VERIFICATION OF PROCEDURAL REASONING SYSTEM (PRS) PROGRAMS USING COLOURED PETRI NETS (CPN)

Ricardo Wagner de Araújo
and Adelardo A. Dantas de Medeiros
Federal University of Rio Grande do Norte - Brazil
rwagner,adelardo@dca.ufrn.br

Abstract PRS (a tool based on procedural reasoning) has inspired several works in Artificial Intelligence, mainly in embedded and industrial applications. This paper proposes a verification mechanism of PRS programs, based on equivalence rules with Coloured Petri Nets (CPN). This equivalence allows using existing analysis methods for coloured Petri nets to verify PRS programs.

1. Introduction

One of the biggest difficulties in the project of expert systems for embedded or industrial applications is the real-time control of its execution. Many of the usual techniques of knowledge representation and inference in Artificial Intelligence do not have a guaranteed bounded response time. Purely procedural systems, in the other way, are not flexible enough to allow easily incorporating empirical or heuristic knowledge of human experts.

Declarative representations suffer from a lack of control on the execution of their rules, while imperative ones suffer from limited modularity. A promising trade-off for these situations is using Procedural Reasoning. Procedural Reasoning is particularly well suited for problems where implicit or explicit knowledge is already formalized as procedures or plans.

Procedural Reasoning [Georgeff and Lansky, 1986; Ingrand et al., 1992] is a set of tools and methods for representing and executing plans and procedures. These plans or procedures are conditional sequences of actions which can be run to achieve given goals or to react in particular situations. It differs from other usual knowledge representations because it preserves some flow information (i.e., the sequence of action and tests) associated to declarative aspects.

Some of the widely used implementations of procedural reasoning are Reactive Planning (RAP) [Firby, 1987] and PRS (Procedural Reasoning System)

system, originately developed at SRI International [SRI International, 1994]. In our work, we use PRS because it expresses very weel the trade-off between purely declarative and strictly imperative representations and it has good temporal properties and high expressive power. PRS has been used in practical situations as real-time applications – such as supervision and control of mobile robots [Ingrand and Despouys, 2001; Ingrand et al., 1996], failure detection in space shuttles [SRI International, 1994] as well as in the management of communication networks and alarm systems.

In complex knowledge-based systems, however, sometimes the real-time properties and the power of expression are not enough, and we also need verifying such systems as for correctness, reliabilitiy, performance and efficiency [Bench-Capon et al., 1993; Lee and O'Keefe, 1994]. PRS does not offer a verification mechanism. This work proposes some equivalence rules between programs written using a PRS subset and Coloured Petri Nets (CPNs). Thus, the existing methods to analyze CPNs can be used to validate the equivalent PRS program.

Petri nets are widely used to verify knowledge-based systems, as in the PREPARE [Zhang and Nguyen, 1993] and Task Structures [Lee and Lai, 2002] systems. We adopted Petri nets as the verification tool for modelling PRS programs because of their capability to model parallelism and non-determinism and also because it is possible to determine exact conversion rules between PRS structures and their equivalent in Petri nets.

2. PRS language

PRS main characteristics will be better explained in section 4. For now, we can say that a PRS agent consists of:

- `Routine library:` each routine, or KA (knowledge area), is a sequence of action and/or tests that can be executed to reach goals or to react to situations;
- `Database:` it contains the system current beliefs about the world;
- `Current goals sets:` in PRS, the goals describe desired tasks or behaviors. In the logic used by PRS, the goal to achieve a certain condition `C` is written as `(! C)`; to test the condition, as `(? C)`; to wait until the condition is true, as `(@ C)`; to maintain `C`, as `(# C)`; to assert the condition `C` in the database, as `(=> C)` and to retract it, as `(Λ>C)`; and
- `Intention graph:` a dynamic set of routines, structured as a graph, where the system keeps information in real-time about the state of the routines chosen for execution and of their posted subgoals.

Each KA has a body and some invocation conditions. Figure 1 shows a KA that solves Tower of Hanoi. This KA is executed when an order to reach a goal is posted (as the `INVOCATION` field shows). It can be selected if the `CONTEXT`

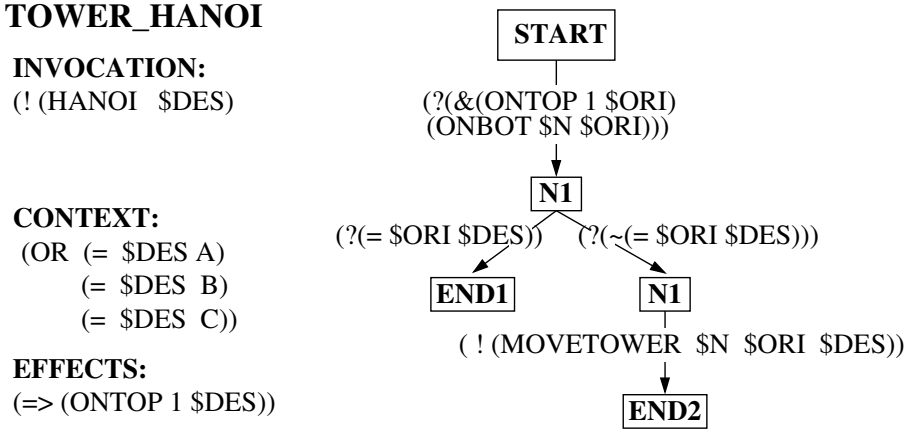

Figure 1. Tower of Hanoi KA.

field conditions are satisfied. Conditions contain variables to bound (`$DES`). There are two variable types in PRS: logical variables `$var` have the classic behavior of variables in logical programming (once bound, they do not change their value), while the program variables `@var` can be re-attributed.

The KA's body describes its execution. The execution initiates in `START` node and continues following the arcs through the net. If more than an arc leaves a node, any of them can be crossed. To cross an arc, the system must either test if the goal associated to the arc was already established in the database or launch a KA that reaches the goal associated to arc. This new goal will be incorporated to the intention graph and has to be satisfied before the current KA can be satisfied. If the system cannot satisfy any of the goals associateds to the arcs that leave a node, the entire KA fails. But when a terminal node (`END` node) is reached, the goal is considered satisfied and the facts listed in `EFFECTS` are concluded in the database.

3. The Coloured Petri Net - CPN

The Coloured Petri Nets theory is widely known and can be found in the literature [Jensen, 1994]. A modeling example [Jensen, 1998] of an enabled transition for a resource allocation system by CPNs is shown in Figure 2.

Each place (circles that describe the system's states - B,C,S) contains a set of markers called `tokens` which carries a data value and belongs to `type` (*P* and *E*, in this case). The place `B` has two tokens in the initial state of color (p,0), *P* type. The place `S` has three tokens of color(e), *E* type.

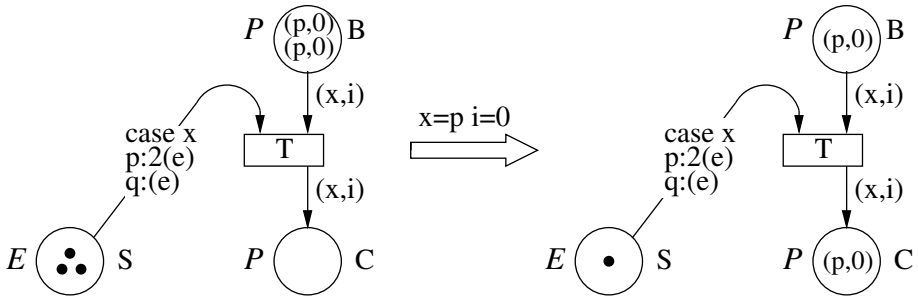

Figure 2. CPN - Transition Faring.

To be fired, a transition (rectangles that describe the actions) must have sufficient tokens on its input places, and these tokens must have values matching the arc expressions (arrows that describe how the CPN's state changes when the transitions occur). Consider the transition `T`. It has two arcs. The two arc expressions involve the variables `x` and `i`. To fire the transition, we have to `bind` these two variables to values in their types, in such a way that the arc expression of each incoming arc evaluates to a token value that is present on the corresponding input place.In this case, `x=p` and `i=0`. The right side of the Figure 2 shows the new state after the `T` transition fires.

4. PRS and the Coloured Petri Nets

Modeling Principle

To determine a CPN equivalent to PRS one associates network places to PRS nodes and transitions or sub-networks to the PRS arcs (see Figure 3). The token displacement models the evolution of the execution and its color contains the variables values currently defined. The number of colors has to be finite to generate a analyzable CPN: this implies we can only have "closed" variables (with a finite domain of values) in the PRS program we want to model.

To each arc we associate two branches in the CPN: one corresponding to case where the goal is reached (the token goes to the place that corresponds to the KA's following node) and the other corresponding to the case where the goal fails (the token's destination varies).

A necessary condition for a sub-net be a possible CPN model of a PRS arc is that it can be reduced to a token "dispatcher". If a sub net consumes a token in its input it has to produce a token in one of its two outputs: one models the attainment of the goal or the other one models its failure (indicated for `T` and `F` respectively, in the drawings).

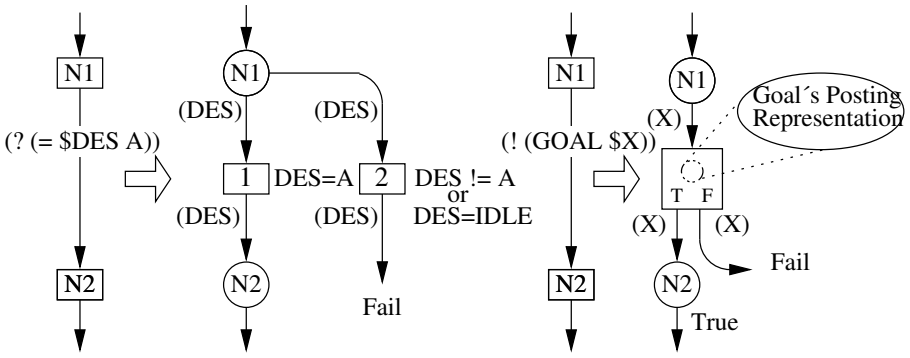

Figure 3. Two examples of equivalence between PRS and CPN.

The Database

There are different predicates types in PRS. We will only present the representation of the standard predicates.

Each standard predicate is represented by four places, called `AFF DEF` and `AFF UNDEF`, `NEG DEF` and `NEG UNDEF`. The corresponding places `DEF` and `UNDEF` are complementary (the token that leaves a complementary place must return to the same place or goes to the other complementary place). The presence of one token of a certain color in a `DEF` place indicates that, for the corresponding value, the predicate was concluded affirmatively (`AFF`) or negatively (`NEG`). In a similar way, one token in an `UNDEF` place indicates that the predicate was not concluded (affirmatively or negatively) for this value. To use the technique of complementary places, some constraints are imposed:

- The places have to be bound (have a maximum number of tokens);
- The number of colors has to be finite;
- For each pair of complementary places, the initial marking guarantees the existence of a token of each color in one of the two places;
- In the initial marking, there are not tokens of the same color simultaneously in the `AFF` and `NEG` places.

Consider a predicate `(ONTOP disc peg)`, where `disc` $\in$ `{1,2,3}`, `peg` $\in$ `{A,B,C}` and `ONTOP` is undefined in the beginning for all the combinations `(disc peg)`. Figure 4 shows the predicate's representation after the conclusions `( => (ONTOP 1 A))` , `(=> (ONTOP 3 C))` and `(=> (^ (ONTOP 3 B)))`.

The tests (? (...)) on standard predicates are represented by CPN as shown in Figure 5. We only show the affirmation test model `(? (PEG $ORI))`; the

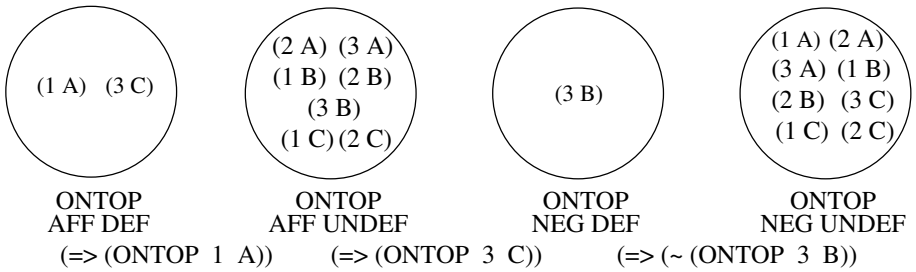

Figure 4. Standard Predicate.

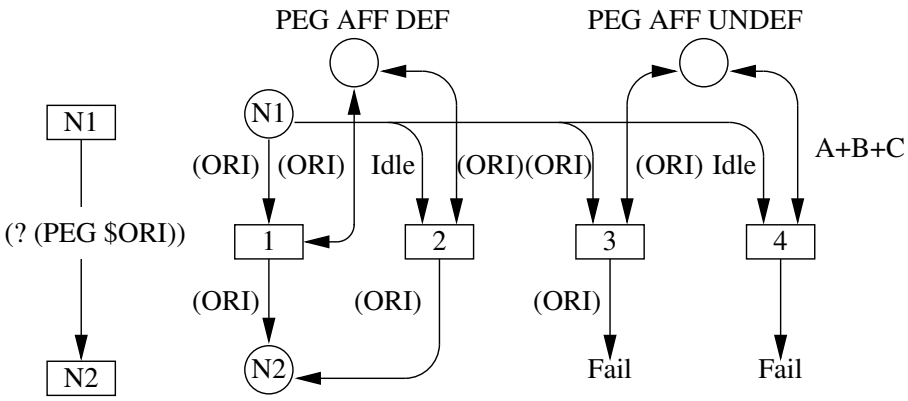

Figure 5. Standard Predicate Test.

negative test model `(? (^ (PEG $ORI)))` is identical, replacing `PEG AFF DEF` and `PEG UNDEF` by `PEG NEG DEF` and `PEG NEG UNDEF`, respectively.

Every token removed of one of the places that models `PEG` must be put in the same place, because a test does not modify the predicate value. The model of behavior of a test arc is different (different transitions are validated) according to predicate's argument. For example:

- The argument is already bound (has a precise value):
  - If for this value the predicate is defined, the test is successful (transition 1). If the predicate is undefined, the test fails (transition 3).
- The argument is not bound (`IDLE`):
  - If there is a value for which the predicate is defined, the test is successful and this value is transmitted (`transition 2`). This rep-

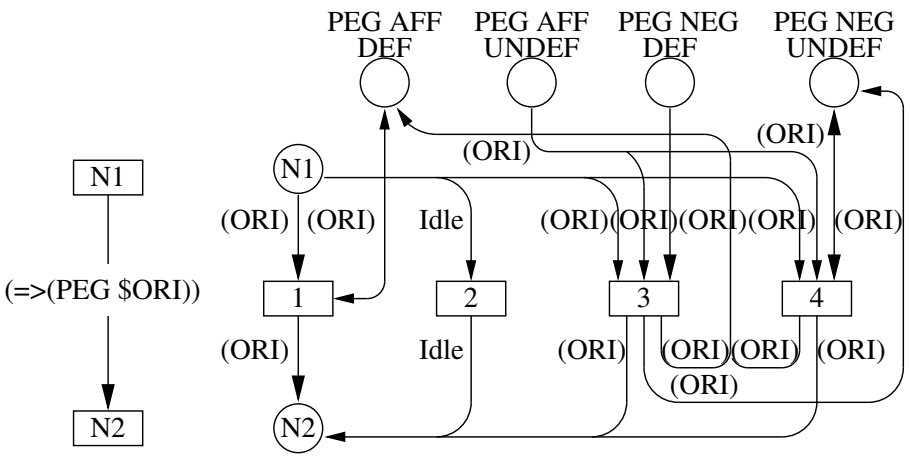

Figure 6. Standard Predicate Assert in Database.

resents unification mechanism of the variable. If `PEG` is undefined for all values, the test fails (`transition 4`).

We show in Fig. 6 the CPN equivalent to the assertive `(=>(PEG $ORI)))`. For the conclusion of the negation, i.e., `(=>(^ (PEG $ORI)`, we have only to exchange the corresponding `AFF` and `NEG` places. We eliminate an eventual affirmation of the predicate when concluding its negation (`transition 3`). A conclusion with non-bound arguments (an error) does not change the predicate's state (`transition 2`).

The Variables

Variables are carried by tokens, since their birth until the moment that they are not necessary anymore. Figure 7 shows an example of variables transmission modeling. Arc models are simplified because the KA's context field guarantees that the variables `$X` and `$Y` are bound.

The token will be a n-tuple with as many elements as PRS variables to preserve. Elements of the n-tuple will have as far possible values as the possible values for the PRS variables they represent, plus one (`IDLE`) to indicate that the variable is non-unified still.

The Goals

When a goal is posted, it can be satisfied either by consulting the database or by calling a KA that satisfies this goal. For each predicate that can be posted, there must be, beyond the places that represent the predicate in the database

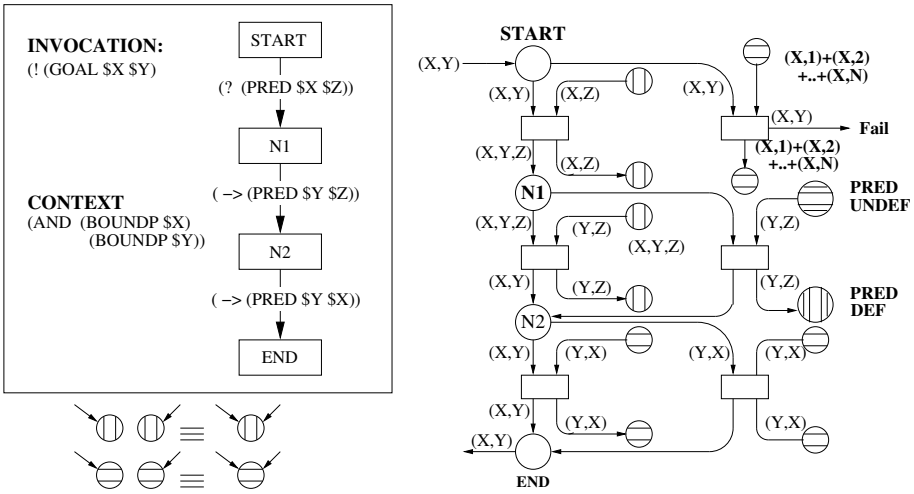

Figure 7. Variables' Transmission Modeling.

(see section 4), three other specific places: `POST`, where the "client" routine indicates that the goal was posted, and `SUCC` and `FAIL`, where the "server" routine indicates the success or failure in attaining of the goal.

Figure 8 shows a goal posting complete model. At the moment of posting the goal, the CPN puts a token (with the goal's arguments colors) in the `POST` place and the execution of this branch enters in a wait state (`place W`) for a reply about the attainment of the goal. This reply will be materialized by the appearance of a token in either `SUCC` or `FAIL` place.

After the goal's posting, the model verifies if the predicate is not already satisfied in the database by a test (as in Figure 5). Otherwise, the system will look for a KA invoked by this goal. If one exists, the KAs will be activated and decide on the satisfaction or not of the goal, as presented in Figure 8.

The Nodes

Normally, each PRS' node corresponds to a place in the CPN, except for the special type: the `(IF-THEN-ELSE)` condition node.The crossing of an arc that arrives at a condition node is always possible. If the goal associated with the arc is reached, the execution continues from the `T` half of the condition node; otherwise, from the `F` half. This node is represented by two places, one for each half. As each model of an arc must generate two possible exits for the token, we drive the branch that represents the success in the attainment of the goal to the `T` place and the branch that models the failure to the `F` place.

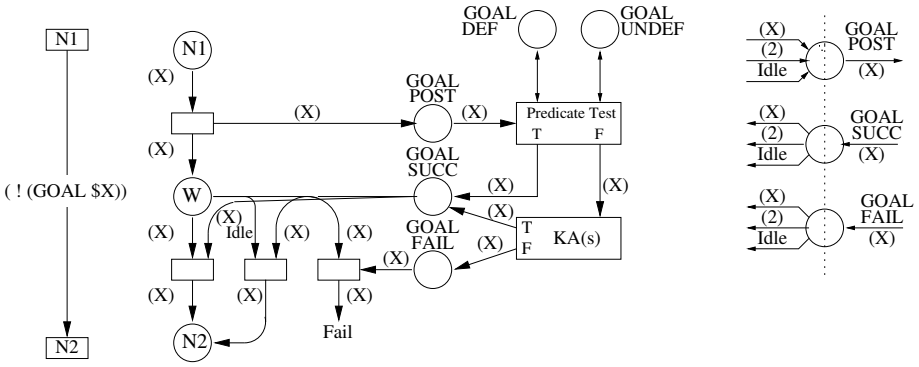

Figure 8. Goal Posting.

The non determinism

In PRS when several arcs leave from the same node, the system chooses one arc at random to try to cross it. If it does not happen, it chooses one another arc and only considers that the KA failed after to have attempted all the arcs. PRS does not perform backtracking: if it gets to cross an arc, but it fails in the continuation of this branch it does not come back to the node in order to test the other unexplored arcs.

Given this behavior, it is necessary to conceive a structure to guarantee that the model tests all the arcs before designating an error. This structure is shown in Figure 9 for a case with two conflict arcs , but it is extensible for any arcs number.

Initially, the `A place` contains two numbered tokens, one for each arc. The only token in `N1*` guarantees the execution of only one arc at a time. If the execution of the arc that caught the token in `N1*` will have success, the model cleans the `A` and `B` places (it consumes all the tokens of the other branches) and the execution continues. In case of failure, the model replaces the token in `N1*`, authorizing another arc to be tested and puts the corresponding token to the arc that failed in `B`. If all the tokens are in `B`, the model indicates an error.

The KAs

The CPN that models a KA has a start place called `INIT`. Between `INIT` and `START` there is a sub-net that tests if the KA's execution context (`CONTEXT` field) was satisfied. This is equivalent to add a supplementary arc in the KA to play the role of `CONTEXT` field, it does not modify the execution of the program (see Figure 10). In the same way, between the `END` place, that plays the role of

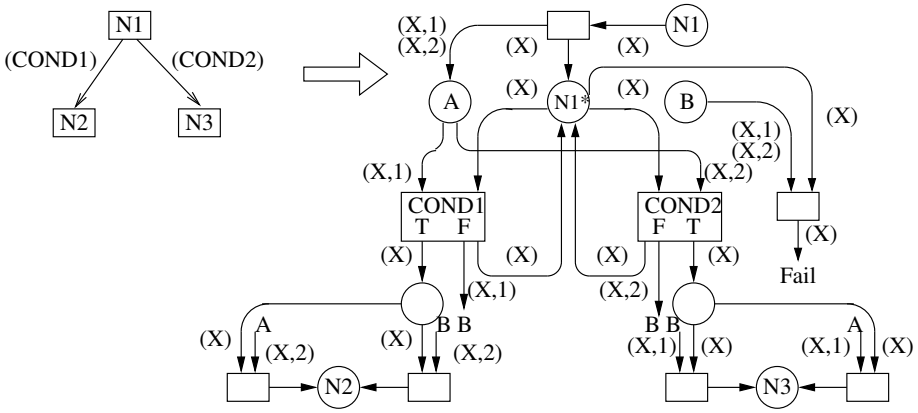

Figure 9. Node with some arcs that leave

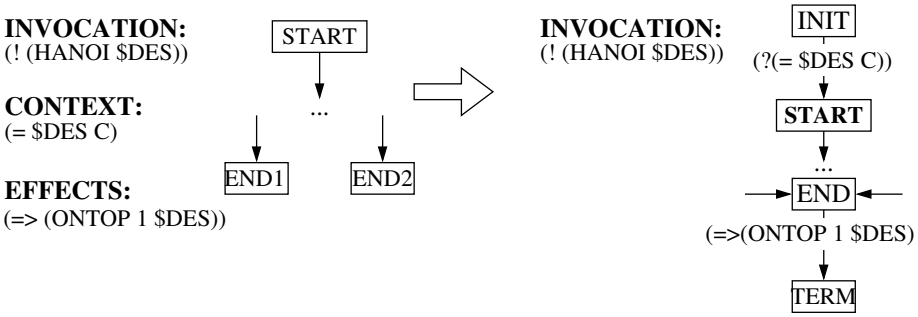

Figure 10. Equivalence between KAs' Fields and Arc.

all the terminals nodes of the KA, and the `TERM` place, that models successfully the KA execution ending, it has a sub-net that models the `EFFECTS` field.

The models in CPN of the KAs must also contain a `FAIL` place, where the appearance of one token indicates that the KA failed. The exit, in case of error (`F`), of all the arcs must be directed to this `FAIL` place, except for the special situations of the arcs that lead to a `IF-THEN-ELSE` node (section 8) and of the arcs that leave from a same node (section 8). Figure 11 shows a CPN equivalent to a generic KA. The arc models are not detailed and we assume that there is only one variable to transmit (`DES`) in the whole KA.

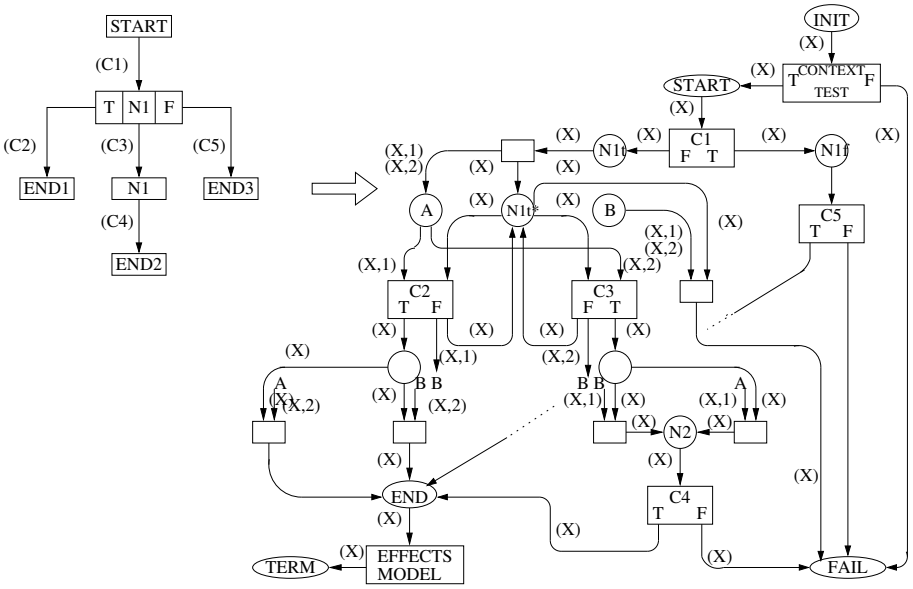

Figure 11. CPN Equivalence to KA.

5. Verification Methods

Summarily, we can say that the CPN can be analyzed in three main ways: by simulation, by acessibility graph (state space or occurrence graph) and by invariant methods (of place or transition).

Analysis methods allow to prove a number certain of Petri nets properties. These properties can be behavioral (for each initial marking) or structural (only depending on the structure of the net). The invariants establish certain structural properties and the accessibility graph allows, by inspection of the states, to determine all the net's properties, but only for a well accurate initial marking.

For the CPN of our model, the properties of interest are mainly the ones which were proved for a defined initial marking, for two reasons:

- There are complementary places in the net. So, the initial marking must guarantee the existence of the mandatory tokens in one of the complementary places.
- Normally, we are interested in guaranteeing the behavior of the system in specific situations, corresponding to well-defined initial markings.

The properties demonstrated for the CPN must be interpreted knowing that the net models a set of KAs and what the initial marking represents. A non-limited place in number of marks, for example, indicates almost certainly an error in the writing of the KAs (as a KA that recursively calls itself without limits). The existence of a blockage (when there is no fireable transition and the net "dies") can indicate a conception error or, in contrast, prove that the functioning of the system is correct:

- If the expected behavior of the system for one given initial marking is that it enters in an endless cycle, the existence of a blockage possibility indicates a project error.
- For an initial marking where there is a posted goal, it is necessary that CPN finishes in blockage, with only one mark in the places that indicate the goal success or failure.

6. Conclusion

Because of space limitation, we presented only some of the equivalence rules between PRS programs and CPN that allows using the existing analysis tools, as CPN TOOL [Jensen, 2001], to verify PRS programs.

The main limitations of the proposed approach are the obligatory previous knowledge of the variable's finite domain and also the possible considerable size of the generated CPNs. The first limitation is not so restrictive in embedded or industrial applications, where usually the variables can only assume predefined values. The second one imposes the automatic generation and analysis of the Coloured Petri Nets.

The manual conversion between PRS and CPN can introduce errors. To avoid this, we are developing a (PRS to CPN) automatic converter in Common Lisp [Shapiro, 1992]. This is possible because all the equivalence rules presented here (and the other ones) follow generic laws and then can be translated into conversion rules able to be implemented by a conversion software to be used in practical situations.

Hence, we can use the CPN's analysis to investigate the main properties of the generated net and, consequently, indirectly to verify the PRS program that generated the net.

Acknowledgment

The authors would like to thank the anonymous reviewers for their constructive comments as well the Professors Kurt Jensen and François Félix Ingrand by the use of CPN TOOLS and OpenPRS softwares and the Brazilian agency CAPES.

References

Bench-Capon, T., Coenen, F., Nwana, H., and Paton, R. (1993). Two aspects of the validation and verification of knowledge-based systems. In *IEEE Expert*, pages 76–81.

Firby, James R. (1987). An investigation into reactive planning in complex domains. In *Proceedings of the 6th National Conference on Artifical Intelligence*, Seattle, WA, USA.

Georgeff, Michell P. and Lansky, A. L. (1986). Procedural knowledge. In *Proceedings of the IEEE*, volume 10, pages 183–198.

Ingrand, François Felix, Georgeff, Michael P., and Rao, Anand S. (1992). An architecture for real-time reasoning and system control. In *IEEE Expert*, volume 6, pages 34–44.

Ingrand, François Félix, Chatila, Raja, Alami, Rachid, and Robert, Frédéric (1996). PRS: A high level supervision and control language for autonomous mobile robot. In *ICRA - IEEE International Conference on Robotics and Automation*, volume 1, pages 43–49, Minneapolis, Minnesota, USA.

Ingrand, François Félix and Despouys, Olivier (2001). Extending procedural reasoning toward robot actions plannig. In *ICRA - IEEE International Conference on Robotics and Automation*, Seoul, South Korea.

Jensen, Kurt (1994). *Coloured Petri Nets - Analysis Methods*, volume 2. Springer Verlag, Aarhus, Denamark.

Jensen, Kurt (1998). A brief introduction to coloured petri nets. In Brinksma, E., editor, *Lecture Notes in Computer Science: Tools and Algorithms for the Construction and Analysis of Systems. Proceedings of the TACAS'97 Workshop*, pages 201–208, Enschede, The Netherlands. Springer-Verlag.

Jensen, Kurt (2001). CPN TOOLS. `http://www.daimi.au.dk/Acpntools`.

Lee, J. and Lai, Lein F. (2002). A high-level petri nets-based approach to verifying task structures. In *IEEE Transactions on Knowledge and Data Engineering*, volume 14, pages 316–335.

Lee, J. and O'Keefe, R.M. (1994). Developing a strategy for expert system verification and validation. In *IEEE Transactions on Systems, Man, and Cybernetics*, volume 24, pages 643–655.

Shapiro, Stuart Charles (1992). *Common LISP - An Interactive Approach*. W.H. Freeman and Company, USA.

SRI International (1994). Shuttle malfunction handling. `http://www.ai.sri.com/`.

Zhang, D. and Nguyen, D. (1993). A tool for knowledge base verification. In Bourbakis, Nikolaos G., editor, *Knowledge Engineering Shells: Systems and Techniques*, volume 2 of *Advanced Series on Artificial Intelligence*, pages 455–486. World Scientific Pub Co, USA. ISBN: 9810210566.