Heuristic Symbolic Verification of Safety Properties for Parameterized Systems

Qiusong Yang, Mingshu Li
Institute of Software, Chinese Academy of Sciences
4# South Fourth Street, Zhong Guan Cun
Beijing 100190, P.R. China
qiusong_yang, mingshu@iteshs.iscas.ac.cn

ABSTRACT
A parameterized system is a system that involves numerous instantiations of the same finite-state process, and depends on a parameter which defines its size. The backward reachability analysis has been widely used for verifying parameterized systems against safety properties modeled as a set of upward-closed sets. As in the finite-state case, the verification of parameterized systems also faces the state explosion problem and the success of model checking depends on the data structure used for representing a set of states. Several constraint-based approaches have been proposed to symbolically represent upward-closed sets with infinite states. But those approaches are still facing the symbolic state explosion problem and the containment problem, deciding whether a set of concrete states represented by one set of constraints is a subset of another set of constraints, is co-NP complete. As a result, those examples investigated in the literature would be considered of negligible size in finite-state model checking. In this paper, we present several heuristic rules specific to parameterized systems that can help to mitigate the problem. The experiment shows that the efficiency is significantly improved and the heuristic algorithm is several orders of magnitude faster than the original one in certain cases.

Categories and Subject Descriptors
D.2.4 [Software Engineering]: Software/Program Verification—formal methods, model checking, heuristic

General Terms
Validation

Keywords
Parameterized System, Heuristic, Safety Property

1. INTRODUCTION

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.
ciding whether a set of concrete states represented by one set of constraints is a subset of another set of constraints, is co-NP complete (for example, the containment problem for constraints with additions and DV-constraints in [4] and the subsumption problem for sharing tree in [8]). The containment test is the most important operation in the backward reachability analysis and decides when the backward propagation terminates. As a result, those examples investigated in [12, 6, 5] would be considered of negligible size in finite-state model checking.

In this paper, we investigated several heuristic rules specific to parameterized system that can help to relieve the symbolic state explosion and co-NP complexity of containment tests mentioned above. The experiment shows that the efficiency is greatly improved and the heuristic algorithm is almost several orders of magnitude faster than the original one in certain cases. The sequel of the paper is organized as follows: Preliminary definitions of parameterized systems are presented in Section 2. In Section 3, we present a new concrete backward reachability algorithm for the verification of parameterized systems defined in Section 2 which are different from what have been defined in [12] in the sense that the monitor and the process can act non-deterministically.

In Section 4, those heuristic rules are clearly defined. In Section 5, we show that the model checking efficiency is significantly improved through a series of experiments. An outline of related work is presented in Section 6 and Section 7 draws the conclusion.

2. PARAMETERIZED SYSTEMS

2.1 Syntax

Without lost of generality, we consider those parameterized systems consisting of a monitor, \( M \), and an arbitrary number of instances of a finite-state process \( P \). Those instances of \( P \) are concurrently combined together and they only communicate with the monitor by rendezvous or broadcast. Following the notations in [12], we represent a parameterized system as a tuple \( \langle M, P, \Sigma \cup \{\tau\} \rangle \) where:

- \( \Sigma \) is a set of actions and \( \tau \) is an internal behavior
- \( M \) is a finite automaton \( (Q_M, \Sigma \times \{\cdot, !, ??\!] \cup \{\tau\}, s'_M, \delta_M) \), representing the monitor. \( \Sigma \times \{\cdot, !, ??\!] \cup \{\tau\} \) is a set of actions consisted of: the set \( \Sigma \times \{\cdot, !, ??\!] \) of input and output rendezvous labels, the set \( \Sigma \times \{\cdot, !, ??\!] \) of input and output broadcast labels, and the internal behavior \( \tau \). \( s'_M \) denotes the initial state of the automaton. \( \delta_M \in Q_M \times (\Sigma \times \{\cdot, !, ??\!] \cup \{\tau\}) \times Q_M \) specifies the automaton’s transitions.
- \( P \) is a finite automaton \( (Q_P, \Sigma \times \{\cdot, !, ??\!] \cup \{\tau\}, s'_P, \delta_P) \), representing the identical process. Each element in the tuple is similar to what has been defined in \( M \).

The absence of \(!!\) in \( P \)’s actions forbids two instances of \( P \) from communicating by broadcast. The possibility that two instances may communicate by rendezvous actions will be ruled out through the calibration of semantics presented in the next subsection. With this constraint, the algorithm presented in Section 3 can be greatly simplified and, however, the parameterized system is still as powerful as it was. As shown later, an output broadcast transition can be removed from \( P \) and replaced by a new transition in the monitor, so it does a rendezvous communication between instances of \( P \).

As an example, the load balancing example [4] is presented in Figure 1. In the initial configuration of the system, the monitor resides in the state \( idle \) and there are an arbitrary number of processes staying at the state \( req \). When the monitor remains in the state \( idle \), a process can transit from \( req \) to \( use \) or from \( use \) to \( req \) by executing the action \( request \) or \( release \). A process residing in the state \( use \) will non-deterministically transit to the state \( low \) or \( high \) when the message \( swap_out \) is broadcasted by the monitor. A process with a higher priority, in the state \( high \), will transit to the state \( use \) when the monitor broadcasts the message \( swap_in \) and a lower prioritized one will go back to the initial state \( req \).

2.2 Semantics

A configuration, denoted by \( c \), of parameterized systems is a vector \( \langle s_M, n_1, \cdots, n_m \rangle \), where \( s_M \) is the current state of the monitor and, \( m \) denoting the number of states in \( P \), \( n_i \) records the number of processes in the state \( s_i \in Q_P \). In the sequel, we will use \( c_M \) to denote the element \( s_M \) of \( c \), \( c(i) \) to \( n_i \), and \( c_P \) to the vector \( \langle n_1, \cdots, n_m \rangle \). In addition, \( c(i) : n'_i \) is equal to \( c \) other than substituting \( c(i) \) with \( n'_i \). The set of configurations, denoted by \( C \), of a parameterized system is the subset of \( Q_M \times \{\mathbb{N} \cup \{\omega\}\}^m \), \( \omega \) denoting a number that is larger than any specific natural number. Thus, the semantics of a parameterized system is the smallest subset of \( C \times \Sigma \times \mathbb{C} \) such that the following conditions are satisfied, where a triple \( (c, a, c') \in C \times \Sigma \times \mathbb{C} \) is denoted by \( c \xrightarrow{a} c' \):

- if \( s_M \xrightarrow{\cdot} s'_M \) then \( c \xrightarrow{\cdot} c' \) for every \( c, c' \) such that \( c_M = s_M, c'_M = s'_M \), and \( c_P = c_P \)
- if \( s_i \xrightarrow{!} s_j \) then \( c \xrightarrow{!!} c' \) for every \( c, c' \) such that \( c(i) > 0 \) and \( c' = c(i : (c(i) - 1, j : c(j) + 1)) \)
- if \( s_M \xrightarrow{!!} s'_M \) and \( s_i \xrightarrow{!!} s_j \) then \( c \xrightarrow{!!} c' \) for every \( c, c' \) such that \( c(i) > 0 \) and \( c' = c(i : (c(i) - 1, j : c(j) + 1)) \) other than \( c_M = s_M \) and \( c'_M = s'_M \)

![Figure 1: Load Balancing Example](image-url)
if $\delta_s s' \neq s' s$, then $c \not\rightarrow c'$ for every $c, c'$ such that $c(i) > 0$ and $c' = c(i - 1, j : c(j)) + 1$ otherwise $c = s s'$ and $c' = s s'$

- if $s s' \neq s' s$ then $c \rightarrow c'$ for every $c, c'$ such that:

$$V_i = \{ v | v = \sum_{(s_k, v)^{\tau_{(s_k)}}_{s_k}} t_k \cdot u_k, \}
\left\{ \begin{array}{ll}
c(i) = \sum_{(s_k, v)^{\tau_{(s_k)}}_{s_k}} t_k, & 1 \leq i \leq m
\end{array} \right.
$$

where $|u_k| = m$ and $u_k(j) = \left\{ \begin{array}{ll}0 & \text{if } i \neq j
1 & \text{if } i = j
\end{array} \right.$

$$c' \in \{ \langle s s', v \rangle | v = v_1 + \cdots + v_m \text{ and } v_i \in V_i \}$$

In [12], the authors use affine transformations to describe the semantics of deterministic broadcast protocols. However, the one-to-one correspondence between transformation matrices and actions in deterministic systems does not exist when non-determinism is introduced.

### 2.3 Safety Properties

As for the verification of parameterized systems, we focus on those problems that can be reduced to the backward reachability analysis from an upward-closed set. Given a quasi-ordering relation (reflective and transitive), a set $S \subseteq X$ is upward-closed if every $y \geq x \in S$ entails $y \in S$. For any $x \in I$, let $\uparrow x = \{ y | y \geq x \}$ and for a set $S$, $\uparrow S = \bigcup_{x \in S} \uparrow x$. A basis of an upward-closed set $S$ is a set $S_0 \subseteq S$ such that $S = \uparrow S_0$. The quasi-ordering relation $\preceq$ over a parameterized system is defined as: $\langle c, c' \rangle \in \preceq$ if and only if $c = c_0$ and $c(i) \leq c'(i)$ for each $1 \leq i \leq m$.

Let $B$ be an upward-closed set, defining the set of configurations that should be avoided in an execution. With the backward reachability analysis, the execution of a system is safe if and only if there is no $c_0$ such that $c_0 \in \text{BackReach}(B)$, where $c_0$ is an initial configuration and BackReach$(B)$ denotes the set of configurations that can be backwardly reached from $B$. The mutual exclusion property of the load balancing system shown in Figure 1 can be stated as no initial configuration $c_0 \models \langle s_{idle}, n_{req} = \omega, n_{use} = 0, n_{low} = 0, n_{high} = 0 \rangle$. As for the verification of parameterized systems, we focus on those problems that can be reduced to the backward reachability analysis from an upward-closed set. Given a quasi-ordering relation (reflective and transitive), a set $S \subseteq X$ is upward-closed if every $y \geq x \in S$ entails $y \in S$.

In addition to explicitly presenting the set of bad configurations, we can use a safety property to regulate the behaviors of a parameterized system. In the automata-theoretic approach, a linear safety property will be represented as a regular set of dangerous traces accepted by a FSA [18, 10]. The model checking procedure conducts a reachability analysis on the synchronous product of the concurrent system and the FSA to verify whether a safety property holds. Given an automaton $A = \langle Q_A, \Sigma, \delta_A, q_0, F_A \rangle$ and a parameterized system $B = \langle M, P, \Sigma \cup \{ \tau \} \rangle$, the combined system is defined as follows: the configuration set is a subset of $Q_A \times Q_M \times \{ \omega \}_m$ and $(q_A, c_0) \rightarrow (q_A', c_0')$ if and only if $\delta_A a \rightarrow q_A'$ and $c_B a = c_B'$ in $B$.

Whether the set of bad configurations is presented explicitly or not, it does not make a difference to the model checking algorithm. In the latter, the set of bad configurations is the upward-closed set $\uparrow \{ F_A \times Q_M \times \{ 0, \cdots, 0 \} \}$. It’s still explicit with regard to the newly generated combined system. In the sequel, we only focus those properties explicitly defined by a set of bad configurations.

### 3. BRUTE-FORCE SEARCH

#### 3.1 Overall Approach

**Definition** A well-quasi-ordering (a wqo) is any quasi-ordering relation (reflective and transitive) $\preceq$ over some set $X$ such that, for any infinite sequence $x_0, x_1, x_2, \cdots$ in $X$, there exists indexes $i < j$ with $x_i \preceq x_j$.

**Definition** A well-structured transition system (WSTS) is a transition system $S = \langle S, \rightarrow, \preceq \rangle$ equipped with a $\preceq \subseteq S \times S$ such that the two following conditions hold:

- well-quasi-ordering: $\preceq$ is a wqo, and
- compatibility: $\preceq$ is (upward) compatible with $\rightarrow$, i.e. for all $s_1 \preceq s_2$ and transition $s_1 \rightarrow s_3$, there exists a sequence $t_1 \rightarrow t_2$ such that $s_2 \preceq t_2$.

The quasi-ordering relation $(\preceq, C)$ over a parameterized system defined previously, $(c, c') \in \preceq$ if and only if $c = c_0$ and $c(i) \leq c'(i)$ $(1 \leq i \leq m)$, is well-ordered because of the following theorem:

**Lemma 3.1** (Dickson’s Lemma, [5]). Let $v_1, v_2, \cdots$ be an infinite sequence of elements of $\mathbb{N}^k$. There exists $i < j$ such that $v_i \preceq v_j$ (pointwise order).

With regard to the quasi-ordering $(\preceq, C)$, the upward compatibility condition obviously holds in a parameterized system. It is because that there must exist $c'' \in C$ such that $c' \rightarrow c''$ for each $c_1$ and $c_2$ such that $c_1 \preceq c_2$ and $c_1 \rightarrow c_2$.

As a result, a parameterized system is also a well-structured system. The following backward reachability analysis [13] of well-structured systems can be used for analyzing parameterized systems. The algorithm will terminate because of the well-quasi-ordering condition in the previous definition, otherwise an infinite sequence in which there are no two elements $c_i$ and $c_j$ $(i < j)$ such that $c_i \preceq c_j$.

**BackReach** $(S_0)$ // an upward-closed set of targeted configurations

```plaintext
1: let $S_{new} \leftarrow S_0, S_{old} \leftarrow \emptyset$
2: while $S_{new} \neq \emptyset$ do
3: $S_{old} \leftarrow S_{new}$
4: $S_{new} \leftarrow \text{predecessors}(S_{old})$
5: $S_{new} \leftarrow S_{new} \cup S_{old}$
6: end while
7: return $S_{new}$
```

Figure 2: Backward Reachability Analysis

In this paper, we use a slightly different backward reachability algorithm. Unlike the previous algorithm running until a fixed-point is reached, it will terminate when an initial configuration $s$, such that $s \models \langle s_M, n_{req} = \omega, 0, \cdots, 0 \rangle$, is reached. The BFS version of the algorithm is presented in Figure 3. The construction of a corresponding DFS version is rather straightforward. The set of bad configurations, $\text{bad-States}$, is an upward-closed set. The procedure $\text{PRE}$, called in $\text{BBFS}$, returns a given set’s immediate predecessors.
\textbf{BBFS} \,(\text{badStates}) \quad // \text{a set of bad configurations}

1: \text{visitedStates} := \emptyset

2: \text{workSet} := \text{badStates}

3: \textbf{while} \text{workSet} \text{ is not empty } \textbf{do}

4: \text{predecessors} := \text{PRE(workSet)}

5: \text{visitedStates} := \text{visitedStates} \cup \text{workSet}

6: \text{workSet} := \text{predecessors}

7: \textbf{for each } \text{s} \in \text{workSet} \textbf{ do}

8: \textbf{if } a? \text{ \textbf{visitedStates} then}

9: \text{\textbf{print counterexample}_{\text{shortest}}}

10: \textbf{end if}

11: \textbf{if } \exists \text{s} \in \text{visitedStates} \text{ such that } \text{s} \not\preceq \text{s} \text{ then}

12: \text{\textbf{delete s from workSet}}

13: \textbf{end if}

14: \textbf{end for}

15: \textbf{end while}

\textbf{Figure 3: Backward BFS Algorithm}

3.2 Procedure \text{PRE}

In the verification of parameterized systems, the set of bad configurations and all its predecessors are upward-closed, but tend to be infinite. To make the backward reachability analysis be practical, those sets have to be represented in a finite space. Fortunately, any upward-closed set has a finite basis because of the well-quasi-ordering. At the same time, the set of immediate predecessors of an upward-closed set is also upward-closed and has a finite basis. Based on these results, the procedure \text{PRE} restricts the backward reachability analysis to operations on the finite bases of those potential infinite sets. Being specific to the parameterized system defined in Section 2 allowing rendezvous, broadcast, and non-determinism, we present a procedure \text{PRE} in Figure 6. In the procedure, each element in the current set’s basis is enumerated and those configurations that can be reachable from those elements by rendezvous or broadcast form the basis of the current set’s immediate predecessors.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure4.png}
\caption{Predecessors of Rendezvous Actions}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure5.png}
\caption{Predecessors of Broadcast Actions}
\end{figure}

When a rendezvous action is carried out by the monitor and the processes, there are two ways to construct an predecessor for each element in \text{currentBasis}. Let \text{c} be the current configuration. Without lost of generality, we assume that the monitor issues the action \text{a}! and one of the processes will respond with the action \text{a}?. As shown in Figure 4, \text{c}(k : n_k + 1, l : n_l - 1) will be the predecessor of \text{c} if \text{c}(l) > 0 and there is a transition \text{c}_{n} \rightarrow \text{c}_{s} in the process. In the figure, another transition \text{s}_{n} \rightarrow \text{s}_{j} is also enabled, but with \text{c}(j) = 0. It is obvious that \text{c}(i : n_i + 1, j : n_j - 1) could not be the predecessor of \text{c} because \text{n}_j - 1 will be less than zero when \text{n}_j = 0. However, the current set that is reachable from the initial set of bad configurations is the upward-closed set \uparrow \text{currentBasis}. Thus, \text{c}(j : u_j + 1) is in the set \uparrow \text{currentBasis} because of \text{c} \leq \text{c}(j : u_j + 1) and is the successor of \text{c}(i : u_i + 1). As a result, we know that \text{c}(i : u_i + 1) is the predecessor of \text{c}(j : u_j + 1) and is one of predecessors of \uparrow \text{currentBasis}.

4. HEURISTIC SEARCH

The algorithm presented in the previous section will terminate as the necessity outcome of the well-quasi-ordering of well-structured systems. When the search procedure terminates, a property is violated if an legitimate initial configuration is reached or it holds when the state space is exhausted and no initial configurations are visited. As for the formal model checking of systems, the time and space efficiency of an algorithm has always been a cause for concern.
The execution time and number of reachable configurations of the BBF S algorithm in Figure 3 and its corresponding DFS version, running on the load balancing example, are presented in Figure 7. The verification accomplished by the algorithm is universal in the sense that a conclusion will hold and be conclusive, regardless of the number of identical processes. Instead of sampling the size, number of processes, of a concurrent system, the x-axis consists of seven different properties that are verified. Those properties are explicitly specified by an upward-closed set with a finite basis and the principles of selecting those properties will be thoroughly discussed shortly. What we need to know at this point is that the properties become more and more complicated as the index is increased. As shown in Figure 7, the execution time and reachable configurations grow dramatically with regard to the complexity of the property to be verified. It is analogous with the experience of model checking of normal concurrent systems, in which the complexity is an exponential function of the size of a system.

4.1 Number of States In $s^0_P (H_\Phi)$

In this heuristic rule, the promise of an intermediate configuration is calculated based on the number of processes in the state $s^0_P$, the initial state of the identical process. As for the algorithm in Figure 3, it will terminate when an initial configuration $c$, such that $c \preceq (s^0_M, n_1, \ldots, n_m)$, is reached. Given an intermediate configuration $c' = (s_M', n_1', \ldots, n_m')$, the algorithm intends to find a series of backward transitions $a_1, \ldots, a_m$ and an initial configuration $c'' = (s_M', n_1', \ldots, \theta m)$ such that $c'' \preceq \ldots \preceq c'$. According to the procedure $PRE$, we will have $n_1' = \sum_{i=1}^{m} n_i + \theta$ and $\theta$ is the number of rendezvous actions conducted during the backward search that are labeled with a double circled 1 as illustrated in Figure 4. A natural idea is that the configuration, having a larger $n_1$, will be picked up first. Thus, the evaluating function of a configuration $c$ is defined as:

$$f(c) = c(1), \text{number of processes in } s^0_P$$

4.2 Internal Height ($H_{IH}$)

Compared to a backward rendezvous action, a broadcast action may result in a dramatic increase of reachable intermediate configurations during the backward search. It gets worse when a state in $\mathcal{P}$, the identical process, can backwardly reach to multiple states when a broadcast action is...
conducted. Given a configuration \( c = (s_M, \cdots, n_k, \cdots, n_j = \lambda, \cdots, n_i, \cdots) \) where \( n_i = 0 \) for each \( i \neq j \) \((0 \leq i \leq m)\) and there is a broadcast action \( a \) such that \( s_k \xrightarrow{a??} s_j \in \delta_P \), \( s_j \xrightarrow{a?} s_j \in \delta_P \) and \( s_M \xrightarrow{a!!} s'_M \). According to the procedure PRE, the following configurations will be added to the list of \( c \)'s immediate predecessors:

\[
\begin{align*}
&s'_M, \cdots, n_k = 0, \cdots, n_j = \lambda, \cdots, n_i = 0, \cdots \\
&s'_M, \cdots, n_k = 1, \cdots, n_j = \lambda - 1, \cdots, n_i = 0 \\
&s'_M, \cdots, n_k = 0, \cdots, n_j = \lambda - 1, \cdots, n_i = 1 \\
&s'_M, \cdots, n_k = 2, \cdots, n_j = \lambda - 2, \cdots, n_i = 0 \\
&s'_M, \cdots, n_k = 1, \cdots, n_j = \lambda - 2, \cdots, n_i = 1 \\
&s'_M, \cdots, n_k = 0, \cdots, n_j = \lambda - 2, \cdots, n_i = 2 \\
&\cdots \cdots \\
&s'_M, \cdots, n_k = \lambda, \cdots, n_j = 0, \cdots, n_i = 0 \\
&s'_M, \cdots, n_k = \lambda - 1, \cdots, n_j = 0, \cdots, n_i = 1 \\
&\cdots \cdots \\
&s'_M, \cdots, n_k = 1, \cdots, n_j = 0, \cdots, n_i = 1 \\
&s'_M, \cdots, n_k = 0, \cdots, n_j = 0, \cdots, n_i = \lambda 
\end{align*}
\]

In the list, the value \( n_j \) is incrementally distributed to \( n_k \) and \( n_i \). There is no two configurations \( c' \) and \( c'' \) in the list such that \( c' \preceq c'' \) or \( c'' \preceq c' \). All configurations listed above should be added to the set of \( c \)'s immediate predecessors. The number will get larger when there are more states in \( \mathcal{P} \) that can transit to \( s_j \) through the broadcast action \( a?? \).

To speed up the search process, we introduce a rule that attempts to pick up those configurations which have a steeper increase or decrease with regard to their predecessors first. The evaluating function \( f(c) \) is defined as:

\[
f(c) = \max_{i=1}^{m} c(i) - \min_{i=1}^{m} c(i)
\]

### 4.3 Beam Search (HBS)

In the best-first search, an intermediate configuration with the largest \( f(c) \) will be unfolded and moved from the open list to the closed list and its immediate predecessors are added to the open list. Then another configuration with the largest \( f(c) \) in the open list will be selected and unfolded again, and so on. The search radius is always equal to one. However, the beam search, with a parameter \( m \), will select \( m \) best configurations from the open list and add their predecessors to the open list. We also apply the beam search to the verification of parameterized systems. It is shown that the efficiency can be further improved when the parameter \( m \) is appropriately selected.

### 4.4 Implementation Issues

In general, the heuristic search algorithms discussed above are not complete, in the sense that the targeted configuration will be reached if there exists one. To get a complete solution, those configurations that are not selected are also stored in the open list and they will be finally visited if those configurations having a larger promise can not lead to initial configurations. The open list is a ordered list with regard to the promise obtained through various heuristic rules. A semi-search algorithm is used to speed up the locating process when a new configuration needs to be added to the list.

However, those search algorithms are still not optimal. In this paper, we only care about whether a property holds other than the minimum bound of the number of identical processes in a parametrized system enabling the existence of a counter-example.

### 5. EXPERIMENT RESULTS

#### 5.1 Methodology

To demonstrate the improvement of efficiencies with those heuristic rules, the following four parameterized systems are used as experiment systems: the Load Balancing (LB) system in Figure 1, the Broadcast Protocol (BP) [12], the Shared File Access (SFA) system [6], and the Exclusive CPU Access (ECA) system [5]. The shared file access system used here is a simplified version of the original system in [6]. We consider the case there is only one shared file in the system instead of two for the reason of simplicity.

Compared to the semantics of parameterized systems defined in Section 2.2 in which an output broadcast action is not allowed in the identical process \( \mathcal{P} \), several output broadcast actions, such as \( \text{lock!!} \), \( \text{unlock!!} \), \( \text{reada!!} \) and \( \text{writea!!} \), appear in the processes of LFAF and EAC. A transition with an output broadcast action, like \( s_1 \xrightarrow{a??} s_2 \), can be removed from \( \mathcal{P} \) by the following transformation rules: if the monitor is empty, in the case of LFAF and EAC, then add a state \( s \) to \( Q_M \) and a transition \( s \xrightarrow{!!} (s \xrightarrow{!!}) \) in \( \delta_P \); add a new state \( t \) to \( Q_P \), a new action \( \text{sig} \) into \( \Sigma \), and two transitions \( s_1 \xrightarrow{\text{sig}!!} t, t \xrightarrow{!!} s_2 \) to \( \delta_P \); delete the transition \( s_1 \xrightarrow{a??} s_2 \) from \( \delta_P \); add a new state \( t \) to \( Q_M \) and two transitions \( s \xrightarrow{\text{sig}!!} t, t \xrightarrow{a!!} \delta_M \) for each state \( s \in Q_M \) other than the newly added state \( t \).

#### Table 1: Parameterized Systems in the Experiment

| Name                          | \( |Q_M| \) | \( |Q_P| \) | Property       |
|-------------------------------|--------|--------|---------------|
| Load Balancing (LB)           | 2      | 4      | (low, high)   |
| Broadcast Protocol (BP)       | 3      | 3      | \( \langle c_1, c_2 \rangle \) |
| Shared File Access (SFA)      | 3      | 6      | \( \langle I, S_A \rangle \) |
| Exclusive CUP Access (ECA)    | 3      | 5      | \( \langle \text{use, wait} \rangle \) |

#### Table 2: Safety Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>D5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle c_1, c_2 \rangle )</td>
<td>( (5, 10) )</td>
<td>( (6, 6) )</td>
<td>( (7, 7) )</td>
<td>( (8, 8) )</td>
<td>( (9, 9) )</td>
</tr>
<tr>
<td>( \langle I, S_A \rangle )</td>
<td>( (2, 2) )</td>
<td>( (3, 3) )</td>
<td>( (4, 4) )</td>
<td>( (5, 5) )</td>
<td>( (6, 6) )</td>
</tr>
<tr>
<td>( \langle \text{use, wait} \rangle )</td>
<td>( (5, 5) )</td>
<td>( (6, 6) )</td>
<td>( (7, 7) )</td>
<td>( (8, 8) )</td>
<td>( (9, 9) )</td>
</tr>
</tbody>
</table>

After necessary transformations, the basic information of those parameterized systems used in the following experiments is listed in Table 1. \( |Q_M| \) and \( |Q_P| \) denote the size of \( Q_M \) and \( Q_P \), respectively. The Property column specifies the properties to be verified. As an example, the safety property for the load balancing system will be an upward-closed

---

1Further details about those systems and the raw data collected during the following experiments can be seen from [http://www.xxx.xxx.xx/xxx.html](http://www.xxx.xxx.xx/xxx.html)
set \( \langle s_{bus}, 0, 0, n_{low}, n_{high} \rangle \). Each assignment to \( n_{low} \) and \( n_{high} \) corresponds to a safety property. The state space needed to be explored during the backward search will grow dramatically when \( n_{low} \) and \( n_{high} \) increase. The adjustable state space provides a perfect platform to demonstrate that a search algorithm equipped with those heuristic rules outperforms those without heuristic rules. Thus, we focus on the sort of properties listed in Table 2. Those properties do not hold in corresponding experiment systems except that the property \( \langle use, wait \rangle \) holds in the the Exclusive CPU Access (ECA) system.

Those algorithms discussed previously have been implemented in JAVA. Each data sample collected in the experiment consists of the execution time and the number of reachable configurations of an algorithm running on a parameterized system to verify a certain safety property. The programs are run on a IBM ThinkPad platform with a DualCore T2400@1.83GHz CPU and 2GB of memory running Windows XP. As for the measurement of an algorithm’s execution time, the JNI technology is employed to directly measure the execution time of a JAVA thread instead of the CPU time of the computer system. A relatively more accurate result is expected as the interference from other processes running in the same computer system is separated out.

5.2 Results

Instead of using the raw data to present the result, we use ratios of one algorithm’s performance data to another one’s. The ratios clearly tell which algorithm is more or less efficient than another. The performance data consists of the execution time and the number of reachable configurations of a backward search. In general, the improvement gained by a heuristic algorithm may vary from one parameterized system to another and sometimes it may fluctuate dramatically, as shown later. For each comparison, the result will be presented in two separated figures and those systems having a similar range of gain are drawn in the same figure.

As for the time efficiency of the two algorithms, BBFS outperforms BDFS in the exclusive CPU access (ECA) system and, however, BDFS does better in the load balancing (LB) system. They end almost in a tie in the broadcast protocol (BP). The most interesting one is the trend \( T(SFA) \) shown in Figure 9. BDFS has a higher efficiency than BBFS when the property is relatively simple and the time ratio plummets when the property get complicated. On the other side, the number of reachable configurations shows almost the same trend except that the \( RC(SFA) \) remains flat when \( T(SFA) \) drops rapidly.

![Figure 9: BFS vs. DFS (2)](image)

**Result 2:** With the heuristic rules \( H_{IH} \) and \( H_{sp} \), the efficiency can be greatly improved and even some-
times dramatically.

Compared to the brute-force backward search algorithm, a heuristic search continually expands the node with the largest promise calculated by the corresponding evaluation function other than the first one in the open list or a randomly selected one. In general, it works in the same way as a DFS algorithm does except that the visiting order of intermediate nodes is different. To measure the effect of the introduction of a heuristic rule, we use the ratio of a DFS search’s performance data to the corresponding heuristic search’s data as the $y$-axis of the figure. A larger ratio implies that a better result is obtained by introducing the heuristic rule.

The experiment results of $H_{IH}$ are shown in Figure 10 and Figure 11. With the introduction of $H_{IH}$, the DFS algorithm’s efficiency is obviously increased in the load balancing (LB) system and the broadcast protocol (BP). The increase ranges from 25 percent to almost 150 percent. The impressive increase comes from the share file access (SFA) system, in which the improved algorithm is more than eleven times faster than the original one. Although the change is not remarkable, it is really improved in the exclusive CPU access (ECA) system.

The heuristic rule $H_{IH}$ is rather simple and to an extent intuitive. From what’s shown in Figure 12 and Figure 13, however, we can see that the result is further improved on the basis of $H_{IH}$. The former outperforms the latter at each sampling point. Compared to the original DFS algorithm, the verification of the load balancing (LB) system against the safety property $(low, high) = (9,9)$ is almost two orders of magnitude faster and only about fifty percent memory is used. To clearly present the time and space ratios in a graph, we use two $y$-axes in Figure 13. The time ratios marked by a filled black circle refers to the left $y$-axis while the space ratios marked by a blank triangle refers to the right one.

Result 3: With the beam search, a further improvement is possible as long as the parameter, $m$, is appropriately selected.

The beam search can be integrated with any heuristic or non-heuristic DFS algorithms. To demonstrate that a further improvement is possible, we apply both the beam search algorithm and the heuristic rule $H_{IH}$ to the original DFS algorithm. In the experiment, we gradually adjust the parameter $m$, the number of states to expand at each iteration, to see how the performance will change.

The final result is shown in Figure 14. The line $T(SFA)$ connects those data points, collected when the shared file access (SFA) system is verified against the property $(low, high) = (9,9)$ and the parameter $m$ changes from 1 to 100 with a step length 5. The second line $T(LB)$ is associated with the verification of the load balancing (LB) system against the property $(I, S_a) = (6,6)$. The first data point with $m = 1$ is actually the execution time when only the heuristic rule $H_{IH}$ is used. From the figure, we can see that a maximize optimization is got when $m$ is about 70 in the LB system, and 65 in the SFA system. Compared to the the heuristic rule $H_{IH}$, the efficiency is increased by 78.4 and 62.5 percent, respectively.

The problem with the beam search is that, unlike the heuristic rules $H_{IH}$ and $H_{IH}$, its performance highly depends on the parameter $m$.

Result 4: Although the heuristic algorithms are compared to the concrete backward reachability algorithm in which an upward-closed set of configurations is symbolically represented by a NA-constraint, the model checking efficiencies can also be greatly improved by the introduction of heuristic rules when DV-constraints are used.

A NA-constraint is a conjunction of atomic constraints of the form $n_i \geq k$, where $n_i \in \{n_1, \ldots, n_m\}$ and $k$ is a positive integer. In the procedure PRE, we actually use the disjunction of a set of NA-constraints to represent the preimage of an upward-closed set because $\neg c$ is a NA-constraint, where $c$ is one of minimal elements of the preimage. A DV-constraint is a constraint with additions of the form

$x_{1,1} + \cdots + x_{1,n_1} \geq k_1 \land \cdots \land x_{m,1} + \cdots + x_{m,n_m} \geq k_m$
where \( x_{i,j} \) and \( x'_{i',j'} \) are distinct variables for all \( i, j, i', j' \).

The full strength of DV-constraints comes from the fact that a set of NA-constraints may be represented by only one DV-constraint.

As discussed in [4], the containment problem for NA-constraints, whether a set of concrete configurations represented by one set of NA-constraints is a subset of another set of NA-constraints, can be solved in polynomial time. However, the cardinality of \( \text{pre}_{NA}(\Phi) \), the preimage of a NA-constraint, is \( O(|\Phi| \ast a \ast C_{n+e}) \) where \( n, a \) are the number of states and actions in a parameterized system and \( c \) is the biggest constant occurring in \( \Phi \). It leads to an exponential blow-up. It explains that the searched state space grows dramatically when the properties listed in Table 2 go from left to right because a larger initial assignment will lead to a larger \( c \). Although the containment problem for DV-constraints is co-NP Complete, the entailment problem, whether a set of configurations represented by one NA-constraint is a subset of another NA-constraint, can be solved in polynomial time. The cardinality of \( \text{pre}_{DV}(\Phi) \) is \( O(|\Phi| \ast a \ast c') \).

We also investigated what if an upward-closed set is represented by a DV-constraint. Another version of the procedure \( \text{PRE} \) for deterministic parameterized systems is implemented. The procedure will get much more complicated and the approach based on DV-constraints becomes unattractive when non-determinism is introduced. It is because that the affine transformation matrix does not exist and extra time and space have to be used for transforming non-DV-constraints created in each backward propagation into DV-constraints. It is similar to the transformation from a \( \text{U} \)-formula to a normal disjunctive formula as discussed in [8], facing an exponential blow-up.

As a conclusion, the DV-constraint approach does not certainly outperforms the NA-constraint one. In Figure 16, the former does much better than the latter in the the \textit{Shared File Access (SFA)} experiment system. However, the NA-constraint approach spends less time in the \textit{Broadcast Protocol} experiment system as shown in Figure 15. Figure 17 shows the improvement of the DV-constraint approach in SFA brought by the heuristic rules \( H_{\Phi} \) and \( H_{IH} \). We also use two y-axes to clearly present the result. It can be seen that the improvement brought by the heuristic rules is overwhelming in SFA. We also made similar comparisons on \( BP \) and \( ECA \), another two deterministic experiment systems listed in Table 1, but the improvement is not obvious. It is because that the SFA system has more broadcast input and output actions resulting in a lot of backtrack searches. The heuristic rules help the algorithm to find the path with the most promise.

6. RELATED WORK

Several abstract backward reachability algorithms have been proposed for verifying well-structures systems [13] and infinite-state systems [1]. To calculate the finite basis of an upward-closed set’s immediate predecessors, the authors in [12] give a pseudo concreate procedure based on the rules: 

\[
\text{c} \cdot \text{e} \Rightarrow \text{c} \cdot \text{e} \cdot \text{r} + \text{v}_a = \text{c} \cdot \text{r},
\]

where \( \text{e} \) and \( \text{e}' \) belong to the basis of an upward-closed set and its immediate predecessor, respectively. \( \text{M}_a \) and \( \text{v}_a \) are the transformation matrix and vector associated with an action \( a \). One problem in the algorithm is that the second rule does not always hold. As an example, if a property is specified by an automata \( A \), then the set of bad states is the upward-closed set \( \uparrow S_b = \{ (q, q', n_1 = 0, \ldots, n_m = 0) \mid q \in F_A, q' \in Q_M \} \). Let \( \uparrow S_b' \) be the set of predecessors of \( \uparrow S_b \). It is impossible that there exists \( \text{c} \in S_b \) and \( \text{c}' \in S_b' \) such that \( \text{M}_a \cdot \text{c} \cdot \text{r} + \text{v}_a = \text{c} \cdot \text{r} \).

If \( \text{v}_a \) is a vector for the transition \( s_i \rightarrow s_j \), then we have \( \text{v}_a = \{ n_1 = 0, \ldots, n_j = 1, \ldots, n_m = 0 \} \).

Then, \( c(i) \) can not be zero because each element in \( M_a \) and \( c' \) is greater or equal than zero and the entailment of \( c'(i) = 0 \) requires that \( (M_a \cdot c' \cdot r)(i) = 0 \) in addition, the authors only consider those parameterized systems in which an action, such as \( \text{a}, \text{a}!' \), can only appear once in all transitions and the whole system is deterministic. As for the concrete algorithm proposed in this paper, those limitations are dis-
cared and non-determinism is allowed in the monitor and the identical process.

As discussed previously, several symbolic approaches based on constraint logic programming have been investigated for verifying parameterized systems with an infinite-state space [7, 4, 8]. They are still suffering from the symbolic state explosion and the co-NP complexity of containment tests. In [3], the author uses a BDD-based approach to verify parameterized systems through iteratively computing the backward reachability set for constituent systems of increasing size until a certain convergence condition is reached. Since a parameterized system with specific size has a finite state space, the BDD approach can be used in each iteration. However, the convergence condition is only applicable to some special kinds of systems, called δ-defectable DWSTS (Discrete Well Structured Transition Systems), and the sharing tree approach outperforms it by sever orders of magnitude. The heuristic rules presented in this paper can mitigate the inefficiencies of these existing approaches.

In [9], the authors propose a new heuristic rule based on the structural invariant of Petri Nets for parameterized systems that can be modeled as Petri Nets and (Lossy) Vector Addition Systems. The underlying symbolic structure is based on the sharing tree. We are preparing to investigate what if our heuristic rules are applied on those systems listed in [9].

7. CONCLUSION

Parameterized systems consisting of arbitrary number of identical processes are commonly seen in various application domains. The verification of such kind of systems is universal in the sense that a conclusive conclusion needed to be drawn regardless of the number of identical processes. The dynamic creation of identical processes contributes to the symbolic state explosion in several constraint-based symbolic model checking approaches. At the same time, the containment problem, one of the most important and time-consuming operations during backward reachability analysis, tends to be co-NP complete.

In this paper, we firstly propose a new concrete procedure PRE to calculate the finite-basis of an upward-closed set’s immediate predecessors for parameterized systems. To simplify the procedure, the output broadcast action is forbidden to appear in the identical process. With the rules presented in Section 5.1, an output broadcast transition can be removed from the identical process and replaced by a new transition in the monitor. Thus, the constraint does not result in a loss of expressiveness. In addition, the procedure allows non-determinism in the monitor and the identical process. With a computable procedure PRE, the verification of parameterized systems against safety properties will be transformed into standardized backward reachability analysis. To speed up the backward search process, several heuristic rules are investigated. Although, in general, the performance of a heuristic rule is to an extent related to the system to be verified, the experiments show that the efficiency for each experiment system is significantly improved and sometimes the surge is tremendous.

8. REFERENCES

[16] F. Rossi. Constraint (logic) programming: A survey on research and applications. In Selected papers from

