UNIVERSITE LIBRE DE BRUXELLES
Faculté des sciences
Section Informatique

Attacking Symbolic State Explosion

Mémoire présenté en vue de l'obtention d'un diplôme d'études approfondies en Sciences (filière mathématique et informatique)

Promoteurs: Thierry MASSART
Jean-François RASKIN
Présenté par: Laurent VAN BEGIN

année académique 2000/2001

Ce mémoire est présenté en vue de l'obtention du diplôme d'études approfondies (DEA) en sciences décerné par l'*l'Université Libre de Bruxelles.* Il présente un travail de recherche ayant pour but l'implantation d'outils informatique mettant en œuvre des techniques efficaces pour l'analyse automatique de réseaux de Petri.

Le travail qui suit a été réalisé en collaboration avec Jean-François Raskin, chargé de cour à *l'Université Libre de Bruxelles*, et Giorgio Delzanno, professeur assistant à *l'Université de Gènes* (Italie).

Ce travail sera présenté lors de la 13^{ieme} conférence annuelle *Computer Aided Verification (CAV)*, tenue à Paris (France) du 18 au 23 Juillet 2001.

Durant de ce travail,

- Laurent Van Begin fut financé par une bourse de la *Région Wallonne* "First Europe"

- Giorgio Delzanno fut financé par une bourse de *l'Université Libre de Bruxelles* "Crédit d'ouverture internationale" lors de sa visite à cette même université

- Jean-François Raskin fut partiellement financé par une bourse du *Fond National de la Recherche Scientifique (FNRS)* "Crédit aux chercheurs"

Remerciements

Je tiens tout d'abord à remercier les professeurs Thierry Massart et Jean François Raskin de m'avoir donné la possibilité de faire de la recherche scientifique au sein du *département d'informatique* de la *faculté des sciences* de *l'Université Libre de Bruxelles.*
Je remercie également les autres membres du *service des systèmes distribués* pour leur collaboration éclairée.
Je remercie plus particulièrement Anthony Piron qui a le triste privilège de partager mon bureau.
Finalement, je remercie Marie Cooremans de me supporter jour après jour.

Abstract

We propose a new symbolic model checking algorithm for parameterized concurrent systems modeled as (Lossy) Petri Nets, and (Lossy) Vector Addition Systems, based on the following ingredients: a *rich assertional language* based on the *graph-based* symbolic representation of upward-closed sets introduced in [DR00], the combination of the *backward reachability algorithm* of [ACJT96] lifted to the symbolic setting with a *new heuristic rule* based on *structural properties* of Petri Nets. We evaluate the method on several Petri Nets and parameterized systems taken from the literature [ABC$^+$95, EM00, Fin93, MC99], and we compare the results with other finite and infinite-state verification tools.

Contents

1 Introduction

The theory of well-structured systems [ACJT96, FS01] gives us decision procedures to verify safety properties of *parameterized* systems modeled as Petri Nets [ACJT96, FS01], Lossy Vector Addition Systems [BM99], and Broadcast Protocols [EFM99]. The decision procedures are based on *backward reachability* algorithms like the one proposed in [ACJT96], whose termination (for Petri Nets and their extensions) is guaranteed by Dickson's lemma. It is important to recall that forward approaches like Karp-Miller's coverability tree are not robust when applied to extensions of Petri Nets like Broadcast protocols [EFM99].

Differently from the finite-state case, in parameterized verification *symbolic* representations are ineluctable in order to make the approach *effective*: we *need* to finitely represent infinite collections of states. In the backward approach of [ACJT96, FS01] we need to represent infinite, *upward-closed* sets of *markings*, when we restrict our attention to Petri Nets. Two examples of symbolic representations for *upward-closed sets* of marking are collections of minimal points [ACJT96], and *linear arithmetic constraints* [DEP99]. The complexity of the algorithm of [ACJT96] is non-elementary. For this reason, naive implementations of the backward approach suffer from the *symbolic state explosion* problem: the number of minimal points or the size of the constraints become unmanageable after few iterations. Symbolic state explosion is the counterpart of the state explosion problem we known from finite-state verification.

In our previous work [DR00] we proposed a new *rich assertional language*, in the terminology of [KMM+97], for representing compactly upward-closed sets of markings. Our data structure, we will call here *Covering Sharing Trees* (CSTs), are *directed graphs* in which we store the minimal points of an upward-closed set as a collection of tuples, and for which we allow the maximal sharing of prefixes and suffixes. To obtain efficient operations, it is crucial to avoid enumerating the paths of a CST. Working on the graph structure of CSTs, we defined all operations needed for lifting the backward reachability algorithm of [ACJT96] to the symbolic level. In the preliminary results given in [DR00], we managed to prove properties of Petri Nets (of small size) that could not be managed from other infinite-state model checkers (working backwards) like HyTech [HHW97].

Following our line of research, the *conceptual contribution* of this paper is a *new* heuristic rule for attacking symbolic state explosion based on the combination of CSTs and well known techniques for the static analysis of Petri Nets. More precisely, the heuristic rule is based on *structural properties* [STC98] of Petri Nets, i.e., on a *fully automatic* static analysis, whose results can be used during the backward reachability algorithm to significantly cut the search space. As the other techniques presented in [DR00], our structural heuristic works in polynomial time on the graph structure of CSTs. When combined with our CSTs-symbolic representation, the heuristic rule allow us to scale up the dimension of the case-studies of one order of magnitude.

As *practical contribution*, we describe a set of benchmarks we obtained with an optimized implementation of the CST-library, integrated with the above mentioned structural heuristic. We have applied the resulting model checking algorithm to a large set of examples of parameterized verification problems that can be solved using decision procedures for coverability of Petri Nets (e.g. mutual exclusion for the parametric models like the Mesh and Multipoll examples of [ABC+95, MC99], and semi-liveness for the PNCSA protocol of [Fin93]). We have also applied our method to verify safety properties of *finite-state* systems (e.g. some of the above mentioned examples for fixed values of the parameter). For these examples, we have compared our results with the results obtained with the specialized tool GreatSPN [CFGR95] for computing the reachability set of Petri Nets. As foreseen by Bultan in [Bul00], in most of the cases proving a parameterized property turns out to be more efficient than proving its finite-state instances.

Before entering in more details, in Section 2 we will briefly recall the main ideas behind the connection between parameterized systems, Petri Nets, backward reachability, and in Section 3 the basics of CSTs. The new heuristic rule is presented in Section 4. The new symbolic algorithm is presented in Section 5; its practical evaluation is presented in Section 6. We finish the paper discussing related works and drawing some conclusions.

2 Petri Nets and Verification of Safety Properties

Following [GS92], asynchronous concurrent systems (possibly with internal states modeled via Boolean variables [BCR01]) can be naturally represented as Petri Nets in which places and transitions are used to model *local states*, *internal actions* and communication via *rendez-vous*. At this level of abstraction, processes can be viewed as undistinguishable black *tokens*. A *marking* $\vec{m} = \langle m_1, \ldots, m_n \rangle$, 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 *initial marking* $\vec{m_0}$. The *backward reachability* approach for verification of safety properties of Petri Nets is based on the following notions, taken from [ACJT96, FS01]. Given $\vec{m} = \langle m_1, \ldots, m_n \rangle$ and $\vec{m'} = \langle m'_1, \ldots, m'_n \rangle$, we say that $\vec{m} \preccurlyeq \vec{m'}$ ($\vec{m'}$ is *subsumed* by $\vec{m}$) if and only if $m_i \leq m'_i$ for $i : 1, \ldots, n$. A set of markings U is upward-closed if for any $\vec{m} \in U$ and any $\vec{m'}$ such that $\vec{m} \preccurlyeq \vec{m'}$, we have that $\vec{m'} \in U$. Any upward-closed sets in $\mathbb{N}^m$ can be finitely represented by its *finite set* of *minimal points*, we will call $gen(U)$.

The relation $\preccurlyeq$ is a *well-quasi ordering*. This property ensures the termination of backward reachability, whenever the starting point of the exploration is an upward-closed set of markings.

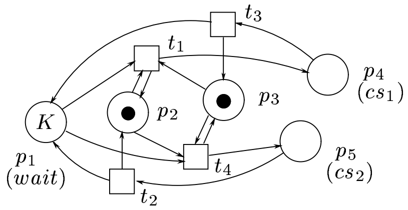

Figure 1: An example of Petri Net with parametric initial marking $\langle K, 1, 1, 0, 0 \rangle$, $K \geq 1$.

As an example, consider the Petri Net of Fig. 1, a 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_2, and locks cs_1 using p_3 (transition t_1), and vice versa. Processes leave the critical section using transitions t_3 and t_4. Note that the

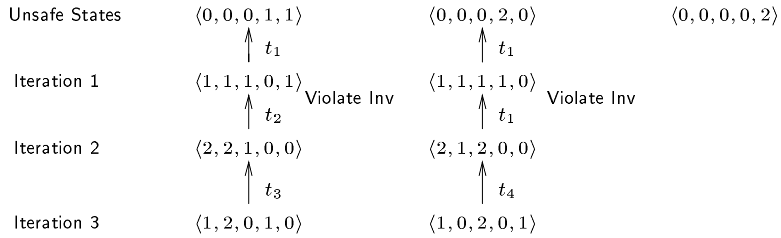

Figure 2: Backward reachability graph.

set U of violations to mutual exclusion is the *upward-closed* set generated by the *minimal violations* $\langle 0,0,0,2,0\rangle$, $\langle 0,0,0,0,2\rangle$, and $\langle 0,0,0,1,1\rangle$ (*at least* 2 tokens in $p_4{+}p_5$). To prove that the protocol guarantees *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 (Pre is the operator that returns the set of markings that reach some marking in U by *firing* a transition). To compute $\mathsf{Pre}^*(U)$, we iterate the application of the predecessor operator Pre until we reach a fixpoint. During the computation, every newly generated marking is stored only if it is *not subsumed* by an already visited one. The backward reachability graph of our example is given in Fig. 2 (ignore the annotations for the moment). In Fig. 2 we have omitted all redundant markings (about 30). As mentioned in the introduction, the symbolic backward approach based on the enumeration of minimal points of sets of markings suffers from the symbolic state explosion problem. More sophisticated data structures are necessary to make the approach feasible in practice.

3 The Assertional Language: Covering Sharing Trees

In [DR00], we studied the mathematical foundations of Covering Sharing Trees (CSTs), a new data structure to symbolically manipulate upward-closed sets. CSTs are based on the Sharing Trees of [ZL94]. A k-sharing tree $\mathbf{S}$ is a rooted acyclic graph with nodes partitioned in *k-layers* (apart from the special *root* and *end* nodes) $N = \{root\} \cup N_1 \cup \ldots \cup N_k \cup \{end\}$, successor relation $succ : N \rightsquigarrow 2^N$, and labeling function $val : N \rightsquigarrow \mathbb{Z} \cup \{\top, \bot\}$, such that: (1) all nodes of layer i have successors in the layer $i+1$; (2) a node cannot have two successors with the same label; (3) two nodes with the same label in the same layer do not have the same set of successors. The *flat denotation* of a sharing tree is defined as follows

$$elem(\mathbf{S}) = \{\langle val(n_1), \ldots, val(n_k) \rangle \mid \langle \top, n_1, \ldots, n_k, \bot \rangle \text{ is a path of } \mathbf{S}\}.$$

Conditions (2) and (3) ensure the maximal sharing of prefixes and suffixes among the tuples of the flat denotation of a sharing tree. The *size* of a sharing tree is the number of *nodes* and *edges*. The number of tuples in $elem(\mathbf{S})$ can be exponentially larger than the size of $\mathbf{S}$. As shown in [ZL94], given a set of tuples $\mathcal{A}$ of size k, there exists a unique (modulo isomorphisms of graphs) sharing tree such that $elem(\mathbf{S}_\mathcal{A}) = \mathcal{A}$. A CST is a sharing tree obtained by lifting the denotation of a sharing tree from the flat one of [ZL94] to the following *rich* one

$$cones(\mathbf{S}) = \{\vec{m} \mid \vec{n} \preccurlyeq \vec{m},\ \vec{n} \in elem(\mathbf{S})\}.$$

Given an upward closed set of markings U, we define the CST $\mathbf{S}_U$ as the k-sharing tree such that $elem(\mathbf{S}_U) = gen(U)$. Thus, $\mathbf{S}_U$ can be used to *compactly* represent $gen(U)$, and to *finitely* represent U. In the best case the size of $\mathbf{S}_U$ is *logarithmic* in the size of $gen(U)$. A CST $\mathbf{S}_U$ can also be viewed as a compact representation of the formula: $\vee_{\vec{m} \in elem(\mathbf{S}_U)} (x_1 \geq m_1 \wedge \ldots \wedge x_n \geq m_n)$. As an example, the CST $\mathbf{S}$ that symbolically represents the set of violations of our example is given in Fig. 3. Let us note that any $\mathbf{S}'$, such that $gen(U) \subseteq elem(\mathbf{S}')$ and such that all additional elements are *redundant* (i.e., are subsumed by elements in $gen(U)$) can still be used to represent U. We will call such a CST *redundant*. In the following we will show that it is often more efficient to work with redundant CSTs. In [DR00], we have

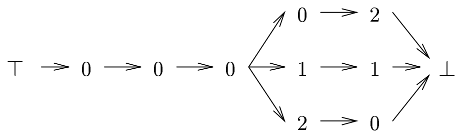

Figure 3: An example of CST.

defined the operations needed to implement a CST-based backward reachability procedure. The operations work on the graph structure of CSTs. In the following we will use $Union_{CST}(\mathbf{S}, \mathbf{T})$ to indicate the CST whose denotation is $cones(\mathbf{S}) \cup cones(\mathbf{T})$, and $\mathsf{Pre}_{CST}(\mathbf{S}, t)$ to indicate the CST whose denotation is $cones(\mathsf{Pre}(cones(\mathbf{S}), t))$ for some transition t. Checking subsumption between CSTs, namely whether $cones(\mathbf{S}) \subseteq cones(\mathbf{T})$ holds, the complexity of this test is CO-NP hard (event if the two CSTs are not redundant). In [DR00], we have defined a set of *polynomial time* sufficient conditions (with different precision) to check subsumption for CSTs, based on *simulation relations* between *nodes* of the corresponding sharing trees. Formally, a node n in the i-th layer of $\mathbf{S}$ is forward-simulated by node m in the i-th layer of $\mathbf{T}$ if and only if $val(n) \geq val(m)$ and for every successor node n' of n there exists a successor m' of m that forward-simulates n'. If the the root node of $\mathbf{S}$ is forward simulated by the root node of $\mathbf{T}$ than $\mathbf{S}$ is subsumed $\mathbf{T}$. Similar definitions and properties can be given for backward and mixed forward-backward simulations. The operations Pre_{CST} and $Union_{CST}$ do not guarantee to generate CSTs that contain *only* the minimal points. However, removing all redundancies is CO-NP hard. As shown in [DR00], simulation relations helped us again to obtain polynomial algorithms to partially eliminate redundancies. (As a technical remark, we point out that these techniques allow us to remove tuples of a given CST that are subsumed either by tuples of *another* CST or by tuples of the *same* CST.) Unfortunately, CST and simulation-based heuristics are not enough to mitigate symbolic state explosion. New heuristics for pruning backward search seem necessary in order to handle large examples.

4 Structural Heuristic

In the backward reachability approach, every place of a Petri Net is initially considered as *unbounded* (in fact, unsafe states are expressed via constraints like $x_1 \geq c_1$, etc.). In many practical cases however, some places are bounded for *any value* of the parameters in the initial configuration. The *Structural Theory* of Petri Nets [STC98] can help us to distinguish between bounded and unbounded places. Let N be a Petri Net with n places, m transitions, and *token flow matrix* $\mathbf{C}$ ($\mathbf{C}$ describes how tokens are moved in the net by the transitions; rows corresponds to places, and columns to transitions). Furthermore, let $\cdot$ denote the vector product $\vec{a}^T \cdot \vec{b} = a_1 b_1 + \ldots a_n b_n$, where $\vec{a}^T$ indicates the transpose of vector $\vec{a}$. *Place invariants* [STC98] are one of the possible informations we can compute via a static analysis of N. A place invariant (also called P-semiflow) is a vector $\vec{p} = \langle p_1, \ldots, p_n \rangle$ (non-negative) solution of the equation

$$\vec{x}^T \ \cdot \ \mathbf{C} \ = \ 0, \ \vec{x} \geq \vec{0},$$

where $\vec{x}$ is a vector of variables of dimension n. Given an initial marking $\vec{m_0}$, and a place invariant $\vec{p}$, the set $\mathcal{O}(\vec{m_0}, \vec{p}) = \{\vec{m} \mid \vec{p}^T \cdot \vec{m} = \vec{p}^T \cdot \vec{m_0}\}$ over-approximates the reachability set of the Petri Net. This property follows from the definition of place invariant, and from the state equation $\vec{m} = \vec{m_0} + \mathbf{C} \cdot \vec{\sigma}$ that characterizes a generic marking $\vec{m}$ reachable from $\vec{m_0}$ via the sequence of transitions represented by the firing vector σ (see [STC98]). As a consequence, the equation

$$\vec{p}^T \cdot \vec{x} = \vec{p}^T \cdot \vec{m_0}$$

for some place invariant $\vec{p}$ gives us a *structural invariant* we can use to analyze the net. Let us consider our running example. The three following equations are invariants of the net in Fig. 1 with the parametric initial marking $\langle K, 1, 1, 0, 0 \rangle$: (i) $x_2 + x_5 = 1$, (ii) $x_3 + x_4 = 1$, (iii) $x_1 + x_4 + x_5 = K$. Unfortunately, the invariants are not sufficient to prove our mutual exclusion property $x_4 + x_5 \leq 1$. Still the invariants contain information that we can exploit during the backward search. A possible way to use the structural analysis would be to make what is usually called *program specialization*, i.e., we can replace the subnet involving places linked by structural invariants (e.g. p_2, p_5 for $x_2 + x_5 = 1$) with a control part (a finite-state automata). This way however, the net resulting from the specialization may become of unmanageable size. As an alternative, we propose to use the structural invariants directly as heuristics for efficient backward reachability.

4.1 Pruning the Backward Search Space

Let U be an upward-closed set of markings denoting *unsafe states*, and let $U' = U \cap \mathcal{O}(\vec{m_0}, \vec{p})$ for some place invariant $\vec{p}$. We first note that if $U' = \emptyset$, then we can immediately infer that the net is *safe*. However, as in our example, invariants might not be sufficient to directly verify the property. We will use them to prune the backward search as follows. Let us consider again our running example and the backward reachability graph of fig. 2. After the first iteration, two generators $\langle 1,1,1,0,1\rangle$ and $\langle 1,1,1,1,0\rangle$ that are not subsumed by previous elements are computed. The first generator defines a set of markings that has no intersection with the set of markings defined by the invariant $x_2+x_5 = 1$, while the second generator defines a set of markings that have no intersection with the set of markings defined by the invariant $x_3 + x_4 = 1$. As a consequence, we deduce that no markings defined by those two generators can be reached from an instance of the parametrized initial marking (recall that the markings satisfying the invariant over-approximate the set of reachable markings.) As a consequence, we can stop the backward search after the first iteration instead of having to consider 3 iterations as in the naive search. Let us now examine how we can incorporate this idea in our CST-based backward search. Since U' is not upward-closed, it cannot be used as the starting point of our symbolic backward search. The following theorem however gives us indications on how to proceed.

Theorem 1 *Given a Petri Net N with initial marking $\vec{m_0}$, a place invariant $\vec{p}$, and an upward-closed set of markings U represented by a CST $\mathbf{S}$, suppose $cone(\vec{m}) \cap \mathcal{O}(\vec{m_0}, \vec{p}) = \emptyset$ for some $\vec{m} \in elem(\mathbf{S})$. Furthermore, let $\mathbf{S}'$ be the CST such that $elem(\mathbf{S}') = elem(\mathbf{S}) \setminus \{\vec{m}\}$, and $\vec{m'_0}$ be any instance of $\vec{m_0}$. Then,*

$$\vec{m'_0} \in \mathsf{Pre}^*(cones(\mathbf{S})) \text{ iff } \vec{m'_0} \in \mathsf{Pre}^*(cones(\mathbf{S}')).$$

The theorem shows that during the computation of $\mathsf{Pre}^*(U)$ we can prune the search space by *safely* removing *all* elements $\vec{m} \in elem(\mathbf{S}_U)$ (redundant or not) such that $cone(\vec{m})$ has empty intersection with the set of markings defined by a structural invariant. We call such elements *useless*. To prune the space efficiently, we must avoid the explicit enumeration of all elements

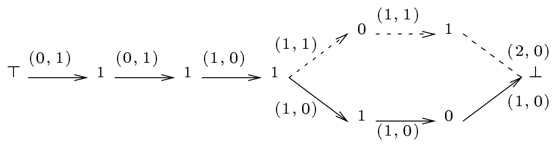

Figure 4: The CST $\mathbf{S}$ denoting $\mathsf{Pre}_{CST}(\mathbf{S})$, where $\mathbf{S}$ is given in Fig 3.

stored in a CST. In fact, the number of those elements is potentially exponential in the size of the CST. Instead of trying to remove all the useless elements for a give invariant $\vec{p}$, we use an *heuristic* rule that works directly on the graph structure of the CST and does not enumerate its paths. To describe the heuristic rule, we need the following definitions. Let $\mathbf{S}$ be a CST , and let $e = (\mathbf{v}, \mathbf{w})$ be an edge of $\mathbf{S}$ connecting nodes of two adjacent layers. We define $elem_e(\mathbf{S})$ as the set of tuples from $elem(\mathbf{S})$ denoted by paths of $\mathbf{S}$ passing through e. Formally, $\vec{m} = \langle val(v_1), \ldots, val(v_n)\rangle \in elem_e(\mathbf{S})$ iff there exists a path $\langle \top, v_1, \ldots, \mathbf{v}, \mathbf{w}, \ldots, v_n, \bot\rangle$ in $\mathbf{S}$ such that $e = (\mathbf{v}, \mathbf{w})$. Consider now a *structural invariant*, say $\mathcal{I}$, having the form $\vec{p}^T \cdot \vec{x} = \vec{p}^T \cdot \vec{m}_0$, where $\vec{p}$ is a place invariant (hence, $\vec{p} \succcurlyeq \vec{0}$) and, such that $\vec{p}^T \cdot \vec{m}_0$ is an integer, i.e., $\vec{p}^T \cdot \vec{m}_0$ does not contain occurrences of the parameters (e.g. we keep $x_2 + x_5 = 1$ and $x_3 + x_4 = 1$, and discharge $x_1 + x_4 + x_5 = K$). Our heuristic rule works by removing an edge e of $\mathbf{S}$, whenever we can prove that the elements in $elem_e(\mathbf{S})$ denote cones that do not intersect with the structural invariant $\mathcal{I}$. To check this condition on the edge e connecting a node of layer $\mathbf{i}$ and a node of layer $\mathbf{i+1}$, we first compute the two values $min_{\prec}(e)$ and $min_{\succ}(e)$ defined as follows: $min_{\prec}(e)$ is the minimal value of *prefixes* $\langle m_1, \ldots, m_{\mathbf{i}}\rangle$ of tuples in $elem_e(\mathbf{S})$ evaluated on the function $\vec{x} \rightsquigarrow \vec{p}^T \cdot \vec{x}$; symmetrically, $min_{\succ}(e)$ is computed for *suffixes* $\langle m_{\mathbf{i+1}}, \ldots, m_n\rangle$.
Specifically, we define

$$min_{\prec}(e) = min\{\vec{p}^T \cdot \langle m_1, \ldots, m_{\mathbf{i}}, 0, \ldots, 0\rangle | \langle m_1, \ldots, m_n\rangle \in elem_e(\mathbf{S})\},$$
$$min_{\succ}(e) = min\{\vec{p}^T \cdot \langle 0, \ldots, 0, m_{\mathbf{i+1}}, \ldots, m_n\rangle | \langle m_1, \ldots, m_n\rangle \in elem_e(\mathbf{S})\}.$$

The following two properties characterize our heuristic rule.

Theorem 2 *Given the initial marking $\vec{m}_0$, the CST $\mathbf{S}$, the structural property $\vec{p}^T \cdot \vec{x} = \vec{p}^T \cdot \vec{m}_0$, and the edge e of $\mathbf{S}$, if $min_{\prec}(e) + min_{\succ}(e) > \vec{p}^T \cdot \vec{m}_0$, then $cone(\vec{m}) \cap \mathcal{O}(\vec{m}_0, \vec{p}) = \emptyset$ for any $\vec{m} \in elem_e(\mathbf{S})$*

Theorem 3 *Given a CST* $\mathbf{S}$*, an edge* e*, and the invariant* $\vec{p}^T \cdot \vec{x} = \vec{p}^T \cdot \vec{m}_0$ *such that* $\vec{p}^T \cdot \vec{m}_0 \in \mathbb{Z}$*, there exists a polynomial time algorithm that computes the values* $min_{\prec}(e)$ *and* $min_{\succ}(e)$*.*

Based on the previous property, we can devise a procedure to heuristically cut the CSTs produced during the backward search. As an example of application of the structural heuristic, consider the CST of fig. 4. The CST $\mathbf{S}$ contains the elements obtained at iteration 1, the pairs of values on the arcs are the values $min_{\prec}(e)$ and $min_{\succ}(e)$ for the place invariant $x_2 + x_5 = 1$, the dashed edges can be removed and thus the useless element $\langle 1, 1, 1, 0, 1 \rangle$ is removed from the CST. Note that if we use the invariant $x_3 + x_4 = 1$ then the last element can also be eliminated. The heuristic rule simply traverses a CST layer by layer, removing all edges that satisfy the hypothesis of Theorem 2. To complete the scenario, we need to compute automatically the structural invariants. This can be done using specialized libraries to compute place invariants like the one available with GreatSPN [CFGR95].

5 Symbolic Backward Reachability

The three main problems we had to solve to obtain an efficient CST-based backward reachability algorithms were: (1) avoid to generate too many redundant elements during the fixpoint computation; (2) use an efficient fixpoint test using sufficient conditions for CST-subsumption; (3) remove useless elements (elements that cannot be reached from the given initial state). As a practical solution to those problems, we propose the algorithm of Fig. 5. The algorithm uses simulation-based heuristics to remove redundancies and for testing subsumption between CSTs, in combination with the heuristic rule proposed in the previous section. Let us give some more detail on the algorithm of Fig. 5. The variable **S** stores the current *frontier* of the breadth-first performed by the algorithm. The variable **T** stores the set of visited generators.

Proc $\mathsf{Pre}^*_{CST}(\mathbf{S}_U : CST)$
  $\mathbf{S} := \mathbf{S}_U$; $\mathbf{T} := empty_{CST}$;
  while $not(Subsumes_{CST}(\mathbf{T}, \mathbf{S}))$ **do**
    $\mathbf{T} := Union_{CST}(\mathbf{T}, \mathbf{S})$; $\mathbf{R} := empty_{CST}$;
    for each *transition* t **do**
      $\mathbf{N} := \mathsf{Pre}_{CST}(\mathbf{S}, t)$;
      $structural\_reduction_{CST}(\mathbf{N})$;
      $remove\_redundancies_{CST}(\mathbf{N}, \mathbf{R}, \mathbf{T})$;
      $minimize_{CST}(\mathbf{N})$;
      $\mathbf{R} := Union_{CST}(\mathbf{N}, \mathbf{R})$;
    $\mathbf{S} := \mathbf{R}$;
  return $\mathbf{T}$;

Figure 5: The CST-based symbolic model checking algorithm.

Before entering the main loop, we need to test subsumption between **S** and **T**. For this purpose, the following heuristic seems to work well in practice. We first compute the forward and backward simulation relations between the nodes of **S** and the nodes of **T**. If the root of **S** is forward simulated by the root of **T** or if the end node of **S** is backward simulated by the end node of **T**, then we know that all the generators of **S** are subsumed by some generators of **T** (see [DR00]), thus the fixpoint is reached. If the test fails, we perform a depth-first, top-down visit of the CST **S** in order to compare its tuples with those of **T**. During the depth-first visit, we use however the information previously computed via the forward simulation as follows. Each time we reach a node n that is forward simulated by a node of $\mathbf{T}$, we stop the exploration: all the elements in the subtree rooted at n will be subsumed by elements of $\mathbf{T}$. In the main loop, we compute the new frontier $\mathbf{N}$ transition by transition via the symbolic operator $\mathsf{Pre}_{CST}(\mathbf{S}, t)$. In order to keep the size of $\mathbf{N}$ small, after computing $\mathsf{Pre}_{CST}(\mathbf{S}, t)$, we first apply the new heuristic rule (via the function $structural\_reduction_{CST}$), and then we apply simulation-based heuristics to remove redundancies. The function $remove\_redundancies_{CST}(\mathbf{N}, \mathbf{R}, \mathbf{T})$ uses simulation relations between nodes of $\mathbf{N}$ and nodes of $\mathbf{R}$ (the CST collecting the generators created via all transitions) and $\mathbf{T}$; the function $minimize_{CST}(\mathbf{N})$ uses simulation relations of nodes of $\mathbf{N}$. We discuss the practical evaluation of the resulting algorithm in the following section.

6 Experimental Results

Based on a new optimized implementation of the CST-library presented in [DR00], and using the library for computing *minimal* place invariants (a system of generators for the positive solutions of $\vec{x}^T \cdot \mathbf{C} = 0$) coming with GreatSPN [CFGR95], we have implemented the algorithm of Fig. 5 and tested on several types of verification problems expressible in terms of *coverability* of markings for Petri Nets. The parameters taken into considerations in our evaluation are listed in Fig. 6.

Parameterized Problems. More precisely, we have considered *mutual exclusion* properties for the parameterized, concurrent and production systems like the Multipoll of [MC99], the Mesh 2x2 of [ABC$^+$95] (Fig. 130, p. 256), its extension to the 3x2 case, the CSM of [ABC$^+$95] (Fig. 76, p. 154), and for an extension of the Readers-Writers example given in [Rei86] in which we use several buffers with 45 slots. Furthermore, we have considered *semi-liveness* and *coverability* problems for the PNCSA communication protocol analyzed in [BF99, Fin93]. The experimental results are listed in Fig. 7. We performed every example either enabling or disabling the structural heuristic rule and the reductions based on simulation relations. As shown in Fig. 7, the heuristics turned out to be fundamental to ensure termination in reasonable time for most of the examples. To compare our results with other infinite-state systems, we ran some of the parameterized examples like CSM and Mesh using the efficient model checker based on polyhedra (i.e. constraint solver over the reals) HyTech [HHW97]. In the experiments on the largest examples (using backward analysis) HyTech was still computing after more than *one day*.

Finite-state Problems. After having fixed the value of the parameter K in the initial marking, we have also tested some case-studies using the specialized Petri Net tool GreatSPN [CFGR95]. GreatSPN uses efficient encodings of markings and simplification rules that reduce the input net to produce the *reachability set* of *bounded* Petri Nets. We performed our experiments on a Pentium 133Mhz measuring the value of K from which GreatSPN is *not able* to compute the entire reachability graph: $K = 3$ for the Mesh 2x2; $K = 9$ for Multipoll, and $K = 115$ for CSM. In contrast, as shown in Fig. 7, we managed to verify mutual exclusion properties for any value of K (assuming $K \geq 1$ in the initial marking) with the following execution times: 1.26s for Mesh 2x2; 1.05s and 324s for Multipoll; and 0.04s for CSM. As already no-

Size of the Petri Net
P=Number of places;
T = Number of transitions.
Verification problem (VP) (only the type of property)
ME=Mutual exclusion property;
C=Covering for a random marking;
SL=Semi-liveness.
Use of heuristics
I=Invariant-based reductions (structural heuristic rule);
S=Simulation-based reductions.
Statistics: execution
EX=Execution time (in seconds) on an AMD Athlon 900 Mhz;
NI=N. of iterations before termination (with *=before stopping the execution).
Quality of analysis
R=An initial state has been reached.
Statistics: use of memory
MaxE(N)=N. of *elements* (*nodes*) of the biggest CST associated to **S** of Fig. 5;
NE(N)=Number of *elements* (*nodes*) of the CST for the fixpoint;
Ratio of memory saving (using CSTs)
RM=MAX-N/(MAX-E $\times$ P) in pct.;
RN=NN/(NE $\times$ P) in pct.

Figure 6: Parameters of the experimental evaluation.

ticed by Bultan in other case-studies [Bul00], lifting a verification problem from the *finite-state* to the *parameterized* case can make its solution easier! Also note that the use of invariants makes the backward analysis sensible to the initial marking. This effect is clear looking at the execution times obtained using different values for K for the Mesh2x2 in Fig. 7 (e.g., we found more useful invariants for $K = 1$ than for $K \geq 1$).
Finally, we have also considered safety properties for *non-parametric examples* (i.e., where it makes no sense to put parameters in the initial marking) like the classical Peterson's and Lamport's mutual exclusion algorithms [MC99, EM00]. As a result, we managed to prove safety properties for all these examples with negligible execution times.

7 Related Works and Conclusions

In this paper we have presented new heuristic rule, based on the structural theory of Petri Nets, to be used in the backward approach of [ACJT96, FS01]. Efficient algorithms allow us to apply the heuristic rule avoiding the enumeration of the minimal points of upward-closed sets generated in the computation of Pre^*. This way, we manage to mitigate the symbolic state explosion in practical examples we did not manage to handle with previous *backward* technology. With the set of benchmarks of Fig. 7, we hope it will be possible to establish connections with other recent attempts of attacking symbolic state explosion [AN00, BLP$^+$99]. The combination of structural and enumerative techniques has been studied before in the context of *forward reachability*, where invariants are used as heuristic for efficient encodings of markings [CFGR95, PCP99]. Structural properties are also used to statically compute over-approximations of the reachability set of a Petri Net [EM00, STC98]. We are not aware of previous attempts of combining structural heuristics and backward reachability.

References

[ABC$^+$95] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* Series in Parallel Computing. John Wiley & Sons, 1995.

[AN00] P. A. Abdulla and A. Nylén. Better is Better than Well: On Efficient Verification of Infinite-State Systems. In *Proc. LICS 2000*, pages 132-140, 2000.

[ACJT96] P. A. Abdulla, K. Cerāns, B. Jonsson and Y.-K. Tsay. General Decidability Theorems for Infinite-State Systems. In *Proc. LICS '96*, pages 313–321, 1996.

[BCR01] T. Ball, S. Chaki, S. K. Rajamani. Parameterized Verification of Multithreaded Software Libraries. MSR Technical Report 2000-116. In *Proc. TACAS 2001*, LNCS 2031, pages 158-173, 2001.

[BLP$^+$99] G. Behrmann, K. G. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient Timed Reachability Analysis Using Clock Difference Diagrams. In *Proc. CAV '99*, LNCS 1633, pages 341–353, 1999.

[BF99] B. Bérard and L. Fribourg. Reachability analysis of (timed) Petri nets using real arithmeti In *Proc. CONCUR '99*, LNCS 1664, pages 178-193, 1999.

[BM99] A. Bouajjani, and R. Mayr. Model Checking Lossy Vector Addition Systems. In *Proc. STACS '99*, LNCS 1563, pages 323-333, 1999.

[Bul00] T. Bultan. BDD vs. Constraint-Based Model Checking: An Experimental Evaluation for Asynchronous Concurrent Systems. In *Proc. TACAS 2000*, LNCS 1785, pages 441-455, 2000.

[BGP97] T. Bultan, R. Gerber, and W. Pugh. Symbolic Model Checking of Infinite-state Systems using Presburger Arithmetics. In *Proc. CAV '97*, LNCS 1254, pages 400–411, 1997.

[CFGR95] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. GreatSPN 1.7: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets. *Performance Evaluation*, 24(1-2), pages 47-68, 1995.

[Del00] G. Delzanno. Automatic Verification of Parameterized Cache Coherence Protocols. In *Proc. CAV 2000*, LNCS 1855, pages 53–68, 1996.

[DEP99] G. Delzanno, J. Esparza, and A. Podelski. Constraint-based Analysis of Broadcast Protocols. In *Proc. CSL '99*, LNCS 1683, pag. 50–66, 1999.

[DR00] G. Delzanno, and J. F. Raskin. Symbolic Representation of Upward-closed Sets. In *Proc. TACAS 2000*, LNCS 1785, pages 426-440, 2000.

[EN98] E. A. Emerson and K. S. Namjoshi. On Model Checking for Non-deterministic Infinite-state Systems. In *Proc. LICS '98*, pages 70–80, 1998.

[EFM99] J. Esparza, A. Finkel, and R. Mayr. On the Verification of Broadcast Protocols. In *Proc. LICS '99*, pages 352–359, 1999.

[EM00] J. Esparza and S. Melzer. Verification of safety properties using integer programming: Beyond the state equation. Formal Methods in System Design, 16:159-189, 2000.

[Fin93] A. Finkel. The minimal coverability graph for Petri nets. In In *Advances in Petri Nets '93*, LNCS 674, pages 210-243. Springer, 1993.

[FS01] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63-92, 2001.

[GS92] S. M. German, A. P. Sistla. Reasoning about Systems with Many Processes. *JACM* 39(3): 675–735 (1992)

[HHW97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: a Model Checker for Hybrid Systems. In *Proc. CAV '97*, LNCS 1254, pages 460–463, 1997.

[KMM+97] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, E. Shahar. Symbolic Model Checking with Rich Assertional Languages. In *Proc. CAV '97*, LNCS 1254, pages 424–435, 1997.

[MC99] A. Miner, and G. Ciardo. Efficient Reachability Set Generation and Storage using Decision Diagrams. In *Proc. of ICATPN '99*, pages 6-25, 1999.

[PCP99] E. Pastor, J. Cortadella, M. A. Peña. Structural Methods to Improve the Symbolic Analysis of Petri Nets. In *Proc. ICATPN '99*, LNCS 1639, pages 26-45. Springer, 1999.

[Rei86] W. Reisig. Petri Nets. An introduction. EATCS Monographs on Theoretical Computer Science, Springer 1986.

[STC98] M. Silva, E. Teruel, and J. M. Colom. Linear Algebraic and Linear Programming Techniques for Analysis of Place/Transition Net Systems. In W. Reisig and G. Rozenberg, editors. Lectures on Petri Nets I: Basic Models. *Advances in Petri Nets*, LNCS 1491, pages 308-309. Springer, 1998.

[ZL94] D. Zampuniéris, and B. Le Charlier. Efficient Handling of Large Sets of Tuples with Sharing Trees. In *Proceedings of the Data Compressions Conference (DCC'95)*, 1995.

Case-study	P	T	VP	I	S	EX(sec)	NI	R	MAX-E	MAX-N	RM	NE	NN	RN
Ex-Fig. $1_{K>1}$	5	4	ME	$\surd$	$\surd$	0.00	1		1	7	140	1	7	140
Ex-Fig. $1_{K>1}$	5	4	ME		$\surd$	0.02	4		4	17	85	9	29	64.4
Ex-Fig. $1_{K>1}$	5	4	ME			0.01	4		7	22	62.9	15	32	42.7
Ex-Fig. $1_{K>1}$	5	4	ME	$\surd$		0.00	1		1	7	140	1	7	140
$\mathrm{CSM}_{K>1}$	14	13	ME	$\surd$	$\surd$	0.08	9		15	73	35	70	68	7
$\mathrm{CSM}_{K>1}$	14	13	ME		$\surd$	0.19	11		29	115	28	152	109	5
$\mathrm{CSM}_{K>1}$	14	13	ME			39.25	11		33737	7342	1	25220	4680	1
$\mathrm{CSM}_{K=2}$	14	13	ME	$\surd$	$\surd$	0.08	9		15	73	35	70	68	7
$\mathrm{CSM}_{K=75}$	14	13	ME	$\surd$	$\surd$	0.08	9		15	73	35	70	68	7
$\mathrm{CSM}_{K=115}$	14	13	ME	$\surd$	$\surd$	0.08	9		15	73	35	70	68	7
$\mathrm{Multip.}_{K>1}$	18	21	ME		$\surd$	2.98	18		881	435	3	5641	196	0.2
$\mathrm{Multip.}_{K>1}$	18	21	ME			>20000	18*		8544367	60922	0.04	–	–	–
$\mathrm{Multip.}_{K>1}$	18	21	ME		$\surd$	117.21	21		9416	2404	1	77015	652	0.04
$\mathrm{Multip.}_{K=1}$	18	21	ME	$\surd$	$\surd$	1.96	18		649	341	2.9	4361	140	0.2
$\mathrm{Multip.}_{K=4}$	18	21	ME	$\surd$	$\surd$	3.06	18		881	435	2.7	5641	196	0.2
$\mathrm{Multip.}_{K=9}$	18	21	ME	$\surd$	$\surd$	3.06	18		881	435	2.7	5641	196	0.2
$\mathrm{Mesh2x2}_{K>1}$	32	32	ME	$\surd$	$\surd$	1.11	15		54	340	20	429	277	2
$\mathrm{Mesh2x2}_{K>1}$	32	32	ME		$\surd$	1.14	15		54	340	20	429	294	2
$\mathrm{Mesh2x2}_{K>1}$	32	32	ME	$\surd$	$\surd$	7.37	20		272	921	11	2138	935	1
$\mathrm{Mesh2x2}_{K>1}$	32	32	ME		$\surd$	9.34	20		342	951	9	2418	1157	1
$\mathrm{Mesh2x2}_{K>1}$	32	32	ME		$\surd$	16.65	16		282	1367	15	2130	1741	2
$\mathrm{Mesh2x2}_{K>1}$	32	32	ME	$\surd$		>20000	14*		433665	105607	1	–	–	–
$\mathrm{Mesh2x2}_{K=1}$	32	32	ME	$\surd$	$\surd$	0.69	13		30	272	28	211	326	5
$\mathrm{Mesh2x2}_{K=2}$	32	32	ME	$\surd$	$\surd$	1.11	15		51	353	22	407	294	2
$\mathrm{Mesh2x2}_{K=3}$	32	32	ME	$\surd$	$\surd$	1.11	15		54	340	20	429	277	2
$\mathrm{Mesh3x2}_{K>1}$	52	54	ME	$\surd$	$\surd$	11.18	21		335	1192	7	2441	1074	1
$\mathrm{Mesh3x2}_{K>1}$	52	54	ME		$\surd$	12.06	21		343	1198	7	2447	1076	5
$\mathrm{Mesh3x2}_{K=1}$	52	54	ME	$\surd$	$\surd$	8.4	21		153	1183	15	1159	1643	3
$\mathrm{R/W}_{K>1}$	24	22	ME	$\surd$	$\surd$	3215.34	122		12590	5911	2	29119	1936	0.3
$\mathrm{R/W}_{K>1}$	24	22	ME		$\surd$	>20000	16*		160197	30941	1	–	–	–
$\mathrm{R/W}_{K>1}$	24	22	ME			>20000	9*		1239466	73724	0.2	–	–	–
$\mathrm{R/W}_{K>1}$	24	22	ME	$\surd$		>20000	13*		1672064	122510	0.3	–	–	–
PNCSA	31	36	SL	$\surd$	$\surd$	0.98	10	$\surd$	55	418	24	117	594	16
PNCSA	31	36	SL		$\surd$	6.98	10	$\surd$	268	1590	19	356	1744	16
PNCSA	31	36	C	$\surd$	$\surd$	52.6	33	$\surd$	590	1604	9	604	751	4
PNCSA	31	36	C		$\surd$	25871.28	33	$\surd$	12495	8174	2	19435	10886	2
Peterson	14	12	ME	$\surd$	$\surd$	0.01	2		2	28	100	3	28	67
Peterson	14	12	ME		$\surd$	0.18	9		36	118	23	181	169	7
Peterson	14	12	ME			23.12	9		15732	7212	0.3	12125	4331	2
Peterson	14	12	ME	$\surd$		0.01	2		2	28	100	3	28	67
Lamport	11	9	ME	$\surd$	$\surd$	0.01	1		1	13	118	1	13	118
Lamport	11	9	ME		$\surd$	0.08	10		16	66	37	57	80	13
Lamport	11	9	ME			1.09	10		4100	1864	4.1	4260	1396	2.9

Figure 7: The experimental results have been obtained using an AMD Athlon 900 Mhz. The parameters of the evaluation are described in Fig. 6.