Symbolic Representation of Upward-Closed Sets

Giorgio Delzanno[1] and Jean-François Raskin[2,3]

[1] Max-Planck-Institut für Informatik
Im Stadtwald, 66123 Saarbrücken, Germany
delzanno@mpi-sb.mpg.de
[2] Electrical Engineering and Computer Sciences, University of California at Berkeley
Berkeley, CA 94720-1770
jfr@eecs.berkeley.edu
[3] Département d'Informatique, Université Libre de Bruxelles
Blvd Du Triomphe, 1050 Bruxelles, Belgium

Abstract. The *control state reachability problem* is decidable for well-structured infinite-state systems like unbounded Petri Nets, Vector Addition Systems, Lossy Petri Nets, and Broadcast Protocols. An abstract algorithm that solves the problem is given in [AČJT96,FS99]. The algorithm computes the closure of the *predecessor* operator w.r.t. a given upward-closed set of target states. When applied to this class of verification problems, traditional (infinite-state) symbolic model checkers suffer from the *state explosion problem* even for very small examples. We provide BDD-like data structures to represent in a compact way collections of upwards closed sets over numerical domains. This way, we turn the abstract algorithm of [AČJT96,FS99] into a practical method. Preliminary experimental results indicate the potential usefulness of our method.

1 Introduction

In the last years many efforts have been made to extend the theoretical results and practical methods developed for finite-state systems [BCB+90] to systems with *infinite* state space (see e.g. [AČJT96,BM99,BW98,EFM99,FS99]). This class of systems comprises well-known examples like Vector Addition Systems [Min67], extensions of Petri Nets [Cia94,Ter94], Integral Relational Automata [Čer94], and more recent examples like Broadcast Protocols [EN98] and Lossy Petri Nets [BM99]. The *control state reachability problem* is decidable for all previous systems [AČJT96,BM99,EFM99,Fin90,FS99]. The abstract algorithm of [AČJT96,FS99] computes the closure of the *predecessor* operator w.r.t. a given upward-closed set of states. The algorithm can be used to check, e.g., invariant properties like *mutual exclusion* [DEP99] and *coverability for markings of Petri Nets* [AJKP98].

As in the finite-state case, the success of symbolic model checking for this class of problems depends on the data structures used as implicit representation of sets of states. Over numerical domains, upward-closed sets can be represented via a sub-class of integer arithmetic constraints (see e.g. [AČJT96,DEP99]). In this setting, the state-space generated by the algorithm of [AČJT96,FS99] can

S. Graf and M. Schwartzbach (Eds.): TACAS/ETAPS 2000, LNCS 1785, pp. 426–441, 2000.

be represented as a large *disjunction* of arithmetic constraints. In [DEP99], the authors tested constraint-based model checking methods over integers (based on a solver for Presburger arithmetic [BGP97]) and reals (based on polyhedra [HHW97]) on verification problems that can be expressed via upward-closed sets. Though the examples in [DEP99] would be considered of negligible size in finite-state model checking (e.g. 6 transitions and 10 variables, cf. [BCB+90]), the methods taken into consideration suffer from the state explosion problem[1]. Some of the experiments required (when terminating) execution times in the order of days. Based on these observations, it seems natural to look for BDD-like data structures to represent 'compactly' the generalization of boolean formulas we are interested in.

In this paper we propose a new symbolic representation for upward-closed sets based on the *sharing trees* of Zampuniéris and Le Charlier [ZL94]. Sharing trees are acyclic graphs used to represent large sets of tuples, e.g., of integer numbers. The intuition behind the choice of this data structure is the following. An upward-closed set U is determined by its finite set of generators (tuples of integers). Thus, we can represent the set U via a sharing tree whose paths correspond to its generators. This way, we managed to turn the abstract algorithm of [AČJT96,FS99] into a 'practical method' working on the examples studied in [DEP99,Ter94] in acceptable time cost.

Technically, our contributions are as follows. We introduce a logic (the logic $\mathcal{U}$) where collections of upward-closed sets can be represented as disjunctive formulas. $\mathcal{U}$-formulas are used for verification problems of infinite-state systems as boolean formulas are used for the finite-state case. We show that sharing trees can be used to obtain compact representations of $\mathcal{U}$-formulas (there exist a $\mathcal{U}$-formula that can be represented using a sharing tree whose size is logarithmic in the size of the formula). We show how basic operations on $\mathcal{U}$-formulas (e.g. conjunction and disjunction) can be implemented symbolically on the corresponding sharing trees. Sharing trees can be viewed as the BDDs for $\mathcal{U}$-formulas.

In practical cases (e.g., during the symbolic computation of the closure of the predecessor operator), sharing trees may still become very large. For this reason, we propose polynomial time algorithms that can be used to eliminate redundant paths. As we prove in the paper, the problem of removing all redundancies from a sharing tree representing a $\mathcal{U}$-formula is co-NP hard (in the size of the sharing tree). The same techniques can be used to give sufficient conditions for the subsumption test of sharing trees representing $\mathcal{U}$-formulas (i.e. for the termination test of the algorithm of [AČJT96]). The complete test is co-NP hard in the size of the sharing trees.

As an application of our method, we have implemented the algorithm of [AČJT96] in the case of Vector Addition Systems. The implementation makes use of the sharing tree library of [Zam97]. First experimental results indicate the potential usefulness of our method.

[1] One would say 'symbolic state explosion problem', in fact, the above cited methods operate on implicit representations of infinite sets of states.

Plan of the Paper. In Section 2, we define the logic $\mathcal{U}$. In Section 3, we introduce the symbolic representation of $\mathcal{U}$-formulas via sharing trees. In Section 4, we define simulation relations for nodes of sharing trees and discuss their application in the operations for $\mathcal{U}$-formulas. In Section 5, we define a symbolic model checking procedure for Vector Addition Systems. In Section 6, we present related work. In Section 7, we address some conclusions and future perspectives for our work.

The extended version of the paper (containing all proofs) is available as technical report MPI-1999-2-07 of Max-Planck-Institut für Informatik [DR99].

2 The Logic of Upward-Closed Sets

In this section we introduce the logic $\mathcal{U}$ that we use to define collections of upward-closed sets. Let $V = \{x_1, \ldots, x_k\}$ be a finite set of variables. The set of $\mathcal{U}$-formulas is defined by the following grammar.

$$\Phi \quad ::= \quad x_i \geq c \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \Phi[x_i + c/x_i],$$

where $c \in \mathbb{Z} \cup \{-\infty\}$. $\mathcal{U}$-formulas are interpreted over $\mathbb{Z}^k$. We use $\boldsymbol{t}$ to denote the valuation $\langle t_1, \ldots, t_k \rangle$, where $t_i \in \mathbb{Z}$ is the valuation for variable x_i. We consider the following order over tuples: $\boldsymbol{t} \preccurlyeq \boldsymbol{t'}$ iff $t_i \leq t'_i$ for $i : 1, \ldots, k$ ($-\infty \leq c$ for any $c \in \mathbb{Z}$). When restricted to positive values, $\preccurlyeq$ is a well-quasi ordering (see e.g. [AČJT96,FS99]). Given a tuple $\boldsymbol{t}$, we define $\boldsymbol{t}^{\uparrow}$ as the *upward-closed set* generated by $\boldsymbol{t}$, namely, $\boldsymbol{t}^{\uparrow} = \{ \boldsymbol{t'} \mid \boldsymbol{t} \preccurlyeq \boldsymbol{t'} \}$. Satisfaction of a formula wrt. a valuation $\boldsymbol{t}$ is defined as follows:

- $\boldsymbol{t} \models x_i \geq c$ iff $t_i \geq c$;
- $\boldsymbol{t} \models \Phi_1 \wedge \Phi_2$ iff $\boldsymbol{t} \models \Phi_1$ and $\boldsymbol{t} \models \Phi_2$;
- $\boldsymbol{t} \models \Phi_1 \vee \Phi_2$ iff $\boldsymbol{t} \models \Phi_1$ or $\boldsymbol{t} \models \Phi_2$;
- $\boldsymbol{t} \models \Phi[x_i + c/x_i]$ iff $\boldsymbol{t'} \models \Phi$ and $\boldsymbol{t'}$ is obtained from $\boldsymbol{t}$ replacing t_i with $t_i + c$.

The *denotation* of a formula Φ, namely $[\![\Phi]\!]$, is defined as the set of all evaluations $\boldsymbol{t}$ such that $\boldsymbol{t} \models \Phi$. A formula Φ_1 is *subsumed* by a formula Φ_2, written $\Phi_1 \models \Phi_2$, if $[\![\Phi_1]\!] \subseteq [\![\Phi_2]\!]$. Two formulas are equivalent if their denotations coincide.

Note that, whenever we restrict the domain of interpretation of our formulas to *positive* integers (say $\mathbb{Z}_+$), the class of $\mathcal{U}$-formulas denotes all upward-closed sets, i.e., all sets $I \subseteq \mathbb{Z}_+^k$ such that if $\boldsymbol{t} \in I$ then $\boldsymbol{t}^{\uparrow} \subseteq I$.

All formulas can be reduced to *disjunctive formulas*, i.e., to formulas having the following form[2]:

$$\bigvee_{i \in I} (x_1 \geq c_{i,1} \wedge \ldots \wedge x_k \geq c_{i,k}).$$

Notation. In the rest of the paper we use Φ, Ψ, etc. to denote arbitrary $\mathcal{U}$-formulas, and ϕ, ψ, etc. to denote disjunctive formulas.
The set of *generators* of a disjunctive formula φ are defined as

[2] Adding formulas of the form $x_i \geq -\infty$ when necessary.

$$gen(\varphi) = \{\ \langle c_1, \ldots, c_k\rangle \mid (x_1 \geq c_1 \wedge \ldots \wedge x_k \geq c_k) \text{ is a disjunct in } \varphi\ \}.$$

Thus, disjunctive formulas are in one-to-one correspondence with their set of generators modulo logical equivalences. The minimal elements (wrt. $\preccurlyeq$) of $gen(\varphi)$ are denoted by $min(\varphi)$. Note that $[\![\varphi]\!] = \bigcup_{\boldsymbol{t}\in min(\varphi)} \boldsymbol{t}^{\uparrow}$. We say that a disjunctive formula is in *normal form* whenever $gen(\varphi) = min(\varphi)$. As an example, consider the formula $\varphi = (x \geq 1 \wedge y \geq 2) \vee (x \geq 3 \wedge y \geq 1) \vee (x \geq 2 \wedge y \geq 0)$. Then, $gen(\varphi) = \{\langle 1,2\rangle, \langle 3,1\rangle, \langle 2,0\rangle\}$, and $min(\varphi) = \{\langle 1,2\rangle, \langle 2,0\rangle\}$, i.e., φ is *not* in normal form.

2.1 Operations on Formulas in Disjunctive Form

Disjunction and Conjunction. Formulas in disjunctive form are closed under $\vee$. Furthermore, given the disjunctive formulas φ and ψ, the disjunctive formula for $\varphi \wedge \psi$ is defined as follows: $\bigvee_{\boldsymbol{t}\in gen(\varphi),\boldsymbol{t'}\in gen(\psi)}(x_1 \geq max(t_1, t'_1) \wedge \ldots \wedge x_k \geq max(t_k, t'_k))$, i.e., $gen(\varphi \wedge \psi) = \{\boldsymbol{s} \mid \exists \boldsymbol{t} \in gen(\varphi), \boldsymbol{t'} \in gen(\psi)$ and $s_i = max(t_i, t'_i)\}$. Note that the resulting formula may not be in normal form.

Substitution. The formula $\varphi[x_i + c/x_i]$ is equivalent to the formula φ' obtained from φ by replacing every atom $x_i \geq d$ with $x_i \geq d - c$ ($-\infty - c = -\infty$ for any $c \in \mathbb{Z}$), i.e., $gen(\varphi[x_i + c/x_i]) = \{\boldsymbol{t'} \mid t'_i + c = t_i,\ t'_j = t_j\ j \neq i,\ \boldsymbol{t} \in gen(\varphi)\}$.

Satisfaction wrt. a tuple. Given a valuation $\boldsymbol{t}$, we first note that $\boldsymbol{t} \models \varphi$ iff there exists $\boldsymbol{t'} \in gen(\varphi)$ such that $\boldsymbol{t'} \preccurlyeq \boldsymbol{t}$. Thus, checking $\boldsymbol{t} \models \varphi$ can be done in time linear in the size of φ.

Subsumption. Let φ and ψ be in disjunctive form. We can check $\varphi \models \psi$ in time quadratic in the size of the formulas. In fact, $\varphi \models \psi$ holds iff for all $\boldsymbol{t} \in gen(\varphi)$ there exists $\boldsymbol{t'} \in gen(\psi)$ such that $\boldsymbol{t'} \preccurlyeq \boldsymbol{t}$.

It is important to remark that the subsumption test is much harder for *arbitrary* $\mathcal{U}$-formulas, as stated in the following theorem.

Theorem 1. Given two arbitrary $\mathcal{U}$-formulas Φ and Ψ, checking that $\Phi \models \Psi$ is co-NP complete in the size of the formulas.

Reduction in normal form. Given a disjunctive formula φ we can reduce it in normal form by eliminating all 'redundant' generators from $gen(\varphi)$, i.e., all $\boldsymbol{t} \in gen(\varphi)$ such that there exists $\boldsymbol{t'} \in gen(\varphi)$, $\boldsymbol{t} \neq \boldsymbol{t'}$, $\boldsymbol{t'} \preccurlyeq \boldsymbol{t}$. The reduction can be done in time quadratic in the size of φ.

All previous operations depend on the set of 'generators' of disjunctive formulas. In the following section we introduce a special data structure, called sharing tree [ZL94], for handling large sets of generators. We show how to use this data structure to represent and manipulate symbolically formulas of the logic $\mathcal{U}$.

3 Sharing Trees

In this paper we specialize the original definition of [ZL94] as follows. We call a k-sharing tree a rooted directed acyclic graph $(N, V, root, end, val, succ)$ where $N = \{root\} \cup N_1 \ldots \cup N_k \cup \{end\}$ is the finite set of *nodes*, (N_i is the set of nodes of *layer* i and, by convention, $N_0 = \{root\}$ and $N_{k+1} = \{end\}$), $val : N \rightsquigarrow \mathbb{Z} \cup \{\top, \bot\}$ is a labeling function for the nodes, and $succ : N \rightsquigarrow 2^N$ defines the successors of a node. Furthermore,

1. $val(n) = \top$ if and only if $n = root$;
2. $val(n) = \bot$ if and only if $n = end$;
3. $succ(end)=\emptyset$;
4. for $i : 0, \ldots, k$, forall $n \in N_i$, $succ(n) \subseteq N_{i+1}$ and $succ(n) \neq \emptyset$;
5. forall $n \in N$, forall $n_1, n_2 \in succ(n)$, if $n_1 \neq n_2$ then $val(n_1) \neq val(n_2)$.
6. for $i : 0, \ldots, k$, forall $n_1, n_2 \in N_i$ s.t. $n_1 \neq n_2$, if $val(n_1) = val(n_2)$ then $succ(n_1) \neq succ(n_2)$.

In other words, a k-sharing tree is an acyclic graph with root and terminal node such that: all nodes of layer i have successors in the layer $i+1$ (cond. 4); a node cannot have two successors with the same label (cond. 5); finally, two nodes with the same label in the same layer do not have the same set of successors (cond. 6). We say that S is a *pre-sharing* tree if it respects conditions (1)-(4) but possibly not (5) and (6).
Notation. In the rest of the paper we use $root^S$, N^S, $succ^S$ *etc. to refer to the root, set of nodes, successor relation etc. of the sharing-tree* S.
A path of a k-sharing tree is a sequence of nodes $\langle n_1, \ldots, n_m \rangle$ such that $n_{i+1} \in succ(n_i)$ $i : 1, \ldots, m$-1. Paths will represent tuples of size k of integer numbers. Formally, we use $elem(S)$ to denote the set of elements represented by the k-sharing tree S:

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

Condition 5 and 6 ensure the maximal sharing of prefixes and suffixes among the paths (elements) of the sharing tree. We define the 'size' of a sharing tree as the number of its *nodes* and *edges*. Note that the number of tuples in $elem(S)$ can be exponentially larger than the size of S. Given a node n of the i-th layer of a k-sharing tree S, the sub-(sharing)tree S_n rooted at n is the $k - i + 1$-sharing tree obtained as follows. We first isolate the graph rooted at n and consisting of all nodes reachable from n (this subgraph has $k - i + 1$ layers and a terminal node). Then, we add a layer with the single node *root* and we set $succ(root) = \{n\}$. From the previous definition, $elem(S_n)$ consists of all tuples $\langle val(n), m_{i+1}, \ldots, m_k \rangle$ obtained from tuples $\langle m_1, \ldots, val(n), m_{i+1}, \ldots, m_k \rangle$ of $elem(S)$.

As shown in [ZL94], given a set of tuples F of size k, there exists a unique (modulo isomorphisms of graphs) sharing tree such that $elem(S) = F$. In the same paper the authors give algorithms for the basic set-operations on the set of elements represented by the sharing trees. The table in Fig. 1 gives the specifi-

Operation	Complexity
$elem(union(S,T)) = elem(S) \cup elem(T)$	$\mathcal{O}(max(edges(S), edges(T)) + Red)$
$elem(intersection(S,T)) = elem(S) \cap elem(T)$	$\mathcal{O}(min(edges(S), edges(T)) + Red)$
$member(\boldsymbol{t}, S)$ *iff* $\boldsymbol{t} \in elem(S)$	$\mathcal{O}(size(\boldsymbol{t}))$
$contained(S,T)$ *iff* $elem(S) \subseteq elem(T)$	$\mathcal{O}(edges(S))$
$is\_empty(S)$ *iff* $elem(S) = \emptyset$	$\mathcal{O}(const)$

Fig. 1. Operations on the sharing trees S and T: $edges(S)$=No.edges of S

cation and the complexity in terms of the size of sharing trees, for the operation we will consider in the rest of the paper: union, intersection, emptiness, containment, and equality test. In [ZL94], the authors show that the operations in Fig. 1 can be safely applied to pre-sharing trees that satisfy condition 5 only. The cost for intersection and union depends also on the cost *Red* in Fig. 1, of re-arranging condition 6. This task can be achieved using the algorithm presented in [ZL94] with cost quadratic in the number of nodes of the resulting sharing trees.

3.1 Symbolic Representation of $\mathcal{U}$-Formulas

We first show how to represent $\mathcal{U}$-formulas in disjunctive form, and then show how to define disjunction, conjunction, subsumption and reduction in normal form over the resulting data structure.

Let φ be a $\mathcal{U}$-formula in disjunctive form over $x_1, \ldots, x_k$. We define S_φ as the k-sharing tree such that $elem(S_\varphi) = gen(\varphi)$. The *denotation* of a k-sharing tree S is then defined as $[\![S]\!] = \bigcup_{\boldsymbol{t} \in elem(S)} \boldsymbol{t}^{\uparrow}$. Clearly, $[\![\varphi]\!] = [\![S_\varphi]\!]$. We say that S_φ is irredundant if φ is in normal form, i.e., there exists no $\boldsymbol{t} \in elem(S_\varphi)$ such that $\boldsymbol{t'} \preccurlyeq \boldsymbol{t}$ for $\boldsymbol{t'} \in elem(S_\varphi)$ distinct from $\boldsymbol{t}$. The following proposition explains the advantages of using sharing trees for representing $\mathcal{U}$-formulas.

Proposition 1. There exist a disjunctive formula in normal form φ such that the corresponding sharing tree S_φ has size (no. of nodes and arcs) logarithmic in the size of φ.

As an example, consider the $\mathcal{U}$-formula $\Phi = \bigvee_{i=1}^{m} (v_i \geq 1 \wedge w_i \geq 0) \vee (v_i \geq 0 \wedge w_i \geq 1)$. The corresponding disjunctive formulas φ (obtained by distributing $\wedge$) is $\bigvee_{\boldsymbol{c}+\boldsymbol{d} \geq 1} \bigwedge_{i=1}^{m} (v_i \geq c_i \wedge w_i \geq d_i)$ for $\boldsymbol{c}, \boldsymbol{d} \in \mathbb{Z}_+^m$. This formula has $\mathcal{O}(2^m)$ disjuncts. Also note that φ is in normal form. In contrast, the sharing tree S_φ shown in Fig. 2 has size logarithmic in φ and polynomial in Φ (each layer has two nodes and at most four arcs).

3.2 Symbolic Operations for $\mathcal{U}$-Formulas

In this section we show how operations on the disjunctive formulas φ and ψ can be defined symbolically on their representations S_φ and S_ψ. We use the term *symbolically* because the algorithms that we propose work directly on the graphs S_φ and S_ψ and not by enumerating the elements that they represent.

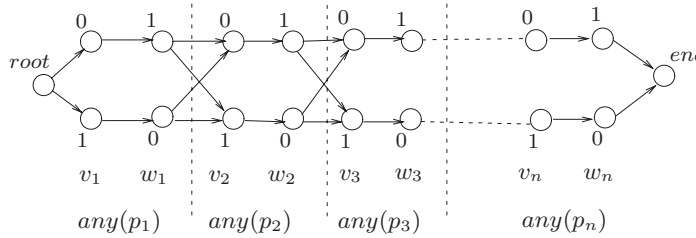

Fig. 2. Sharing tree for an exponential formula in DNF

Disjunction. Let S_φ and S_ψ be the k-sharing trees representing the formulas (in disjunctive form) φ and ψ. To build a sharing tree with $elem(S_{\varphi\vee\psi}) = gen(\varphi)\cup gen(\psi)$, we define $S_{\varphi\vee\psi}$ as $union(S_\varphi, S_\psi)$, where *union* is the operation in Fig. 1. In [ZL94], it has been show that the size of the sharing-tree $S_{\varphi\vee\psi}$ is at most quadratic in the size of S_φ and S_ψ, and can be computed in time quadratic in the size of the input sharing-trees.

Conjunction. Given S_φ and S_ψ, we build $S_{\varphi\wedge\psi}$ as follows. We first define a pre-sharing tree P with the following components: (i) $N^P = \{root\}\cup N_1\cup\ldots\cup N_k\cup\{end\}$ with $N_i = \{(n,m) \mid n\in N_i^{S_\varphi}, m\in N_i^{S_\psi}\}$ (i.e. a node in P correspond to a pair consisting of a node of S_φ and a node of S_ψ (at the same layer); (ii) $val^P((n,m)) = max(val^{S_\varphi}(n), val^{S_\psi}(m))$, and (iii) for all $(n,m)\in N_1\cup\ldots\cup N_n$, we set $succ^P((n,m)) = \{(n',m') \mid n'\in succ^{S_\varphi}(n), m'\in succ^{S_\psi}(m)\}$, $succ^P(root) = N_1$, and for all $(n,m)\in N_k$ we set $succ^P((n,m)) = \{end\}$. We obtain the sharing-tree $S_{\varphi\wedge\psi}$ from P by enforcing conditions (5-6) of Section 3 with the algorithms proposed in [ZL94].

Substitution. Given the sharing tree S_φ we build a new sharing tree $S_{\varphi[x_i+c/x_i]}$ such that $elem(S_{\varphi[x_i+c/x_i]}) = gen(\varphi[x_i+c/x_i])$ as follows. $S_{\varphi[x_i+c/x_i]}$ has the same components as S_φ except from the valuation function: for every node $n\in N_i^{S_{\varphi[x_i+c/x_i]}}$ $val^{S_{\varphi[x_i+c/x_i]}}(n) = val^{S_\varphi}(n)-c$, and for every node $n\in N_j^{S_{\varphi[x_i+c/x_i]}}$, with $j\neq i$, $val^{S_{\varphi[x_i+c/x_i]}}(n) = val^{S_\varphi}(n)$. This way, we obtain a well-formed sharing tree. The complexity of the construction is linear in the number of nodes of S_φ, i.e., potentially logarithmic in the number of its elements.

Satisfaction wrt. a tuple. Checking that $\boldsymbol{t}\models\varphi$ on the sharing tree S_φ has cost linear in the size of φ, i.e., following from Remark 1, possibly logarithmic in the size of φ. In fact, the following theorem holds.

Theorem 2. Let S be a k-sharing tree and $\boldsymbol{t}$ be a vector of length k. We can check if $\boldsymbol{t}$ is subsumed by S in time linear in the number of edges of S.

Subsumption. The subsumption problem is harder: the best possible algorithm for subsumption is exponential in the size of the trees, as shown by the following theorem.

Theorem 3. The subsumption problem for two (irredundant) k-sharing trees is co-NP complete in the size of the sharing trees.

Following from the previous result, the cost of checking subsumption may be exponential in the number of edges of the input sharing trees. The result follows from the fact that $\mathcal{U}$-formulas in disjunctive form can be represented compactly via sharing-tress.

Reduction in normal form. Let S_φ be the sharing tree associated to a disjunctive formula φ. We consider now the following problem: what is the complexity of computing the sharing-tree for the normal form of φ (i.e. the sharing-tree S such that $elem(S) = min(\varphi)$)? The following theorem shows that it is as hard as checking subsumption.

Theorem 4. Given a k-sharing tree S, computing the irredundant k-sharing tree S' such that $[\![S]\!] = [\![S']\!]$ is co-NP hard.

Let S_1 and S_2 be two k-sharing trees. Note that, if $elem(S_1) \subseteq elem(S_2)$ then $[\![S_1]\!] \subseteq [\![S_2]\!]$. Besides giving a sufficient condition for checking subsumption, the previous fact suggests a possible strategy to reduce the cost of the 'complete' test. We first compute $T = minus(S_1, S_2)$ (polynomial in the size of S_1, S_2) and then test $T \models S_2$ on the (possibly) smaller sharing tree T.

In the next section we give more interesting polynomial-time *sufficient conditions* for the subsumption test, based on a notion of *simulation* between nodes of k-sharing trees. We will see that this notion of simulation is also useful to reduce sharing-trees and "approximate" the reduction in normal form.

4 Simulations for Nodes of a k-Sharing Tree

In the previous section we have proved that the subsumption problem for two $\mathcal{U}$-formulas represented as sharing-trees and the computations of generators of the normal form of a $\mathcal{U}$-formula represented as a sharing-tree, are co-NP hard. In this section we will introduce 'approximations' of the subsumption relation that can be tested more efficiently. More precisely, given two nodes n and m of a sharing tree S we are looking for a relation $\curvearrowright_F$ such that: $n \curvearrowright_F m$ 'implies' $[\![S_n]\!] \subseteq [\![S_m]\!]$.

Definition 1 (Forward Simulation). Given two sharing tree S and T, let n be a node of the i-th layer of S, and m be a node of the i-th layer of T. We say that n is *simulated by* m, written $n \curvearrowright_F m$, if $val^S(n) \geq val^T(m)$ and for all $s \in succ^S(n)$ there exists $t \in succ^T(m)$ such that $s \curvearrowright_F t$.

Note that, if $S = T$ then the simulation relation is *reflexive* and *transitive*.

Let $father(n)$ be the set of fathers of a node n at layer i ($fathers(n) \subseteq N_{i-1}$). We define the backward simulation as follows:

Definition 2 (Backward simulation). Given two sharing tree S and T, let n be a node of the i-th layer of S, and m be a node of the i-th layer of T. We say that n is *backwards simulated by* m, written $n \curvearrowright_B m$, if $val^S(n) \geq val^T(m)$ and for all $s \in fathers^S(n)$ there exists $t \in fathers^T(m)$ such that $s \curvearrowright_B t$.

The following result (taken from [HHK95]) shows that the previous simulations can be computed efficiently.

Theorem 5 (From [HHK95]). The forward and backward simulation relations between the nodes of the sharing tree S and the nodes of the sharing tree T can be computed in $O(m \cdot n)$ where m is the sum of the number of nodes in S and in T, and n is the sum of the number of edges in S and in T.

In the rest of this section we will focus on properties and algorithms for the forward simulation. The results and algorithms can be reformulated for the backward simulations by replacing the successor relation with the father relation.

4.1 Properties of the Simulation

The following propositions relate subsumption and the simulation $\curvearrowright^F$.

Lemma 1. Given the sharing trees S and T, let S_n and T_m be the sub-sharing trees rooted at nodes n and m, respectively. If $n \curvearrowright^F m$ then $[\![S_n]\!] \subseteq [\![S_m]\!]$.

The converse does not hold (in accord with the co-NP hardness result for subsumption). As a counterexample, take the two trees in Fig. 3. The curly arrows represent the simulation relation between nodes of S and T. Note that none of the nodes of layer 2 in T simulates the single node of S at the same layer. However, the denotation of S are contained in that of T. In fact, $\langle 1,1,2,0\rangle \preccurlyeq \langle 1,2,2,1\rangle$ and $\langle 1,0,0,2\rangle \preccurlyeq \langle 1,2,1,2\rangle$. The following theorem follows from Lemma 1.

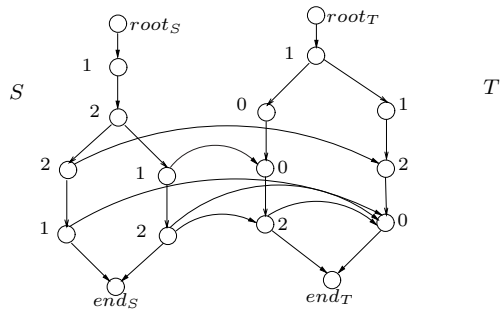

Fig. 3. The forward simulation is incomplete wrt. subsumption

Theorem 6. Let $root_S, end_S$ and $root_T, end_T$ be the root and terminal nodes of S and T, respectively. If $root_S \curvearrowright^F root_T$ then $[\![S]\!] \subseteq [\![T]\!]$. Symmetrically, if $end_S \curvearrowright^B end_T$ then $[\![S]\!] \subseteq [\![T]\!]$.

The previous theorem gives us sufficient conditions for testing subsumption.

4.2 Use of Simulations to Remove Redundancies

As for the subsumption test, the simulations we introduced in Section 4 can be used to 'approximate' the exact normalization procedure. For this purpose, we introduce a rule that allows us to exploit the information given from (one of) the simulation relation(s) in order to 'locally' remove edges of a sharing tree.

Definition 3 (Edge Removal). Given a sharing tree S with node N and successors $succ$, let us assume that for $n \in N$ there exist $s, t \in succ(n)$ $(s \neq t)$ such that $s \curvearrowright^F t$. Then, we define $remove(S, n)$ as the pre-sharing tree with successor relation $succ'$ obtained from S by setting $succ'(n) = succ(n) \setminus \{s\}$.

The following proposition states the 'correctness' of the previous rule.

Proposition 2. (1) S and $remove(S, n)$ have the same denotations, i.e., $[\![S]\!] \equiv [\![remove(S, n)]\!]$; (2) the simulation relation $\curvearrowright^F$ for S and $remove(S, n)$ coincides.

A possible strategy to apply Def. 3 consists of the 'on-the-fly' removal of edges during the computation of the simulation relation. Specifically, during a bottom-up traversal of the sharing tree, we first apply Rule 3 to every node of a layer, then compute the simulation relation for the nodes of the same layer, move to the next layer, and so on. The rule to remove edges is applied exhaustively at each step. In fact, given $s \in succ(n)$, let us assume that there exists $u, t \in succ(n)$ such that $u \curvearrowright^F s$, and $s \curvearrowright^F t$. By transitivity, $u \curvearrowright^F t$ holds, as well, i.e., we can still remove u after having removed s. The pre-sharing tree $remove(S, n)$ may violate the condition 6 of the definition of sharing trees (Section 3). However, as already mentioned in the course of the paper, condition 6 can be restored using an algorithm proposed in [ZL94]. Similar algorithms can be defined for the *backward* simulation. It is important to note that, though an application of Rule 3 does not change the forward simulation (Fact (2) of Prop. 2), it may change the backward simulation (and, vice versa, the removal of edges according to the backward relation may change the forward simulation). As a consequence, we get better and better results iterating the application of the algorithm for removing edges for the backward-forward simulations. A simplified version of Rule 3 that requires only a local test for every node of a sharing tree is given as follows.

Definition 4 (Local Edge Removal). Given a sharing S tree with node N and successors $succ$, let assume that for $n \in N$ there exist $s, t \in succ(n)$ $(s \neq t)$ such that $val(s) \geq val(t)$ and $succ(s) \subseteq succ(t)$. Then, we define $local\_remove(S, n)$ as the pre-sharing tree with successor relation $succ'$ obtained from S by setting $succ'(n) = succ(n) \setminus \{s\}$.

Though less effective than Rule 3, Rule 4 can be paired with it in order to simplify the computation of the simulation. In the following section we show how to incorporate the previous ideas in a model checking procedure for an example of integer system.

5 Invariant Checking for Vector Addition Systems

A Vector Addition System (VAS) consists of n variables $x_1, \ldots, x_n$ ranging over positive integers, and m transition rules given as guarded command over the data variables. For every j, transition i contains a guard $x_j \geq c_{i,j}$ and an assignment $x_j := x_j + d_{i,j}$; if $d_{i,j} < 0$ then $c_{i,j} \geq d_{i,j}$. States are tuples of *positive* numbers and executions are sequences of tuples $\boldsymbol{t}_0\boldsymbol{t}_1 \ldots \boldsymbol{t}_i \ldots$ where $\boldsymbol{t}_{i+1}$ is obtained from $\boldsymbol{t}_i$ by applying (non-deterministically) one of the transition rules. The predecessor operator *pre* takes as input a set of states (tuples) F and returns the set of predecessors of F. Properties like *mutual exclusion* and *coverability* can be represented through upward-closed sets [AČJT96,DEP99]. Checking safety properties expressed as upward-closed set for Petri Nets is decidable using the following algorithm taken from [AČJT96]. Let F be an upward-closed set (denoting unsafe states). To test the safety property 'always $\neg(F)$' we compute symbolically the closure of the relation *pre*, say $pre^*(F)$, and then we check that the initial configuration is not part of the resulting set. From [AČJT96], $pre^*(F)$ is still an upward-closed set. The termination test is based on the containment relation, namely we stop the computation whenever $pre^{n+1}(F) \subseteq \bigcup_{i=0}^{n} pre^i(F)$.

U-Logic based Model Checking. Let φ_U a $\mathcal{U}$-formula representing a collection of upward-closed set U. The predecessor relation for VAS can be represented as the following $\mathcal{U}$-formula:

$$pre(\varphi_U) = \bigvee_{i=1,\ldots,m} (\varphi_i \quad \wedge \quad \varphi_U[x_1 + d_{i,1}/x_1] \ldots [x_k + d_{i,k}/x_k])$$

where $\varphi_i = x_1 \geq c_{i,1} \wedge \ldots \wedge x_n \geq c_{i,n}$. In other words, by using the results in the previous sections, starting from S_{φ_U} we can compute the sharing tree S_{pre^i} that represents $\bigvee_{k=0}^{i} pre^k(U)$. The termination test is implemented by the subsumption test for sharing trees. The algorithms based on the simulations that we described in this paper can be used for a weaker termination test and for removing redundancies from the intermediate results. We have define a prototype implementation for the algorithm using the library of [Zam97]. We discuss next some preliminary experimental results.

5.1 A Case-Study: The Table Manufacturing System

The *table manufacturing system* of Teruel [Ter94] is a production system modeled via a *weighted Petri Net* with 7 places and 6 transitions. The values of the weights on the transitions vary from 2 to 4. This Petri Net can be modeled using a VAS with 13 variables, 7 for the places and 6 extra variables, say $T_1, \ldots, T_6$, to keep track of the number of firings of each transition. As shown in [Ter94], all deadlock-free initial markings are exactly the predecessors of the set of states where the first transition is fired at least 3 times, and all the others are fired at least twice (i.e. $U = T_1 \geq 3 \ \wedge \ \bigwedge_{j>1} T_j \geq 2$). In [DEP99], different general purpose constraint-based checkers have been used to compute the set of predecessors via backward reachability. While other tools fail from terminate within

Example	NV	NR	NS	LR	FSR	ET	NE	Nodes	Ratio
Man. Sys.	13	6	24	no	_	39s	7563	4420	4%
Man. Sys.	13	6	24	yes	5	68s	727	1772	12%

Fig. 4. Flags: NV=No. Variable; NR=No. Rules; ET=Execution Time; LR=Use of local reduction; FSR=Frequency in the use of sim. reduction (_=not used); NS=No Steps; NE=No. of Elem. of the result sharing tree; Ratio=Nodes/(NV*NE)

1 day of execution, the tool based on the Omega library of Bultan, Gerber and Pugh's [BGP97] terminates after 19 hours and 50 minutes on a Sun Ultra Sparc, and the 'specialized' checker proposed in [DEP99] (see Sect. 6) terminates after 1090 seconds. Sharing trees allows us to dramatically speed up the computation. Our prototype implementation (based on the library of [Zam97]) terminates after 39 seconds on a Sun Ultra Sparc (see Fig. 4). In a first experiment we have not removed the redundancies from S_{pre^i}, whereas in a second experiment we have applied the reduction based on the forward simulation (every 5 steps) (see Fig 4). Simulation-based reduction (every 5 steps) allows us to reduce the set of states of a factor of ten (note: removing *all* redundancies yields 450 elements). Other examples are considered in the extended version of this paper [DR99].

6 Related Work

In [AČJT96], the authors introduce a symbolic representation (*constraint* system) for collections of upward-closed sets. Their representation corresponds to disjunctive $\mathcal{U}$-formulas. Traditional symbolic methods for handling linear constraints (e.g. polyhedra or Presburger arithmetics) suffer however from the state-explosion problem when applied to this type of 'constraints'. In [DEP99], a more efficient representation based on sequences of pairs *bitvector-constant* is proposed for representing the state-space of broadcast protocols, and, as special case, of VAS. In this paper we have shown how to obtain more compact and efficient representations via sharing trees.

In [Zam97,GGZ95], the authors apply sharing trees to represent the state-space of concurrent systems: a *state* is a *tuple* of values and a *set* of states is represented as a *sharing tree*. Note the difference with our approach. We represent a *set of states* via a *tuple*, and *collections of sets* of states via a *sharing tree*. The complexity issues are different when lifting the denotation to collections of sets of states (see Section 3). In [Zam97], Zampuniéris makes an accurate comparison between sharing trees and *binary decision diagrams* (BDDs) [Bry86]. When the aim is to represent tuples of (unbounded) integers (as in our case), the layered structure of sharing trees allows optimizations that seem more difficult using BDDs (or extensions like *multi-valued* DDs [SKM+90] or *multi-terminal* DDs [CFZ96]).

Our approach shares some similarities with recent works on *interval decision diagrams* (IDDs) [ST99] and *clock decision diagrams* (CDDs) for timed automata [BLP+99]: all approaches make use of *acyclic* graphs to represent disjunctions of interval constraints. However, the use of simulations as abstractions for handling efficiently large disjunctions has not been considered in the other approaches. More experimentations are needed for a better comparison of all those methods. Finally, the PEP tool [Gra97] provides a BDD-based model checking method for Petri Nets (with a fixed-a-priori number of tokens) [Wim97]. We are not aware of BDDs-based representations for the 'constraints' we are interested in, e.g., for verification problems of Petri Nets with a possibly unbounded number of tokens.

7 Conclusions and Future Work

We have proposed a new symbolic representation for 'constraints', we called $\mathcal{U}$-formulas, that can be used in verification problems for infinite-state integer systems (e.g., coverability of Petri Nets). The representation is based on the sharing trees of Zampuniéris and Le Charlier. For our purposes, we lift the denotation of a sharing tree to the upward-closed set 'generated' by the tuples contained in the sharing tree. We have studied the theoretical complexity of the operations for sharing trees wrt. this denotation. Furthermore, we have given sufficient conditions for testing subsumption (co-NP hard for $\mathcal{U}$-formulas) we discover thanks to the view of $\mathcal{U}$-formulas as acyclic graphs. In fact, the conditions are based on simulations relations for nodes of sharing trees.

Though the termination test for the algorithm of [AČJT96] applied to collections of upward-closed sets ($\sim \mathcal{U}$-formulas $\sim$ sharing trees) may be very costly[3], testing for membership of the initial configuration (when it can be expressed with a conjunctive formula) can be done efficiently (Theorem 2). This gives us an effienct method to detect *violations* of safety properties.

The implementation is currently being optimized, but the preliminary experimental results are already promising. The type of optimizations we are interested in are: heuristics for finding 'good' orderings of variables; symbolic representation of the transition system (e.g. PADs [ST99]); partial order reductions (see e.g. [AJKP98] for an application to the coverability problem of Petri Nets). Finally, it would be interesting to extend our techniques to more general classes of constraints.

Acknowledgments. The authors would like to thank Jean-Marc Talbot, Andreas Podelski, and Witold Charatonik for fruitful discussions, and Denis Zampuniéris for having made the sharing tree library available for our experiments.

[3] Quadratic for disjunctive formulas, but disjunctive formulas suffers from the state explosion; exponential for sharing trees or arbitrary $\mathcal{U}$-formulas.

References

AČJT96. P. A. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay. General Decidability Theorems for Infinite-state Systems. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science (LICS'96)*, pages 313–321. IEEE Computer Society Press, 1996. 426, 427, 428, 436, 437, 438

AJKP98. P. A. Abdulla, B. Jonsson, M. Kindahl, and D. Peled. A General Approach to Partial Order Reductions in Symbolic Verification. In *Proceedings of the 10th Conference on Computer Aided Verification (CAV'98)*, volume 1427 of *LNCS*, pages 379–390. Springer, 1998. 426, 438

BLP+99. G. Behrmann, K. G. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient Timed Reachability Analysis Using Clock Difference Diagrams. In *Proceedings of the 11th Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *LNCS*, pages 341–353. Springer 1999. 438

BW98. B. Boigelot, P. Wolper. Verifying systems with infinite but regular state space. In *Proceedings of the 10th International Conference on Computer Aided Verification (CAV'98)*, volume 1427 of LNCS, pages 88–97. Springer, 1998. 426

BM99. A. Bouajjani and R. Mayr. Model Checking Lossy Vector Addition Systems. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS'99)*, volume 1563 of LNCS, pages 323–333. Springer, 1999. 426

Bry86. R. E. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transaction on Computers*, C-35(8):667-691, August, 1986. 437

BGP97. T. Bultan, R. Gerber, and W. Pugh. Symbolic Model Checking of Infinite-state Systems using Presburger Arithmetics. In *Proc. 9th Conf. on Computer Aided Verification (CAV'97)*, LNCS 1254, pages 400–411. Springer-Verlag, 1997. 427, 437

BCB+90. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of the 5th IEEE Symposium on Logic in Computer Science*, pages 428-439. IEEE Computer Society Press, 1990. 426, 427

Čer94. K. Čerāns. Deciding Properties of Integral Relational Automata. In *Proceedings of the International Conferences on Automata and Languages for Programming (ICALP 94)*, volume 820 of LNCS, pages 35-46. Springer, 1994. 426

Cia94. G. Ciardo. Petri nets with marking-dependent arc multiplicity: properties and analysis. In *Proceedings of the 15th Int. Conf. on Application and Theory of Petri Nets 1994*, LNCS 815, pages 179–198. Springer-Verlag, 1994. 426

CFZ96. E. Clarke, M. Fujita, and X. Zhao. Multi-terminal Binary Decision Diagrams and Hybrid Decision Diagrams. In *Representations of Discrete Functions*, pages 93-108. Kluwer Academic Publishers, 1996. 437

DEP99. G. Delzanno, J. Esparza, and A. Podelski. Constraint-based Analysis of Broadcast Protocols. In *Proceedings of the Annual Conference of the European Association for Computer Science Logic (CSL'99)*, volume 1683 of LNCS, pag. 50–66. Springer, 1999. 426, 427, 436, 437

DP99. G. Delzanno and A. Podelski, Model Checking in CLP. In W. R. Cleaveland, editor, *Proc. 5th Int. Con. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99).*, LNCS 1579, pages 223–239. Springer-Verlag, 1999.

DR99. G. Delzanno and J. F. Raskin. Symbolic Representation of Upward-Closed Sets. Technical report MPI-I-1999-2-007, Max-Planck-Institut für Informatik, November 1999. Available on the web at http://www.mpi-sb.mpg.de/units/ag2/. 428, 437

EN98. E. A. Emerson and K. S. Namjoshi. On Model Checking for Non-deterministic Infinite-state Systems. In *Proceedings of the 13th Annual Symposium on Logic in Computer Science (LICS '98)*, pages 70–80. IEEE Computer Society Press, 1998. 426

EFM99. J. Esparza, A. Finkel, and R. Mayr. On the Verification of Broadcast Protocols. In *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, pages 352–359. IEEE Computer Society Press, 1999. 426

Fin90. A. Finkel. Reduction and Covering of Infinite Reachability Trees. *Information and Computation* 89(2), pages 144–179. Academic Press, 1990. 426

FS99. A. Finkel and P. Schnoebelen. Well-structured Transition Systems Everywhere! *Theoretical Computer Science*, 1999. To appear. 426, 427, 428

GGZ95. F. Gagnon, J.-Ch. Grégoire, and D. Zampuniéris. Sharing Trees for 'On-the-fly' Verification. In *Proceedings of the International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE'95)*, 1995. 437

Gra97. B. Grahlmann. The PEP Tool. In *Proceedings of the 9th Conference on Computer Aided Verification (CAV'97)*, volume 1254 of LNCS, pages 440–443. Springer, 1997. 438

HHK95. M. R. Henzinger, T. A. Henzinger, P. K. Kopke. Computing Simulations on Finite and Infinite Graphs. In *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS'95)*, pages 453–462. IEEE Society Press, 1995. 434

HHW97. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: a Model Checker for Hybrid Systems. In *Proceedings of the 9th Conference on Computer Aided Verification (CAV'97)*, volume 1254 of *LNCS*, pages 460–463. Springer, 1997. 427

McM93. K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* Kluwer Academic, 1993.

Min67. N. M. Minsky. Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, N.Y., 1967. 426

SKM+90. A. Srinivasan, T. Kam, S. Malik, and R. K. Brayton. Algorithms for Discrete Functions Manipulation. In *Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD'90)*, 1990. 437

ST99. K. Strehl, L. Thiele. Interval Diagram Techniques For Symbolic Model Checking of Petri Nets. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE'99)*, pages 756–757, 1999. 438

Ter94. E. Teruel. *Structure Theory of Weighted Place/Transition Net Systems: The Equal Conflict Hiatus.* Ph.D. Thesis, University of Zaragoza, 1994. 426, 427, 436

Wim97. G. Wimmel. A BDD-based Model Checker for the PEP Tool. Technical Report, University of Newcastle Upon Tyne, 1997. 438

Zam97. D. Zampuniéris. The Sharing Tree Data Structure: Theory and Applications in Formal Verification. PhD Thesis. Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, May 1997. 427, 436, 437

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. 427, 429, 430, 431, 432, 435