Reachable state space analysis of LOTOS specifications[*] [†]

A. Kerbrat
VERIMAG[‡]

We present a symbolic analysis technique for LOTOS programs with integer variables on which only linear expressions are allowed. The technique is applicable to models generated by the LOTOS compiler of the CÆSAR-ALDÉBARAN toolbox which are Petri nets extended with guarded commands. It allows to compute a predicate on variables characterizing the set of the reachable states or an upper approximation of it. Predicates are represented as systems of linear inequalities on program variables. We implemented a tool for performing the operations necessary for the analysis such as conjunction, disjunction, widening operation as well as comparison of predicates. The method is applied to two examples showing that non trivial relations between program variables can be discovered

1. Motivations

Most formal verification methods are based on *model checking*: starting from a specification written in any Formal Description Technique with well defined operational semantics, a model is build and properties are checked on this model. The majority of the model checking verification techniques are based on a more or less exhaustive exploration of the graph, either for deadlock searching, checking of *logical properties* or checking of *equivalence* between the model and a property. The performances and applicability of these methods depend greatly on the size of the model to be checked. Unfortunately, real life examples tend to produce huge models: this problem is known as the *state explosion problem*.

Various improvements have been introduced on the classical enumerative exploration of the fully generated model in order to cope with this problem. The first class of improvement try to optimize the exploration of the model, either by avoiding its complete generation (*"on the fly" techniques* [11, 24]) or its complete exploration (*Partial order techniques* [15, 24]), or by exploration of an *abstraction* of the model [16] . A second class of improvement concerns the *representation* of the model: the aim of *symbolic techniques* is to avoid the explosive enumerative representation of a model by the representation and manipulation of *sets* of states instead of individual states. One particular technique, Binary Decision Diagrams [5] (BDD) has already a story of success in hardware verification [7, 23] and has been applied successfully in other concurrent programming domains [3, 12]. However, the effectiveness of BDDs depends greatly on the structure

[*]This work has been partly supported ESPRIT-BRA action "REACT"
[†]Author address: VERIMAG, Miniparc ZIRST, 38330 Montbonnot, France.
e-mail: Alain.Kerbrat@imag.fr
[‡]Verimag is a joint laboratory of CNRS, Institut National Polytechnique de Grenoble, Université Joseph Fourier and Verilog SA associated with IMAG.

of the model to represent. In particular, one major cause of the state explosion is the presence of variables in the program such as counters, which cannot be handled directly by BDDs.

Another idea is to use symbolic techniques based on an *implicit representation* of *numerical* values. Such techniques are widely applied in the context of semantic analysis of imperative languages (e.g. [9, 17, 4]) and are now introduced for the analysis of real-time systems (e.g. [1, 21]).

The aim of this paper is to present an analysis technique based on a partly symbolic representation of the model. The idea is to keep the control structure of the program explicit and to represent its data part symbolically by systems of linear inequalities known as *polyhedra*. This approach offers several advantages, such as the possibility to deal with infinite models and the checking of properties expressed in terms of variables of the program instead of actions.

Together with this model, we present a *approximate forward analysis* method successfully applied in other contexts [9, 19]. Applied with polyhedra, this method allows the analysis of *linear relations* among variables of the program and the computation of an approximation of the *reachable states space*. We will show the various applications of these methods to the analysis of LOTOS specifications with integer variables and the results given by our implementation in the CÆSAR-ALDÉBARAN toolbox [13] context.

2. Definitions

We propose to work on a model which keeps a distinction between the *control part* and the *data part*, presented below as an *extended automaton*.

An extended automaton is a tuple $M = (C, V, T, q_0)$ where

- C is a finite set of *locations*
- V is a finite set $\{x_1, x_2, \dots, x_n\}$ of *variables* of $\mathbb{R}$. We will call a *valuation* the function v which associates a real value $v(x)$ to each variable $x \in V$. A *state* of the underlying model is defined as a couple (l, v) with $l \in C$ and v is a valuation. We will note $Q \subseteq C \times \mathbb{R}^n$ the (possibly infinite) set of all possible states.
- T is a finite set of transitions. Each transition is a tuple $(l_1, l_2, \gamma, \alpha)$ where :
  - $l_1, l_2 \in C$ are respectively the *source* location and the *target* location
  - γ is the *guard* of the transition. It is a conjunction of m *linear constraints*, each constraint being defined as a triple (a, b, r) where a is a vector of $\mathbb{Z}^n$, $b \in \mathbb{Z}$ and r is a comparison operator belonging to $\{<, \leq, =, \geq, >\}$. The guard defines a *convex* set of points of $\mathbb{R}^n$ and is called a *linear system.* Given a set Φ of valuations, the value of $\gamma(\Phi)$ is given by the intersection of the set of points defined by γ and the set Φ. If the set Φ is convex, then the intersection $\Phi \cap \gamma$ is convex too.
  - α is an action, defined as a *linear assignment* from valuations to valuations. It is given as a couple (A, B) such that $A \in \mathbb{Z}^{n \times n}$ is a square matrix of integer coefficients and $B \in \mathbb{Z}^n$ is a vector of integer coefficients. Given a

linear assignment $\alpha = (A, B)$ and a set Φ of valuations, we note $\alpha(\Phi)$ the transformation by α of Φ:

$\alpha(\Phi) = \{Av + B \mid v \in \Phi\}$

- $q_0 = (l_0, v_0)$ is the *initial* state.

We will note $q_1 \xrightarrow{\gamma,\alpha} q_2$ the transition between $q_1 = (l_1, v_1)$ and $q_2 = (l_2, v_2)$ such that $\exists t = (l_1, l_2, \gamma, \alpha) \in T, v_1 \in \gamma \wedge v_2 = \alpha(v_1)$.

A *run* of this automaton is a finite or infinite sequence $q_0 \xrightarrow{\gamma_0,\alpha_0} q_1 \xrightarrow{\gamma_1,\alpha_1} \ldots$. We will say that a state $q = (l, v)$ is *reachable* if and only if there exists a run $q_0 \xrightarrow{\gamma_0,\alpha_0} q_1 \xrightarrow{\gamma_1,\alpha_1} \ldots$ such that $\exists i \geq 0, q = q_i$.

We will note *Reach* the set of all reachable states.

More precisely, we are interested in the computation of the *set of reachable valuations* for any location $l \in \mathrm{L}$, that we will note ϕ_l:

$\phi_l = \{v \mid (l, v) \in \phi_l\}$

In the following, we will call ϕ_l the *context* of the location l.

2.1. From Lotos specification to extended automaton

Given the type of variables used in an extended automaton, we will first restrict ourselves to LOTOS specifications with integer or enumerated variables, and linear expressions on these variables. The translation of a LOTOS program into an extended automaton suitable for analysis is done in two steps, via the *interpreted Petri Net* provided by the CÆSAR [14] compiler. This interpreted Petri Net is a *safe* Petri Net decorated with guards and operations on variables. The Petri Net is then simulated without taking account of the variables, to provide the control part of the extended automaton. This simulation can in some cases generate a extended automaton with unreachable locations. However, this automaton is generally much smaller than the explicit model, due to the implicit representation of the numerical variables. The whole process is best resumed by the figure 1.

During the translation, all the correspondences between locations of the automaton and places of the Petri Net are kept. In particular, all transitions in the extended automaton refer to a transition in the Net. As the inverse relation is also kept, we can associate to a given transition in the Net the set of corresponding transitions in the extended automaton. These informations give the possibility to interpret *automatically* the properties deduced on the automaton as properties on the Net.

The current version of the CÆSAR compiler dœs not allow us to extend this backward interpretation up to the LOTOS program, but still the Petri Net generated matches closely the structure of the source program, so any information obtained on the Net is easy to understand on the LOTOS specification.

3. Forward analysis

The computation of a set of reachable states from the initial state is usually performed by the resolution of a *least fixpoint* equation $x = F(x)$, where x represents sets of states

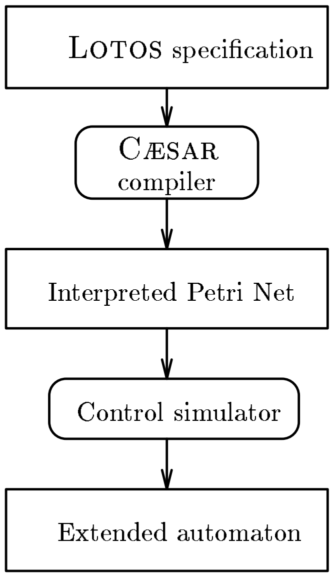


Figure 1. LOTOS to extended automaton

and F is defined as a transformation on sets of states. When dealing with programs using numerical variables, we face two problems:

1. How do we represent sets of states ? We need an *efficient* representation which can be *normalized* (for comparison of elements) and allows the definition of an *ordering relation* (for fixpoint computation).

2. As the set of all states can be infinite, how do we perform (possibly) infinite fixpoint computations ?

Both problems can find an answer in the context of *abstract interpretation.*

3.1. Abstract interpretation

Abstract Interpretation is a useful method proposed by [8] for computing *formally* and *automatically* an approximation of the dynamic behaviour of a program. It gives a way to establish a correspondence between the *concrete* domain and an *abstract* domain on which representation of elements becomes simpler and more efficient. An application of this framework proposed in [9] gives us an answer to our first question: they propose to represent a concrete set ϕ of valuations of variables by an *abstract* representation given by the *convex hull* Φ of the set ϕ. This convex hull can be defined as the set of solutions of a system of linear constraints, i.e. a *convex polyhedron.*

We first recall the definition of a convex polyhedron, then we present how the usual operators on sets such as inclusion, union, ... translate in terms of operators on polyhedra.

Convex polyhedron

A convex polyhedron P can be expressed as one of the following dual representations:

- a *system of linear inequalities* also called its *constraint system*: $P = \{\vec{x} \mid A\vec{x} \geq B\}$ where $\vec{x} = \{x_i \mid i \in [1 \ldots n] \wedge x_i \in \mathbb{R}\}$, $A \in \mathbb{Z}^{m \times n}$ and $B \in \mathbb{Z}^m$.

- a *system of generators*, which is a couple (V, R) where V and R are finite sets of vectors of $\mathbb{R}^n$, called respectively *vertices* and *rays*:

$$P = \{\sum_{v_i \in V} \lambda_i . v_i + \sum_{r_j \in R} \mu_j . r_j \mid \lambda_i \geq 0, \mu_j \geq 0, \sum_i \lambda_i = 1\}$$

We use an efficient algorithm initially proposed by [6] then improved and implemented by [25] to compute one representation from the other. This algorithm also allows to minimize each representation, by the detection of redundant constraints in the constraint system and conversely redundant generators.

Most of the operators needed for the computations of fixpoints are directly defined as operators on polyhedra. Given two convex polyhedra P and P', their *minimal* system of inequalities (A, B) and (A', B') and their *minimal* system of generators (V, R) and (V', R'), we define the following operators:

Test operators:

- **Emptyness:** P is empty if and only if $V = \emptyset$

- **Inclusion $\sqsubseteq$:** $P \sqsubseteq P' \Leftrightarrow \forall v \in V, A'v \geq B'$ and $\forall r \in R, A'r \geq 0$
(each vertex and ray of P satisfies the system of inequalities of P')

- **Equality $=$:** $P = P' \iff P \sqsubseteq P'$ and $P' \sqsubseteq P$

Set operators:

- **Intersection $\sqcap$:** the polyhedron $P \sqcap Q$ is defined by the system of inequalities given by the conjunction of (A, B) and (A', B')

- **Union:** the union of two convex polyhedra is not (in general) a convex polyhedron. We will approximate this operator by the *convex hull* operator

- **Convex hull $\sqcup$:** The convex hull of P and P' is the convex polyhedron defined by the system of generators $(V \cup V', R \cup R')$

- **Linear assignment:** given $\alpha = (A'', B'')$ a linear assignment, then $\alpha(P)$ is the convex polyhedron defined by the system of generators (V'', R'') where
$V'' = \{A''v + B'' \mid v \in V\}$ and $R'' = \{A''r \mid r \in R\}$

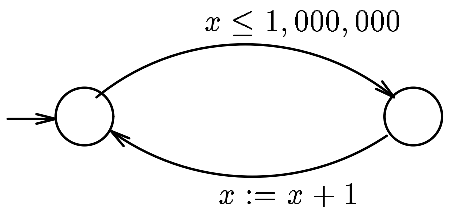


Figure 2.

3.2. Fixpoint computation acceleration

Choosing convex polyhedra as abstract domain dœs not provide an answer to our second problem: the lattice of convex polyhedra is of infinite depth, so we can still be faced with infinite fixpoint computations. It would be possible to choose an abstract domain either finite or of finite depth: in that case, all fixpoint computations converge in a finite number of steps. However, abstract domains of this kind are usually uninteresting, giving too inaccurate approximations. Furthermore, even if a fixpoint terminates, it may do so in too many steps to be of any practical interest: if we consider the example of figure 2, a classical iteration starting with $x := 0$ would take 1,000,000 steps before giving the obvious result $\{0 \leq x \leq 1,000,000\}$.

To complete the framework of abstract interpretation, [8] propose another kind of approximation, which enforces the convergence of an infinite fixpoint computation by extrapolation of its limit. This technique is based on the use of a *widening operator* and allows to compute an *upper approximation* of a least fixpoint, giving a safe approximation of the desired result.

If we consider the fixpoint equation $\Phi = F(\Phi)$, it can be written as the limit of the sequence

$$\begin{cases} \Phi^{(0)} = \bot \\ \Phi^{(n+1)} = F(\Phi^{(n)}) \end{cases} \tag{1}$$

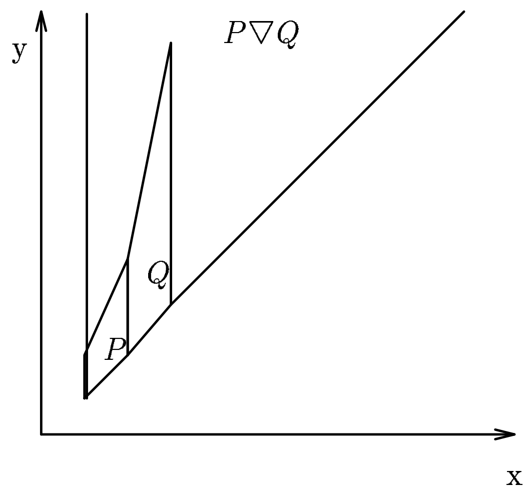
We define a widening operator as follows:

Definition 3.1 (Widening operator)
Given a complete lattice L, a widening operator $\nabla : L \times L \to L$ satisfies the following properties:

- $\forall x, y \in L, x \cup y \sqsubseteq x \nabla y$
- *for all increasing chains $x_0 \sqsubseteq x_1 \sqsubseteq \cdots$ the chain $y_0 = x_0, y_{i+1} = y_i \nabla x_{i+1}$ is not strictly increasing, i.e. the limit of the sequence can be computed in a finite number of steps.*

The sequence (1) then becomes:

$$\begin{cases} \phi^{(0)} = \bot \\ \phi^{(n+1)} = \phi^{(n)} \nabla F(\phi^{(n)}) \text{ if } F(\phi^{(n)}) \not\sqsubseteq \phi^{(n)} \\ \phi^{(n+1)} = \phi^{(n)} \text{ if } F(\phi^{(n)}) \sqsubseteq \phi^{(n)} \end{cases}$$



$$P = \left\{ (x,y) \mid \begin{pmatrix} 1 & \leq & x & \leq & 2 \\ x & \leq & y & \leq & 2x \end{pmatrix} \right\}$$

$$Q = \left\{ (x,y) \mid \begin{pmatrix} 2 & \leq & x & \leq & 3 \\ x & \leq & y & \leq & 3x \end{pmatrix} \right\}$$

$$P\nabla Q = \{ (x,y) \mid (\; 1 \; \leq \; x \; \leq \; y \;) \}$$

Figure 3. Widening operator

3.3. Widening operator for polyhedra

A widening operator for polyhedra has been first proposed in [18]. Given two polyhedra P and Q, the basic idea for computing $P\nabla Q$ is to keep only the constraints of P satisfied by Q:

Given $P = \{(x,y) \mid 1 \leq x \leq 2, x \leq y \leq 2x\}$ and $Q = \{(x,y) \mid 2 \leq x \leq 3, x \leq y \leq 3x\}$ then (see figure 3)

$P\nabla Q = \{(x,y) \mid 1 \leq x, x \leq y\}$

This widening operator can be improved when P is not full dimensional. In that case, its system of inequalities can be rewritten in order to maximize the number of constraints satisfied by Q:
Given $P = \{(x,y) \mid 1 \leq x \leq 2, y = 1\}$ and $Q = \{(x,y) \mid 1 \leq y \leq x \leq 2\}$, if we apply the basic idea of the widening operator, we obtain

$P\nabla Q = \{(x,y) \mid 1 \leq x \leq 2, 1 \leq y\}$

If we first rewrite P into $P = \{(x,y) \mid 1 \leq y \leq x \leq 2, y \leq 1\}$, the result of the widening becomes
$P\nabla Q = \{(x,y) \mid 1 \leq y \leq x \leq 2\}$ This operator satisfies the properties of a widening operator: the result of the widening contains its operands and it cannot be applied infinitely without convergence, as if $P \neq Q$, the system of inequalities of $P\nabla Q$ will be a strict subset of the system of inequalities of P.

3.4. Analysis of extended automaton with polyhedra

We can now gather the results of abstract interpretation with convex polyhedra to apply them on extended automata.

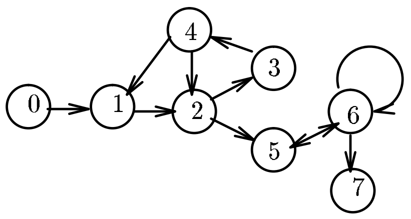


Figure 4. SCCs computation example

As the set of states Q of an extended automaton is partitioned with respect to the set of locations C, the computation of the reachable states is given by a *system* of fixpoint equations. Each equation is associated with a given location l and defines the context of this location as follows:

$$\phi_l \quad = \quad F_l(\{\phi_{l_1} \mid l_1 \in C\}) \tag{2}$$

In particular, the context of a location l depends only on the contexts of locations l_1 such that $\exists(l_1, l, \gamma, \alpha) \in T$. So the equations (2) become:

$$\phi_l \quad = \quad \bigcup_{(l_1,l,\gamma,\alpha)\in T} \{\alpha(\phi_{l_1} \cap \gamma)\}$$

We can write this system of fixpoint equations directly in terms of polyhedra and operators on polyhedra. In particular, the guard γ of each transition is itself a polyhedron (conjunction of linear inequalities) and the action α can be directly defined in terms of linear assignments on polyhedra:

$$\Phi_l = \bigsqcup_{(l_1,l,\gamma,\alpha)\in T} \{\alpha(\Phi_{l_1} \sqcap \gamma)\} \tag{3}$$

Due to the introduction of the convex hull operator, the forward analysis using polyhedra computes an upper approximation of the reachable states.

3.5. Choice of widening locations

The basic idea to ensure the convergence of the system of fixpoint equation (3) is to introduce a widening operation in every equation. This system then becomes:

$$\Phi_l \quad = \quad \Phi_l \nabla \bigsqcup_{(l_1,l,\gamma,\alpha)\in T} \{\alpha(\Phi_{l_1} \sqcap \gamma)\}$$

As the widening operator is an important source of loss of accuracy, it is important to apply it only when necessary. A good intuition is that every cycle of the extended automaton must contain a widening location, in order to ensure convergence. This condition is not always necessary, depending on the nature of the automaton: it is not necessary to set a widening location for a loop doing only an assignment of the form $X := K$.

We need to choose a *minimal* (in number of elements) set of widening locations, i.e. a minimal set of locations v such that every cycle of the automaton contains at least one state of v. This problem being NP-complete, we use an algorithm of graph decomposition into Strongly Connected Components (SCC), this decomposition being applied recursively until no further decomposition is possible. The algorithm gives also for each SCC its *entry point* for a depth-first exploration, we will choose this entry point as the widening location for the SCC. The result of this decomposition is an *admissible* set of widening locations, possibly not minimal. The decomposition into SCCs of the example of Figure 4 gives the following result:

$0\ (\underline{1}\ (\underline{2}\ 3\ 4))(\underline{5}\ (\underline{6})\ )\ (7)$

where the $(\dots)$ operator indicates a SCC. The *admissible* set of widening locations computed is $\{1\ 2\ 5\ 6\}$ and a *minimal* set of widening locations computed would be $\{2\ 6\}$.

3.6. Iteration strategy

Given this system of equations and an admissible set of widening locations, we still have to choose an iteration order for the computation and stabilization of each fixpoint equation. All orders are not equivalent: it is better to delay the computation of a context Φ_l until a maximum of the contexts Φ_{l_1} it depends on are themselves updated. Hopefully the decomposition in SCCs of the extended automaton will help us to choose a consistent iteration strategy, as it allows to sort the locations in a *topological* order.

The iteration strategy itself consists in a *recursive* iteration as presented in [4]. The whole stabilization algorithm (Figure 5) is based on a depth first exploration of the hierarchy of SCCs. During this exploration, we will stabilize first the innermost SCCs of the automaton.

The main call to this algorithm is *Stabilize*(C) (C is the set of all locations). The context of the initial location is defined by the initial value of the variables. The context of all other locations is empty. The successors of the initial location are marked not stable.

The application of this strategy on the example of figure 4 could be given by the expression

$0\ (\underline{1}\ (\underline{2}\ 3\ 4)^*)^*(\underline{5}\ (\underline{6})^*\ )^*\ (7)$

where the * operator's meaning is " iterate until stabilization".

4. Analysis example

We have developed a prototype implementation of the approximate analysis described in the previous sections. We present now the application of this tool on two different LOTOS programs.

4.1. Read-write protocol

The first example is a simple readers-writers protocol. A common resource is shared by a pool of potential writers and readers. Reading and writing are mutually exclusive, only one writer is allowed at a time, but all readers can read together if they want. All writing demands are counted, and a reader can start to read only when there is no more writers waiting to write, thus giving priority to the writers over the readers.

The process controlling the access to the resource is given by the following LOTOS process:

```
procedure Stabilize(Loc : set of locations)

while there exists a non stable location in Loc do
    let $l \in Loc$ be the least (wrt iteration order) non stable location
    $\Phi'_l = \bigsqcup_{(l_1,l,\gamma,\alpha)\in T} \{\alpha(\Phi_{l_1} \sqcap \gamma)\}$
    Mark $l$ as stable
    if $\Phi_l \sqsubset \Phi'_l$ then
        if $l$ is a widening location then
            $\Phi_l = \Phi_l \nabla \Phi'_l$
        else
            $\Phi_l = \Phi_l \sqcup \Phi'_l$
        fi
        foreach $(l, l_2, \gamma, \alpha) \in T$ do
            Mark $l_2$ as not stable
        od
    fi
    if $l$ is an entry point of a sub SCC then
        Stabilize(SCC(l))
    fi
od
```

Figure 5. Stabilization algorithm

```
process RESOURCE[S_READ,E_READ,WRITE_DEMAND,S_WRITE,E_WRITE](N_READERS, N_W_DEMANDS, WRITING:NAT) : noexit :=
(* N_READERS gives the number of readers currently reading *)
(* N_W_DEMANDS gives the number of writers asking to write *)
(* WRITING indicates that a writer is currently writing *)
[(WRITING eq 0)]->
(
  [N_W_DEMANDS eq 0] ->
     S_READ ; RESOURCE [S_READ,E_READ,DEMAND_WRITE,S_WRITE,E_WRITE](N_READERS+1, N_W_DEMANDS, WRITING)
  []
  [(N_READERS eq 0) and (N_W_DEMANDS gt 0)] ->
     S_WRITE ; RESOURCE [S_READ,E_READ,DEMAND_WRITE,S_WRITE,E_WRITE](N_READERS, N_W_DEMANDS-1,1)
)
[]
WRITE_DEMAND ; RESOURCE [S_READ,E_READ,DEMAND_WRITE,S_WRITE,E_WRITE](N_READERS, N_W_DEMANDS+1,0)
[]
E_WRITE ; RESOURCE [S_READ,E_READ,DEMAND_WRITE,S_WRITE,E_WRITE](N_READERS, N_W_DEMANDS,0)
[]
E_READ ; RESOURCE [S_READ,E_READ,DEMAND_WRITE,S_WRITE,E_WRITE](N_READERS-1, N_W_DEMANDS,WRITING)
endproc
```

The global parallel expression for 3 readers and 2 writers is

```
   (READER[S_READ,E_READ] |||  READER[S_READ,E_READ])
        |[S_READ,E_READ]|
   RESOURCE[S_READ,E_READ,S_WRITE,E_WRITE](0, 0)
        |[S_WRITE,E_WRITE]|
   (WRITER[S_WRITE,E_WRITE] )
```

The following results are given for M readers and N writers if we compute the union of the non empty contexts of the extended automaton after forward analysis, we obtain the following convex polyhedron, which represents an approximation of the smallest invariant of the program:

$$P = \left\{ \begin{array}{lcl} \text{N\_W\_DEMANDS} & \geq & 0 \\ \text{N\_READERS} & \geq & 0 \\ \text{WRITING} & \geq & 0 \\ \text{N\_W\_DEMANDS} + \text{WRITING} & \leq & N \\ \text{N\_READERS} + M * \text{WRITING} & \leq & M \end{array} \right\}$$

As WRITING can only take the value 0 and 1, this invariant shows that the number N_READERS can vary from 0 to M *if and only if* WRITING is equal to 0, thus ensuring the mutual exclusion. Note that the constraint $N\_READERS \leq M$ dœs not appear anywhere in the LOTOS program, this information has been deduced by the analysis process and is derived from the control part, not only from the data part of the program.

Another interesting result of this analysis is the detection of unreachable locations (empty context) in the extended automaton (see the size of the *explored* part of the automaton in table 1). This unreachability detection could be later used by the simulation phase of CÆSAR in order to remove some transitions of its Petri Net and reduce its state vector size .

4.2. Scheduler

This example is a task scheduler which handles several classes of tasks, each class is assigned a specific job whose completion takes a certain number of time units (TASK#LENGTH). All these tasks are constrained to run on a single processor. Each class is activated by a specific interrupt, these interrupts being generated at constant time intervals (INT#LENGTH). We describe further the version of the protocol running with two classes of tasks.

The LOTOS process generating the interrupts for two classes of tasks is the following:

```
process Interrupts[TICK,INT1,INT2](t1,t2:nat):noexit :=
[t1 ge Int1Length] -> INT1 ?X:nat;Interrupts[TICK,INT1,INT2](0,t2)
[]
[t2 ge Int2Length] -> INT2 ?X:nat;Interrupts[TICK,INT1,INT2](t1,0)
[]
TICK ;Interrupts[TICK,INT1,INT2](t1+1,t2+1)
endproc
```

When no tasks are running, the control is taken by an *Idle* process which reacts to the first interruption coming:

```
process Idle[TICK,INT1,INT2]:noexit :=
   INT1 ;Task1[TICK,INT1,INT2](0,1)
   []
   INT2 ;Task2[TICK,INT1,INT2](0,0,0,1)
   []
   TICK;Idle[TICK,INT1,INT2]
endproc
```

For each class of task, the number of incomplete tasks (either running or waiting) is accumulated in a variable (O#). Furthermore, we want to give priority to interrupts of the second class other the first class. The first class has just to keep count of its own interrupts and of the time it has already spent running a job. When interrupted by a second class interrupts, it saves its context by giving it to task2.

```
process Task1[TICK,INT1,INT2](L1,O1:nat): noexit :=
(
   TICK;(
     [L1 eq Task1Length]-> (* Current task completed *)
       ([O1 le 1]->Idle[TICK,INT1,INT2] (* No more tasks *)
        []
        [O1 gt 1]->Task1[TICK,INT1,INT2](0,O1-1) (* begin another task *)
       )
    []
    [L1 lt Task1Length]->Task1[TICK,INT1,INT2](L1+1,O1) (* Running the current task*)
   )
   []
   INT1 ;Task1[TICK,INT1,INT2](L1,O1+1) (* Keeping count of interrupts of same class*)
   []
   INT2 ;Task2[TICK,INT1,INT2](L1,0,O1,1) (* Prioritary interrupt *)
)
endproc
```

As the second class has maximum priority, it must keep a trace of all interrupts counters and length of incomplete tasks of lower priority.

```
process Task2[TICK,INT1,INT2](L1:nat,L2:nat,O1:nat,O2:nat): noexit :=
(
   TICK;(
     [L2 eq Task2Length]-> (* Current task completed *)
       ([O2 le 1]-> (* No more task of this class *)
        (
        [O1 ge 1]-> Task1[TICK,INT1,INT2](L1,O1) (* Continue or begin a task of class 1*)
         []
         [O1 eq 0]-> Idle[TICK,INT1,INT2] (* No task of any class *)
        )
        []
        [O2 ge 2]->Task2[TICK,INT1,INT2](L1,0,O1,O2-1) (* Begin another task of this class *)
       )
    []
    [L2 lt Task2Length]->Task2[TICK,INT1,INT2](L1,L2+1,O1,O2) (* Running current task *)
   )
   []
   INT1 ;Task2[TICK,INT1,INT2](L1,L2,O1+1,O2) (* Keeping count of lower priority interrupts *)
   []
   INT2 ;Task2[TICK,INT1,INT2](L1,L2,O1,O2+1) (* Keeping count of same interrupts *)
)
endproc
```

Given that priority is given to tasks of the second class over tasks of the first class, we want to verify that tasks of the second class never wait if the time interval between the generation of interrupts of class 2 is greater than the duration of tasks of class 2 (INT2LENGTH > TASK2LENGTH). It is equivalent to verify that the number of waiting *or* running tasks of class 2 never exceed 1 ($0 \leq$ O2 ≤ 1) under the previous condition.

An important remark is that exact forward analysis on this example will not terminate, as the variables of the interrupts generator are not bounded. It is therefore necessary to approximate the computations in order to terminate. The results obtained for this example need a little more interpretation than the previous example: if we compute only the union of all non empty contexts, we obtain $\mathbb{R}^n$ (n being the number of variables) as a result. To obtain more accurate results, it is necessary to investigate at the transitions level of the Petri Net. In particular, we want to know the possible values of the variable O2 in the process task2. As the only line of this process able to change the value of O2 is the line

```
INT2 ;Task2[TICK,INT1,INT2](L1,L2,O1,O2+1)
```

we have to check the corresponding transition of the Petri Net, which expands to a set of transitions in the extended automaton. This is done automatically by the tool, by the insertion of a question mark {O2?} at this transition in the Petri Net. The analysis gives the following results: the union of the contexts of *source* and *target* locations of these transitions is computed, giving the following polyhedra (for all examples such that INT2LENGTH > TASK2LENGTH):

$$\textbf{Source locations } P = \left\{ \begin{array}{rcl} 0 & \leq & \mathrm{T1} \\ 0 & \leq & \mathrm{T2} \leq \mathrm{TASK2LENGTH} \\ 0 & \leq & \mathrm{L1} \leq \mathrm{TASK1LENGTH} \\ \mathrm{T2} & = & \mathrm{L2} \\ \mathrm{O2} & = & 1 \\ 0 & \leq & \mathrm{TASK1LENGTH} * \mathrm{O1} - \mathrm{L1} \\ 0 & \leq & \mathrm{T1} + \mathrm{T2} + \mathrm{TASK2LENGTH} * \mathrm{T1} \end{array} \right\}$$

$$\textbf{Target locations } P' = \emptyset$$

and the answer to the question mark is $\{\mathrm{O2} = 1\}$.
This result shows that:

1. Whenever the control arrives at this transition, O2 = 1
2. The transition is never enabled (target contexts are all empty), so O2 cannot be incremented.

So O2 can never be greater than 1; tasks of the second class can never wait.

Note that the analysis performances in this case do not depend on the size of the variables: the example has been run with values for (TASK1LENGTH,INT1LENGTH) and (TASK2LENGTH, INT2LENGTH) going from {(2,4) (3,5)} or {(4,8) (10,20)} up to {(30,40) (40,50)}, with the same performances, thanks to the widening operator. The limitation here comes from the internal representation of numbers in our package, leading to some overflows during the minimization of the polyhedra representations.

Another important remark is that the scheduler example is difficult to verify with enumerative tools like CÆSAR: the 2 tasks version remains workable, as the graphs generated by CÆSAR are in the order of 100,000 states, but if we consider the 3 tasks version of the scheduler, even if we bound the variables $T1$, $T2$ and $T3$ to at most $max(\mathrm{INT1LENGTH}, \mathrm{INT2LENGTH}, \mathrm{INT3LENGTH})$ (otherwise, the resulting graph is infinite), and we choose small values ($\leq$ 10) for the durations of the different tasks and the time intervals between interrupts, then the resulting graph generated by CÆSAR is more than two millions states big (when we stopped the generation), well beyond the capacity of classical verification tools.

Program name	Petri Net		Extended automaton		Explored automaton		Variables number	Analysis time
	places	transitions	states	transitions	states	transitions		
3 readers 2 writers	20	19	320	608	166	250	3	1.3s
4 readers 3 writers	25	24	2156	4748	754	1257	3	15s
5 readers 3 writers	27	26	4364	10412	1454	2677	3	42s
Scheduler 2 tasks	17	27	27	51	24	48	8	4.4s
Scheduler 3 tasks	26	46	46	91	43	88	15	25.9s

Table 1
Analysis performances on a SPARC station 10

5. Conclusion

We have presented a symbolic representation of models well-suited for the approximate analysis of reachable states space. Although the analysis technique as well as the symbolic representation presented are not new [18] , their application to concurrent programming languages like LOTOS is as far as we know an innovation. Similar works for protocols can be found in [22] and [2], but based on a less powerful extrapolation operator than the widening operator used in our work.

One of the applications of this analysis technique is of course the verification of some invariant properties on the variables of the program, as shown on the examples presented. But we also use our implementation as an *abstract debugging* tool: it is often useful to know what are the bounds on variables or relations between them to understand some behaviours (and often unwanted behaviours) of the program. As our tool allows finely tuned analysis, at the level of places and transitions of the Petri Net, we can see as a *symbolic* debugger of LOTOS programs.

The analysis technique proposed here is based on a semi-symbolic model, an automaton being used to describe the control part and a symbolic representation being used for the data part. The control part of this automaton is derived from a simulation of a Petri Net. Yet, this automaton is only used for a partitioning of the analysis, in order to obtain more accurate results. An improvement would be to explore the automaton "on the fly" during the analysis, to avoid storing it completely and generating unreachable states and transitions, in the same spirit as those developed in [11, 24], and also partial-orders methods such as [15, 24].

A last direction of interest is the application of these techniques to timed automata, already under investigation in [19] for delay analysis and [20] for linear hybrid systems.

An adaptation of our tool to this kind of systems could complement the tool presented in [10] for the verification of an extension of LOTOS with timing requirements .

Acknowledgments

Thanks are due to Jean Claude Fernandez and Laurent Mounier for their helpful comments and careful readings of this paper.

REFERENCES

1. R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of timed transition systems (extended abstract). In *CONCUR'92, Stony Brook*. LNCS 630, Springer Verlag, august 1992.
2. Bernard Boigelot and Pierre Wolper. Symbolic verification with periodic sets. *6th Workshop on Computer-Aided Verification,Stanford*, 1994.
3. A. Bouali and R. de Simone. Symbolic bisimulation minimisation. In G. Bochmann, editor, *Proceedings of the fourth workshop on Computer-Aided Verification (Montreal,Canada)*, june 1992.
4. Francois Bourdoncle. *Semantique des languages imperatif d'ordre superieur et interpretation abstraite*. PhD thesis, Ecole Polytechnique, 1992.
5. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8), 1986.
6. N. V. Chernikova. Algorithm for discovering the set of all solutions of a linear programming problem. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 8(6):282–293, 1968.
7. O. Coudert, J.C. Madre, and C. Berthet. Verifying temporal properties of sequential machine without building their state diagram. In R.P. Kurshan and E.M. Clarke, editors, *Proceedings of the Workshop on Computer-Aided Verification (Rutgers, USA)*. DIMACS, June 1990.
8. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th POPL*, January 1977.
9. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th. Annual Symp. on Principles of Programming Languages*, pages 84–87, 1978.
10. C. Daws, A. Olivero, and S. Yovine. Verifying et-lotos programs with the tool kronos. In *7th international Conference on Formal Description Techniques for Distributed Systems and Communication Protocols*, 1994.
11. J.C. Fernandez, C. Jard, T. Jéron, and L. Mounier. "on the fly" verification of finite transition systems. *Formal Methods in System Design, Kluwer Academic Publishers*, 1992.
12. J.C. Fernandez, A. Kerbrat, and L. Mounier. Symbolic equivalence checking. In C. Courcoubetis, editor, *Proceedings of the 5th Workshop on Computer-Aided Verification (Heraklion, Greece)*, 1993.
13. Jean-Claude Fernandez, Hubert Garavel, Laurent Mounier, Anne Rasse, Carlos Rodríguez, and Joseph Sifakis. A toolbox for the verification of lotos programs. In

Lori A. Clarke, editor, *Proceedings of the 14th International Conference on Software Engineering ICSE'14 (Melbourne, Australia)*, pages 246–259, New-York, May 1992. ACM.

14. Hubert Garavel and Joseph Sifakis. Compilation and verification of lotos specifications. In L. Logrippo, R. L. Probert, and H. Ural, editors, *Proceedings of the 10th International Symposium on Protocol Specification, Testing and Verification (Ottawa, Canada)*, Amsterdam, June 1990. IFIP.
15. P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In Kluwer Academic Publishers, editor, *Formal Methods in System Design*, April 1993.
16. S. Graf and C. Loiseaux. A tool for symbolic program verification and abstraction. In *5th Conference on Computer-Aided Verification (Elounda, Greece)*. LNCS 697, Springer Verlag, jul 1993.
17. Philippe Granger. Static analysis of linear congruences. In *International Journal of Computer Mathematics*, 1989.
18. N. Halbwachs. *Détermination automatique de relations linéaires vérifiées par les variables d'un programme.* Thèse de 3e cycle, Université de Grenoble, March 1979.
19. N. Halbwachs. Delay analysis in synchronous programs. In *5th Conference on Computer-Aided Verification (Elounda, Greece)*. LNCS 697, Springer Verlag, jul 1993.
20. N. Halbwachs, Y.E. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In *First International Static Analysis Symposium*. Springer Verlag, September 1994.
21. T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. In *LICS'92*, June 1992.
22. A.S. Krishnakumar. Reachability and recurrence in finite state machines: Modular vector addition systems. In C. Courcoubetis, editor, *Proceedings of the 5th Workshop on Computer-Aided Verification (Heraklion, Greece)*, 1993.
23. H. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit state enumeration of finite state machines using BDDs. In *International Conference on Computer Aided Design (ICCAD)*, November 1990.
24. A. Valmari. On-the-fly verification with stubborn sets. In C. Courcoubetis, editor, *Proceedings of the fifth workshop on Computer-Aided Verificatio n (Elounda, Crete)*. DIMACS, june 1993.
25. H. Le Verge. *Un environnement de transformations de programmes pour la synthèse d'architectures régulières.* PhD thesis, Université de Rennes, 1992.