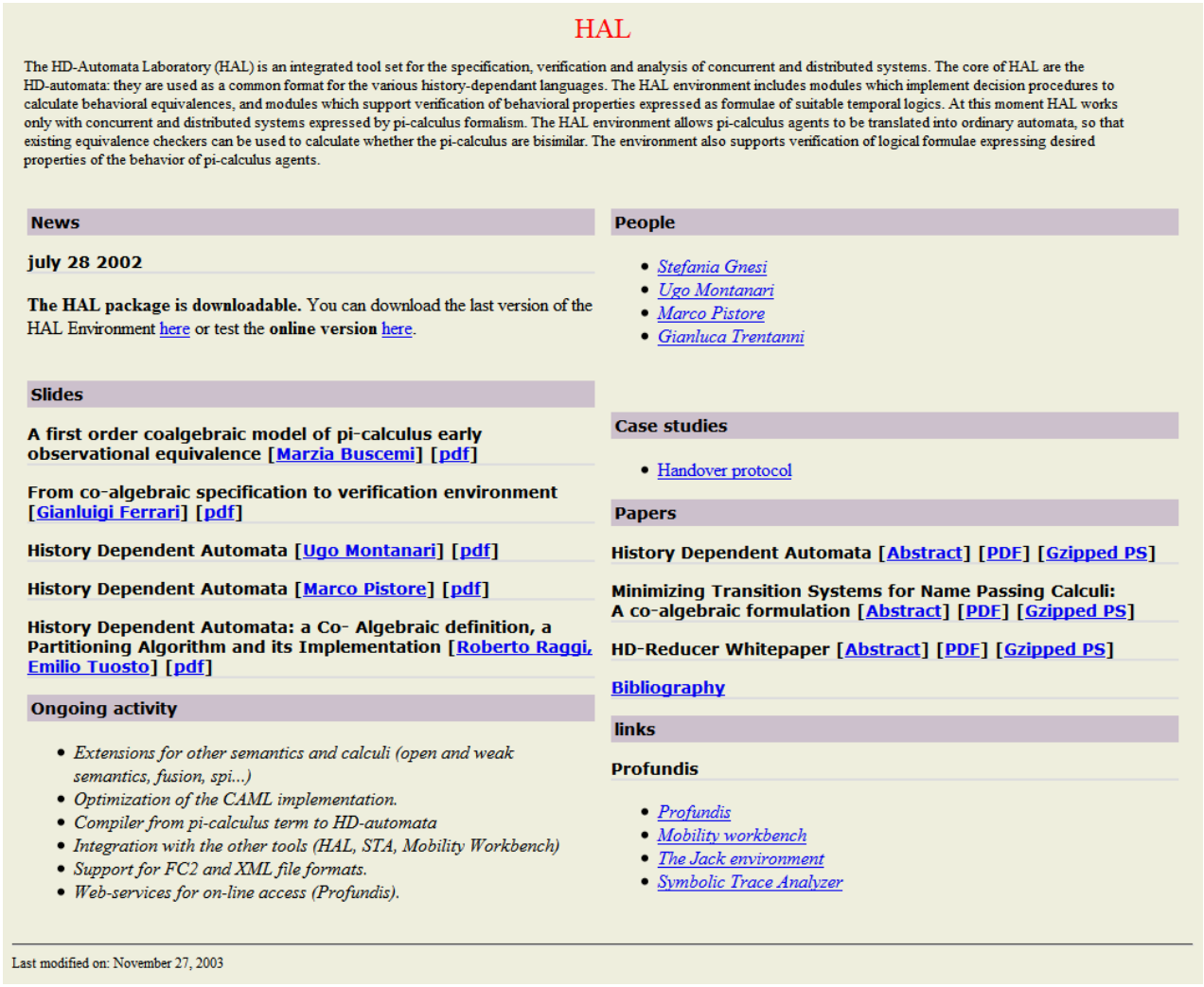


Figura 1 - HAL-online starting page: http://fmt.isti.cnr.it:8080/hal

HAL has been developed exploiting Zope [56], an open source web and application server that allows dynamic server pages generation and interaction with the server le system through the highly compatible Python platform. Starting from the HAL-online start page, the user can upload browsing the local system or by means of a cut-and-paste from plain text files.

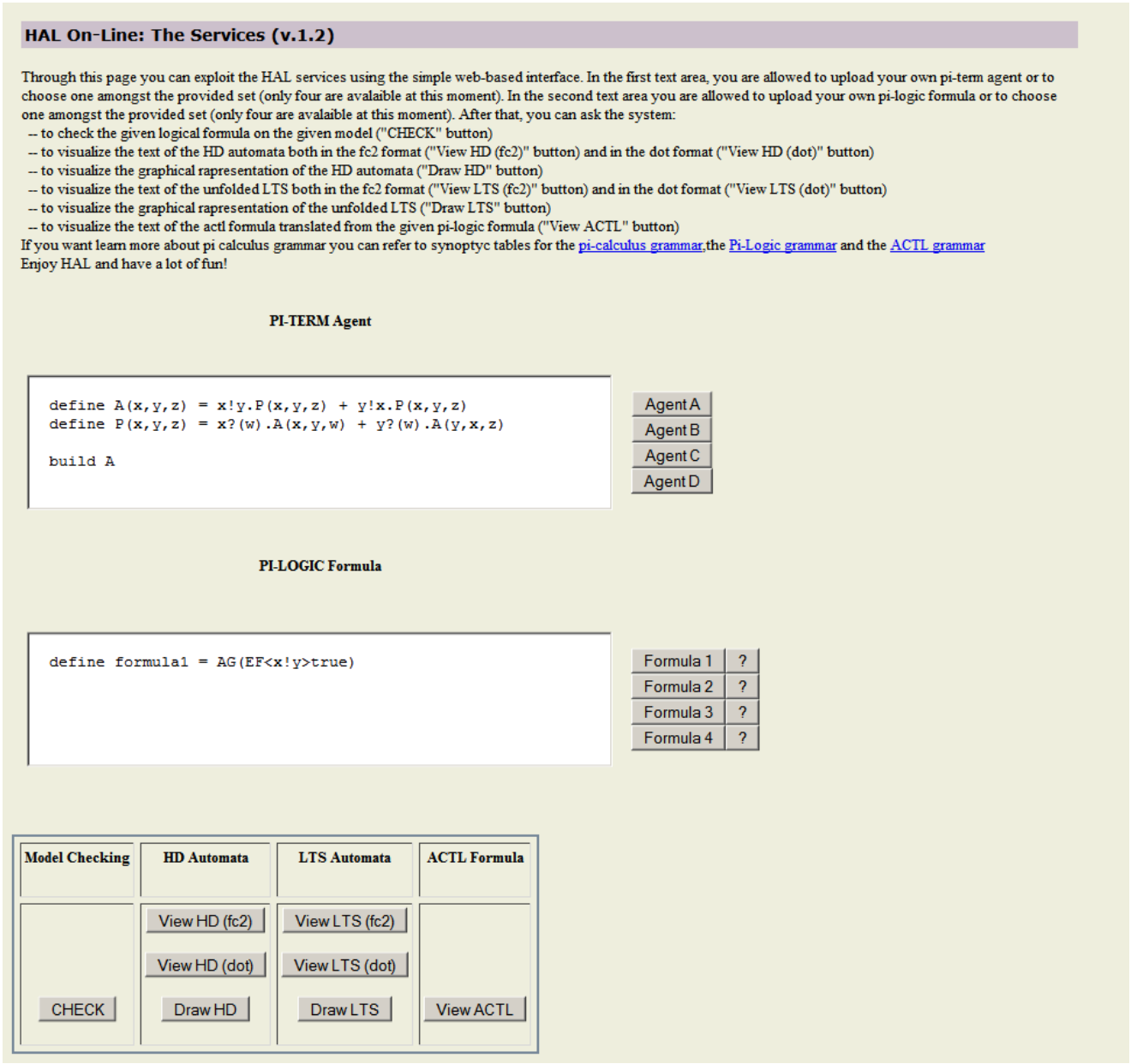


Figura 2 - HAL-online TOOLs page: http://fmt.isti.cnr.it:8080/hal/bin/HALOnLine

In **Fig. 2** the HAL-online TOOLs page

http://fmt.isti.cnr.it:8080/hal/bin/HALOnLine

is shown. The ability to browse among local is usually delegated to common web clients. The ability to retrieve the from the client system, and visualize it, is realized taking advantage of simple DTML code calling the Zope built-in read

function. HAL-online exploits simple javascript and further DTML [56] code to open the output window and to control that any request is denitely sent in the correct form.

The goal of the HAL toolkit is to verify properties of mobile systems specified in the π-calculus.

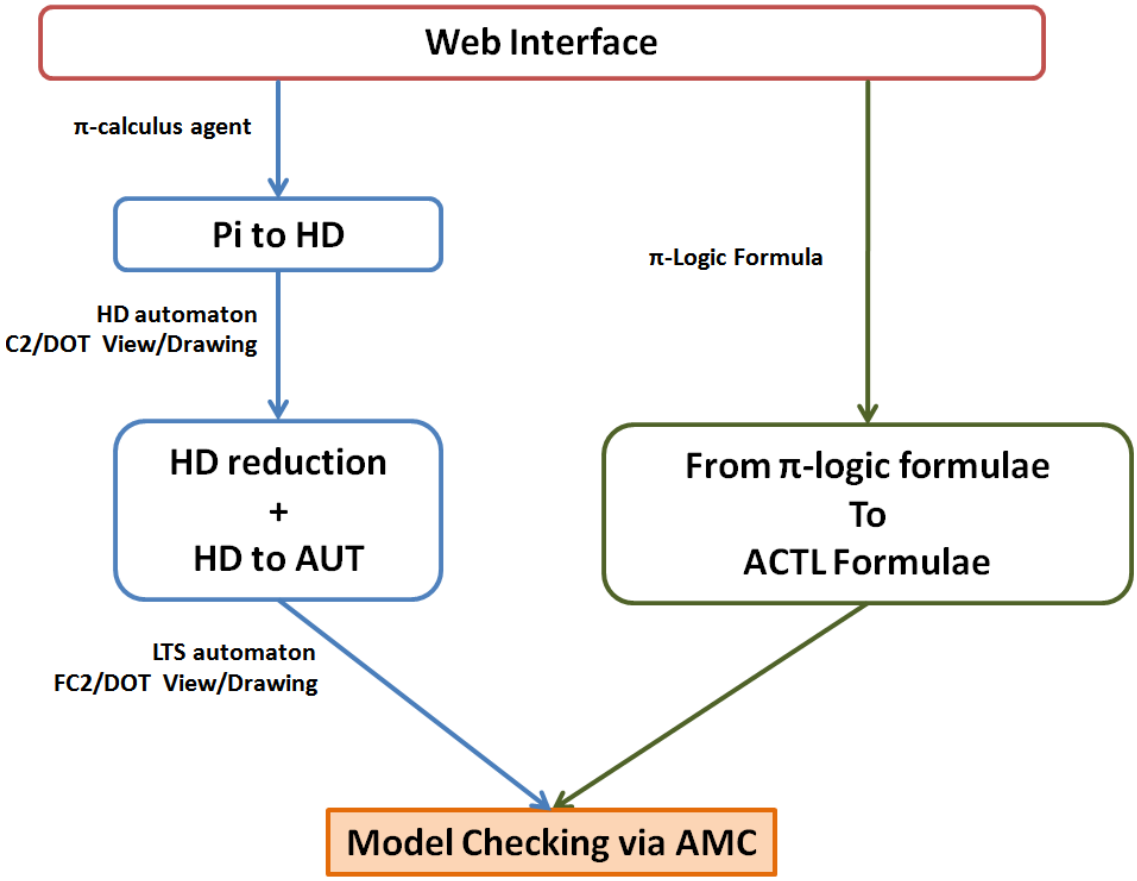


Figura 3 - HAL-online architecture overview

In **Fig. 3** The HAL-online internal architecture is shown.

Exploiting HAL facilities, π-calculus specifications are translated first into HD-automata and then in ordinary automata. Hence, the bisimulation checking performed by the AMC module can be used to verify (strong and weak)

bisimilarity. Automata minimization, according to weak bisimulation is also possible.

HAL supports verification of logical formulae expressing properties of the behaviour of π-calculus specifications.

The Actl model checker can be used for verifying properties of π-calculus specifications, after that the π-logic formulae expressing the properties have been translated into Actl formulae.

Notice that the complexity of the model checking algorithm depends on the construction of the state space of the π-calculus agent to be verified,which is, in the worst case, exponential in the syntactic size of the agent.

In the current implementation the HAL-online environment consists essentially of five modules: three modules perform the translations from π-calculus agents to HD-automata (pi-to-hd), from HD-automata to ordinary automata after hd reduction, (hd-reduce and hd-to-aut) and from π-logic formulae to ordinary ACTL formulae (pl-to-actl).

The fifth module works at the level of ordinary automata and performs the standard operations on them like behavioral verification and model checking.

Latest function is represented by a tiny module called "*trace*", developed ad-hoc by Franco Mazzanti, that exports the textual formal description of automata (both HD and LTS) in the drawable "dot" [50, 51, 52, 53, 54, 55] format allowing a visual representation by means of a gif image.

Online User Interface

The upper part of the HAL-online user interface (**Fig. 4**) allows to specify the π-automaton and the π-formula by hand (or to choose among four presets).

The default formula presets are briefly explained too ("?" buttons).

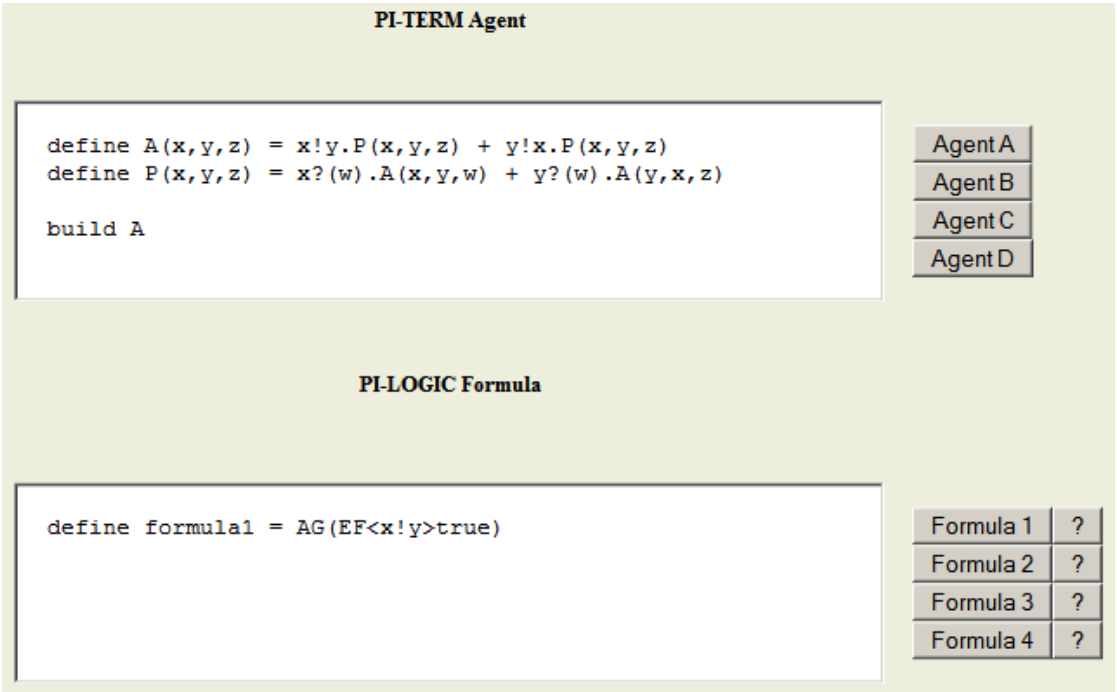


Figura 4 - HAL-online inputs boxes

The function button panel perform several functions of transformation and visualization on automata and formulas.

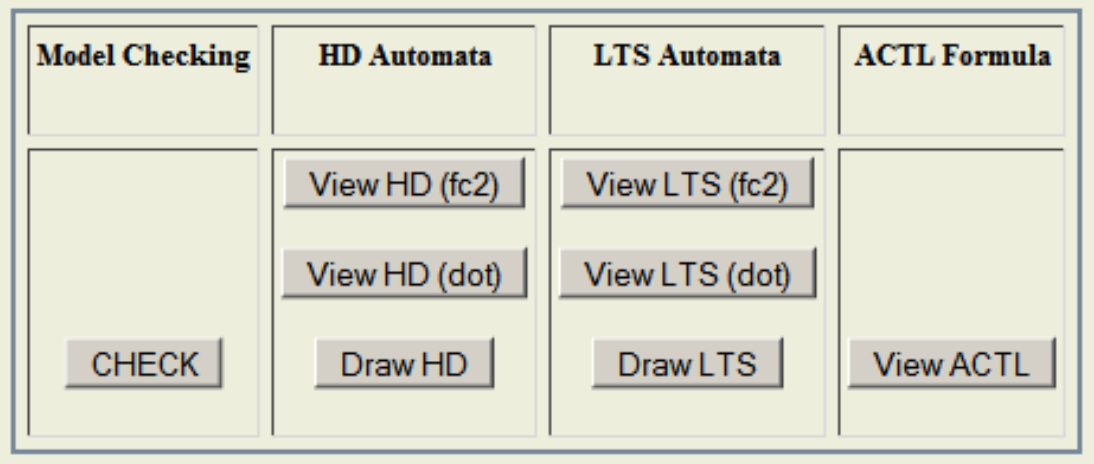


Figura 5 - HAL-online control buttons set

The "**HD Automata**" buttons column allows a π-calculus agent to be transformed into a HD-automaton and to view the resulting automaton in both the textual formats fc2 and dot and graphically as gif image.

The "**LTS Automata**" buttons column allows a HD agent to be transformed into a LTS automaton (i.e. an ordinary automaton) and to view the resulting automaton in both the textual formats fc2 and dot and graphically as gif image.

The "**ACTL Formula**" column button "***View Actl***" allows a π-logic formula to be translated into an ordinary ACTL one and visualizes it in text format in the related result pop-up window.

The "**Model Checking**" column "***Check***" button allows to verify the equivalence of the ordinary automata corresponding to the generated HD automata from the π-calculus agents specified in the input text box by means of the related choosen formula.

Several optimizations have been implemented. These optimizations reduce the state space of HD-automata, thus allowing a more efficient generation of the ordinary automata associated with π-calculus agents. An example of optimization is given by the reduction of tau chains (that are unbranched sequences of tau

transitions) to simple tau transitions (option Reduce). Another optimization consists of the introduction of constant declarations. Constant names are names that cannot be used as objects of input or output actions (for instance, names that represent stationary communication topologies, namely communication topologies which cannot be modified when computations progress). Since constant names are not consider as possible input values, the branching structure of input transitions is reduced. The semantic handling of constants is presented in [38]. Constants have to be declared in the π-calculus specifications.

Summarizing schematically, the available actions we can perform are:

- **Model Checking**
    - **Button "*Check*":** automatically checks the π-automaton against the π-formula performing silently all the transformations, translations, unfolding and reductions actions needed exploiting as final step the AMC model checker. An Example of the output is shown in **Fig. 6**.
- **HD Automata**
    - **Button "*View HD (FC2)*":** shows the HD automaton in the textual fc2 formalism. An Example of the output is shown in **Fig. 7**.
    - **Button "*View HD (dot)*":** shows the formula in the textual dot formalism. An Example of the output is shown in **Fig. 8**.
    - **Button "*Draw HD*":** draws the HD automata exploiting the dot format and the "*trace*" tool. An Example of the output is shown in **Fig. 9**.

- **LTS Automata**
    - **Button "*View LTS (FC2)*":** shows the LTS automaton in the textual fc2 formalism. An Example of the output is shown in **Fig. 11**.
    - **Button "*View LTS (dot)*":** shows the LTS automaton in the textual dot formalism. An Example of the output is shown in **Fig. 12**.
    - **Button "*Draw LTS*":** draws the HD automata exploiting the dot format and the "*trace*" tool . An Example of the output is shown in **Fig. 13**.
- **ACTL Formulae**
    - **Button "*View ACTL*":** shows the ACTL formula translated starting from the pi-formula input text-box. An Example of the output is shown in **Fig. 10**.

Thus, any action is the result of chained commands/programs and script execution that usually give a feedback on the successfully or not execution itself, a view on the internal actions log is provided for any action. An Example of the output log for an action "***Check***" is shown in **Fig. 11**.

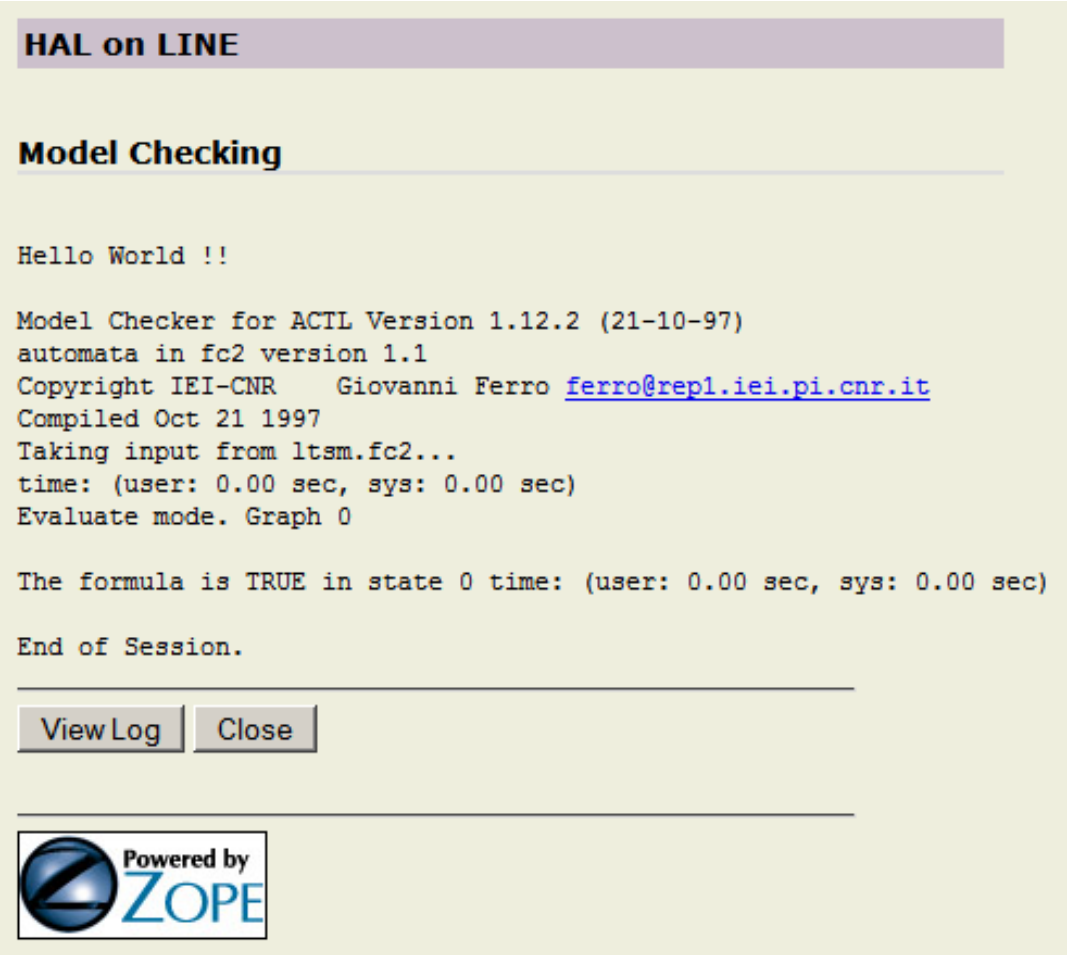
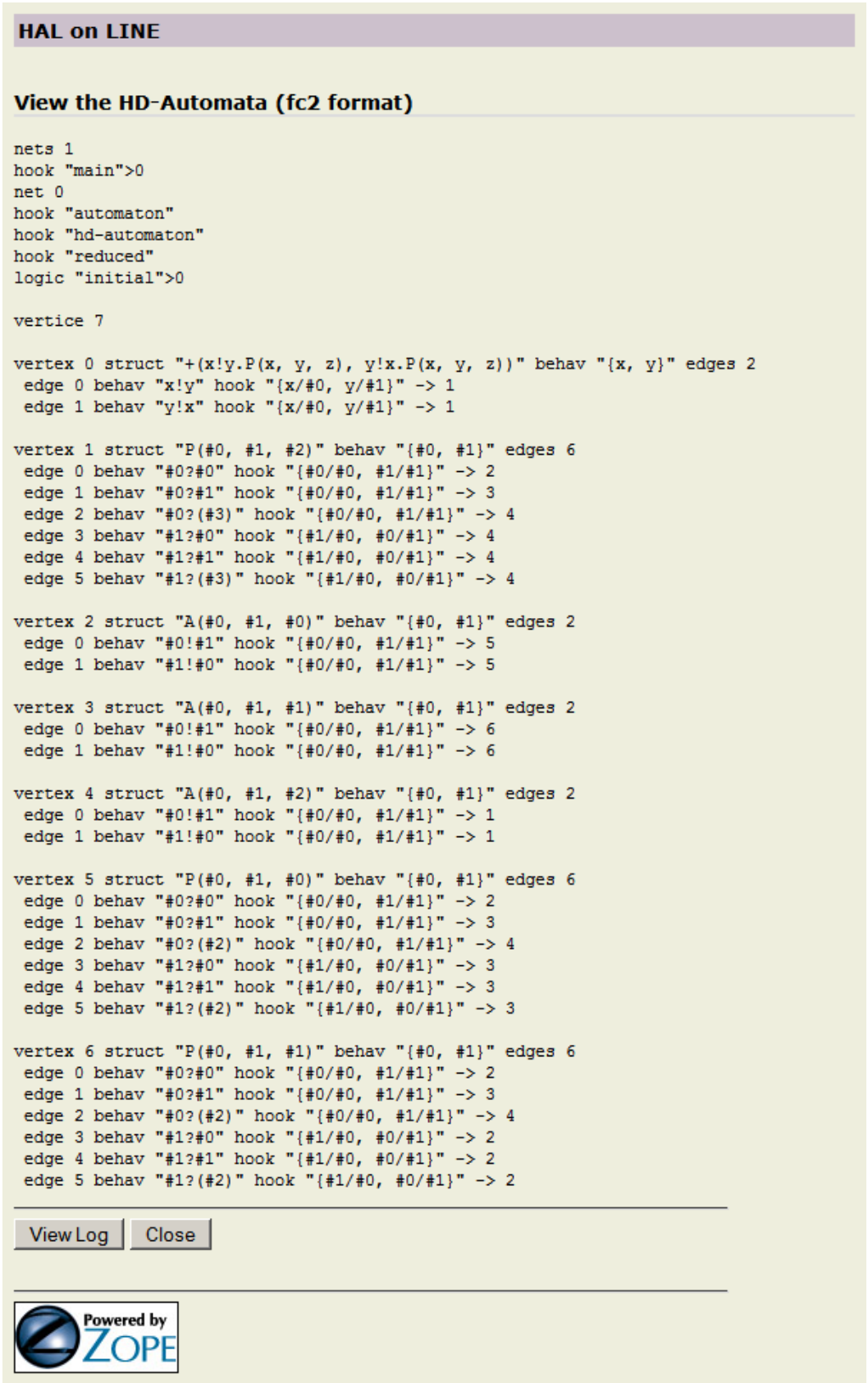


Figura 6 – Example of HAL-online "*Check*" output



HAL on LINE

View the HD-Automata (fc2 format)

```
nets 1
hook "main">0
net 0
hook "automaton"
hook "hd-automaton"
hook "reduced"
logic "initial">0

vertice 7

vertex 0 struct "+(x!y.P(x, y, z), y!x.P(x, y, z))" behav "{x, y}" edges 2
 edge 0 behav "x!y" hook "{x/#0, y/#1}" -> 1
 edge 1 behav "y!x" hook "{x/#0, y/#1}" -> 1

vertex 1 struct "P(#0, #1, #2)" behav "{#0, #1}" edges 6
 edge 0 behav "#0?#0" hook "{#0/#0, #1/#1}" -> 2
 edge 1 behav "#0?#1" hook "{#0/#0, #1/#1}" -> 3
 edge 2 behav "#0?(#3)" hook "{#0/#0, #1/#1}" -> 4
 edge 3 behav "#1?#0" hook "{#1/#0, #0/#1}" -> 4
 edge 4 behav "#1?#1" hook "{#1/#0, #0/#1}" -> 4
 edge 5 behav "#1?(#3)" hook "{#1/#0, #0/#1}" -> 4

vertex 2 struct "A(#0, #1, #0)" behav "{#0, #1}" edges 2
 edge 0 behav "#0!#1" hook "{#0/#0, #1/#1}" -> 5
 edge 1 behav "#1!#0" hook "{#0/#0, #1/#1}" -> 5

vertex 3 struct "A(#0, #1, #1)" behav "{#0, #1}" edges 2
 edge 0 behav "#0!#1" hook "{#0/#0, #1/#1}" -> 6
 edge 1 behav "#1!#0" hook "{#0/#0, #1/#1}" -> 6

vertex 4 struct "A(#0, #1, #2)" behav "{#0, #1}" edges 2
 edge 0 behav "#0!#1" hook "{#0/#0, #1/#1}" -> 1
 edge 1 behav "#1!#0" hook "{#0/#0, #1/#1}" -> 1

vertex 5 struct "P(#0, #1, #0)" behav "{#0, #1}" edges 6
 edge 0 behav "#0?#0" hook "{#0/#0, #1/#1}" -> 2
 edge 1 behav "#0?#1" hook "{#0/#0, #1/#1}" -> 3
 edge 2 behav "#0?(#2)" hook "{#0/#0, #1/#1}" -> 4
 edge 3 behav "#1?#0" hook "{#1/#0, #0/#1}" -> 3
 edge 4 behav "#1?#1" hook "{#1/#0, #0/#1}" -> 3
 edge 5 behav "#1?(#2)" hook "{#1/#0, #0/#1}" -> 3

vertex 6 struct "P(#0, #1, #1)" behav "{#0, #1}" edges 6
 edge 0 behav "#0?#0" hook "{#0/#0, #1/#1}" -> 2
 edge 1 behav "#0?#1" hook "{#0/#0, #1/#1}" -> 3
 edge 2 behav "#0?(#2)" hook "{#0/#0, #1/#1}" -> 4
 edge 3 behav "#1?#0" hook "{#1/#0, #0/#1}" -> 2
 edge 4 behav "#1?#1" hook "{#1/#0, #0/#1}" -> 2
 edge 5 behav "#1?(#2)" hook "{#1/#0, #0/#1}" -> 2
```

View Log | Close

Powered by ZOPE

Figura 7 - Example of HAL-online "*View HD (fc2)*" output

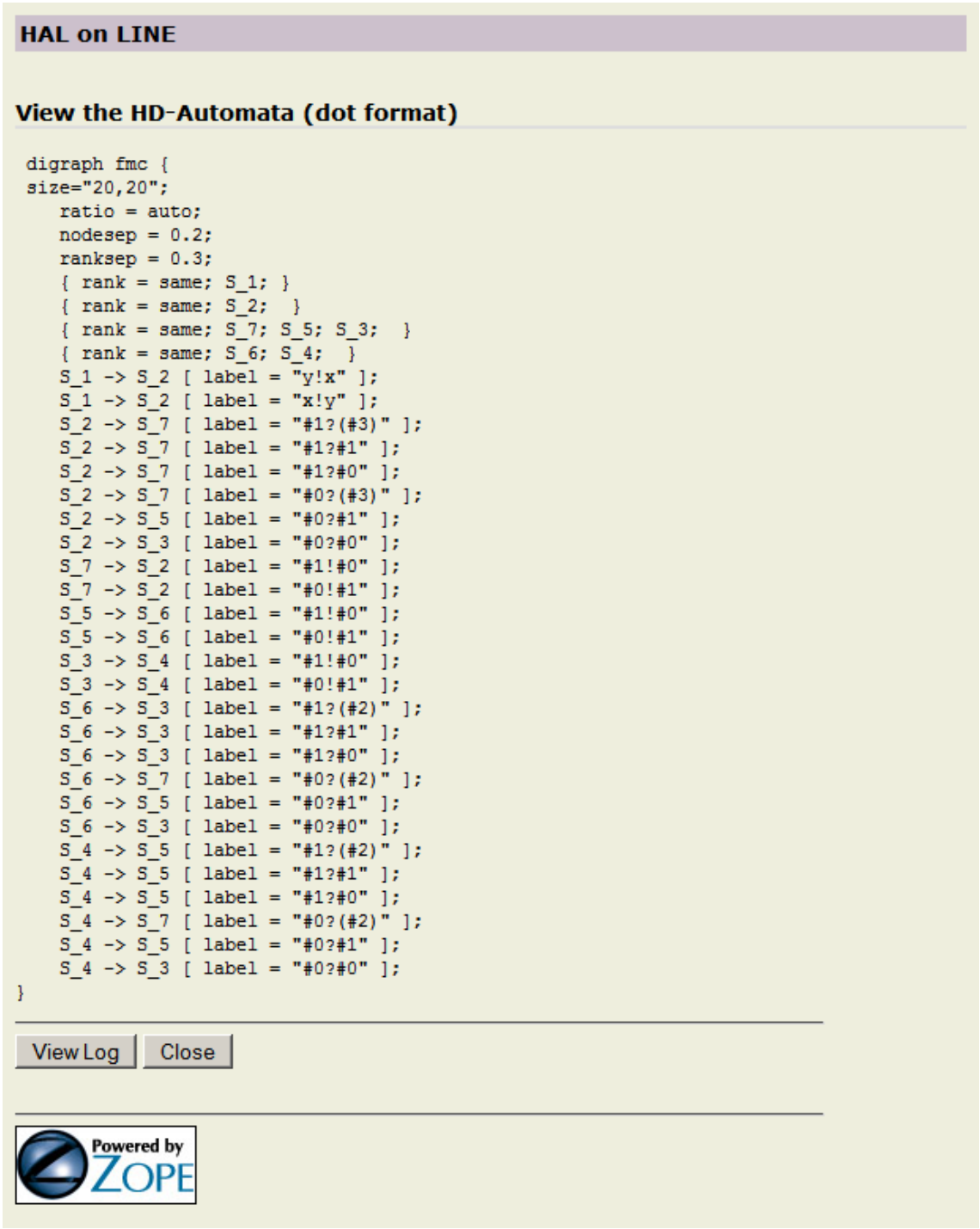


Figura 8 - Example of HAL-online "*View HD (dot)*" output

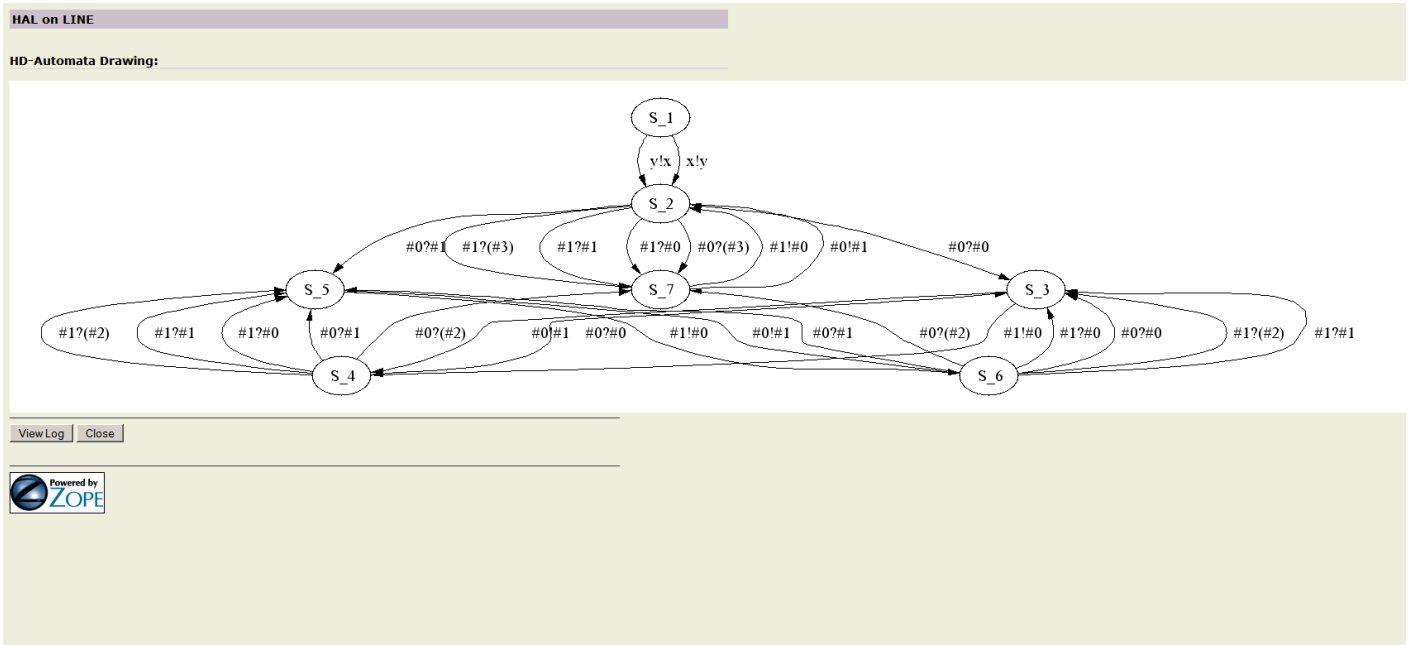


Figura 9 - Example of HAL-online "Draw HD" output

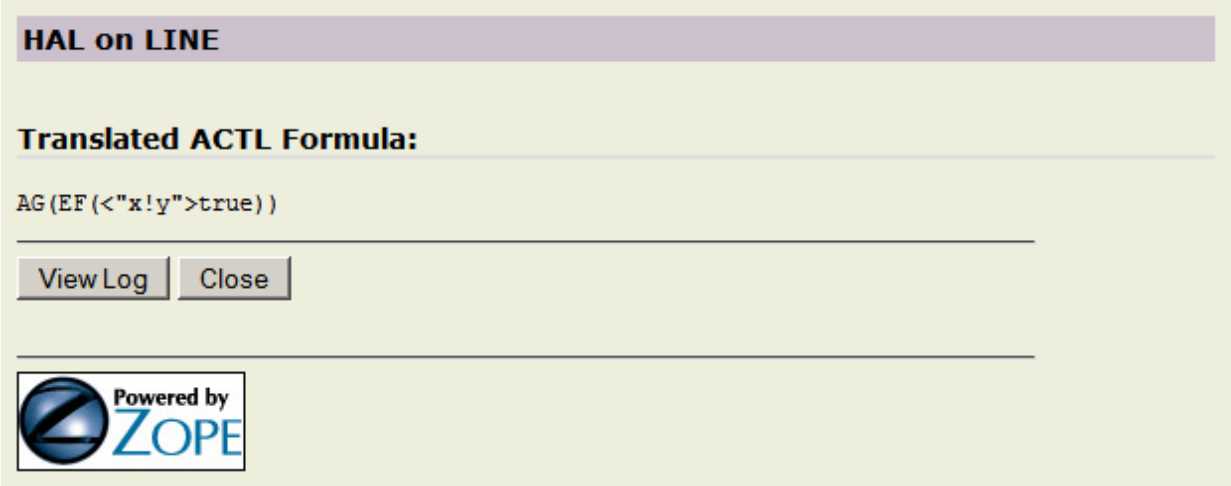


Figura 10 - Example of HAL-online "*View ACTL*" output

HAL on LINE

View the unfolded HD-Automata in its LTS form (fc2 format)...

...either minimized by fc2min (according to the strong bisimulation minimization)...and not minimized...

Minimized LTS

```
% Fc2 file generated by the fc2 package
nets 1
net 0
l "initial">0 h "automaton"
V2
v0 E2
e0 b "x!y" r 1
e1 b "y!x" r 1
v1 E6
e0 b "x?x" r 0
e1 b "x?y" r 0
e2 b "x?(#0)" r 0
e3 b "y?x" r 0
e4 b "y?y" r 0
e5 b "y?(#0)" r 0
```

LTS not Minimized

```
nets 1
hook "main">0
net 0
hook "automaton"
logic "initial">0
vertice 13

vertex 0 struct "0"
  edges 2
    edge 0
      behav "x!y"
      -> 1
    edge 1
      behav "y!x"
      -> 1

vertex 1 struct "1"
  edges 6
    edge 0
      behav "x?x"
      -> 2
    edge 1
      behav "x?y"
      -> 3
    edge 2
      behav "x?(#0)"
      -> 4
    edge 3
      behav "y?x"
      -> 5
    edge 4
      behav "y?y"
      -> 5
    edge 5
      behav "y?(#0)"
      -> 5

vertex 2 struct "2"
  edges 2
    edge 0
      behav "x!y"
      -> 12
    edge 1
      behav "y!x"
      -> 12
```

Figura 11 - Example of HAL-online "*View LTS (fc2)*" output

HAL on LINE

View the unfolded HD-Automata in its LTS form (dot format)...

...either minimized by fc2min (according to the strong bisimulation minimization) ...and not minimized...

Minimized LTS

```
digraph fmc {
size="20,20";
   ratio = auto;
   nodesep = 0.2;
   ranksep = 0.3;
   { rank = same; S_1; }
   { rank = same; S_2;  }
   S_1 -> S_2 [ label = "y!x" ];
   S_1 -> S_2 [ label = "x!y" ];
   S_2 -> S_1 [ label = "y?(#0)" ];
   S_2 -> S_1 [ label = "y?y" ];
   S_2 -> S_1 [ label = "y?x" ];
   S_2 -> S_1 [ label = "x?(#0)" ];
   S_2 -> S_1 [ label = "x?y" ];
   S_2 -> S_1 [ label = "x?x" ];
}
```

LTS not Minimized

```
digraph fmc {
size="20,20";
   ratio = auto;
   nodesep = 0.2;
   ranksep = 0.3;
   { rank = same; S_1; }
   { rank = same; S_2;  }
   { rank = same; S_12; S_7; S_5; S_3;  }
   { rank = same; S_13; S_6; S_4;  }
   { rank = same; S_10; S_8;  }
   { rank = same; S_11; S_9;  }
   S_1 -> S_2 [ label = "y!x" ];
   S_1 -> S_2 [ label = "x!y" ];
   S_2 -> S_12 [ label = "y?(#0)" ];
   S_2 -> S_12 [ label = "y?y" ];
   S_2 -> S_12 [ label = "y?x" ];
   S_2 -> S_7 [ label = "x?(#0)" ];
   S_2 -> S_5 [ label = "x?y" ];
   S_2 -> S_3 [ label = "x?x" ];
   S_12 -> S_13 [ label = "x!y" ];
   S_12 -> S_13 [ label = "y!x" ];
   S_7 -> S_2 [ label = "y!x" ];
   S_7 -> S_2 [ label = "x!y" ];
   S_5 -> S_6 [ label = "y!x" ];
   S_5 -> S_6 [ label = "x!y" ];
   S_3 -> S_4 [ label = "y!x" ];
   S_3 -> S_4 [ label = "x!y" ];
   S_13 -> S_7 [ label = "x?(#0)" ];
   S_13 -> S_7 [ label = "x?x" ];
   S_13 -> S_7 [ label = "x?y" ];
   S_13 -> S_12 [ label = "y?(#0)" ];
   S_13 -> S_10 [ label = "y?x" ];
   S_13 -> S_8 [ label = "y?y" ];
   S_6 -> S_8 [ label = "y?(#0)" ];
   S_6 -> S_8 [ label = "y?y" ];
   S_6 -> S_8 [ label = "y?x" ];
   S_6 -> S_7 [ label = "x?(#0)" ];
   S_6 -> S_5 [ label = "x?y" ];
   S_6 -> S_3 [ label = "x?x" ];
   S_4 -> S_10 [ label = "y?(#0)" ];
   S_4 -> S_10 [ label = "y?y" ];
   S_4 -> S_10 [ label = "y?x" ];
   S_4 -> S_7 [ label = "x?(#0)" ];
   S_4 -> S_5 [ label = "x?y" ];
   S_4 -> S_3 [ label = "x?x" ];
   S_10 -> S_11 [ label = "x!y" ];
   S_10 -> S_11 [ label = "y!x" ];
   S_8 -> S_9 [ label = "x!y" ];
   S_8 -> S_9 [ label = "y!x" ];
   S_11 -> S_3 [ label = "x?(#0)" ];
   S_11 -> S_3 [ label = "x?x" ];
   S_11 -> S_3 [ label = "x?y" ];
   S_11 -> S_12 [ label = "y?(#0)" ];
   S_11 -> S_10 [ label = "y?x" ];
   S_11 -> S_8 [ label = "y?y" ];
   S_9 -> S_5 [ label = "x?(#0)" ];
   S_9 -> S_5 [ label = "x?x" ];
   S_9 -> S_5 [ label = "x?y" ];
   S_9 -> S_12 [ label = "y?(#0)" ];
   S_9 -> S_10 [ label = "y?x" ];
   S_9 -> S_8 [ label = "y?y" ];
}
```

View Log | Close

Powered by ZOPE

Figura 12 - Example of HAL-online "*View LTS (dot)*" output

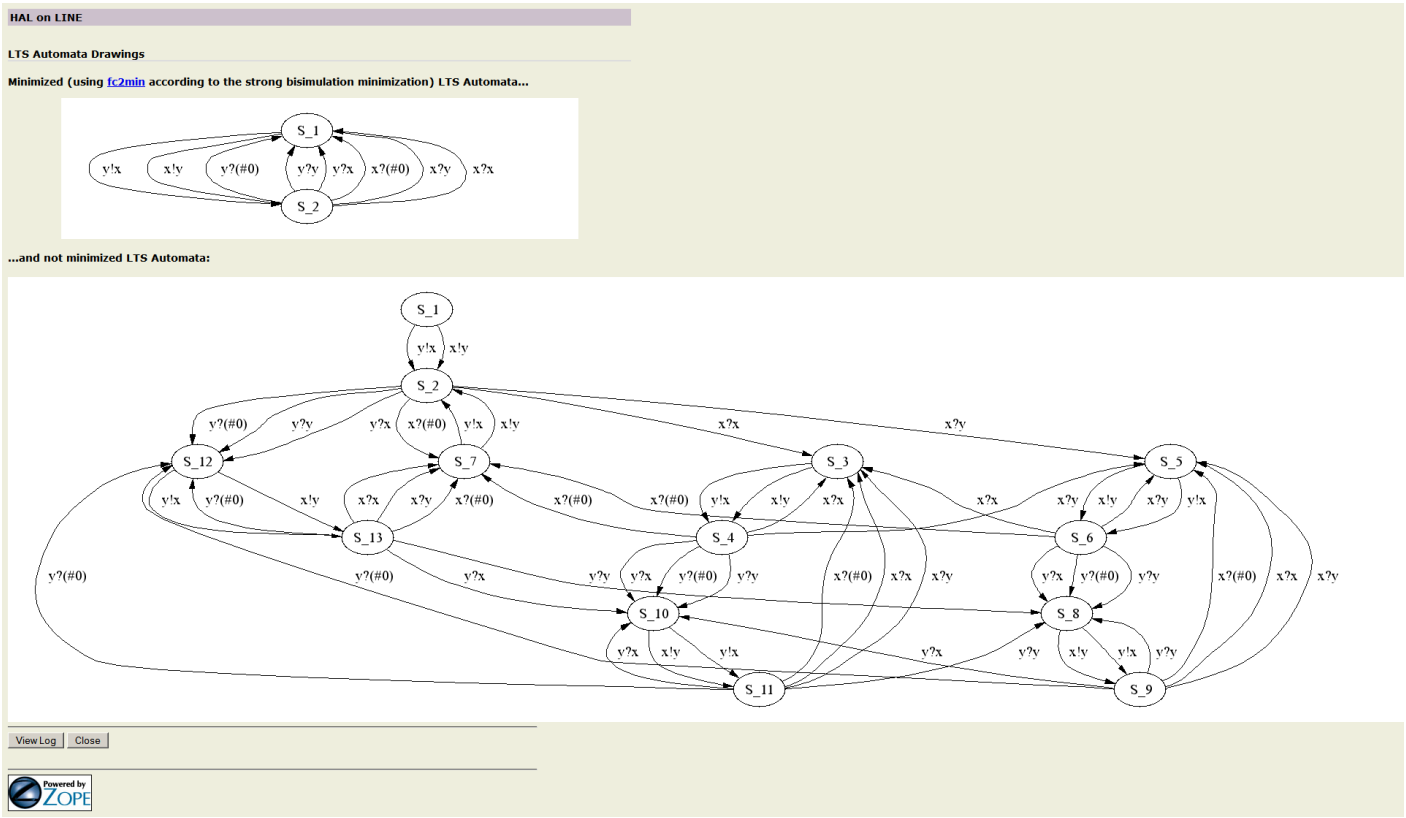


Figura 13 - Example of HAL-online "Draw LTS" output

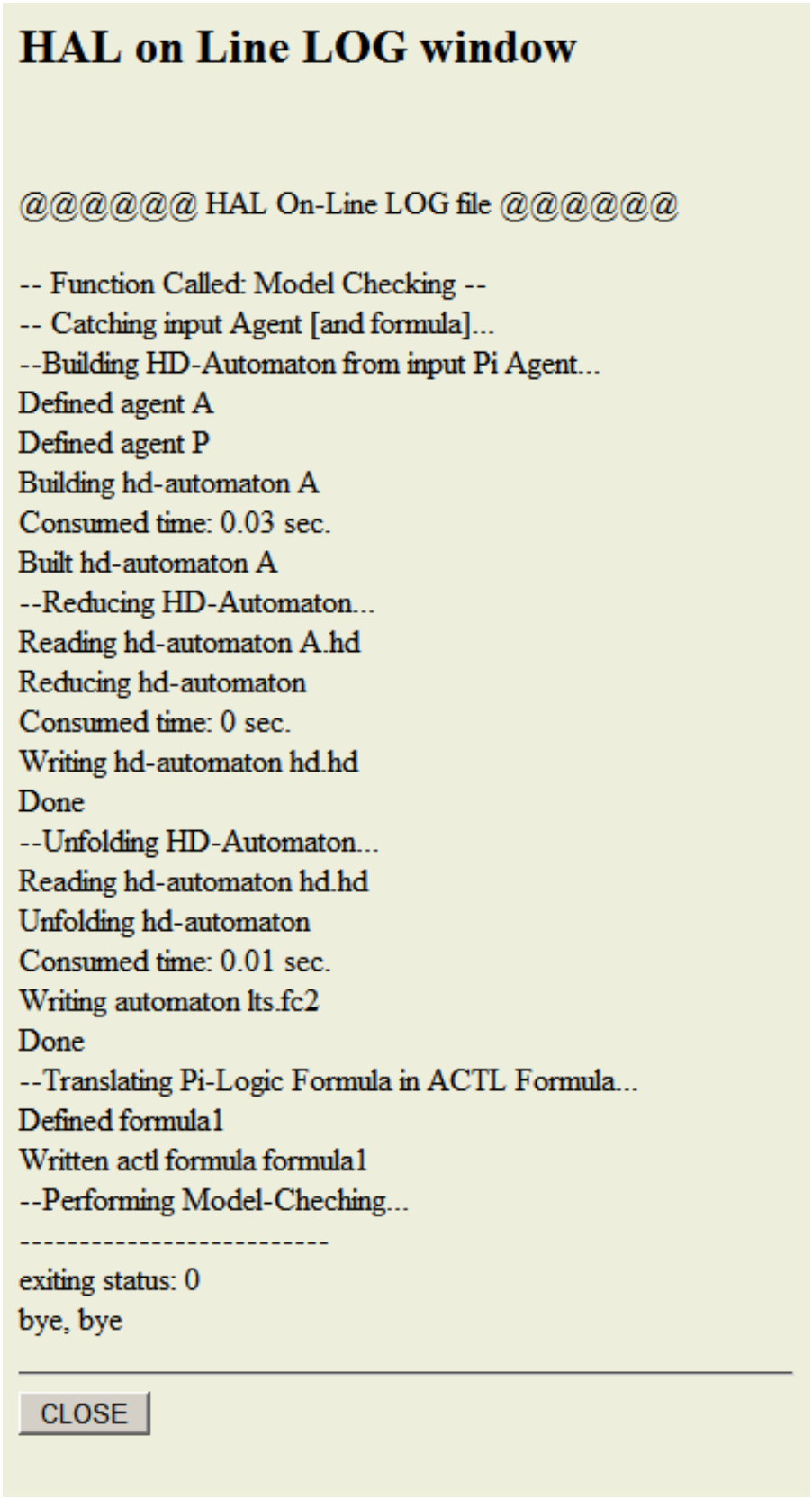


Figura 14 - Example of HAL-online output LOG for the action "*Check*"

References

[1] http://matrix.iei.pi.cnr.it/projects/JACK/hal.html

[2] M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. Information and Computation, 148(1):1–70, 1999

[3] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Proc. Royal Society of London A 246, 233-271, 1989

[4] A. Bouali and R. de Simone. Symbolic bisimulation minimization. In Proc. CAV'92 LNCS 663. Springer-Verlag, 1992

[5] A. Bouali, A. Ressouche, V. Roy, and R. De Simone. The FC2Tools set. In Proc. CAV'96, LNCS 1102. Springer Verlag, 1996

[6] L. Caires and L. Cardelli, A spatial logic for concurrency (Part II), In Proc. CONCUR'02, LNCS 2421, Springer, 2002

[7] L. Caires and L. Cardelli, A spatial logic for concurrency (Part I), Information and Computation, special issue on TACS 2001

[8] S. Chaki, S. Rajamani, and J. Rehof. Types as Models: model checking message-passing programs. In Proc. POPL'02, ACM Press, 2002

[9] E. Clarke and J. Wing Eds. Formal Methods: State of the Art and Future Directions. Strategic Directions in Comp. Res. Formal Methods WG Rep. ACM Comp. Surv., December 1996

[10] E. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In Proc. IFIPWorking Conference on Programming Concepts and Methods (PROCOMET), 1998

[11] M. Dam. Model checking mobile processes. In Proc. CONCUR'93, LNCS 715, Springer Verlag, 1993

[12] 12. M. Dam. Proof systems for π-Calculus Logics. To appear on Logics for Concurrency and Synchronization. Studies in Logics and Computation, Oxford University Press, 2001

[13] R. De Nicola, A. Fantechi, S. Gnesi, and G. Ristori. An action-based framework for verifying logical and behavioural properties of concurrent systems. Computer Networks and ISDN Systems 25(7):761-778. North-Holland, 1993.

[14] R. De Nicola and F. W. Vaandrager. Action versus state based logics for transition systems. In Proc. Ecole de Printemps on Semantics of Concurrency, LNCS 469. Springer Verlag, 1990

[15] E.A. Emerson and J.Y. Halpern. Sometimes and Not Never Revisited: on Branch-ing Time versus Linear Time Temporal Logic. Journal of ACM, 33(1), January 1986, pp. 151–178

[16] Ferrari, U. Montanari, and M. Pistore. Minimizing Transition Systems for Name Passing Calculi: A Co-algebraic Formulation. In Proc. FOSSACS'02, LNCS 2303, 2002

[17] J.-C. Fernandez and L. Mounier. "On the Py" verifcation of behavioral equivalences and preorders. In Proc. CAV'91, LNCS 575. Springer Verlag, 1991

[18] G. Ferro. AMC: ACTL Model Checker. Reference Manual. IEI-Internal Report, B4-47, 1994

[19] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In 14th Annual Symposium on Logic in Computer Science. IEEE Computer Society Press, 1999

[20] R. Focardi and R. Gorrieri. The Compositional Security Checker: A tool for the verification of information how security properties. IEEE Transaction on Software Engineering, 23(9):550–571, 1997

[21] M.J. Gabbay and A.M. Pitts. A new approach to abstract syntax involving binders. In 14th Annual Symposium on Logic in Computer Science. IEEE Computer Society Press, 1999

[22] A. Gordon, Notes on Nominal Calculi for Security and Mobility, In Proc. FOSAD Sumer School, LNCS 2171, Springer, 2001

[23] S. Gnesi and G. Ristori. A Model Checking Algorithm for π-calculus agents. In Advances in Temporal Logic. Kluwer Academic Publishers, pp.339-357, 2000

[24] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. Journal of ACM 32(1), pp. 137-161, 1985

[25] K. Honda, Elementary Structures for Process Theory (1): Sets with Renaming. Mathematical Structures in Computer Science 10:617–613, Cambridge University Press, 2000

[26] D. Kozen. Results on the propositional μ-calculus. Theoretical Computer Science 27, pp. 333-354, 1983

[27] M. Lowe. An attack on the needham-schroeder public-key authentication protocol, Information Processing Letter, 56(3):131–133, 1995.

[28] E. Madelaine and D. Vergamini. AUTO: A verification tool for distributed systems using reduction of finite automata networks. Formal Description Techniques II, 1990

[29] R. Milner. Communication and Concurrency. Prentice-Hall, 1989

[30] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (parts I and II). Information and Computation, 100:1–77, 1992

[31] R. Milner, J. Parrow, and D. Walker. Modal logic for mobile processes. In Theoretical Computer Science 114:149-171, 1993

[32] U. Montanari and M. Pistore. Checking bisimilarity for unitary PI-calculus. In Proc. CONCUR'95, LNCS 962. Springer Verlag, 1995.

[33] U. Montanari and M. Pistore. π-calculus, structured coalgebras and minimal HD-automata. In Proc. MFCS'00, LNCS 1893. Springer, 2000.

[34] U. Montanari and M. Pistore. Structured Coalgebras and Minimal HD-automata for the PI-Calculus, MFCS 2000 special issue of Theoretical Computer Science.

[35] R. Needhan. In Distributed Systems (Mullender, Ed). Addison-Wesley, 1989

[36] 36. M. Pistore. History Dependent Automata. PhD. Thesis TD-5/99, Universit`a di Pisa, Dipartimento di Informatica, 1999

[37] Montanari, U. and Pistore, M., Checking Bisimilarity for Finitary pi-calculus, in: Insup Lee, Scott A. Smolka, Eds., CONCUR'95: Concurrency Theory, Springer LNCS 962, pp. 42-56.

[38] Montanari, U., Pistore, M., and Yankelevich, D., Efficient Minimization up to Location Equivalence, in: Hanne Riis Nielson, Ed., Programming Languages and Systems - ESOP'96, Springer LNCS 1058, pp. 265-279

[39] Montanari, U. and Pistore, M., History Dependent Verification for Partial Order Systems, in: D. Peled, V.Pratt, and G. Holzmann, Eds., Procs. Partial Order Methods in Verifications, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol.29, 259-272, 1996

[40] Montanari, U. and Pistore, M., Minimal Transition Systems for History-Preserving Bisimulation, in: Ruediger Reischuk, Michel Morvan, Eds., STACS 97, Springer LNCS 1200, 1997, pp. 413-425

[41] Ferrari, G., Ferro, G., Gnesi, S., Montanari, U., Pistore, M. and Ristori, G., An Automata Based Verification Environment for Mobile Processes, in: Ed Brinksma, Ed., Tools and Algorithms for the Construction and Analysis of Systems, Springer LNCS 1217, 1997, pp. 275-289

[42] Ferrari, G., Gnesi, S., Montanari, U., Pistore, M. and Ristori, G., Verifying Mobile Processes in the HAL Environment, in: Alan J. Hu and Moshe Y. Vardi, Eds., CAV'98, Springer LNCS 1427, pp.511-515.

[43] Montanari, U. and Pistore, M., An Introduction to History Dependent Automata, in: Andrew Gordon, Andrew Pitts and Carolyn Talcott, Eds, Second Workshop on Higher-Order Operational Techniques in Semantics (HOOTS II), ENTCS, Vol. 10, 1998

[44] Montanari, U. and Pistore, M., Finite State Verification for the Asynchronous Pi-Calculus, in: W. Rance Cleaveland, Ed., TACAS'99, Springer LNCS 1579, pp.255-269, 1999.

[45] Montanari, U. and Pistore, M., Pi-Calculus, Structured Coalgebras and Minimal HD-Automata, in: Mogens Nielsen and Branislav Roman, Eds., Proc. MFCS 2000, Springer LNCS 1983

[46] Ferrari, G., Montanari, U. and Pistore, M., Minimizing Transition Systems for Name Passing Calculi: A Co-algebraic Formulation, in: Mogens Nielsen and Uffe Engberg, Eds., FOSSACS 2002, Springer LNCS 2303, pp.129-143

[47] Buscemi, M. and Montanari, U., A First Order Coalgebraic Model of Pi-Calculus Early Observational Equivalence, Proc. CONCUR 2002, Springer LNCS

[48] Montanari, U. and Pistore, M., History-Dependent Automata, Technical Report TR-98-11, Dipartimento di Informatica, Pisa

[49] Montanari, U. and Pistore, M., Structured Coalgebras and Minimal HD-Automata for the pi-Calculus, Technical Report 0006-02, IRST-ITC, 2000

[50] M. Carpano. Automatic display of hierarchized graphs for computer aided decision analysis. IEEE Transactions on Software Engineering, SE-12(4):538–546, April 1980

[51] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem-Phong Vo. A Technique for Drawing Directed Graphs. IEEE Trans. Sofware Eng., 19(3):214–230, May 1993

[52] Frances J. Newbery. Edge Concentration: A Method for Clustering Directed Graphs. In 2nd International Workshop on Software Configuration Management, pages 76–85, October 1989. Published as ACM SIGSOFT Software Engineering Notes, vol. 17, no. 7, November 1989

[53] Stephen C. North. Neato User's Guide. Technical Report 59113-921014-14TM, AT&T Bell Laboratories, Murray Hill, NJ, 1992

[54] K. Sugiyama, S. Tagawa, and M. Toda. Methods for Visual Understanding of Hierarchical System Structures. IEEE Transactions on Systems, Man, and Cybernetics, SMC-11(2):109–125, February 1981

[55] JohnWarfield. Crossing Theory and HierarchyMapping. IEEE Transactions on Systems, Man, and Cybernetics, SMC-7(7):505–523, July 1977

[56] The Zope Book the o cial on line Zope manual. http://zope.org/Documentation/Books/ZopeBook/2 6Edition/

Appendix A: Grammars

Formal syntax of Pi Agents

```
commands
        : /* empty */
        | command commands
        ;
command
        : "define" IDENT "(" param ")" "=" pi_term
        | "build" IDENT
        | "write_hd" IDENT
        | "const" IDENT
        ;

pi_term
        : "nil"
        | IDENT "?" "(" IDENT ")" "." pi_term
        | IDENT "!" IDENT "." pi_term
        | "tau" "." pi_term
        | pi_term "|" pi_term
        | "|" "(" pi_term_list ")"
        | pi_term "+" pi_term
        | "+" "("  pi_term_list ")"
        | "(" IDENT ")" pi_term %prec "."
        | "[" IDENT "=" IDENT "]" pi_term %prec "."
        | IDENT "(" param ")"
        | "(" pi_term ")"
        ;

param   : /*empty*/
        | param_aus
        ;

param_aus
        : IDENT
        | param_aus "," IDENT
        ;

pi_term_list
        : pi_term
        | pi_term_list "," pi_term
        ;
```

Formal syntax of PI-LOGIC formulae

Token Definitions:

```
letters     [A-Z][a-z]
SYMBOLS     , . / ; ' ' < > \ ? : ~ [ ] { } - _ = + ! ^ | @ ( )
IDENT        letters{letters} | SYMBOLS
STRING      SYMBOLS | letters{letters}* | "{any ascii char}*"
CONSTANT    SYMBOLS | letters{letters}*
INFIX       SYMBOLS | letters{letters}*
UNARY       SYMBOLS | letters{letters}
PREFIX      SYMBOLS | letters{letters}
```

PI-LOGIC Formulae Syntax:

```
<estate> :=
             true
           | false
           | "~" <estate>                          NOT
           | <estate> "&" <estate>            AND
           | <estate> "|" <estate>            OR
           | <estate> "->" <estate>
           | "A" <epath>                           FOR ALL PATH START FROM A STATE
           | "E" <epath>                           FOR ANY PATH START FROM A STATE
           | "[" <afor> "]" <estate>
           | "<" <afor> ">" <estate>
           | "E""X" "{"  <afor> "}" <estate>
           | "A""X" "{"  <afor> "}" <estate>
           | "E""F" "{"  <afor> "}" <estate>
           | "A""G" "{"  <afor> "}" <estate>
           ;

<afor> :=
             true
           | false
           | <afor> "&" <afor>                        AND
           | <afor> "|" <afor>                        OR
           | <exp>                                          ACTION
           ;

<exp> :=
             STRING
           | CONSTANT
           | UNARY <exp>
           | <exp> INFIX <exp>
           | PREFIX "(" <exp> ")"
           ;
```

Formal syntax of ACTL formulae

Token Definitions:

```
letters     [A-Z][a-z]
SYMBOLS     , . / ; ' ' < > \ ? : ~ [ ] { } - _ = + ! ^ | @ ( )
IDENT       letters{letters} | SYMBOLS
STRING      SYMBOLS | letters{letters}* | "{any ascii char}*"
CONSTANT    SYMBOLS | letters{letters}*
INFIX       SYMBOLS | letters{letters}*
UNARY       SYMBOLS | letters{letters}*
PREFIX      SYMBOLS | letters{letters}*
```

ACTL Formulae Syntax:

```
<estate> :=
          true
        | false
        | "~" <estate>                    NOT
        | <estate> "&"  <estate>            AND
        | <estate> "|"  <estate>           OR
        | <estate> "->" <estate>
        | "A" <epath>                     FOR ALL PATH START FROM A STATE
        | "E" <epath>                     FOR ANY PATH START FROM A STATE
        | "[" <afor> "]" <estate>
        | "<" <afor> ">" <estate>
        | <estate> "<" <afor> ">" <estate>
        | <macroname>                     CONSTANT MACRO
        | <macroname> "[" <parms> "]"     MACRO WITH PARAMETERS
        ;

<epath> :=
          "X" "{" <afor> "}" <estate>     NEXT OPERATOR
        | "T" <estate>                    NEXT WITH TAU
        | "G" <estate>                    ALWAYS
        | "F" <estate>                    EVENTUALLY UNTIL OPERATORS
        | "[" <estate> "{" <afor> "}"
        | "[" <estate> "{" <afor> "}" "U" <estate> "]"
        | "[" <estate> "{" <afor> "}" "U" "{" <afor> "}" <estate> "]"
        ;

<afor> :=
          true
        | false
        | <afor> "&" <afor>                   AND
        | <afor> "|" <afor>                   OR
        | <fc2exp>                            ACTION  (AN FC2 EXPRESSION)
        | <macroname>                         CONSTANT MACRO OF ACTION FORMULA
        | <macroname> "[" <action_parms> "]" ACTIONS MACRO WITH PARAMETERS
        ;

<fc2exp> :=
          STRING
        | CONSTANT
        | UNARY <fc2exp>
        | <fc2exp> INFIX <fc2exp>
        | PREFIX "(" <fc2exp> ")"
        ;
```

```
The ACTL actions are FC2 formulae.

<parms> :=
          <estate>
        | "{" <afor> "}"
        | <estate> "," <parms>
        | "{" <afor> "}" "," <parms>
        ;

<action_parms> :=
          "{" <afor> "}"
        | "{" <afor> "}" "," <action_parms>
        ;

<formal_parms> :=
          STRING
        | "{" STRING "}"
        | STRING "," <formal_parms>
        | "{" STRING "}" "," <formal_parms>
        ;

<action_formal_parms> :=
          "{" STRING "}"
        | "{" STRING "}" "," <action_formal_parms>
        ;

<macroname> := STRING;
```