A Temporal Specifications Checker for Petri Nets

Pascal Racloz
Swiss Federal Institute of Technology
DI-LCO
In-Ecublens CH-1015 Lausanne

ABSTRACT

In this article we present a model-checking method for systems modelled with Petri nets. Our approach is based on a definition of a symbolic representation of sets of markings, with the intended properties of the net expressed by using a specific temporal logic. On one hand, the symbolic representation may structure the state space of the net, thus allowing the consideration of larger systems while, on the other hand, the temporal properties are expressed in a very intuitive temporal language.

1. INTRODUCTION

When the modelling of the system is accomplished with a Petri net [8], its properties can be studied by means of the graph of the reachable states of the net. Unfortunately, in the case of large systems, the size of the graph may be very large and difficult, if not impossible, to analyse.

Attempts to overcome this difficulty can be classified roughly into two approaches. The first approach takes advantage of the symmetric aspects of the system, with respect to either its components or its behavior. The second, alternative approach defines efficient representations of the states of a Petri net.

Our method corresponds to the second approach, as it defines a symbolic representation of a set of states of a Petri net, and it has the purpose of analysing the system by expressing first its intended behavior by means of a temporal formula and then verifying the accuracy of the modelling with respect to these specifications, i.e. is the net a model of the formula?

The temporal logic [4] provides a formal framework for describing and reasoning about the correct behavior of systems. Within such a logic, a formula expresses qualitative properties related to the occurrence of the events of a system over a period of time. Its basic properties are based on the notions of invariance, liveness and precedence. The traditional proof method, called model-checking [3], is an algorithm which, exhaustively and iteratively, labels the states of the system with particular temporal formulae. Given a formula which expresses the intended behavior of the system, once the algorithm is terminated, a state satisfies the formula when the latter belongs to the label of the state. This technique can only be applied in the case of finite state systems which have a reasonable number of states. By structuring the states space of a system, the symbolic representation defined may then handle systems with large number of states.

Within our framework, proofs are accomplished by first computing incrementally the symbolic representation for which the formula is true and then checking to see if the state of the system on which the formula must be proved belongs to this set.

This article is structured as follows. The next section, Section 2, introduces the Petri net formalism. Section 3 gives an informal presentation of the temporal logic used in the context of Petri nets. Our symbolic representation with its main features are then defined in Section 4. The approach is illustrated by applying it to the very simple example of the mutual exclusion problem in Section 5. Section 6 presents the SANDS environment in which our temporal tool is integrated, before the final, concluding remarks made in Section 7.

2. PETRI NETS

Informal Introduction

The Petri net formalism is used to model and analyse discrete systems in which notions such as interactive components, concurrence and synchronization play a key role. A Petri net model embeds structural aspects of the system as well as its evolution.

Stated briefly, the intuitive notion of a Petri net is as follows. The states of a system are represented by means of *places*, which can contain a varying number of *tokens*. A given distribution of tokens in the places represents a state and is called a *marking*. The events of the system are represented by *transitions* and the occurrence of an event corresponds to the *firing* of a transition which, in turn, depends on the existence of a minimal number of tokens at some places in the net. This minimal distribution of tokens at the places in the net is called the *input marking* of the transition. When an event may occur, its corresponding transition is *firable*. The firing of a transition generates a new state given by a new distribution of the tokens in the net. The tokens can be viewed as resources needed to generate the occurrence of events. The firing of a transition consumes some resources and generates new ones.

Definitions

In this paper, the usual Petri nets, known specifically as place-transition systems, are used. In the interest of simplicity, the term Petri nets (or nets) will be used to designate them.

A *Petri net* is a quadruplet (P, T, FR, λ) in which

- P is a set of *places* (local states) and T is a set of *transitions*. P and T are disjoint sets.
- FR $\subseteq$ (PxT) $\cup$ (TxP) is a *flow relation* representing the connections place-transition and transition-place.
- $\lambda : FR \rightarrow \mathbb{N}^+$ is a *weight function* which associates a weight to each connection.

The rule of the evolution of a net is defined with the following notions:

- Each state of a net is given by a particular function $M: P \rightarrow \mathbb{N}$ which determines the number of tokens that each place contains. Such a function is called a *marking*.
- A transition t is *firable* for the marking M, written $M \xrightarrow{t}$, iff (if and only if) $\forall\ p \in P,\ \lambda(p,t) \leq M(p)$.
- When a transition is firable, the *occurrence* or the *firing* of the transition can occur. The new marking (i.e. the new distribution of tokens) which result from the firing of t is given by M' which is defined as $\forall\ p \in P$, $M'(p) = M(p) - \lambda(p,t) + \lambda(t,p)$.
- For each transition, we define the functions $\lambda^{in}(t), \lambda^{out}(t) : P \rightarrow \mathbb{N}$ defined by $\forall\ p \in P$, $\lambda^{in}(t)(p) = \lambda(p,t)$ (number of tokens consumed by t) and $\forall\ p \in P$ $\lambda^{out}(t)(p) = \lambda(t,p)$ (number of tokens generated by t). $\lambda^{in}(t), \lambda^{out}(t)$ are respectively the *input* and *output markings* of t.
- The functions with the domain P and the co-domain $\mathbb{N}$, i.e. $P \rightarrow \mathbb{N}$, are represented by a vector which is indexed by P with components in $\mathbb{N}$. In this way, the firability condition of a transition t for M is simply written $\lambda^{in}(t) \subseteq M$ where $\subseteq$ represents the usual inclusion of markings, i.e. $M_1 \subseteq M_2$ iff for every place " p ", $M_1(p) \leq M_2(p)$.

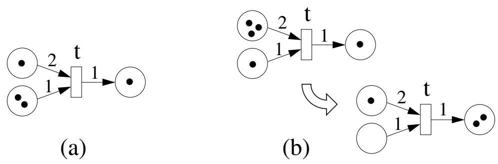

Figure 1: The Evolution of a Net: (a) t is not firable, (b) the firing of t

A net has been represented graphically in the preceding Figure 1. A place is represented by a circle and a transition by a rectangle. The connections between places and transitions are represented by labelled arcs, where the label is given by the weight function λ. The places' tokens are represented by black dots.

3. TEMPORAL LOGIC

An Overview

Temporal logic allows the expression of qualitative properties related to the arrangement of the events of a system over a period of time. By means of the temporal constraints on events, these properties characterize some aspects of the evolution of the system. Typical properties which may be expressed include:

- *Potentiality*, which specifies the existence of an evolution in the system, leading to a state satisfying a given property.
- *Invariance*, which specifies that the set of the evolutions of the system satisfies a given property. For instance, the invariance allows to express that the mutual exclusion must be fulfilled.
- *Precedence*, which specifies a temporal relation existing between some properties. Typically, precedence deals with statements in the form, 'As long as a given property is satisfied (or not), another one is (or is not) fulfilled'.
- *Fairness*, which specifies that, in cases of competition between entities for a shared resource or service, this one is attributed fairly. The fairness property can pertain to a number of different aspects.

We use the Computation Tree Logic (CTL) [3], which offers a trade-off between its expressive power and the difficulty of its decision procedures. Moreover, this logic has the small model property which states that each satisfiable formula has a finite model. CTL is a branching time logic, i.e. a state has a single predecessor and may have several direct successors.

Temporal Logic and Petri Net

The arrangement of the computations of a net in a tree-like way allows the use of branching time logic for reasoning about the net. This tree-like structure is called the *reachability tree* of the net and the direct successors of a marking are computed by the occurrence of the firable transitions from this marking (cf. Figure 2). Temporal logic specifications may then be conceived of making statements about pattern of behavior in the reachability tree.

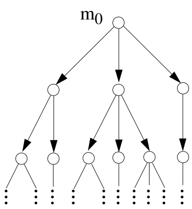

Figure 2: The Reachability Tree from the Initial State " m_0 "

Figure 3 shows a graphical interpretation of some temporal operators of CTL, namely the operator **EF**, **AF** and **AG** which deals with properties of potentiality, inevitability and, invariance, respectively. The fact that a marking " m " satisfies the property represented by a formula " p " is written " $m \models p$ ". The states represented by " ○ " and " ● " are such that the formula " p " holds, i.e. " $● \models p$ " and " p " does not hold, i.e. " $○ \not\models p$ ".

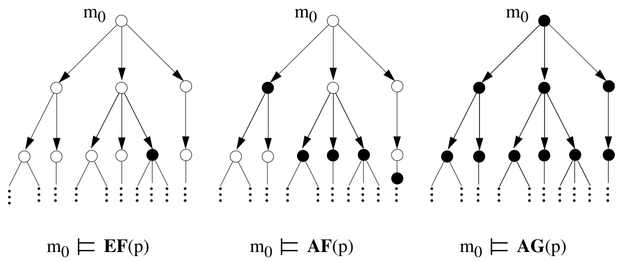

Figure 3: Some Temporal Operators of CTL

Formally, " $m_0 \models \mathbf{EF}(p)$ " iff there exists some path from " m_0 " that leads to a state at which " p " holds, " $m_0 \models \mathbf{AF}(p)$ " iff the formula " p " holds in the future along every path from " m_0 ", and " $m_0 \models \mathbf{AG}(p)$ " iff the formula " p " holds at every state on every path from " m_0 ".

In order to formulate various properties of the net behavior, we have to be able to describe that a transition is firable at some particular marking. Thus, the set of atomic propositions, " AP ", is given by the set of the transitions of the net:

$$AP = T$$

For all transitions " t " in a given Petri net, the notation *t* represents its corresponding atomic proposition. An atomic proposition *t* is satisfied by a marking " m " iff the transition " t " can be fired from " m ":

$$m \models t \Leftrightarrow m \xrightarrow{t}$$

Examples of Temporal Properties for a Net

Deadlock: Since a deadlock is a marking from which no transition can occur, " m " is a deadlock $\Leftrightarrow \forall t \in T\ m \not\models t$. Thus, the system represented by Petri net can enter in a deadlock state iff $m_0 \models \mathbf{EF}(\bigwedge_{t \in T} \neg t)$.

Activity: The quasi-live or live character of the transitions characterize the activity for a net. A transition " t " is quasi-live if it can be fired at least once, i.e. " t " is quasi-live $\Leftrightarrow m_0 \models \mathbf{EF}(\ t\ )$. The liveness of a transition is a stronger notion which states that from each marking of the reachability tree, the transition is quasi-live, i.e. " t " is live $\Leftrightarrow m_0 \models \mathbf{AG}(\ \mathbf{EF}(\ t\ )\ )$.

Temporal Operators and Predicate Transformers

By identifying the common notion of a unary predicate on states with the set of states which satisfy it, a *predicate transformer* is thus a function which maps a set of states into another set of states, i.e. a predicate into another predicate. The mapping function depends on the underlying transitions system of the system considered.

Sifakis [10] shows how some properties of invariance and non-invariance of a system can be expressed by means of predicate transformer expressions. In this context, the predicate transformer *pre* ("predecessor") is at the basis of the approach taken as a whole. This predicate transformer associates to a predicate the set of states from which the states of this predicate can be reached in a single step in the evolution of the system (i.e. its direct predecessors).

Relative to CTL, the set of states which verify a formula can be computed using predicate transformer expressions based on *pre*. For instance, Figure 4 illustrates the computation of

$$pre^{*}\ (P)\ =\ \bigcup_{i \geq 0} pre^{i}\ (P)$$

where " $pre^0(P) = P$ " and " $pre^{n+1}(P) = pre(pre^n(P))$ " for $n \geq 0$, which relates to the temporal operator **EF**.

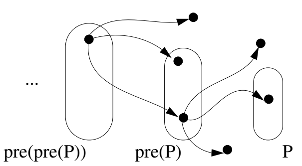

Figure 4: Computation of the Predecessors of the Predicate "P"

By construction, each state of " $pre^{*}(P)$ " is a source of a path which leads to a state of the predicate " P ". Then, by considering that the states of " P " fulfill a property " p ", we can state that

$$\forall q \in pre^{*}(P),\ q \models \mathbf{EF}(p)$$

From an operational point of view, the computations of the predicate transformer expressions are performed iteratively until a fixed point is reached. As mentioned before, all the CTL operators have a correspondence in terms of predicate transformer expressions based on *pre*. Within the framework of Petri nets, the underlying transitions system is represented by the reachability tree of the net.

4. THE SYMBOLIC REPRESENTATION

Predicate Structure

Our definition of the symbolic representation of markings is based on *predicate structures*. A predicate structure has two components which impose some constraints on the particular set of markings it represents.

Relative to a given Petri net, if " $\mathcal{M}$ " denotes the set of all possible markings, then a predicate structure is a couple " (A,B) " where " A " and " B " are subsets of " $\mathcal{M}$ ". The predicate structure " (A,B) " represents the following set of markings:

$$\{\ M \in \mathcal{M}\ /\ \forall M_a \in A,\ M_a \subseteq M \text{ and } \forall M_b \in B,\ M_b \not\subseteq M\ \}$$

Our *symbolic representation* of markings is a set of predicate structures. In general, several predicate structures are needed in order to treat the notion of disjunction of properties.

The CTL elaborate properties of Petri nets are based on their basic properties which state simultaneously that the firing of some specific transitions are possible while the firing of some other specific transitions are impossible. A predicate structure deals with a basic property as follows: the markings verifying that each transition of the set $\{t_1, ..., t_n\}$ can be fired and each transition of the set $\{t'_1, ..., t'_m\}$ can not be fired are represented by the predicate structure " (A,B) " with

$$A = \{\lambda^{in}(t_1), ..., \lambda^{in}(t_n)\} \text{ and } B = \{\lambda^{in}(t'_1), ..., \lambda^{in}(t'_m)\},$$

($\lambda^{in}(t)$ denotes the input marking of the transition " t ".)

The use of a symbolic representation for states modifies the context of the temporal proofs. The operations performed on states, such as, for instance, union or negation as well as the computation of the result of the predicate transformer *pre* must be defined within the framework of the symbolic representations.

The closeness of these symbolic operations must also be established. Let " τ " denote the transformation which associates a set of states or an operation on states with its symbolic representation and let " τ^{-1} " denote the inverse process. Then, in the particular case of unary operations, for instance, this closeness is respected when the following holds:

$$\forall Q \subseteq \text{'Set of states'},\ \forall OP \in \text{' unary operations on states'}$$
$$OP(Q) = \tau^{-1}((\tau(OP)(\tau(Q))$$

In other words, the set of states represented by the symbolic operation performed on the symbolic representation must be the same as the resulting states of the operation performed directly on the states.

The definition of the symbolic representations of the operations may be found in [6] while their closeness was already proven in [7].

Example of an Operation on a Predicate Structure

The chosen operation is the symbolic representation of the predicate transformer *pre*. This operation, written pre_S, allows the treatment of the temporal operators of CTL in the symbolic domain. To simplify the definition of pre_S, the operation preT: $\mathcal{M} \times T \rightarrow \mathcal{M}$ is introduced:

$$\forall\ (M,t) \in \mathcal{M} \times T,\ \forall\ p \in P$$

$$\mathrm{preT}(M,t)(p) = \begin{cases} \lambda^{in}(t)(p) & \text{if } M(p) \leq \lambda^{out}(t)(p) \\ M(p) + \lambda^{in}(t)(p) - \lambda^{out}(t)(p) & \text{if } M(p) \geq \lambda^{out}(t)(p) \end{cases}$$

For a marking " M " and a transition " t ", the operation " preT(M,t) " calculates the minimum marking such that, after the occurrence of " t ", the resulting marking contains " M ".

The definition of pre_S is as follows: for all predicate structures " (A,B) ",

$$pre_S(A,B) = \bigcup_{t \in T} (\mathrm{preP}_t(A),\ \mathrm{preN}_t(B))$$

where $\forall\ t \in T$, $\forall\ S \subseteq \mathcal{M}$ the auxiliary functions preP_t, preN_t are defined by:

$$\mathrm{preP}_t(S) = \begin{cases} \{\ \lambda^{in}(t)\ \} & \text{if } S = \varnothing \\ \bigcup_{M \in S} \{\ \mathrm{preT}(M,t)\ \} & \text{if } S \neq \varnothing \end{cases}$$

$$\mathrm{preN}_t(S) = \begin{cases} \varnothing & \text{if } S = \varnothing \\ \bigcup_{M \in S} \{\ \mathrm{preT}(M,t)\ \} & \text{if } S \neq \varnothing \end{cases}$$

To illustrate the use of pre_S let us consider the nets in Figure 5. For both of these, we compute the predicate structures which represent the set of markings which verify the property that after the firing of the transition " t' " the transition " t " can respectively be fired or not be fired. The predicate structures representing the markings for which " t " is firable and is not, are given respectively by " $(\{\lambda^{in}(t)\},\varnothing)$ " and " $(\varnothing,\{\lambda^{in}(t)\})$ ". So, in case (a) this gives " $(\{(7)\},\varnothing)$ ", " $(\varnothing,\{(7)\})$ " and, in case (b), it gives " $(\{(3)\},\varnothing)$ ", " $(\varnothing,\{(3)\})$ ".

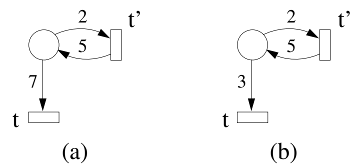

Figure 5: Petri Nets for the Computation of Predicate Structures Predecessors

Firability of " t ": The predicate structure fulfilling the property that its markings are such that the transition " t " can be fired after the firing of the transition " t' " is calculated by " $(\mathrm{preP}_{t'}(\{\lambda^{in}(t)\}),\ \mathrm{preN}_{t'}(\varnothing))$ ". The computation gives in case (a) " $(\{(4)\},\varnothing)$ " and " $(\{(2)\},\varnothing)$ " in case (b). In other words, the set of markings represented are such that, in the first case the place must have at least 4 tokens, and in the second case, the place must have at least 2 tokens. These conditions can easily be checked in Figure 5.

Non-firability of " t ": Seeing as, in this case, the property of interest is the non-firability of the transition " t " following the firing of the transition " t' ", the searched predicate structure is given by " $(\mathrm{preP}_{t'}(\varnothing),\ \mathrm{preN}_{t'}(\{\lambda^{in}(t)\}))$ ". In case (a) this leads to " $(\{(2)\},\{(4)\})$ " and in case (b) to " $(\{(2)\},\{(2)\})$ ". In the first case, this means that the place must contain at least 2 tokens and less than 4. In case (b), the place should have simultaneously at least 2 tokens and less than 2! This condition is clearly unsatisfied and the set of markings represented by " $(\{(2)\},\{(2)\})$ " is thus empty. This result was foreseeable since, in case (b), after the firing of the transition " t' ", the transition " t " was inevitably firable.

Management of the Predicate Structures

A set of rules has been formulated to simplify the components of a predicate structure and to eliminate both inconsistency and redundancy from a set of predicate structures.

Simplification: This process eliminates the useless markings from both of the components of a predicate structure. For instance, let us consider " (A,B) " such that $A=\{\ldots, M_a', M_a, \ldots\}$ with $M_a \subseteq M_a'$, with permits the marking " M_a " to simply be removed from the component " A ". A similar rule can be applied to markings in part " B ".

For instance, the predicate structure " $(\{\lambda^{in}(t),\ \lambda^{in}(t')\},\varnothing)$ " represents the markings from which the transitions " t " and " t' " may be fired. If the Petri net considered is that represented in Figure 6, this predicate structure can be simplified into " $(\{\lambda^{in}(t')\},\varnothing)$ ". Indeed, the firability of " t' " induces the firability of " t ".

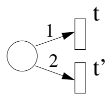

Figure 6: A Petri net for the Management of the Predicate Structure

Inconsistency: A predicate structure is inconsistent when the constraints specified by its components are in opposition, which creates a situation in which these opposing constraints cannot be simultaneously fulfilled. In this case, the set of markings represented by the predicate structure is empty. " $(\{\ldots, M_a, \ldots\}, \{\ldots, M_b, \ldots\})$ " with $M_b \subseteq M_a$ is such an example.

Let us consider now, the predicate structure " $(\ (\{\lambda^{in}(t')\})\ ,\ (\{\lambda^{in}(t)\})\ )$ ". It represents the markings from which the transition " t' " is firable while the transition " t " is not. If the Petri net considered is once again that one of Figure 6, this predicate structure is inconsistent. Indeed, the firability of " t' " induces the firability of " t " ($M_b = \lambda^{in}(t) \subseteq \lambda^{in}(t') = M_a$).

Redundancy: This feature is detected by inspecting pairs of predicate structures and by establishing that a certain relation between their components exists.

In Figure 6, in the set of predicate structures "{ ({λ^{in}(t), λ^{in}(t')},∅) , ({λ^{in}(t')},∅) } ", " ({λ^{in}(t), λ^{in}(t')},∅)" is redundant with respect to the set of markings represented.

The set of rules defined leads to the definition of an equivalency relation between predicate structures. If two predicate structures belong to the same class of equivalency, this implies that they represent the same set of markings. This equivalency relation is expanded to the set of predicate structures, namely to our symbolic representation.

Summary

The equivalency established between the process of the temporal proofs with states and that which utilizes our symbolic representation is symbolically depicted in Figure 7.

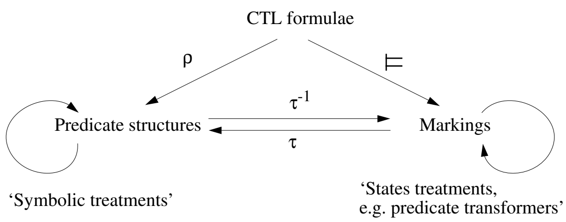

Figure 7: Equivalency of processes of proofs

The arrow labelled with " ρ " represents the computation of temporal proofs in the symbolic domain, i.e. the sets of predicate structures. According to the preceding arguments, the following result holds:

$$\forall\ f \in \mathrm{CTL}\ \forall\ m \in M,\quad m \models f \Leftrightarrow m \in \tau^{-1}(\rho(f))$$

5. A SIMPLE EXAMPLE

So as to obtain an impression of how a temporal proof is handled by means of our symbolic representation, we consider the simple example of the mutual exclusion problem (cf. Figure 8). The markings are displayed in the vector organization (NCS_1, NCS_2, R, CS_1, CS_2).

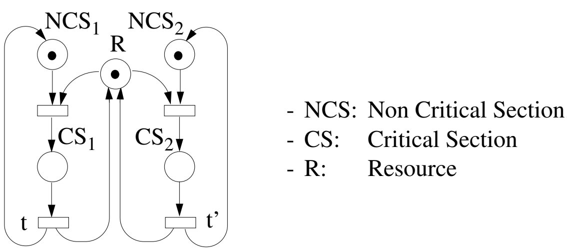

Figure 8: The Mutual Exclusion Case

Let us suppose that we want to find all the markings from which the net evolves so as to lead to a state in which the mutual exclusion is not respected. These markings fulfill the formula

$$\mathbf{EF}(\,t \wedge t'\,)$$

since in this case, both the places " CS_1 " and " CS_2 " must contain at least one token. The predicate structure representing the markings such that the transitions " t " and " t' " may be fired is the result of the intersection operation on the predicate structures of " ({λ^{in}(t)},∅) " and " ({λ^{in}(t')},∅) ". It gives the predicate structure " ({(0,0,0,1,1)},∅) ". In the symbolic domain, the calculation of " **EF**($t \wedge t'$) " is performed by the iterative application of pre_S applied first to " ({(0,0,0,1,1)},∅) ". The iteration stops when, according to the equivalency relation, no new predicate structure can be added. The result of the iteration is the following:

({(0,0,0,1,1)},∅) ∪ ({(0,1,1,1,0)},∅) ∪ ({(1,0,1,0,1)},∅) ∪ ({(1,1,2,0,0)},∅) ∪ ({(0,1,0,2,0)},∅) ∪ ({(1,0,0,0,2)},∅)

If the initial marking of the net, " m_0 ", is that of Figure 8, i.e. " (1,1,1,0,0) ", it is easy to check that its reachable markings verify the mutual exclusion property, since " m_0 " is not represented by any of the preceding predicate structures. Stated differently, " m_0 " does not belong to the set of markings represented in the preceding symbolic representation.

6. SANDS

SANDS [2] (Structured Algebraic Net Development System) is the environment developed for the specification language CO-OPN [1] (Concurrent Object-Oriented Petri Nets). CO-OPN permits the abstract description of both the data-structures and the concurrent aspects of a system. The concurrent aspects are described by means of Petri nets, while the data aspects are described with algebraic abstract data types. This association, called algebraic nets [9], is structured in CO-OPN with the notion of objects.

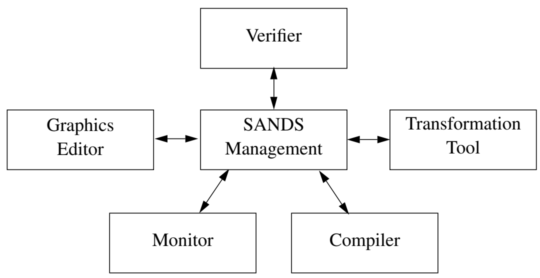

Figure 9: SANDS environment

The Components

The tools embedded in SANDS (cf. Figure 9) have been developed so as to cover all the steps in the development of CO-OPN specifications. It includes a graphics editor, which handles the objects and their organization, a compiler for checking and producing parallel code, in accordance with the semantics of the system, a monitor for testing some parts of the application, a transformation tool as well as its library for the parallelization of specifications, and, finally, the verifier of temporal properties.

The Temporal Tool

The current state of the temporal part of the tool includes the definition and manipulation of predicate structures by means of the calculation of the CTL's operators. This prototype version is semi-automatic, in that the user still exercises a large degree of control under the fixed point calculations.

The temporal tool developed thus far is a prototype which has been used on different modellings of moderate dimension (for instance the Alternating Bit Protocol) and it establishes that our method is correct. Up till now, modular modellings are not handled by the tool, it can be applied to usual Petri nets and has some extensions to algebraic nets.

7. CONCLUSION

As we have seen, the definition of the symbolic representation of a set of (possibly infinite) markings may be integrated into a model-checking process of CTL formulae. In spite of the similarity to the construction of the reachability tree of a net, the overall approach is different. Indeed, for instance, it is not dependant on the initial marking of the net.

The complexity of a proof depends on the number of transitions in the net. The computation of the predecessor, i.e. pre_S, of a symbolic representation composed of “ n ” predicate structures of a Petri net with “ |T| ” transitions, will produce a symbolic representation of “ n * |T| ” components in just one step. The experiments undertaken at this point show that the structuring power of the symbolic representation is real, in that most of these predecessors are removed by the process of elimination of redundancies and inconsistencies. It is worth mentioning that, even when the fixed point computation converges, its speed may not been foreseen.

Our method, at its present state, contends with properties which require that some places in the nets must be unbounded, i.e. must contain a large and unspecified number of tokens, so as to be fulfilled. A simple net which illustrates this point is shown in Figure 10.

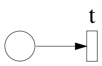

Figure 10: A Endless Computation

Indeed, for this net, the CTL formula “ **AG** (*t*) ”, stating that the transition “ t ” is always firable from every reached state of the computation, is only verified when the place is unbounded. Besides, at the n^{th} iteration, the symbolic representation is “ $\{(\{(n)\},\emptyset)\}$ ”. This limitation could be surmounted by using a technique similar to that used for building the reachability tree of an unbounded net, in which some places in the net can accumulate an arbitrary number of tokens. However, this would certainly be at the expense of a loss of some information.

Presently, we are experimenting with the possibility of incorporating some external constraints into nets, such as the boundaries of some places and the definition of relationships between places. Some experiments have shown that the size of the states space of the proof can be drastically reduced.

Our efforts are concerned also with the efficient implementation of rules and operations which act on the set of predicate structures. These include functions already defined and implemented in SANDS which, unfortunately, are presently inefficient.

Acknowledgements

The author is pleased to acknowledge the helpful discussions held with Didier Buchs.

8. REFERENCES

[1] D. Buchs, N. Guelfi. CO-OPN, “A Concurrent Object Oriented Petri Net Approach”. *12th Int. Conf. on Application and Theory of Petri Nets*, 1991, pp. 432-454.

[2] D. Buchs, J. Flumet, P. Racloz, “SANDS: Structured Algebraic Net Development System”, *14th Int. Conf. on Application and Theory of Petri Nets*, Tool presentation, 1993, pp. 25-29.

[3] E.M. Clarke, E.A. Emerson, A.P. Sistla “Automatic Verification of Finite State Concurrent Systems Using Temporal Logics Specifications: A Practical Approach”, *ACM Transaction. on Programming Languages and Systems*, vol.8, no.2, 1986, pp. 244-263.

[4] E.A. Emerson, “Temporal and Modal Logic”, *Handbook of Theoretical Computer Science*, Chapter 16, 1990, pp. 995-1072.

[5] J.P. Queille, J. Sifakis, “Iterative methods for the analysis of Petri nets”, *First European Workshop on Application and Theory of Petri Nets*, 1980, pp. 161-167.

[6] P. Racloz, D. Buchs, “Symbolic Proof of CTL Formulae over Petri Nets”, 8*th Int. Symp. on Computer and Information Sciences*, 1993, pp. 189-196.

[7] P. Racloz, “Symbolic proofs of temporal properties for various kinds of Petri nets”, Ph.D. thesis, University of Geneva, april 1994.

[8] W. Reisig, “Petri nets: an introduction”, *EATCS Monograph on Theoretical Computer Science*, Vol. 4, Springer Verlag, 1985.

[9] W. Reisig, “Petri nets and algebraic specifications”, *Theoretical Computer Science*, No. 80, 1991, pp. 1-34.

[10] J. Sifakis, “A Unified Approach for Studying the properties of transition systems”, *Theoretical Computer Science*, No. 18, 1982, pp. 257-259.