Symbolic Representation of Upwards Closed Sets

Giorgio Delzanno[1] and Jean-Francois 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 reachability problem for a wide class of infinite-state systems is decidable when the initial and the final set of configurations are given as upwards closed sets. Traditional symbolic model checking methods suffer from the state explosion problem when applied to this class of verification problems.
We provide new data structures and algorithms for an efficient manipulation of upwards closed sets. These operations can be incorporated into model checking procedures for integer systems with infinite-states space. We report on experiments for verification problems of Vector Addition Systems.

1 Introduction

Symbolic model checking [BCB+90] has been successfully applied to verification of finite-state systems. This approach is based on the efficient representation of sets of states via *binary decision diagrams* (BDDs) [Bry86]. In the last years, many efforts have been made to extend the results and methods developed for finite-state systems to systems with *infinite-state* space.

Many interesting theoretical results have been obtained for systems with variables ranging over integer values (e.g. [AČJT96,AJ99,BF99,BM99,BW94,BW98,CJ98,EFM99,FS99]). This class of systems comprises well-known examples like Vector Addition Systems [Min67], Petri Nets [KM69], Integral Relational Automata [Čer94], and more recent examples like Broadcast Protocols [EN98] and Lossy Petri Nets [BM99].

The reachability problem for all these systems is decidable when the initial set of configurations is *upwards closed* [AČJT96,Fin90,FS99]. Properties like *mutual exclusion* and *coverability* can be expressed via upwards closed sets (see e.g. [AJKP98,DEP99]). As for the finite state case, the success of symbolic model checking for this class of problems seems to depend on the representation chosen for implicitly represent sets of states.

Upwards closed sets can be expressed via a sub-class of integer arithmetic constraints (see e.g. [AJ99,DEP99]). In [DEP99], the authors tested 'traditional' symbolic model checking methods for arithmetic constraints (e.g., Presburger arithmetics [Bul98] and polyhedra [HHW97]) on verification problems that can be expressed via upwards closed sets. All methods suffer from the state explosion problem[1]. 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]), some of the reported execution times were given in days.

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

Based on these observations, it seems natural to look for formulation of BDD-like data structures to represent the 'generalization' of boolean formulas we are interested in. Since upwards closed sets are univocally determined by their minimal elements, the data structure we are looking for should provide for compact representations of possibly huge collections of *tuples*.

In this paper we propose a new symbolic representation for upwards closed sets based on the *sharing trees* of Zampuniéris and Le Charlier [ZL94]. A sharing tree is a directed *acyclic graph* used to represent a set of tuples so as to obtain the maximal sharing of prefixes and suffixes of its elements. In our approach, we lift the denotation of a sharing tree from its set of elements (as in [ZL94]) to the upwards closed sets 'generated' by its elements. The view of *formulas as sharing trees* allowed us to study abstractions based on *simulation* relations usually considered for *transition systems* (see, e.g., [HHK95]).
Technically, our contributions are as follows.

- We introduce a logic (the logic $\mathcal{U}$) that provides connectives to represent union, intersection, and translations of upwards closed sets, i.e., it can be used to express the predecessor operators, e.g., of Vector Addition Systems.
- We show that sharing trees can be used to obtain compact representations of $\mathcal{U}$-formulas: there exist $\mathcal{U}$-formulas that can be represented using sharing trees logarithmic in the size of the formulas.
- We show how to compute operations for $\mathcal{U}$-formulas on the corresponding symbolic representations. In general, the operations can be implemented efficiently (time linear in the size of the sharing tree).
- The subsumption test for arbitrary $\mathcal{U}$-formulas turns out to be co-NP hard. We define a notion of *simulations* between nodes of sharing trees and use it to give polynomial-time sufficient conditions for the subsumption test. We show that the simulations can also be used to remove redundancies (from a sharing tree) in efficient way.

In other words, we set up the stage for a symbolic model checking algorithm for safety properties expressed as upward closed sets. As an application of our method, we give a symbolic model checking algorithm for Vector Addition Systems based on transformations of sharing trees. We report on some experimental results we obtained with a prototype implementation based on the sharing tree library of [Zam97].

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 simulations 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 7, we address conclusions and future works.

2 The Logic of Upwards Closed Sets

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

$$\Phi \quad ::= \quad x_i \geq c \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \exists_{x_i + c}.\ \Phi,$$

where $c \in \mathbb{Z}_\omega$ and $\mathbb{Z}_\omega = \mathbb{Z} \cup \{-\infty, +\infty\}$. $\mathcal{U}$-formulas are interpreted over $\mathbb{Z}_\omega^k$. We use $\boldsymbol{t}$ to denote the valuation $\langle t_1, \ldots, t_k \rangle$, where $t_i \in \mathbb{Z}_\omega$ 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 $i : 1, \ldots, k$ ($-\infty \leq c \leq +\infty$ for $c \in \mathbb{Z}_\omega$). When restricting to positive values, $\preccurlyeq$ is a well-quasi ordering (see e.g. [AČJT96,FS99]). Given a tuple t, we define $\boldsymbol{t}^\uparrow$ as the *upwards closed set* generated by $\boldsymbol{t}$, namely, $\boldsymbol{t}^\uparrow = \{\boldsymbol{t'} \mid \boldsymbol{t} \preccurlyeq \boldsymbol{t'}\}$.
Satisfiability 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 $\boldsymbol{t} \models \Phi_2$;
– $\boldsymbol{t} \models \exists_{x_i+c}.\ \Phi$ where $\boldsymbol{t'} \models \Phi$ and $\boldsymbol{t'}$ is obtained from $\boldsymbol{t}$ replacing t_i with $t_i + c$.

The formula $\exists_{x+c}.\Phi[x]$ corresponds to the formula with explicit equality $\exists_{x'}.x' = x + c \wedge \Phi[x'/x]$. We will use it to represent transformations of $\mathcal{U}$-formulas.

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.

The class of $\mathcal{U}$-formulas denotes all upwards closed set over $\mathbb{Z}_\omega^k$, i.e., all sets $I \subseteq \mathbb{Z}_\omega^k$ 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: $\bigvee_{i \in I}(x_1 \geq c_{i,1} \wedge \ldots \wedge x_k \geq c_{i,k})$[2].

Notation 1 *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

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

Thus, disjunctive formulas are in one-to-one correspondence with their set of generators. 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. A graphical representation of the upwards closed set generated by φ is given in Fig. 1.

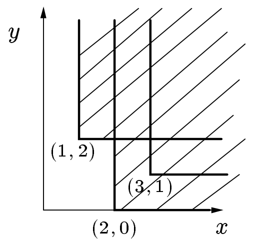

Fig. 1. Generators and minimal points.

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., $gen(\varphi \wedge \psi) = \{\boldsymbol{s} \mid \exists \boldsymbol{t} \in gen(\varphi), \boldsymbol{t'} \in gen(\psi) \text{ and } s_i = max(t_i, t'_i)\}$. Note that the resulting formula may not be in normal form.

Quantification. The formula $\exists_{x_i+c}.\varphi$ is equivalent to the formula φ' obtained from φ by replacing each atom $x_i \geq d$ with $x_i \geq d - c$, i.e., $gen(\exists_{x_i+c}.\varphi) = \{\boldsymbol{t'} \mid t'_i + c = t_i,\ t'_j = t_j\ j \neq i,\ \boldsymbol{t} \in gen(\varphi)\}$.

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

Satisfiability. 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 hard in the size of the formulas.

Proof. We use a reduction from the *validity* problem (see e.g. [Joh90]) for propositional formulas in disjunctive normal form. Given a propositional formula $F = D_1 \vee \ldots \vee D_n$ where each disjunct D_i is a conjunction of literals over the set of propositional variables $p_1, \ldots, p_m$, is F true for all valuations?

The reduction works as follows. Given a propositional formula F, we build a $\mathcal{U}$-formula $\Phi_{val(F)}$ (polynomial in the size of F) that represents all valuations that make F true. Then, we reduce validity for F to the subsumption problem $\Phi_{val(true)} \models \Phi_{val(F)}$.

Without loss of generality, we assume that *all variables* occur in each disjunct of F either as p, $\neg p$ or $any(p)$. The latter means that p can take any value (note: it is equivalent to say that p *does not* occur in the disjunct). The formula $true$ is equivalent to $any(p_1) \wedge \ldots \wedge any(p_m)$. To encode the valuation for p_i, we use two variables from V, namely v_i and w_i. We use the formula $\Phi_1 = (v_i \geq 1 \wedge w_i \geq 0)$ when p_i evaluates to $true$; $\Phi_2 = (v_i \geq 0 \wedge w_i \geq 1)$ when p_i evaluates to $false$. Note that neither $\Phi_1 \models \Phi_2$ nor $\Phi_2 \models \Phi_1$. We use the disjunction $(v_i \geq 1 \wedge w_i \geq 0) \vee (v_i \geq 0 \wedge w_i \geq 1)$ when p_i can be evaluated either to $true$ or $false$. The previous encoding allows us to keep the size of $\Phi_{val(F)}$ polynomial in the size of F.

Formally, the formula $\Phi_{val(F)}$ is defined by induction on F as follows: $\Phi_{val(D_1 \vee D_2)} = \Phi_{val(D_1)} \vee \Phi_{val(D_2)}$, $\Phi_{val(C_1 \wedge C_2)} = \Phi_{val(C_1)} \wedge \Phi_{val(C_2)}$, $\Phi_{val(p_i)} = (v_i \geq 1 \wedge w_i \geq 0)$, $\Phi_{val(\neg p_i)} = (v_i \geq 0 \wedge w_i \geq 1)$, $\Phi_{val(any(p_i))} = (v_1 \geq 0 \wedge w_1 \geq 1) \vee (v_1 \geq 1 \wedge w_1 \geq 0)$.
Based on this construction, it is easy to check that F is valid if and only if $\Phi_{val(true)} \models \Phi_{val(F)}$.
□

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 $\boldsymbol{t'} \in gen(\varphi)$, $t \neq t'$, $\boldsymbol{t'} \preccurlyeq \boldsymbol{t}$. This 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 set 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 $N_{k+1} = \{end\}$), $val : N \rightsquigarrow \mathbb{Z}_\omega \cup \{\top, \bot\}$ 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 $succ(n) \neq \emptyset$;

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

Fig. 2. Operations on the sharing trees S and T: $edges(S)$=No.edges of S.

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).

In teh rest ofthe 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 tuple 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 ensures the maximal sharing of prefixes and suffixes among the tuples represented by 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.

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. Table 2 gives the specification 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, emptyness, containment, and equality test. The cost for intersection and union depend also from the cost (quadratic in the number of nodes) of re-arranging condition 6 (denote by *Red* in Table 2) using the algorithm shown in [ZL94].

Finally, 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 $elem(S)$.

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 $\boldsymbol{t}$ is subsumed by $minus(S_\varphi, \boldsymbol{t})$. 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.*

Proof. Consider the $\mathcal{U}$-formulas $\Phi_{val(D)}$ we used in the proof of Theorem 1 to represent all evaluations that satisfy a disjunct D of a DNF propositional formula F. The transformation of $\Phi_{val(D)}$ to a disjunctive formula may give a formula φ exponential in the size of $\Phi_{val(D)}$. However, the representation of φ using a sharing tree S_φ is polynomial in the size of the *original* formula $\Phi_{val(D)}$, i.e., logarithmic in φ. In other words, we can build S_φ 'directly' from $\Phi_{val(D)}$ (the formal proof is by induction on the structure of the formula).

As an example, $\Phi_{val(true)} = \bigvee_{i=1}^{m}(v_i \geq 1 \wedge w_i \geq 0) \vee (v_i \geq 0 \wedge w_i \geq 1)$. The corresponding disjunctive formulas φ (obtained by distributing $\wedge$) is $\bigvee_{c,d \in \{0,1\}} \bigwedge_{i=1}^{m} (v_i \geq c \wedge w_i \geq d)$, i.e., one conjunct for each possible valuation for the variables $p_1, \ldots, p_m$ of F (exponential in m). Note that φ is in normal form. On the other hand, the sharing tree S_φ has size polynomial in $\Phi_{val(true)}$, i.e., logarithmic in ψ, as shown in Fig. 3 (each layer has two nodes and at most four arcs).

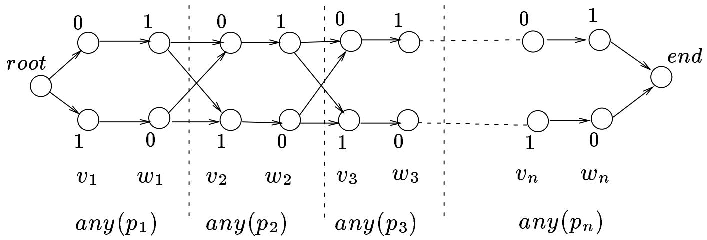

Fig. 3. Sharing tree for $\Phi_{val(true)}$.

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

We show in this subsection how operations on disjunctive formulas φ and ψ can be done symbolically on the sharing-trees representations $S_\varphi, S\psi$. We use the term *symbolically* because the algorithms that we propose work directly on the graphs S_φ and $S\psi$ and not by enumerating the elements that they represent.

Disjunction. Let S_φ and S_ψ be the k-sharing trees representing the formulas (in disjunctive form) φ and ψ. We must construct a sharing tree such that $elem(S_{\varphi\vee\psi}) = gen(\varphi) \cup gen(\psi)$, so we simply have $S_{\varphi\vee\psi} = union(S_\varphi, S_\psi)$. It can be shown 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 those two sharing-trees.

Conjunction. Given S_φ and S_ψ, we construct symbolically $S_{\varphi\wedge\psi}$ as follows:

- we first construct a pre-sharing tree P with the following elements: (i) $N^P = \{root\} \cup N_1 \cup \ldots \mathcal{N}_k \cup \{end\}$ with each $N_i = \{(n,m) \mid n \in N_i^{S_\varphi}, m \in N_i^{S_\psi}\}$ (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 $(n,m) \in N_1 \cup \ldots N_n$, we have $succ^P((n,m)) = \{(n',m') \mid n' \in succ^{S_\varphi}(n), m' \in succ^{S_\psi}(m)\}$, $succ^P(root) = N_1$, and for all $(n,m) \in N_k$ we have $succ^P((n,m)) = \{end\}$.
- we obtain the sharing-tree $S_{\varphi\wedge\psi}$ from P by inforcing in P the rules (5) and (6) of the definition of sharing-trees, with the algorithms proposed in [ZL94].

It is easy to show that $elem(S_{\varphi\wedge\psi}) = gen(\varphi \vee \psi)$.

Quantification. Given the sharing tree S_φ we can compute $S_{\exists_{x+c}\cdot\varphi}$ as follows: we take $S_{\exists_{x+c}\cdot\varphi}$ as S_φ with the exception of the valuation function which has the following definition: for every node $n \in N^{S_{\exists_{x+c}\cdot\varphi}}$ we have $val^{S_{\exists_{x+c}\cdot\varphi}}(n) = val^{S_\varphi}(n) - c$. This operations returns a well-formed sharing tree and has complexity linear in the number of nodes of S_φ. Again this operation is linear in the size of the sharing-tree and thus potentially logarithmic in the number of elements represented by this sharing-tree. Also it is easy to check that $elem(S_{\exists_{x+c}\cdot\varphi}) = gen(\exists_{x+c}\cdot\varphi)$.

Satisfiability. Checking that $\boldsymbol{t} \models \varphi$ on the sharing tree S_φ has cost polynomial 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.

Proof. We exhibit an algorithm linear in the size of the sharing tree. The algorithm is based on a layer-by-layer comparison of the nodes of the sharing-tree and the components of the vector: at a given layer, we mark all nodes that belong to a path (tuple) that is candidate to subsume the vector. Specifically, we first mark $root_S$. Then, if a node m at the i-th layer of S is marked, we mark all successors n of m such that $t_{i+1} \geq val(n)$. At the end of the procedure, the vector $\boldsymbol{t}$ is subsumed by S if and only if the node end_S is marked (note: we assume that $t_0 = \top$ and $t_{k+1} = \bot$). $\square$

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.

Proof. We first show that the problem is in co-NP. We can solve the complement of the subsumption problem, namely $[\![S_1]\!] \not\subseteq [\![S_2]\!]$, by guessing a tuple of length k and testing that it is subsumed by S_1 but not by S_2. Following from Prop. 2, we can check whether a tuple is subsumed or not by a sharing tree in time linear in the size of the tree.

To prove hardness, we use a reduction from *validity*. More precisely, given a propositional formula in DNF $F = D_1 \vee \ldots \vee D_n$ we define the sharing tree $S_{val(F)}$ that encode all evaluations that satisfy F and then reduce validity to the subsumption test between $S_{val(true)}$ and $S_{val(F)}$. Following the proof of Theorem 1, we first build the two $\mathcal{U}$-formulas $\Phi_{val(true)}$ and $\Phi_{val(F)}$ that encode the evaluations for *true* and F, respectively. Following from Prop. 1, we know that the sharing tree $T_{val(true)}$ (with root r_t and terminal node e_t) that encode $\Phi_{val(true)}$, and the sharing tree S_i (with root r_i and terminal node e_i) that encode $\Phi_{val(D_i)}$ have size polynomial in F. (Note: in Prop. 1, the sharing tree $S_{val(F)}$ is obtained from the expansion of $\Phi_{val(F)}$ to a disjunctive formula.) To define an 'irredundant' sharing tree $S_{val(F)}$ that encode all evaluations for F, we merge all S_i's into a single sharing tree with root r and terminal node e such that $succ(r) = \{r_1, \ldots, r_n\}$, $succ(e_1) = \{e\}, \ldots, succ(e_n) = \{e\}$. To ensure that $S_{val(F)}$ is irredundant, we first note that S_i is irredundant by construction. It remains to make all paths in S_i and S_j for $i \neq j$ non comparable. For this, we set $val(r_i) = i$ and $val(e_i) = 2n + 1 - i$ (i.e., from 'left-to-right': increasing values for the r_i's; decreasing values for the e_i's). Clearly, the construction is polynomial in the size of F.

Similarly, we $S_{val(true)}$ is the sharing tree obtained from $T_{val(true)}$ as the sharing tree with root r' and terminal node e' such that $succ(r') = \{r_t\}$ and $succ(e'_t) = \{e_t\}$. Finally, since the values of the nodes r' and e' should not influence the comparison of the paths of $S_{val(true)}$ and $S_{val(F)}$, we set $val(r') = val(e') = 2n + 1$ (a value strictly greater than the max label in $S_{val(F)}$). We give an example of the reduction in Fig. 4. $\square$

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

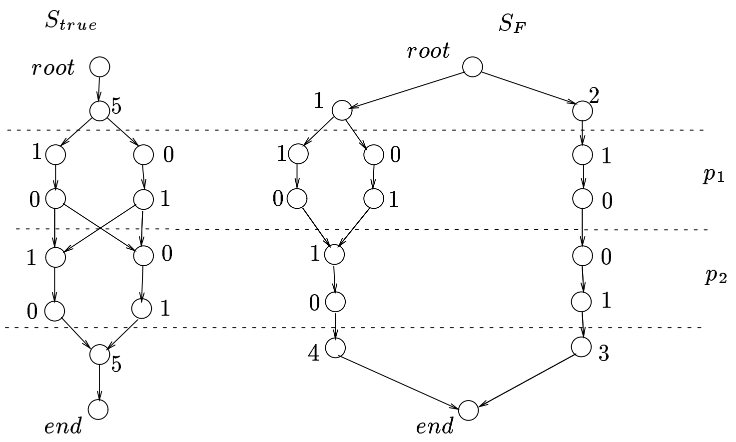

Fig. 4. Validity of $F = p_2 \vee (p_1 \wedge \neg p_2)$ is reduced to $S'_{val(true)} \models S_{val(F)}$.

Reduction in normal form. Let S_φ be the sharing tree associated to a disjunctive formula φ. We now characterize the following problem: what is the complexity of computing the sharing-tree for the normal form of φ ? That is 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.

Proof. Following [Joh90], we exhibit a polynomial-time Turing reduction from VALIDITY (co-NP complete). As in the proof of theorem 3, given a formula F we build the irredundant sharing tree $S_{val(F)}$ and $S_{val(true)}$ (with size polynomial in F). We build an Oracle Turing Machine (OTM) for testing validity of F as follows. Let us assume that we have an oracle that yields the sharing tree R obtained from S after removing all redundant tuples. Following the idea of the proof of Theorem 3, we first define the sharing tree S obtained merging $S_{val(true)}$ and $S_{val(F)}$) in time polynomial in the size of $S_{val(true)}$ and $S_{val(F)}$. Then, we invoke the oracle with S as input. By construction of S, the oracle returns the sharing tree R obtained from S by removing all tuples of $S_{val(true)}$ in S subsumed by tuples in S_F. Finally, we compute $T = intersect(S_{val(true)}, R)$ without adjusting the conditions 5-6 (i.e. we obtain a pre-sharing tree) in time polynomial in the minimum between the size of $S_{val(true)}$ and R (see Table 2). As shown in [ZL94], this step is possible and avoids the cost *Red* of Table 2. Finally, we check emptiness of the pre-sharing tree T in constant time (see Table 2). Clearly, T is empty if and only if F is valid. Thus, for every input formula F, the resulting OTM runs in polynomial time in the size of F and returns 'yes' if and only if F is valid. $\square$

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* bewteen 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). Let n, respectively m, be a node of the i-th layer of a k-sharing tree S, respectively of k-sharing tree 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). Let n, respectively m, be a node of the i-the layer of two k-sharing tree S, respectively of k-sharing tree 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 $s \curvearrowright^B t$.

Theorem 5 ([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 $[\![S_n]\!] \subseteq [\![S_m]\!]$.

Proof. We will show that for every node of S_n in layer $k-l$, for $0 \leq l \leq k$ the property is verified. We reason by induction on the value of l.

Base case: $l = 0$. Let us consider $n \in N^S_{k-l}$ and $m \in N^T_{k-l}$ such that $n \curvearrowright^F m$. By definition of sharing trees, we have that $succ^S(n) = \{end\}$, $succ^T(m) = \{end\}$, $elem(S_{end}) = \emptyset$, and $elem(T_{end}) = \emptyset$. Thus, $elem(S_n) = \langle val^S(n) \rangle$ and $elem(T_m) = \langle val^T(m) \rangle$, and, by definition of $\curvearrowright^F$, we have that $val^S(n) \geq val^T(m)$.

Induction step: $l > 0$. By induction hypothesis, for all nodes $n' \in N^S_{k-i}$ and $m' \in N^T_{k-i}$, with $O \leq i < l$, we know that if $n' \curvearrowright^F m'$ then $[\![S_{n'}]\!] \subseteq [\![S_{m'}]\!]$. Let us now consider $n \in N^S_{k-l}$ and $m \in N^T_{k-l}$ with $n \curvearrowright^F m$. By definition of $\curvearrowright^F$, we know that for $s \in succ^S(n)$ there exists $t \in succ^T(m)$ such that $s \curvearrowright^F t$. We also know that $val^S(n) \geq val^T(m)$. So $elem(S_n) = \{\langle val^S(n) \rangle \times elem(S_{n'}) \mid n' \in succ^S(n)\}$ and $elem(T_m) = \{\langle val^T(m) \rangle \times elem(T_{m'}) \mid n' \in succ^T(m)\}$, and thus we have $[\![S_n]\!] \subseteq [\![T_m]\!]$. □

The converse does not hold (in accord with the co-NP hardness result for subsumption). As a counterexample, take the two trees in Fig. 5. 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.

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

The last theorem gives us sufficient conditions for testing subsumption.

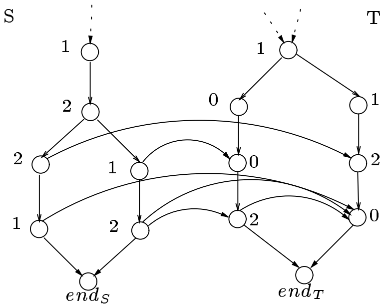

Fig. 5. The forward simulation is incomplete wrt. subsumption.

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 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 $s \curvearrowright^{F} t$. 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 property states the 'correctness' of the rule 3.

Proposition 2. (1) S and $remove(S, n)$ have the same denotations, i.e., $[\![S]\!] \equiv [\![remove(S, n)]\!]$;[3] (2) the simulation relation $\curvearrowright^{F}$ for S and $remove(S, n)$ coincides.

Proof. (1) By definition of $remove$. (2) Let $S' = remove(S, n)$. We first note that: (*) $succ(n)$ cannot become empty after removing the edge of n that satisfy the condition above: if we remove t from $succ(n)$ then there exists $t' \in succ'(n)$ s.t. $t \curvearrowright^{F} t'$. Now, per absurdum, let us assume that, for n, m in N, n is simulated by m in S_n but not in S'. By definition, there exists $s \in succ(n)$ that is simulated by $t \in succ(m)$, whereas, none of the nodes in $succ'(m)$ simulates s, i.e., t has been removed from $succ(m)$ after the application of the rule. Following from observation (*), there exists $t' \in succ'(m)$ such that $t \curvearrowright^{F} t'$. By transitivity, it follows that $s \curvearrowright^{F} t'$ contradicting the hypothesis. Vice versa, let us assume that for two nodes n, m in N n is not simulated by m at step in S_n but it is in S'. This means that we have removed a successor s of n that was not simulated by any of the successors of m. By definition, there exists $s' \in succ(n)$ s.t. $s \curvearrowright^{F} s'$. Following from (*), we can choose s' such that $s' \in succ'(n)$. Since we assume that $n \curvearrowright^{F} m$ in S', it follows that $s' \curvearrowright^{F} t$ for some $t \in succ'(m)$ and, by transitivity, $s \curvearrowright^{F} t$ in S_n contradicting the hypothesis. Thus, the simulation is invariant under application of $remove$. □

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 node of the later, and move to the next layer. 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.

Note that graph $remove(S)$ may violate condition 6 of the definition of sharing trees. For instance, removing the node with label 6 from the sharing tree in Fig. 6 makes the two nodes with label 2 (belonging to vectors that are not in the subsumption relation) have the same set of

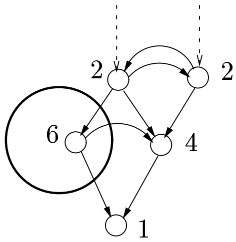

Fig. 6. Violation of cond. 6 after removing two edges and one node.

successors. As already mentioned in the course of the paper, condition 6 can be restored using an algorithm proposed [ZL94] linear in the number of edges and nodes.

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). An

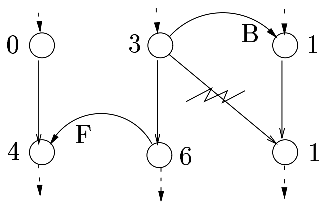

Fig. 7. Removing edge $3 \rightarrow 5$ changes the forward simulation.

example is given in Fig. 7, where B anf F represent backward and forward simulation, respectively. Since $3 \curvearrowright_B 1$, we can remove the arc $3 \rightarrow 5$, doing so we will add $\langle 0, 3 \rangle$ to $\curvearrowright_F$.

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 each 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 the 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 (a.k.a. Petri Nets) 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 each j,

[3] Note that, $[\![G]\!]$ is well-defined even if G is not a sharing tree.

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 $g_{i,j} \geq d_{i,j}$. The prime variable x' represents the new value of x after the execution of a rule. States are tuple of natural numbers and executions are sequences of tuples $t_0 t_1 \ldots t_i \ldots$ where t_{i+1} is obtained from t_i by applying (non-deterministically) one of the transition rules. Invariant properties like *mutual exclusion* can be represented through upwards closed sets [AČJT96,DEP99]. Checking safety properties expressed as upward closed set for Petri Nets is decidable using the algorithm of [AČJT96]. The *predecessor states operator pre* associated to a vector addition system takes as input a set of states (tuples) F and returns the set of predecessors of F. Let F be a set of initial states (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. The termination test is based on the subsumption 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 upwards closed set U. The predecessor relation for VAS can be represented as the following $\mathcal{U}$-formula:

$$pre(\varphi_U) = \bigvee_{i=1,\ldots,m} (\varphi_i \quad \wedge \quad \exists_{x_1+c_{i,1}}.\ldots.\exists_{x_k+c_{i,k}}.\varphi_U),$$

where $\varphi_i = x_1 \geq c_{i,1} \wedge \ldots \wedge x_n \geq x_{i,n}$. In other words, by using the results in the previous sections, starting from S_{φ_U} we can compute the sharing tree $S_{\varphi_{pre(U)}}$ that represents $pre(U)$. 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.

5.1 Some Experimental Results

We have tested the (biggest) examples in [DEP99] using our prototype implementation. The results are shown in Fig. 8: the flag LR denotes the use of the local reduction of 4; the field FSR denotes the frequency in the use of the reduction based on the forward simulation 3; ET denotes the execution time needed to reach a fixpoint. In the first experiment of each example, we do not remove the redundancies from $\bigcup_i pre^i(\phi_0)$, whereas in the second experiment we apply the reduction based on the forward simulation (every 5 steps).

Example (1) corresponds to the manufacturing system of [DEP99]. Sharing trees allows us to dramatically speed up the computation (see [DEP99] for the execution times of other tools methods). Simulation-based reduction (every 5 steps) allows us to reduce the set of states (checking for redundancies) of a factor of ten (removing all redundancies yields 450 elements). The other examples (2-3) give an idea of the ratio (Nodes/NV*NE) of the sharing trees obtained for NV=20 and NV=25, respectively. We did note manage to handle these examples with other methods in acceptable time.

6 Related Work

In [AČJT96,AJ99], the authors introduce a symbolic representation (*constraint* system) for collections of upwards closed sets. Their representation corresponds to disjunctive $\mathcal{U}$-formulas. As mentioned in the introduction, the traditional symbolic methods for handling linear constraints (e.g., *Presburger or real solvers* and *polyhedra*) suffer from the state-explosion problem when applied to this type of 'constraints' (see [DEP99]). 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.

Example	NV	NR	NS	LR	FSR	NS	ET	NE	Nodes	Ratio
1	13	6	1	no	_	24	39s	7563	4420	4%
1	13	6	1	yes	5	24	68s	727	1772	12%
2	20	4	1	no	_	26	13s	1347	5545	20%
2	20	4	1	yes	5	26	44s	1172	5333	22%
3	25	4	2	no	_	31	120s	3682	15315	16%
3	25	4	2	yes	5	31	680s	2339	11647	19%

Fig. 8. Execution times on examples of VAS: NV=No. Variable; NR=No. Rules; NS=No. Initial States; ET=Execution Time; LR=Use of loc.reduction; FSR=Freq. sim. reduction (_=not used); NS=No Steps; NE=No. Elem.; Ratio=Nodes/(NV*NE).

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 [SKMB90] or *multi-terminal* DDs [CFZ96]).

Our approach shares some similarities with recent work on *interval decision diagrams* (IDDs) [ST98,ST99] and *clock decision diagrams* (CDDs) for timed automata [BLP$^+$99], in that all approaches use *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 these 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]. The method works via a translation to SMV [McM93]. We are not aware of BDDs-based representations for the 'constraints' we are interested in, i.e., 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 sets of upwards closed '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 problems via upwards closed sets ($\sim \mathcal{U}$-formulas $\sim$ sharing trees) is generally very costly[4], testing for membership of the initial configuration (when it can be expressed with a conjunctive formula) can be done efficiently (from Theorem 2 logarithmic in the size of the formula). This gives us an effienct method to detect *violations*.

The implementation is currently being optimized, but the preliminary experimental results seems 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).

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

Finally, it would be interesting to apply our techniques based on simulation relations to more general type of constraints (i.e., to the corresponding symbolic representation like IDDs [ST98]).

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.

[AJ99] P. A. Abdulla, and B. Jonsson. Ensuring Completeness of Symbolic Verification Methods for Infinite-State Systems. *Theoretical Computer Science*, 1999. To appear.

[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.

[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.

[BF99] B. Bérard, and L. Fribourg. Reachability Analysis of (Timed) Petri Nets Using Real Arithmetic. In *Proceedings of the 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *LNCS*, pages 178–193. Springer, 1999.

[BW94] B. Boigelot, P. Wolper. *Symbolic verification with periodic sets.* In *Proceedings of the 6th Conference on Computer Aided Verification (CAV'94)*, volume 818 of LNCS, pages 55–67. Springer, 1994.

[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.

[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.

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

[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.

[Bul98] T. Bultan. *Automated Symbolic Analysis of Reactive Systems*, Ph.D. Thesis, Department of Computer Science, University of Maryland, College Park, August 1998.

[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.

[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.

[CJ98] H. Comon, Y. Jurski. Multiple Counters Automata, Safety Analysis, and Presburger Arithmetic. In *Proceedings of the 9th International Conference on Computer Aided Verification (CAV'98)*, volume 1427 of LNCS, pages 268–279. Springer, 1998.

[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.

[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.

[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.

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

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

[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.

[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.

[KM69] R. M. Karp and R. E. Miller. Parallel Program Schemata. *Journal of Computer and System Sciences*, 3, pages 147-195, 1969.

[Joh90] D. S. Johnson. A Catalog of Complexity Classes. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A, Algorithm and Complexity*, Elsevier, 1990.

[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.

[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.

[McA84] K. McAloon. Petri Nets and Large Finite Sets. *Theoretical Computer Science* 32, pages 173–183, Elsevier, 1984.

[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.

[SKMB90] 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.

[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.

[ST98] K. Strehl, L. Thiele. Symbolic Model Checking of Process Networks Using Interval Diagram Techniques. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'98)*, pages 686–692. 1998.

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

[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.

[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.