A New Approach to Upward-Closed Set Backward Reachability Analysis

Jesse Bingham [1,2,3]

Department of Computer Science
University of British Columbia
Vancouver, Canada

Abstract

In this paper we present a new framework for computing the backward reachability from an upward-closed set in a class of parameterized (i.e. infinite state) systems that includes broadcast protocols and petri nets. In contrast to the standard approach, which performs a single least fixpoint computation, we consecutively compute the finite state least fixpoint for constituents of increasing size, which allows us to employ binary decision diagram (BDD)-based symbolic model checking. In support of this framework, we prove necessary and sufficient conditions for convergence and intersection with the initial states, and provide an algorithm that uses BDDs as the underlying data structure. We give experimental results that demonstrate the existence of a petri net for which our algorithm is an order of magnitude faster than the standard approach, and speculate properties that might suggest which approach to apply.

Key words: broadcast protocol, petri net, parameterized model checking, well-structured transition system

1 Introduction

The successes of finite state model checking techniques have motivated much research on automatic verification of infinite state systems. Recent works have investigated a class of infinite systems called *well-structured transition systems* and provided several positive decidability results [1,8,9]. Examples of well-structured transition systems include: basic process algebra, lossy channel systems, timed automata, petri nets, and various species of parameterized

[1] This version corrects a minor error in Fig. 1 that appeared in the INFINITY 04 proceedings.

[2] Thanks to Armin Biere, Anne Condon, Giorgio Delzanno, Pierre Ganty, Alan J. Hu, and Laurent Van Begin

[3] Email: jbingham@cs.ubc.ca

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

finite state systems. An important class of decidable problems, with variants variously called *safety property verification*, *control state reachability*, and *coverability*, ask if a specified set of target (i.e. violating) states is reachable from any of the designated *initial* states.

The standard approach to solving these problems is based on *backward reachability analysis*, in which, starting from the set of violating states, preimages are iteratively computed until a fixpoint is reached. For well-structured systems, convergence is guaranteed, and an abstract algorithm is given in several papers on the topic [1,8,9]; we will call this the *standard algorithm*. Necessary for practical implementation of the standard algorithm is an efficient representation of so-called *upward-closed sets*. Delzanno et al. propose using *covering sharing trees* (CST) for this purpose [5]. One drawback of this technique is that checking for convergence is co-NP hard in the size of the involved CSTs.

In this paper we propose an alternative to the standard algorithm. We focus on a generalization of *broadcast protocols* [8,7]. A broadcast protocol represents the composition of an unbounded number of identical finite state processes that communicate in certain ways. The infinite state space arises because a broadcast protocol consists of an infinite family of systems; for each positive n the system of *size* n involves the composition of n processes. The paramount difference between our approach and the standard is that rather than compute the transitive preimage of the entire set of violating states in one fell swoop, we iteratively compute the backward reachability set for constituent systems of increasing size, until we reach a certain convergence condition. Since the constituent systems are each finite state, we can leverage well-known finite state symbolic model checking [4] and binary decision diagram (BDD) [3] techniques. A primary advantage of our approach is that the necessary convergence checks can be done efficiently. A possible disadvantage is that in some sense we undo a symmetry reduction inherent in the standard approach.

That our procedure eventually covers all elements of the transitive preimage follows trivially from the theory of well-structured transition systems. However, *determining when* we have reached full coverage is unobvious [4]. A key result in this paper is a theorem that gives us a necessary and sufficient condition for detecting this convergence. Through our experimental results we exhibit the existence of a family of petri nets for which our algorithm is two orders of magnitude faster than a state-of-the-art implementation of the standard approach. We emphasize that our approach does not always outperform the standard approach, but there are examples for which each is superior. In our discussion of Sect. 5 we speculate the system properties that might suggest which technique to apply.

[4] In particular, convergence between sizes n and $n+1$ is in general insufficient for full coverage [2].

The paper is organized as follows. Preliminary definitions are given in Sect. 2. Section 3 develops our approach and highlights the differences between the standard approach. In Sect. 4 experimental results for an example petri net are presented. Our discussion of Sect. 5 attempts to explain the strengths of each approach and outlines future work. Due to space constraints, proofs have been omitted throughout. The reader may find the proofs and an outline of related work in the technical report [2].

2 Preliminaries

A *transition system* T is a pair $(S, \rightarrow)$, where S is the *state space* and $\rightarrow \subseteq S \times S$ is the *transition relation*. Associated with T is the *predecessor function* $Pred : 2^S \rightarrow 2^S$ defined by $Pred(X) = \{y \mid \exists x \in X : y \rightarrow x\}$. A *path of* T is a sequence $x_0, \ldots, x_\ell$ over S such that for each $1 \leq i \leq \ell$ we have $x_{i-1} \rightarrow x_i$. The function $Pred^* : 2^S \rightarrow 2^S$ is defined by

$$Pred^*(X) = \{y \mid \text{there exists a path from } y \text{ to some } x \in X\}$$

Let $\mathbb{N}$ and $\mathbb{Z}$ denote the natural numbers and the integers, respectively. For a vector $v \in \mathbb{Z}^m$ we let $v(i)$ denote the ith component of v for each $1 \leq i \leq m$. The *weight* of $v \in \mathbb{Z}^m$ is $|v| = \sum_{i=1}^{m} v(i)$, and for any $X \subseteq \mathbb{N}^m$, denote $\{x \in X \mid |x| = n\}$ by $[X]_n$. We define a reflexive and transitive relation $\preceq$ on $\mathbb{N}^m$ by $x \preceq y$ iff for all $1 \leq i \leq m$ we have $x(i) \leq y(i)$, and we write $x \prec y$ iff $x \preceq y$ and $x \neq y$.

A *broadcast protocol* B is a finite set of pairs $\{(M_1, c_1), \ldots, (M_{|B|}, c_{|B|})\}$ where each M_i is a $m \times m$ binary matrix with all columns being unit vectors, and each c_i is an integer vector of height m such that $|c_i| = 0$. The semantics of B is the transition system $(\mathbb{N}^m, \rightarrow)$, where $\rightarrow \subseteq \mathbb{N}^m \times \mathbb{N}^m$ is such that $u \rightarrow v$ iff $v = Mu + c$ for some $(M, c) \in B$.

It follows that whenever $u \rightarrow v$ in a broadcast protocol we have $|u| = |v|$. Intuitively, a broadcast protocol models a parameterized system consisting of an unbounded number of identical finite state processes, where each process has m states. A vector $v \in \mathbb{N}^m$ represents any state in which there are $v(i)$ processes in local state i for each $1 \leq i \leq m$. Communication between processes can occur as a broadcast, in which all components change state based on the type of broadcast and the local state, along with a rendezvous-like synchronization in which a bounded number of processes collaborate to change state. These two aspects of a transition are respectively modelled by the matrix M and the vector c of each pair (M, c) in the broadcast protocol.

In our verification problem the initial states and target (i.e. violating) states are respectively required to be sets of the following forms. A *parametric set* is $I \subseteq \mathbb{N}^m$ such that $I = \{x \mid x(1) \sim_1 a(1) \wedge \cdots \wedge x(m) \sim_m a(m)\}$, for some $a \in \mathbb{N}^m$ and each $\sim_i$ is either $=$ or $\geq$. The vector a is called the *root vector* of I. An *upward-closed set* [9,1] is a set $U \subseteq \mathbb{N}^m$ such that $x \in U$ and $x \preceq y$ implies $y \in U$. For $X \subseteq \mathbb{N}^m$, the *upward-closure* of X is $\uparrow X =$

```
previous_reach := ∅
reach := gen(U)
while ¬(↑reach ⊆ ↑previous_reach) do
    if (I ∩ ↑reach ≠ ∅) then
        exit with verification failure
    previous_reach := reach
    reach := gen(U) ∪ basis(Pred(↑reach))
exit with verification success
```

Fig. 1. The standard algorithm. Here, $basis(S)$ is used to denote some (not necessarily canonical) finite basis of upward-closed set S.

$\{y \mid \exists x \in X : x \preceq y\}$. If U is upward-closed, a *basis* for U is a set U^b such that $U = {\uparrow}U^b$.

A set $X \subset \mathbb{N}^m$ is *canonical* if for all distinct $x, y \in X$ we have that x and y are incomparable under $\preceq$. It is well-known that any upward-closed set has a canonical finite basis, and that this basis is unique. Given an upward-closed set U, we let $gen(U)$ denote this basis. The *base-weight* of an upward-closed set U is $\max(\{|u| \mid u \in gen(U)\})$, denoted $bw(U)$.

The verification problem we tackle is now defined. The *Broadcast Protocol Reachability Problem* (BPRP) asks, given a broadcast protocol B, a parametric set I, and an upward-closed set U, does there exist $v \in I$ and $u \in U$ such that there is a path of B from v to u? BPRP is decidable; a decision procedure based on the standard algorithm is given by Esparza, Finkel, and Mayr [8]. [5] Our notion of broadcast protocols subsumes petri nets in that any algorithm to solve BPRP can be harnessed to solve a similar problem on petri nets called the *coverability problem* [2].

3 Our Approach

A rendition of the standard algorithm is given in Fig. 1. On the surface, this algorithm resembles the well-known finite state backward reachability analysis, i.e. *least fixpoint computation*, the difference being that the involved sets are upward-closed and hence infinite. The algorithm is guaranteed to converge for well-structured transition systems such as broadcast protocols.

Our approach contrasts with the standard approach and is based on the following observation. A broadcast protocol consists of the disjoint union of a countably infinite number of finite transition systems (this follows from

[5] Our definition of broadcast protocols differs from that of [8] in two ways. First, as a very modest generalization, we allow rendezvous synchronizations to involve any constant number of processes (not just 2). Second, we don't explicitly allow for guards. Guarding is handled indirectly via the negative components of the c vector of a transition. There is a subtle situation wherein our approach is not as general; however this can be remedied by adding a few more local states. Esparza et al.'s algorithm could clearly be applied to our formalism, and vice versa.

```
i := 1
while (¬converged) do
    compute Γ_i := Pred*([U]_i)
    if intersect_check(I, Γ_i) then
        exit with verification failure
    i := i + 1
exit with verification success
```

Fig. 2. The skeleton of our algorithm

the fact that $u \rightarrow v$ implies $|u| = |v|$). For each i, one such subsystem is obtained by restricting $\rightarrow$ to $[\mathbb{N}^m]_i$, and intuitively corresponds to the instance of with i processes. Starting with $i = 1$, we analyze each of these subsystems by computing $Pred^*([U]_i)$ for successive values of i and checking if this set allows us to determine a nonempty intersection with the initial states I. If so, there exists a path from I to U and the procedure terminates. Otherwise i is incremented and the process repeats until we reach a certain convergence condition.

The skeleton of our algorithm is given in Fig. 2. Omitted are the definitions of the termination condition *converged*, and the function *intersect_check*. Note that implicit in the line "`compute` $\Gamma_i := Pred^*([U]_i)$" is an inner loop that performs a least fixpoint computation. However, unlike the fixpoint computation of the standard approach (i.e. the `while` loop of Fig. 1), our fixpoint computations take place in finite domains (namely $2^{[\mathbb{N}^m]_i}$) and are hence amenable to the well-known techniques of finite state symbolic model checking [4].

In Sect. 3.1 we develop a theorem that gives a necessary and sufficient condition for *converged*, while in Sect. 3.2 we present a theorem that conveys the appropriate definition of *intersect_check*. Section 3.3 shows how BDDs can be employed as the underlying data structure in our algorithm. For the remainder of this section we fix an BPRP instance (B, I, U) with corresponding transition system $(\mathbb{N}^m, \rightarrow)$.

3.1 A Convergence Condition

In this section we present Theorem 3.2, which gives us a convergence condition for the algorithm of Fig. 2. The formal proof is provided elsewhere [2]. However the key idea behind the proof, the notion of *eager descent*, is summarized here.

Figure 3 gives a graphical depiction of the notion of eager descent. Informally, an eager descent is a path that alternates between following the transition relation $\rightarrow$ and "jumping down" to a smaller system size (i.e. weight) through $\preceq^{-1}$. These jumps always land as low as possible while remaining in $Pred^*(U)$. Definition 3.1 gives the formal definition. For $v \in Pred^*(U)$, define $\mu(v)$ to be the (nonempty) set of $\preceq$-minimal elements of $\{t \mid t \preceq v\} \cap Pred^*(U)$.

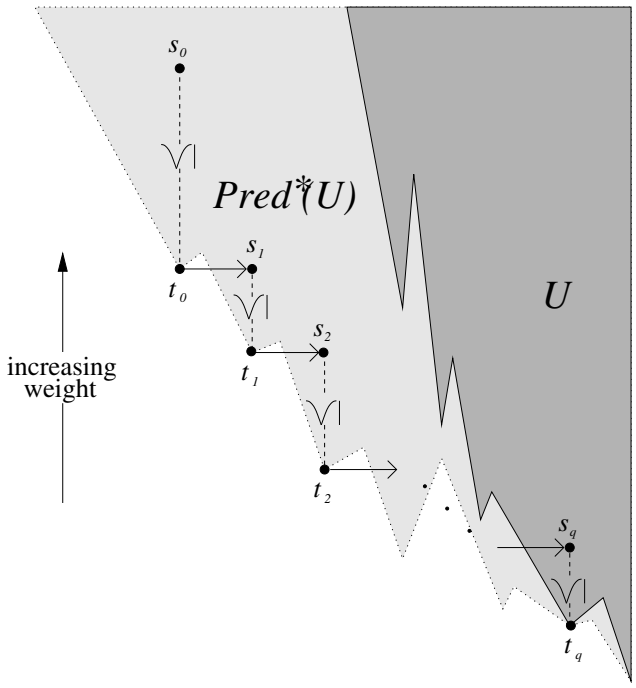

Fig. 3. Eager Descent

Definition 3.1 [eager descent] An *eager descent* from $v \in Pred^*(U)$ to an upward-closed set U is a path $e = s_0, t_0, s_1, t_1, \ldots, s_q, t_q$ through the digraph $(\mathbb{N}^m, \rightarrow \cup \preceq^{-1})$ such that

(i) $v = s_0$, and

(ii) $t_i \in \mu(s_i)$ for all $0 \leq i \leq q$, and

(iii) $t_i \rightarrow s_{i+1}$ for all $0 \leq i < q$, and

(iv) $s_i \neq s_j$ for all $0 \leq i < j \leq q$, and

(v) $t_q \in U$

For $v \in \mathbb{Z}^m$ with weight 0, the *displacement* of v, denoted $dis(v)$ is the sum of the positive components of v. It turns out that (with the exception of the first hop) the distance each downward hop jumps is bounded by $dis(c)$, where c is the vector involved in the previous broadcast protocol transition. Letting $maxdis(B) = \max(\{dis(c) \mid (M, c) \in B\})$, this leads to the following result.

Theorem 3.2 (convergence) *Let U be an upward-closed set and let $n \geq bw(U)$. Then $Pred^*(U) = \uparrow \bigcup_{j=1}^{n} Pred^*([U]_j)$ if and only if*

$$(1) \quad \uparrow Pred^*([U]_{n+maxdis(B)}) \subseteq \uparrow Pred^*([U]_{n+maxdis(B)-1}) \subseteq \cdots \subseteq \uparrow Pred^*([U]_n)$$

Theorem 3.2 tells us that we can terminate our algorithm (Fig. 2) once we arrive at a value of i such that (1) holds for $n = i - maxdis(B)$. In Sect. 3.3 we will show how the set containments of (1) can be checked efficiently.

3.2 Checking for Intersection with the Initial States

Another aspect of our algorithm that is yet to be defined is the function *intersection_check* (cf. Fig. 2). The goal of this function is to return true if we can ascertain that $Pred^*(U) \cap I \neq \emptyset$ by examining $Pred^*([U]_n)$. In this section we provide a necessary and sufficient condition for this intersection being nonempty that is fit for use in our algorithm. For $Y \subseteq \{1, \ldots, m\}$, define the partial order $\preceq_Y$ on $\mathbb{N}^m$ such that $v \preceq_Y u$ iff for all $i \in Y$ we have $v_i \leq u_i$.

Theorem 3.3 (intersection check) *Let $P = \{x \mid x_1 \sim_1 a_1 \wedge \cdots \wedge x_m \sim_m a_m\}$ be a parametric set, let X be an upward-closed set, and let $r \geq bw(X)$. Then $P \cap X = \emptyset$ if and only if $\bigcup_{i=1}^{r} [X]_i$ does not contain v such that $v \preceq_E a$, where $E = \{i \mid \sim_i \ is \ =\}$ and a is the root vector of P.*

Since I is parametric and $Pred^*(U)$ is upward-closed, we can apply Theorem 3.3 to determine if the intersection of these sets is empty. The suggested implementation of *intersection_check* takes $Pred^*([U]_n)$ and simply tests if this set has a nonempty intersection with $[\{v \mid v \preceq_E a\}]_n$. Decidability of this problem follows from the fact that both of these sets are finite.

3.3 Using BDDs

Our discourse so far has dealt with the semantics of a broadcast protocol as a transition system $(\mathbb{N}^m, \rightarrow)$. As previously mentioned, broadcast protocols model the composition of an unbounded number of identical communicating finite state processes, i.e. a parameterized family of finite state systems [7,8]. We let $L = \{\ell_1, \ldots, \ell_m\}$ denote the local states of an individual process. For the system instance with n processes, a *concrete state* is a vector $g \in L^n$, where g_i gives the local state of the ith process. A vector $v \in \mathbb{N}^m$ is said to *abstract* a concrete state $g \in L^{|v|}$ if $|\{j \mid g_j = \ell_i\}| = v_i$ for each $1 \leq i \leq m$. This distinction between abstract and concrete states must be made; our theory of the previous sections pertains to the former while in this section we will instantiate our algorithm to manipulate sets of the latter. The concretization function γ maps abstract states to sets of concrete states: given $v \in \mathbb{N}^m$, $\gamma(v)$ is the subset of $L^{|v|}$ consisting of all concrete states abstracted by v. We extend γ to work on a set A of abstract states by $\gamma(A) = \bigcup_{v \in A} \gamma(v)$.

Binary decision diagrams (BDDs) are a popular data structure for representing and manipulating boolean functions [3]. BDDs can be used to store arbitrary finite sets by encoding elements using boolean variables and building a BDD for the characteristic function, and they can also symbolize finite relations in a similar manner. Assuming some reasonable encoding of the elements of L, we employ BDDs to represent sets of concrete states (namely the transitive preimages) and also the concrete version of the predecessor function [6]

[6] More accurately, there is a different $Pred_\gamma$ BDD for each system size n

$i := 1$
$n := 1$
$\Gamma_0 := \emptyset$
`while` $(n \geq i - maxdis(B) \vee i \leq bw(U))$ `do`
    `compute` $\Gamma_i := Pred^*_\gamma(\gamma([U]_i))$
    `if` $(\gamma([\{v \mid v \preceq_E a\}]_i) \cap \Gamma_i \neq \emptyset)$ `then`
        `exit with verification failure`
    `if` $(\neg(\Gamma_i \Rightarrow \Gamma^{\mathrm{el}}_{i-1}))$ `then`
        $n := i$
    $i := i + 1$
`exit with verification success`

Fig. 4. Our algorithm, version 2

$Pred_\gamma$, which intuitively mimics $Pred$ in the concrete domain.

The following definition defines an operation on concrete state sets that is integral to doing the containment checks of Theorem 3.2 when BDDs are the underlying data structure. We use $\Rightarrow$ and $\Leftrightarrow$ to respectively denote set containment and set equivalence between BDDs (i.e. logical implication and logical equivalence).

Definition 3.4 [existential lifting] Let $S \subseteq L^n$ be a set of concrete states. Then the *existential lifting* S^{el} of S is the subset of L^{n+1} such that

$$(c_1, \ldots, c_{n+1}) \in S^{\mathrm{el}} \;\Leftrightarrow\; \exists i \in \{1, \ldots, n+1\} : (c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_{n+1}) \in S$$

Theorem 3.5 *For $X \subseteq [\mathbb{N}^m]_n$ and $Y \subseteq [\mathbb{N}^m]_{n-1}$, $\uparrow X \subseteq \uparrow Y$ if and only if $\gamma(X) \Rightarrow \gamma(Y)^{\mathrm{el}}$*

Intuitively, the states of S^{el} are precisely those that can be transformed into a state of S by deleting one component. The BDD for S^{el} can be computed from the BDD for S. Theorem 3.5 shows how the upward-closed set containments of the style of (1) are reduced to checking a logical implication involving existential lifting in the concrete domain (BDDs).

Figure 4 shows a detailed version of our BDD-based algorithm. This version elaborates on the skeleton of Fig. 2 in three (overlapping) ways: 1) it includes the convergence condition suggested by Theorems 3.2 and 3.5, 2) it includes a concretized version of the initial states intersection check of Theorem 3.3, and 3) it processes BDDs representing concrete state sets. Note that in Fig. 4, the Γ_is are now BDDs representing sets of concrete states.

Theorem 3.6 *The algorithm of Fig. 4 solves BPRP.*

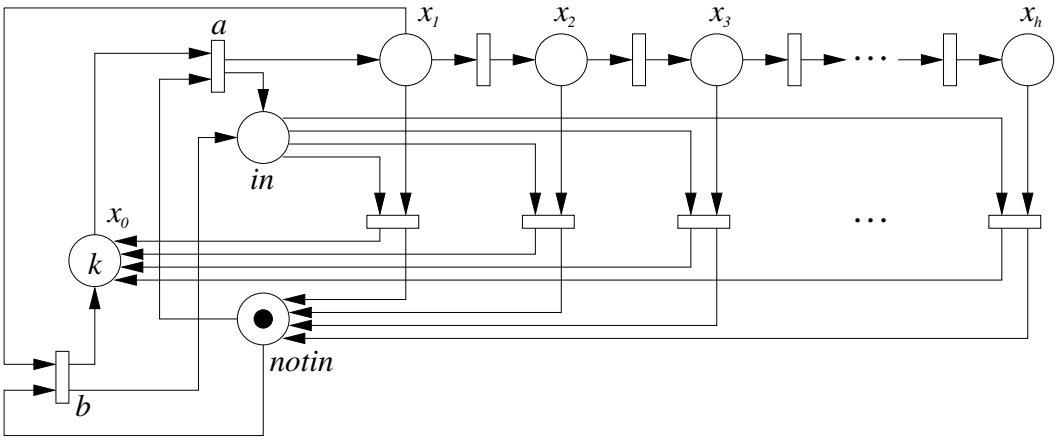

Fig. 5. The petri net $ME(h)$ used in the experiments

4 Experimental Results

As a proof of concept, we have implemented our algorithm as a pair of tools `translate` and `bucsub`. `bucsub` implements our algorithm [7] using the CUDD BDD package [11], and requires its input to be in SMV format. `translate` takes a simple petri net description (extended to handle broadcast transitions) and produces the necessary SMV files for `bucsub`; these SMV files describe the concrete transitions systems.

To demonstrate the existence of examples for which our approach outperforms the standard approach, we constructed the "parameterized petri net" $ME(h)$ depicted in Fig. 5. This family of petri nets is motivated by our discussion of data structure size in Sect. 5. The parameter h dictates the width of the chain of places $x_1, \ldots, x_h$ along the top of the figure. h allows us to control the size of the local state space in the resulting broadcast protocol. For any h, the initial marking of $ME(h)$ places a single token at the place *notin*, and an arbitrary number k at place x_0. A token can move from x_0 to x_1 when there is a token at *notin* by firing transition a. This will move the token at *notin* to *in*, which will disallow any more tokens from entering x_1. The token that is in the chain can move along the chain, and at any point it may hop back to x_0, which will result in the token at *in* being moved back to *notin*. Hence $ME(h)$ implements mutual exclusion in that at most one token can be in the chain at any time. For our experiments, we verified that there can only be at most one token at x_h; i.e. the upward-closed set of markings $\{m \mid m(x_h) \geq 2\}$ is not reachable.

We measured execution times [8] for $h = 25, 50, \ldots, 250$; the results are plotted in Fig. 6. The plot labelled "CST" gives the runtime for the Delzanno

[7] `bucsub` includes an optimization not discussed in this paper; for details see section 3.4 of [2].

[8] Experiments were executed on a 2.6 GHz Intel Pentium 4 machine running Red Hat Linux 9.

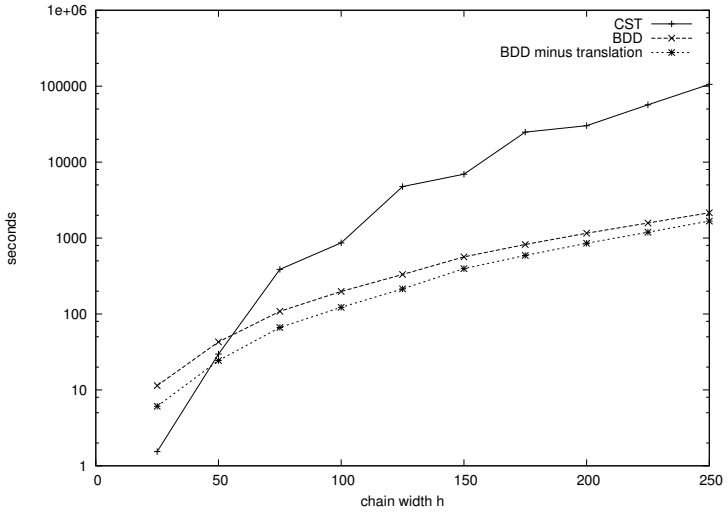

Fig. 6. Experiment execution times for the two approaches. The vertical axis gives the runtime is seconds; the horizontal axis gives the petri net parameter h. Our approach is labelled "BDD", while the CST approach of Delzanno et al. is labelled "CST".

et al.s CST approach [9] [5,6]. The plot labelled "BDD" gives the total runtime for both tools `translate` and `bucsub`, while "BDD minus translation" provides the runtime of `bucsub` only. The latter is presented because the current `translate` phase is suboptimal; in a sophisticated implementation there would be no need for the intermediate SMV files and hence the time in this phase would be highly mitigated.

From Fig. 6 we see that for $ME(h)$ our approach is two orders of magnitude faster than the state-of-the art implementation of the standard approach (for large h).

5 Discussion

In this section we compare our approach with the standard approach (i.e. CSTs) and outline future research directions.

Convergence Given two CSTs C_1 and C_2, the problem of checking if C_1 *subsumes* C_2 (i.e. if the upward-closed set represented by C_1 is a superset of that of C_2) is co-NP hard in the size of the involved CSTs [5]. Unfortunately, checking subsumption is an integral part of the standard algorithm (cf. the `while` condition in Fig. 1). To combat this problem, Delzanno et al. develop a sophisticated heuristic solution in which certain CST simulation relations facilitate pruning of an (exponential time) exact subsumption check [6].

In contrast, subsumption between two BDDs can be decided in time pro-

[9] In fact, the reported run-times are those of software based on *interval sharing trees* (IST) which are an extension of CSTs to handle two-sided constraints [10]. The IST software used is a constant 3 or 4 times slower than a pure CST implementation.

portional to the product of their sizes [3]. In fact, we can correctly replace the condition of the second `if` statement of Fig. 4 with a bidirectional subsumption: $\neg(\Gamma_i \Leftrightarrow \Gamma^{\text{el}}_{i-1})$. This test can be done *in constant time* for BDDs [3].

Data structure size Another indication of the efficiency of the two algorithms is the size of the underlying data structures. Predicting the dynamics of the sizes is a complex problem. Though BDDs compactly represent many practical boolean functions, the worst case size is exponential in their height (i.e. the number of boolean variables). To the author's knowledge, bounds on the size of CST have not been derived in the literature, however, any such bound is at least exponential in the height of the structure. Here we consider data structure height as a course measure of worst-case size.

The height of the CSTs is fixed at $|L|$, while the height of the BDDs is at most $(n_f + maxdis(B))\lceil \log_2 |L| \rceil$, where n_f is the final value of n in our algorithm (which is equal to $\max(bw(Pred^*(U)), bw(U))$ [2]). For example, consider our petri net $ME(250)$, where $|L| = 253$, $n_f = 4$, and $maxdis(B) = 2$. In this case our BDDs have height at most 48, while the CSTs have height 253. This rudimentary analysis suggests that our approach appears to have an advantage when $|L|$ is large and $n_f + maxdis(B)$ is modest. Thus our approach might be more apt at dealing with large local state spaces. Unfortunately, only the lower bound $bw(U)$ of n_f is known a priori.

Future work Clearly further comparison between our approach and Delzanno et al.'s implementation of the standard algorithm is necessary. We also plan on deducing the bound on the size of a BDD for S^{el} in terms of the size of S for symmetric S as such a result would help further quantify the complexity of our algorithm. It is likely that our method can be applied to other discrete well-structured systems, hence we intend to pursue such extensions.

References

[1] P. Aziz Abdulla, K. Cerans, B. Jonsson, and T. Yih-Kuen. General decidability theorems for infinite-state systems. In *10th Annual IEEE Symp. on Logic in Computer Science (LICS'96)*, pages 313–321, 1996.

[2] J. Bingham. A new approach to upward-closed set backward reachability analysis. Technical report, University of British Columbia Department of Computer Science, 2004. TR-2004-07.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Computers*, C-35(8):677–691, August 1986.

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Conf. on Logic in Computer Science*, pages 428–439, 1990.

[5] G. Delzanno and J. F. Raskin. Symbolic representation of upward-closed sets. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 7th Int'l Conf.*, pages 426–440, 2000.

[6] G. Delzanno, J. F. Raskin, and L. Van Begin. Attacking symbolic state explosion. In *Proc. of 13th Int'l Conf. on Computer-Aided Verification (CAV)*, pages 298–310, 2001.

[7] E. A. Emerson and K. S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *12th Annual IEEE Symp. on Logic in Computer Science (LICS'98)*, pages 70–80, 1998.

[8] J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *13th Annual IEEE Symp. on Logic in Computer Science (LICS'99)*, pages 352–359, 1999.

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

[10] P. Ganty and L. Van Begin. Non deterministic automata for the efficient verification of infinite-state. presented at: CP+CV Workshop at European Joint Conferences on Theory and Practice of Software (ETAPS), 2004.

[11] F. Somenzi. Colorado university decision diagram package (CUDD) webpage. http://vlsi.colorado.edu/∼fabio/CUDD/cuddIntro.html.