Symbolic Distributed Verification of a Class of Parametric Concurrent Systems

Thierry Massart [1], Jean-François Raskin [1], Laurent Van Begin [2]

Département d'Informatique
Université Libre de Bruxelles, Blvd Du Triomphe, 1050 Bruxelles, Belgium
{tmassart,jraskin,lvbegin}@ulb.ac.be

Key words: Parameterized Systems, Infinite States Systems, Distributed Verification, Symbolic Data Structures.

---

1 Motivations and backgrounds

Symbolic model-checking have been shown useful to partially solve the state explosion problem for the verification of finite state systems. In [3], we have adapted symbolic Model Checking to an interesting class of infinite states systems. We developed an efficient sequential algorithm to verify an important subclass of *safety* properties (called *upward closed* properties) on Transfer Nets (a monotonic extension of Petri Nets). This paper gives a brief overview of the problems that are related to a parallel and distributed version of the verification algorithm and reports on first results and the gain that we have obtained with a prototype using a cluster of PCs.

Following [4,3], parametric asynchronous concurrent systems can be naturally represented as Transfer Nets in which places, transitions and transfers are used to model *local states*, *internal actions*, communications via *rendez-vous* and *broadcast*, see [3] for more details. At this level of abstraction, processes can be viewed as indistinguishable *tokens*. A *marking* $\mathbf{m} = \langle m_1, \ldots, m_n \rangle$, a mapping from places to *non-negative* integers, can be viewed as an abstraction of a *global system state* in which we only keep track of the number of processes in every state. The number of processes in the system is determined by the (possibly parametric) *initial marking* $\mathbf{m}_0$. The Petri Net of Fig. 1(a)

[1] This author was partially supported by a FRFC grant 2.4530.02.
[2] This author was supported by a Walloon Region grant "First Europe".

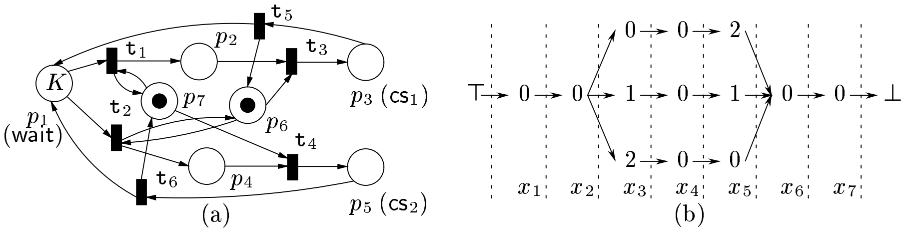

Fig. 1. (a) An example of Petri Net with parametric initial marking $\langle K, 0, 0, 0, 0, 1, 1\rangle$, $K \geq 1$. (b) An example of CST.

models a (erroneous) monitor for a parameterized system with two mutually exclusive critical sections (cs_1 and cs_2). Initially, all K processes are in p_1. To enter cs_1, a process tests for the presence of processes in cs_2 using p_7, and locks cs_1 using p_6 (transition t_1 followed by t_3); the procedure to enter in cs_2 is similar. On that parametric system, we would like to prove that for any K the mutual exclusion property asking that 'at most one token can be simultaneously present in the place p_4 and the place p_5'. To prove that property independently of the value of the parameter K, we can use the *backward reachability* approach which, for the verification of safety properties of Transfer Nets is based on the following notions taken from [1]. Given $\mathbf{m}, \mathbf{m}' \in \mathbb{N}^n$, we say $\mathbf{m} \preccurlyeq \mathbf{m}'$ ($\mathbf{m}$ *subsumes* $\mathbf{m}'$) iff $\mathbf{m}_i \leq \mathbf{m}'_i$ for $i \in [1, n]$. A set of markings U is upward-closed if for any $\mathbf{m} \in U$, for all $\mathbf{m}'$ with $\mathbf{m} \preceq \mathbf{m}'$ we have $\mathbf{m}' \in U$. Any upward-closed sets in $\mathbb{N}^n$ can be finitely represented by its *finite set* of *minimal points*. We note $cone(\mathbf{m})$ for the set $\{\mathbf{m}' | \mathbf{m} \preceq \mathbf{m}'\}$.

In our example of Fig. 1(a), the set U of violations of the mutual exclusion is the *upward-closed* set generated by the *minimal violations* $\mathbf{v}_1 = \langle 0, 0, 2, 0, 0, 0, 0\rangle$, $\mathbf{v}_2 = \langle 0, 0, 0, 0, 2, 0, 0\rangle$ and $\mathbf{v}_3 = \langle 0, 0, 1, 0, 1, 0, 0\rangle$ (*at least* 2 tokens in $p_3 + p_5$), and so $U = cone(\mathbf{v}_1) \cup cone(\mathbf{v}_2) \cup cone(\mathbf{v}_3)$. To prove that the protocol guarantees the *mutual exclusion* for *any* value of K, it is enough to show that no admissible initial marking is in the set of *predecessor* markings $\mathsf{Pre}^*(U)$ of U ($\mathsf{Pre}(U)$ returns the set of markings that permit to reach U by firing a transition) i.e.

$$\mathsf{Init} \cap \mathsf{Pre}^*(U) = \emptyset \qquad (1)$$

$\mathsf{Pre}^*(U)$ can be calculated by fixpoint computation which manipulates finite sets of minimal markings and is always reached within a finite number of iterations as $\preccurlyeq$ is a *well-quasi ordering* and applying Pre on an upward-closed set U returns an upward-closed set. To avoid the problems due to an explosion of the number of minimal markings, we have defined a data structure called CST ([2]) that allows us to compactly represent and efficiently manipulate infinite upward-closed sets of markings. Each path of a CST encodes a minimal marking of the set it represents and the crucial idea of the structure is the maximum sharing of prefixes and suffixes of paths. Fig. 1(b) shows the CST corresponding to the upward-closed set $x_3 + x_5 \geq 2$, where x_3 and x_5 represent the number of tokens in the places p_3 and p_5 respectively. The algorithms of [3] are called *symbolic* because they directly work on the underlying graph structure of CSTs instead of enumerating all the paths and therefore computes in one step several prefixes (suffixes) in common. CSTs have been proven useful for the verification of abstract models of interesting systems [3]. Nevertheless, we hope to be able to treat larger examples with a parallel implementation of the algorithm on a cluster of PCs. We report in the next section on the concepts proposed to obtain a parallel implementation.

2 A Distributed Verification Algorithm Using CSTs

Problem with the partitioning of the state space. The key idea in several distributed versions of verification algorithms for finite state space systems (see e.g. [5]) is the definition of a *partition function* which splits the state space of the system. Each piece of the state space is then assigned to a process which is in charge of its exploration. In the context of the backward (resp. forward) search, process P_i iterates the Pre (resp. Post) operator within its substate-space S_i, when it generates states that belong to a part of the state space owned by another process, it sends those states to the owner process which will continue the exploration from those states.

In our case, the situation is more complicated. In fact, the sets manipulated during the backward exploration of the state space of the class of infinite state systems that we consider are upward-closed sets of markings. Unfortunately, an upward-closed set of markings can *not* be partitioned using upward-closed sets of markings. In fact, the intersection of two non-empty upward-closed sets of markings is always non empty ! To overcome this difficulty, we use ideas from program specialization.

For any over-approximation $\mathcal{O}$ of the reachable markings of a system for which we have to decide (1), we can define a specialized Pre operator :

$$\mathsf{Pre}_{\mathcal{O}}(X) = \mathcal{O} \cap \mathsf{Pre}(X) \tag{2}$$

We call this operator a specialization of the original Pre operator as, if $X \subseteq \mathcal{O}$, it does not explore states that are not in $\mathcal{O}$ and are not reachable from the initial markings (as $\mathcal{O}$ over-approximates the set of reachable markings). It is easy to establish that the following assertion is true:

$$\mathsf{Init} \cap \mathsf{Pre}^*(U) = \emptyset \textbf{ iff } \mathsf{Init} \cap \mathsf{Pre}^*_{\mathcal{O}}(U) = \emptyset$$

The first benefit of this specialization is that it allows us to remove during the computation of the fixpoint all the states that $\mathcal{O}$ proves to be unreachable from the initial states and so potentially speeds up the process of checking (1). The second advantage is that if $\mathcal{O}$ is well chosen, it can allow us to partition the state space in a natural and efficient way.

Partition of the reachable state space using structural invariants. Place invariants of a Petri Net (or an extension of Petri Nets) can be used to obtain an over-approximation of its set of reachable states but also naturally partition its state space. We do not recall the well-known theory about the place invariants and refer the interested reader to [8] for more details. Here we illustrate our method for partitioning the state space using the example of Fig 1(a). For this Petri net with parametric initial marking $\langle K,0,0,0,0,1,1\rangle$, $K \geq 1$, we can automatically compute the following two invariants: (I_1) $x_3 + x_6 = 1$ that expresses 'in every forward reachable states, the marking of the place p_3 plus the marking of the place p_6 is always equal to 1', and (I_2) $x_5 + x_7 = 1$. If we take the set of markings satisfying I_1 and I_2 as over-approximation $\mathcal{O}$ of the forward reachable markings, for any $U \subseteq \mathcal{O}$ we have $\mathsf{Pre}(U) \subseteq \mathcal{O}$ i.e. $\mathsf{Pre}(U) = \mathsf{Pre}_{\mathcal{O}}(U)$.

We call $\mathbf{n}$ a partial marking if it is a function that assigns to a subset of the places a positive integer. If we consider the invariants I_1 and I_2, they give us information about places p_3, p_5, p_6, and p_7. They also define a finite set of partial markings for the variables involved in the invariants. In our cases, there are four partial markings that satisfy I_1 and I_2:
$\mathbf{n}_1$: $x_3 = 0 \wedge x_5 = 0 \wedge x_6 = 1 \wedge x_7 = 1$, $\mathbf{n}_2$: $x_3 = 0 \wedge x_5 = 1 \wedge x_6 = 1 \wedge x_7 = 0$
$\mathbf{n}_3$: $x_3 = 1 \wedge x_5 = 0 \wedge x_6 = 0 \wedge x_7 = 1$, $\mathbf{n}_4$: $x_3 = 1 \wedge x_5 = 1 \wedge x_6 = 0 \wedge x_7 = 0$

Using these partial markings, we can specialize the Petri Net of Fig. 1(a) into the extended automaton of Fig. 2. The locations of the automaton correspond to the partial markings defined by I_1 and I_2. The automaton uses integer variables to represent the number of tokens of places not involved in the invariants. In our example, the variables x_1, x_2 and x_4 are necessary to model the number of tokens in the places p_1, p_2 and p_4 respectively. The transitions of the Petri Net are specialized and give rise to the transitions of the automaton that only refer to these variables. The information about the other places has been "compiled" within the locations of the automaton. It is clear that this automaton contains all the information necessary to answer the safety verification problem of (1) as the Pre operator induced by the automaton is a specialization of the Pre operator of our original Petri Net in the sense of (2). The other advantage of this automaton is that it can be used to naturally partition the state space for its distributed parallel exploration. If we have N processes at our disposal to explore the state space, we partition the set of locations L of the automaton in N subsets $L_1, ..., L_N$ and assign each subset to a process. Each process P_i maintains for each location $l \in L_i$ a CST as explained below. We then apply the classical algorithm to distributively explore the state space.

3 Preliminary Results

We have implemented a distributed version of our algorithm presented in [3] that uses our specialization method to partition the state space. The processes

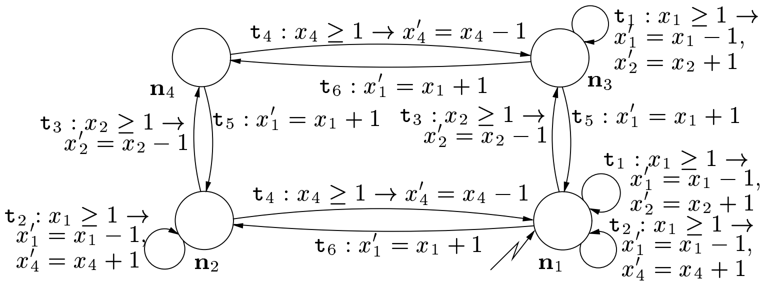

Fig. 2. The extended automaton obtained using the invariants I_1 and I_2.

have disjoint memory and communicate via *message passing*. After each n iterations of the Pre operator, each process sends the sets of computed states that are not in its partition to the corresponding owner process and reads any set of states sent for it. The set of states are pairs (l, S) where l is a location of the automaton obtained by specialization and S is a CST that encodes the upward-closed set of valuations in l for the integer variables not suppressed by the specialization process. These communications are asynchronous i.e. the sendings and the receptions of sets of states are non-blocking. We use an implementation of `MPI` [7] for the communications between processes and the termination of the computations is detected using a standard algorithm (see [6]). Currently our prototype assigns statically a partition (a set of locations of the specialized model) to each process. Using this implementation, we obtain preliminary results for two values of n (the number of local iterations between the communications).

In our previous works, the memory usage was not a bottleneck for the verification of our examples and we are mainly interested in the gain of execution time here. Our set of examples is composed by an abstract model of a JAVA program (see [3]), an abstract model of a protocol (PNCSA) and of a production system (multipoll) (see `www.ulb.ac.be/di/ssd/lvbegin/CST` for more details). We also verify the *specialized models* with the usual sequential version of the algorithm. Those preliminary experiments have been done on a cluster of 8 Bi Intel Pentium III 800Mhz processors with 256MB of memory and results are shown in Fig. 3. Depending on the examples, the gain in execution time is either quite modest or exceeds the number processors used (see the ratios). This is mainly explained by the fact that some examples are very *structured*, in particular the multipoll example. As a consequence the CSTs manipulated in the sequential version are very small and the parallel algorithm does not help in those cases. More experiments have to be done to better explain those first results. In our examples, the ratio with $n = 5$ (the processes communicate every 5 iterations) is always higher than with $n = 1$. The selection of an appropriate value for n is important. A small value increases the communication times and slows down the entire distributed algorithm. In other hand, the transmission of a set S can prevent a process from useless computations on markings subsumed by S, and therefore it is important to communicate frequently. Taking bigger value than 5 for n has not improved our results. Finally, we specialized in two different ways the multipoll example (using different structural invariants). This illustrates that the choice of *structural invariants* used to construct the specialized model can have an non-negligible impact in the execution times: when the sendings/receptions are done every 5 iterations, the second way of specialization is more than 2 times faster. Those preliminary experiences are encouraging

Case Stydy	P	T	NbI	NbP	Seq.	Dist.^{1}	ratio^1	Dist.^{5}	ratio^5
Java$_{bug}$	64	63	1	13	6697.5s	1101.9s	6.1	153.12s	**43.7**
PNCSA	31	36	1	9	680s	16.4s	**41.5**	3.19s	**213.2**
multipoll$_1$	18	21	2	9	3.2s	1.44s	2.2	0.42s	7.6
multipoll$_2$	18	21	2	9	3.2s	1.37s	2.3	0.18s	**17.8**

Fig. 3. Preliminary results : **P**=nb. places, **T**=nb. transitions; **NbI** = nb. invariants used for distribution; **NbP** = nb. processes; **Seq** = sequential ex. time; **Dist**^{i} = distributed ex. time when sendings/attempts to receive are done every i iterations ($n = i$); **ratio**^{i} = **Seq.** / **Dist**^{i}.

but more experiences have to be done to better understand why sometimes we obtain ratios that exceed the number of processors used and sometimes the ratios are very much below the number of processors. Furthermore, as the communications between the processes can be a bottleneck and a *full* use of all the processors is important to have good practical execution times, we are currently working on good techniques to partition the search space in order to minimize the communications. We also plan to include *load balancing techniques* to our implementation.

References

[1] P. A. Abdulla, K. Cerāns, B. Jonsson and Y.-K. Tsay. General Decidability Theorems for Infinite-State Systems. In *Proc. LICS'96*, 1996.

[2] G. Delzanno, and J. F. Raskin. Symbolic Representation of Upward-closed Sets. In *Proc. TACAS 2000*, LNCS 1785, 2000.

[3] G. Delzanno, J.-F. Raskin, and L. Van Begin. Toward the Automated Verification of Multithreaded Java Programs. In *Proc. TACAS 2002*, LNCS 2280, 2002.

[4] S. M. German, A. P. Sistla. Reasoning about Systems with Many Processes. *JACM* 39(3), 1992.

[5] T. Heyman, D. Geist, O. Grumberg, A. Schuster. Achieving Scalability in Parallel Reachability Analysis of Very Large Circuits. In *Proc. CAV 2000*, LNCS 1855, 2000.

[6] F. Mattern. Algorithms for Distributed Termination Detection. *Distributed Computing*,2,1987.

[7] P.S. Pacheco. Parallel Programming with MPI. *Morgan Kaufmann Publishers*, 1997.

[8] M. Silva, E. Teruel, and J. M. Colom. Linear Algebraic and Linear Programming Techniques for Analysis of Place/Transition Net Systems. Lectures on Petri Nets I: Basic Models, LNCS 1491, 1998.