A Complete Bounded Model Checking Algorithm for Pushdown Systems

Gérard Basler*, Daniel Kroening, and Georg Weissenbacher**

Computer Systems Institute, ETH Zurich, 8092 Zurich, Switzerland
{firstname.lastname}@inf.ethz.ch

Abstract. Pushdown systems (PDSs) consist of a stack and a finite state machine and are frequently used to model abstractions of software. They correspond to sequential recursive programs with finite-domain variables. This paper presents a novel algorithm for deciding reachability of particular locations of PDSs. We exploit the fact that most PDSs used in practice are *shallow*, and propose to use SAT-based Bounded Model Checking to search for counterexamples. Completeness is achieved by computing *universal summaries* of the procedures in the program.

1 Introduction

Pushdown systems (PDSs) consist of a finite state machine and stack with a finite set of stack symbols. The use of PDSs as abstractions of software was promoted by the Slam project at Microsoft Research. PDSs are equally expressive as *Boolean Programs* [1]. The finite-state part of the PDS is used to model the global variables and the control location of the Boolean program. The stack can be used to model the call stack of the program, which permits unbounded recursive function calls.

We present a novel algorithm for deciding reachability of particular error locations of PDSs. Our algorithm is tailored to the verification of automatically generated abstractions of commodity software within a program analysis tool such as Slam or SatAbs.

We record two observations about such systems. First of all, most tools that implement an abstraction/refinement framework compute a *conservative abstraction*. If the property holds on the abstract model, it also holds on the original program. As a consequence, the abstraction/refinement loop terminates as soon as an abstraction is built in which the error location is unreachable. In all previous iterations, there exists a path that reaches an error location. It is reported in [2] that the verification of device drivers may require as much as twenty iterations, with an average of 5 iterations. This motivates the need for an algorithm that performs well on models with a counterexample.

* Supported by the Swiss National Science Foundation.
** Supported by Microsoft Research through its European PhD Scholarship Programme.

K. Yorav (Ed.): HVC 2007, LNCS 4899, pp. 202–217, 2008.

Second, we exploit the fact that pushdown systems generated as abstractions of commodity software are typically very *shallow*. Formally, this means that any node of the Kripke structure that is reachable from an initial state is reachable with a few steps.

Bounded Model Checking (BMC) is a perfect fit for this scenario. In BMC, a transition system is unwound together with a property up to a given bound k to form a propositional formula. The formula is satisfiable if and only if there exists a path of length k that refutes the property. If the formula is unsatisfiable, BMC is inconclusive as longer counterexamples might still exist. SAT-based BMC is therefore known as an effective method to discover shallow bugs.

In order prove the absence of errors, we extend BMC with *procedure summarization* [3]. A procedure summary maps a configuration of a PDS at the entry of a procedure to the set of configurations observable upon exit. As there are only finitely many summaries, saturation can be used to compute the set of reachable states [4,5].

Contribution and Outline. We present preliminaries of PDSs, summarization, and symbolic representations in Sec. 2. This paper extends the concept of *Universal Summaries* [6], which are described in Sec. 3. This paper makes two contributions:

1. We present a novel abstraction-based procedure to obtain universal summaries of non-recursive procedures in Sec. 4. The abstraction is refined on-demand if required by a spurious counterexample.
2. We show how to obtain universal summaries for recursive procedures by means of a head-counting argument in Sec 5.

2 Preliminaries

Definition 1 (Pushdown Systems [7]). *A* pushdown system *(PDS) is a transition system in which each state comprises of a control location and a stack. The set of control locations P as well as the stack alphabet Γ is finite. A state is a pair $\langle p, w\rangle$, where $p \in P$ denotes a control location, and $w \in \Gamma^*$ represents the stack content. We use s_0 to denote the initial state of a PDS. The* head *of a state comprises of the control location and the topmost element of the stack. The state space of a PDS may be infinitely large, since the stack height is not bounded. The number of heads is bounded by $|P| \cdot |\Gamma|$.*

The transition relation is defined in terms of a finite set of rules Δ. These rules determine the successors of a state $\langle p, \gamma w\rangle$ $(w \in \Gamma^*)$ based on the head $\langle p, \gamma\rangle$ of this state. Each rule is of the form $\langle p, \gamma\rangle \hookrightarrow \langle q, w\rangle$, where $|w| \leq 2$ (in particular, w may be ϵ, $|\epsilon| = 0$). Depending on the size of w, we distinguish between expansion *rules,* neutration *rules, and* contraction *rules (see Fig. 1). The transition relation $\rightarrow \subseteq (P \times \Gamma^*) \times (P \times \Gamma^*)$ is defined as follows: If $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle \in \Delta$, then $\langle p, \gamma w'\rangle \rightarrow \langle p', ww'\rangle$.*

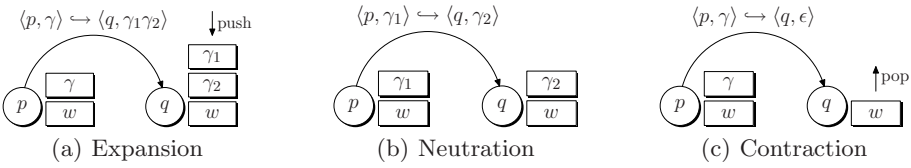


Fig. 1. Transitions of a Pushdown System

We use $\rightarrow^*$ to denote the reflexive transitive closure of $\rightarrow$. A state s_i of a PDS is reachable iff $s_0 \rightarrow^* s_i$. The set of reachable states of a PDS can be represented by means of a finite automaton [8], which may be obtained using a saturation procedure [4,5]. Intuitively, an expansion $\langle p_0, \gamma_0\rangle \rightarrow \langle p_1, \gamma_1\gamma_2\rangle$ followed by neutration that yields $\langle p_2, \gamma_3\gamma_2\rangle$, followed by a contraction $\langle p_2, \gamma_3\gamma_2\rangle \rightarrow \langle p_3, \gamma_2\rangle$ can be *summarized*[1] by $\langle p_0, \gamma_0\rangle \rightarrow^* \langle p_3, \gamma_2\rangle$. Augmenting the transition relation with this summary may give rise to new summaries. The set of states reachable from s_0 is computed by repeatedly applying summarization until a fixpoint is reached (i.e., no new summaries can be constructed).

The efficiency of these saturation based algorithms can be significantly improved by using a *symbolic* BDD-based representation of the PDS. The notion of symbolic PDSs was introduced by Schwoon [7]:

Definition 2 (Symbolic Pushdown Systems). Symbolic pushdown systems *use a propositional encoding for control locations $p \in P$ and stack elements $\gamma \in \Gamma$. A set of $\lceil \log_2(|P|)\rceil + \lceil \log_2(|\Gamma|)\rceil$ Boolean variables is sufficient to encode all heads of a PDS. The right hand side $\langle q, w\rangle$ (where $|w| \leq 2$) of a rule is represented using a separate set of Boolean variables. We use primed variables to denote elements of the latter set. The symbolic rules Δ_S are expressed in terms of a propositional relation over primed and unprimed variables. We use $R^{\nearrow}$, $R^{\rightarrow}$, and $R^{\searrow}$ to refer to the relation for expansions, neutrations, and contractions, respectively.*

Assume, for instance, that P is represented using $\{a_0, a_1\}$, and Γ using $\{b_0, b_1\}$. One way to encode the neutration rule $\langle p_1, \gamma_1\rangle \hookrightarrow \langle p_2, \gamma_3\rangle$ is

$$(\overline{a}_1 \cdot a_0) \cdot (\overline{b}_1 \cdot b_0) \;\cdot\; (a'_1 \cdot \overline{a}'_0) \cdot (b'_1 \cdot b'_0)$$

i.e., to use the variables to represent a binary encoding of the indices i of $p_i \in P$ and $\gamma_i \in \Gamma$ (where a_0 and b_0 correspond to the lower bits). The set of symbolic relations $\Delta_S = \{R^{\nearrow}, R^{\rightarrow}, R^{\searrow}\}$ is a disjunctive partitioning of symbolic relations corresponding to the union of transition rules.

The same technique is used to represent summaries. The symbolic representation of the summary $\langle p_0, \gamma_0\rangle \rightarrow^* \langle p_3, \gamma_2\rangle$ is $\bar{a}_1 \cdot \bar{a}_1 \cdot \bar{b}_1 \cdot \bar{b}_1 \;\cdot\; a'_1 \cdot a'_1 \cdot b'_1 \cdot \bar{b}'_1$. The symbolic representation of summaries (and states) is more succinct than the explicit representation used in [5]. The symbolic summary

[1] The idea of summarization was introduced by Sharir and Pnueli [3].

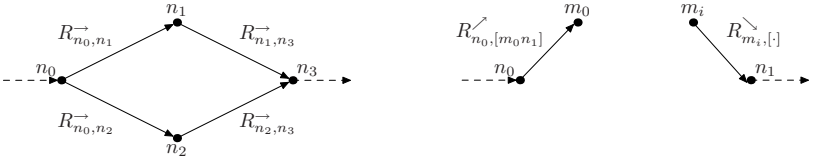
$$(\overline{a}_1 + a_0) \cdot (b_1 \cdot b_0) \ \cdot \ (a'_1 \cdot (a'_0 = a_1)) \cdot (b'_1 \cdot \overline{b}'_0)$$

stands for the three explicit summaries

$$\langle p_0, \gamma_3 \rangle \rightarrow^* \langle p_2, \gamma_2 \rangle, \langle p_1, \gamma_3 \rangle \rightarrow^* \langle p_2, \gamma_2 \rangle, \text{ and } \langle p_3, \gamma_3 \rangle \rightarrow^* \langle p_3, \gamma_2 \rangle.$$

Ordered BDDs are a suitable data structure to encode such formulas, since the canonical representation enables efficient fixpoint detection [7,1]. An alternative algorithm based on satisfiability solvers for propositional logic (SAT) and quantified Boolean formulas (QBF) is presented in [6].

Symbolic PDSs are a popular formalism to represent abstractions of programs [9,10]. Abstraction mechanisms like predicate abstraction [11] preserve the control flow structure of the original program. In that case, it is convenient to maintain an explicit representation of program locations, i.e., to discriminate the rules in Δ_S by their source and target nodes of the control flow graph (CFG).



(a) Conditional instruction (b) Function call

Fig. 2. Symbolic transitions rules with explicit CFG node information

Fig. 2 shows examples for symbolic transitions that are annotated with explicit control flow information (n_i and m_j are control flow nodes two different functions). From these pictures, it becomes intuitively clear how pushdown rules are used to model programs: The neutration rules in Fig. 2(a) model a conditional statement, whereas the expansion and contraction rules in in Fig. 2(b) model a function call and a return statement. A finite number of global (Boolean) variables is modeled by means of the control locations, and the (also finite) local variables of functions are represented using the stack alphabet Γ. For instance, the summary $\langle p_0, \gamma_0 \rangle \rightarrow^* \langle p_3, \gamma_2 \rangle$ modifies the control location (which represents the global variables of a program) and the topmost stack element. Intuitively, the return value of the corresponding function is passed on to the caller via the control location in this example. Throughout this paper, we use the notion of a *function* of a PDS to denote the transition rules associated with a function in the CFG. The correspondence indicates that PDSs are equally expressive as Boolean programs [9].

3 Universal Summaries

The reachability checking algorithms for PDSs presented in [1] and [6] are based on *symbolic simulation*. They determine the *reachability of a given head of a*

state (or a program location, respectively). This is sufficient to verify arbitrary safety properties of a PDS, even though it may be necessary to modify the PDS to state more complicated safety specifications. Each sequence of PDS transitions is represented by a propositional formula that is only satisfiable if the corresponding path is feasible.

Starting from s_0, the search algorithm simulates the transition system. It avoids getting caught in an infinite loop by keeping track of the heads it already visited. Whenever the algorithm encounters a contraction rule, it computes a summary for the corresponding expansion that matches head of the "calling context". This summary relates the heads of two states with the same stack height. The sequences of transitions that may form a summary are described by the grammar in Fig. 3.

$$\begin{aligned}&\text{Summary} &&::= R^{\nearrow}\ \text{Transitions}\ R^{\searrow}\ ;\\ &\text{Transitions} &&::= \text{Transitions Summary} \mid \text{Transitions}\ R^{\rightarrow} \mid \epsilon;\end{aligned}$$

Fig. 3. A grammar for summaries

Each new summary is added to a set Σ (using disjunction), such that summaries can be reused whenever the algorithm encounters a head and an expansion which are already covered. Eventually, Σ converges, since there are at most $|P|^2 \cdot |\Gamma|^2$ summaries that are not logically equivalent. The result is the least fixpoint of the set of summaries for the PDS, i.e., Σ contains only summaries for which the corresponding heads are indeed reachable.

An alternative to computing the set of reachable summaries Σ is to use *universal summaries* instead [6]:

Definition 3 (Universal Summary). *A* universal summary $\Sigma_{\mathcal{U}}$ *for a PDS* $\langle P, \Gamma, \Delta, s_0\}$ *is a relation* $\Sigma_{\mathcal{U}} \subseteq (P \times \Gamma) \times (P \times \Gamma)$ *such that*

$$\begin{aligned}\forall \langle p,\gamma\rangle, \langle p',\gamma'\rangle \in P \times \Gamma.\ (\exists \langle p_1, w_1\rangle, \ldots, \langle p_n, w_n\rangle \in P \times \Gamma^*.\\ \langle p,\gamma\rangle \rightarrow \langle p_1, w_1\rangle \rightarrow \ldots \rightarrow \langle p_n, w_n\rangle \rightarrow \langle p',\gamma'\rangle \wedge\\ \forall i \in \{1..n\}.|w_i| \geq 2) \iff \Sigma_{\mathcal{U}}(\langle p,\gamma\rangle, \langle p',\gamma'\rangle)\end{aligned}$$

holds.

Intuitively, for *any* head $\langle p, \gamma\rangle$, a universal summary subsumes *all* paths that "traverse a function" of the PDS, no matter whether there exists a reachable state $\langle p, \gamma w\rangle$ or not. The restriction $|w_i| \geq 2$ guarantees that $\Sigma_{\mathcal{U}}$ relates expansions to their matching contractions (according to the grammar in Fig. 3). Note that this definition does not rule out *nested* summaries. (A summary is nested if it is entirely contained in another summary according to the grammar in Fig. 3.) In particular, it does not exclude *recursion*, i.e., a summary $\langle p_0, \gamma_0\rangle \rightarrow^* \langle p_0', \gamma_0'\rangle$ may stem from a sequence of transitions that contains a

nested summary $\langle p_1, \gamma_1 \rangle \rightarrow^* \langle p_1', \gamma_1' \rangle$ such that $p_0 = p_1$, $\gamma_0 = \gamma_1$, $p_0' = p_1'$, and $\gamma_0' = \gamma_1'$.

In the following section, we discuss how symbolic summaries are computed. Based on this, we present an algorithm that computes symbolic universal summaries for recursion-free PDSs in Section 3.

3.1 Computing Symbolic Summaries

A symbolic model checking algorithms for PDSs represents a sequence of transitions by a propositional formula that is only satisfiable if the corresponding path is feasible. For instance, the path

$$\underbrace{\langle p_0, \gamma_0 \rangle}_{s_0} \rightarrow \underbrace{\langle p_1, \gamma_1 \gamma_2 \rangle}_{s_1} \rightarrow \underbrace{\langle p_2, \gamma_3 \gamma_2 \rangle}_{s_2} \rightarrow \underbrace{\langle p_3, \gamma_2 \rangle}_{s_3}$$

is represented by following *path formula*:

$$(\overline{a}_1 \cdot \overline{a}_0) \cdot (\overline{b}_1 \cdot \overline{b}_0) \;\cdot\; (\overline{a}_1' \cdot a_0') \cdot (\overline{b}_1' \cdot b_0') \;\cdot\; (a_1'' \cdot \overline{a}_0'') \cdot (b_1'' \cdot b_0'') \;\cdot\; (a_1''' \cdot a_0''') \cdot (b_1''' \cdot \overline{b}_0''')$$

where the variables $\{a_0, a_1, b_0, b_1\}$ are used for the representation of s_0, the variables $\{a_0', a_1', b_0', b_1'\}$ for s_1, and so on. The parts of the formula that refer to s_1 and s_2 constrain only the topmost element of the stack. The content of the bottom element of the stack (γ_2) is determined by the expansion rule $\langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_1, \gamma_1 \gamma_2 \rangle$, but the neutration rule cannot access this element. It only becomes "visible" to subsequently applied transition rules after the contraction $\langle p_2, \gamma_3 \rangle \hookrightarrow \langle p_3, \epsilon \rangle$.

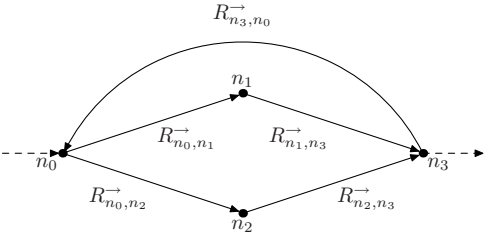
As discussed in Sec. 2, this sequence of transitions gives rise to the symbolic summary $\bar{a}_1 \cdot \bar{a}_1 \cdot \bar{b}_1 \cdot \bar{b}_1 \;\cdot\; a_1' \cdot a_1' \cdot b_1' \cdot \bar{b}_1'$. The summary is obtained by existentially quantifying the variables that represent the intermediate heads (i.e., $\{a_0', a_1', b_0', b_1'\}$ and $\{a_0'', a_1'', b_0'', b_1''\}$ in our example). Each symbolic summary is a propositional relation over a set of primed and unprimed variables.

Merging Paths. Whenever the algorithm encounters a branch (as illustrated in Fig. 2(a)) it splits the path and constructs a new formula for each branch. To avoid an exponential blowup, path formulas are *merged* (by means of a disjunction) when they agree on their initial and final CFG nodes (see, for instance, [12]). A detailed description of an algorithm that performs aggressive merging by delaying transitions until merging becomes possible is given in [6].

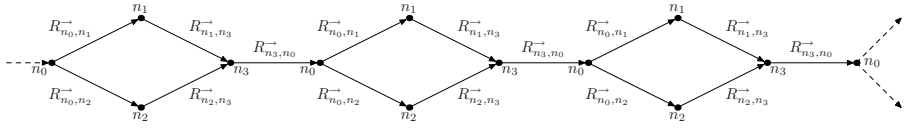
3.2 Using BMC to Compute Universal Summaries

The symbolic algorithms discussed compute the least fixpoint of reachable summaries [1,6]. This fixpoint detection is implemented using either BDDs or a QBF solver. Unfortunately, neither of these approaches scales for a large number of variables. Bounded Model Checking (BMC) addresses this issue by avoiding fixpoint detection altogether: The transition system is simply unrolled up to a

bounded path length k. This idea is illustrated in Fig. 4(b) for the cyclic transition system shown in Fig. 4(a) (instead of unrolling each path separately, we merge paths as discussed in Sec. 3.1). The satisfiability of the resulting path formula can be decided using an efficient SAT-solver (e.g., MiniSat [13]).



(a) A PDS transition system with a cycle



(b) Unrolling of the transitions in Fig. 4(a) for $k = 3$

Fig. 4. Bounded Model Checking for Pushdown Systems

BMC is complete with respect to reachability if (and only if) k is large enough to guarantee that all reachable states of the transition system are considered (the smallest k that has this property is called the *reachability diameter* of a transition system [14]). For a PDS with an infinite state space, there is no such finite k.

However, a function containing only neutrations is a finite state transition system. For a set of neutrations that are represented by $\xrightarrow{R^{\rightarrow}}$ and a symbolic representation $I(s_0)$ of the initial state(s), the constant

$$k = \max\{i \in \mathbb{N} | \exists s_0, \ldots, s_i \in P \times \Gamma^*. I(s_0) \wedge \bigwedge_{j=0}^{i-1} s_j \xrightarrow{R^{\rightarrow}} s_{j+1} \wedge \bigwedge_{j=0}^{i-1} \bigwedge_{l=j+1}^{i} s_j \neq s_l\}$$

is the length of the longest *loop-free* path that contains only neutrations (this corresponds to the *reachability recurrence diameter* of a finite state transition system [14]). Let $\langle p, \gamma\rangle \rightarrow^i \langle p', \gamma'\rangle$ denote the path formula obtained by means of unrolling $\xrightarrow{R^{\rightarrow}}$ exactly i times. Then, the relation

$$R^{\rightarrow^*}(\langle p, \gamma\rangle, \langle p', \gamma'\rangle) \stackrel{\text{def}}{=} \bigvee_{i=1}^{k} \langle p, \gamma\rangle \rightarrow^i \langle p', \gamma'\rangle$$

is sufficient to determine all heads $\langle p', \gamma'\rangle$ that are reachable from $\langle p, \gamma\rangle$ by means of neutrations. Using this technique, we can compute a path formula that

represents all heads reachable from an initial state $\langle p, \gamma \rangle \in I(s_0)$ by the loop in Fig. 4(a). In particular, if $I(s_0) = \mathsf{true}$, then the relation $R^{\rightarrow^*}$ determines the states reachable from an arbitrary initial state.

Given an explicit representation of the CFG (as suggested in Sec. 2), it is possible to determine the *innermost* function f that contains no expansion/contraction rules (at least as long as f is not a recursive function). Let $R_f^{\rightarrow}$ denote the neutrations of this function, and let $R_f^{\nearrow}$ and $R_f^{\searrow}$ the initial expansion and the final contraction, respectively. Furthermore, let $R_f^{\rightarrow^*}$ denote the unrolled path formula for the neutrations $R_f^{\rightarrow}$ and an arbitrary initial state $I(s_0) = \mathsf{true}$. Then, we obtain a universal summary for f by composing $R_f^{\rightarrow^*}$ with $R_f^{\nearrow}$ and $R_f^{\searrow}$:

$$\Sigma_{\mathcal{U}}^f(\langle p, \gamma \rangle, \langle p', \gamma' \rangle) \stackrel{\text{def}}{=} \\ R_f^{\nearrow}(\langle p, \gamma \rangle, \langle p_1, \gamma_1 \gamma' \rangle) \ \wedge \ R_f^{\rightarrow^*}(\langle p_1, \gamma_1 \rangle, \langle p_2, \gamma_2 \rangle) \ \wedge \ R_f^{\searrow}(\langle p_2, \gamma_2 \rangle, \langle p', \epsilon \rangle)$$

We proceed by finding a function g that calls only f and compute the universal summary for $R_g^{\rightarrow}$ using the universal summary for f. Thus, a universal summary for the whole PDS can be obtained in a top-down manner (assuming that the CFG representation of the PDS does not contain recursive function calls).

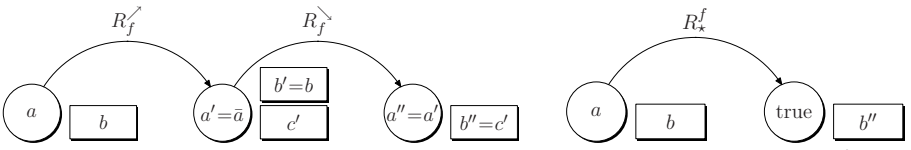
Corollary 1. *Let k be the length of the longest loop-free path through a recursion-free function f of a PDS. Then, the summary that subsumes all loop-free paths through f up to length k is a universal summary for the function f.*

Universal summaries have been presented only recently in [6]. The algorithm described there handles recursion by falling back to QBF-based summarization.

4 Abstraction and Refinement with Summaries

The technique discussed in the previous section (and presented in [6]) applies universal summaries in an *eager* manner: Whenever the search algorithm encounters an expansion rule and an appropriate (universal) summary is available, the summary replaces the expansion transition. If we are only interested in the reachability of a given head (or a program location), this approach is wasteful: A subsequence of a path may be sufficient to show that a head is not reachable. In that case, computing and applying the universal summaries for the nested functions in a top-down manner does not contribute to the infeasibility of the resulting path formula.

Therefore, we propose to compute universal summaries for functions *on demand*, and to apply them in a bottom-up manner. Given the transition rules of a function g of the PDS, we obtain an *over-approximation* of the corresponding universal summary $\Sigma_{\mathcal{U}}^g$ by replacing all occurrences of f in g by a non-deterministic summary $\Sigma_\star^f$:



(a) A path formula for f (b) The summary $\Sigma_\star^f$

Fig. 5. Over-approximation of a universal summary

$$\Sigma_\star^f(\langle p,\gamma\rangle,\langle p',\gamma'\rangle) \stackrel{\text{def}}{=} \exists\langle p_1,\gamma_1\rangle,\langle p_2,\gamma_2\rangle.R_f^{\nearrow}(\langle p,\gamma\rangle,\langle p_1,\gamma_1\gamma'\rangle) \wedge R_f^{\searrow}(\langle p_2,\gamma_2\rangle,\langle p',\epsilon\rangle)$$

The summary $\Sigma_\star^f$ is a conservative over-approximation of $\Sigma_{\mathcal{U}}^f$, since

$$\Sigma_{\mathcal{U}}^f(\langle p,\gamma\rangle,\langle p',\gamma'\rangle) \implies \Sigma_\star^f(\langle p,\gamma\rangle,\langle p',\gamma'\rangle)$$

always holds. A head $\langle p',\gamma'\rangle$ that is *not* reachable via the over-approximated summary for g is also not reachable using $\Sigma_{\mathcal{U}}^g$. The converse does not hold.

Example 1. Consider the following transition rules for a function f:

$$R_f^{\nearrow}(a,b,a',b',c') = (a\cdot b)\cdot(a'=\overline{a})\cdot(b'=b)\cdot c'$$
$$R_f^{\searrow}(a,b,a') \qquad = (a'=a)$$

The composition of $R_f^{\nearrow}$ and $R_f^{\searrow}$ yields the path formula in Fig. 5(a). The corresponding over-approximation of the summary for f is

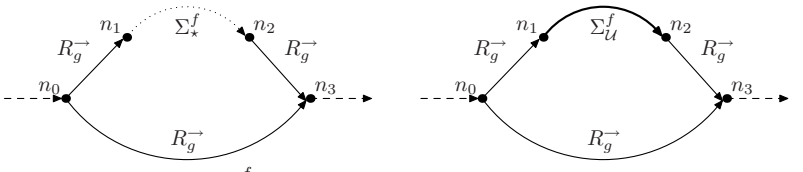
$$\Sigma_\star^f(a,b,a'',b'') = \exists a'b'a_\star.\ (a\cdot b)\cdot(a'=\overline{a})\cdot(b'=b)\cdot b''\cdot(a''=a_\star)$$

The summary $\Sigma_\star^f$ does not constrain the value of a'' (see Fig. 5(b)), even though the control state represented by $a''=1$ on return contradicts the path formula in Fig. 5(a).

Now consider the transitions of a function g shown in Fig. 6, and assume that the rules $R_g^{\rightarrow}$ require that transitions from n_2 to n_3 are only feasible if $a''=1$ holds at n_2 (e.g., $R_g^{\rightarrow}(a,b,a',b') = a\cdot a'\cdot(b'=b)$ for the transition from n_2 to n_3). If we over-approximate the function f as indicated in Fig. 6(a), then there exists a valuation (with $a''=1$ at n_2) to the variables of the corresponding path formula that represents a path through n_0, n_1, n_2 and n_3.

Unfortunately, this path is *spurious*, i.e., there is no feasible corresponding path in the original PDS. Therefore, we have to eliminate it from our over-approximated transition system. This can be achieved by computing $\Sigma_{\mathcal{U}}^f$ and using it to constrain the transition from n_1 to n_2 (as illustrated in Fig. 6(b)).

The reachability of a head $\langle p,\gamma\rangle$ at a node n can be determined by repeatedly refining the transition system until either a feasible path is found, or the head



(a) Applying $\Sigma^f_\star$ in g (b) Refinement

Fig. 6. Refining an over-approximated universal summary

becomes unreachable. Fig. 7 shows the pseudo code for this algorithm. Unfortunately, this algorithm does not work if the CFG contains recursive function calls. Therefore, our implementation still uses the QBF-based approach presented in [6] to compute summaries for recursive functions. Universal summaries and the refinement technique are orthogonal to this approach and can be integrated easily. In the following section, we present a theoretical result that eliminates the need for a QBF-solver.

```
 1: procedure IsReachable(PDS ⟨P, Γ, Δ, s0⟩, head ⟨p, γ⟩, node n)
 2:     for all functions f ∈ CFG of PDS ⟨P, Γ, Δ, s0⟩ do
 3:         Σ(f) := Σ*_f;
 4:     end for
 5:     while true do
 6:         if n contained in function f s.t. Σ(f) = Σ*_f then
 7:             n' := exit node of f;
 8:         else
 9:             n' := n;
10:         end if
11:         Use BMC and Σ to construct a path formula φ ending at n';
12:         if φ(s0, ⟨p, γ⟩) is satisfiable then
13:             if path does not traverse a function f with Σ(f) = Σ*_f then
14:                 return reachable;
15:             else
16:                 Σ(f) := Σ(f)^f_U;            ▷ Use Σ for function calls in f
17:             end if
18:         else
19:             return unreachable;
20:         end if
21:     end while
22: end procedure
```

Fig. 7. Abstraction/Refinement algorithm for PDS

5 Recursion

In this section, we generalize Cor. 1 and extend the algorithm in Fig. 7 in order to enable the construction of universal summaries for recursive functions.

Given a recursive function f, we can compute an over-approximation $\Sigma^f_{\star 1}$ of $\Sigma^f_{\mathcal{U}}$ by replacing all recursive calls to f with $\Sigma^f_\star$. A refined over-approximation $\Sigma^f_{\star 2}$ can then be obtained by applying $\Sigma^f_{\star 1}$ for all calls to f. Unfortunately, this technique may fail to eliminate all spurious paths, since any $\Sigma^f_{\star i}$ contains a nested summary $\Sigma^f_\star$ (i.e., nested according to the grammar in Fig. 3).

We can eliminate these spurious paths by "blocking" all paths in $\Sigma^f_{\star i}$ that traverse $\Sigma^f_\star$, i.e., we can replace the corresponding expansion by false. Unfortunately, this approach may also eliminate feasible paths. The following Theorem states that it is sufficient to consider only paths up to a certain nesting depth:

Theorem 1 (Universal Summaries with Recursion). *Let r be the largest natural number for which both of the following conditions hold:*

1. *There is a feasible path which contains r nested summaries, and*
2. *this path contains no pair of nested summaries $\langle p_0, \gamma_0\rangle \rightarrow^* \langle p'_0, \gamma'_0\rangle$ and $\langle p_1, \gamma_1\rangle \rightarrow^* \langle p'_1, \gamma'_1\rangle$ for which $\langle p_0, \gamma_0\rangle = \langle p_1, \gamma_1\rangle$ and $\langle p'_0, \gamma'_0\rangle = \langle p'_1, \gamma'_1\rangle$ holds.*

We claim that

a) such a finite r always exists and can be computed, and
b) a summary that subsumes all loop-free paths with at most r nested summary applications is a universal summary $\Sigma_{\mathcal{U}}$.

Proof. The proof of claim *a*) is simple: Given a summary $\langle p, \gamma\rangle \rightarrow^* \langle p', \gamma'\rangle$, we call $\langle p, \gamma\rangle$ the *entry*-head and $\langle p', \gamma'\rangle$ the *exit*-head. There are at most $|P|^2 \cdot |\Gamma|^2$ different combinations of entry- and exit-heads. After a path reaches a certain recursion depth $r \leq |P|^2 \cdot |\Gamma|^2$, the pairs of heads inevitably start to repeat. Furthermore, if this path exceeds a length l of at most $|P| \cdot \Sigma_{i=1}^r |\Gamma|^r$, then there is a state $\langle p, w\rangle$ that is visited twice. Thus, r can be computed by examining all paths up to depth $|P|^2 \cdot |\Gamma|^2$ and length $|P| \cdot \Sigma_{i=1}^r |\Gamma|^r$.

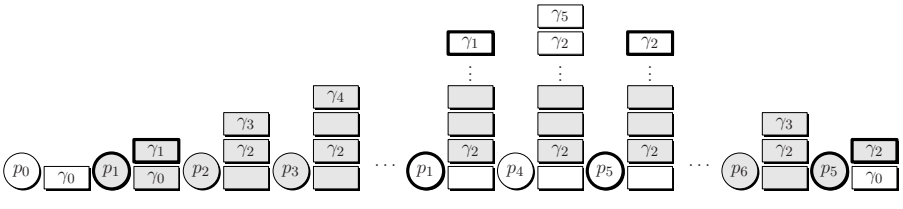


Fig. 8. Nested summaries with repeating entry- and exit-heads

Claim *b*) follows from the observations made above: Fig. 8 shows a path that contains two nested summaries for which the entry-heads and exit-heads are

equal. By eliminating the control states and stack elements that are colored grey in Fig. 8, we obtain a new path with a smaller number of nested summaries. This truncation has no impact on the states reachable from the final state $\langle p_5, \gamma_2\gamma_0\rangle$ of the path. Formally, let

$$\begin{aligned}\langle p_0,\gamma_0\rangle \to \langle p_1,\gamma_1 w_1\rangle \to \langle p_2,\gamma_2 w_2\rangle \to \ldots \to \\ \langle p_i,\gamma_i w_i\rangle \to \langle p_{i+1},\gamma_{i+1} w_{i+1}\rangle \to \ldots \to \langle p_{j-1},\gamma_{j-1} w_{j-1}\rangle \to \langle p_j,\gamma_j w_j\rangle \\ \to \ldots \to \langle p_{k-1},\gamma_{k-1} w_{k-1}\rangle \to \langle p_k,\gamma_k w_k\rangle \to \langle p_{k+1},\gamma_{k+1}\rangle\end{aligned}$$

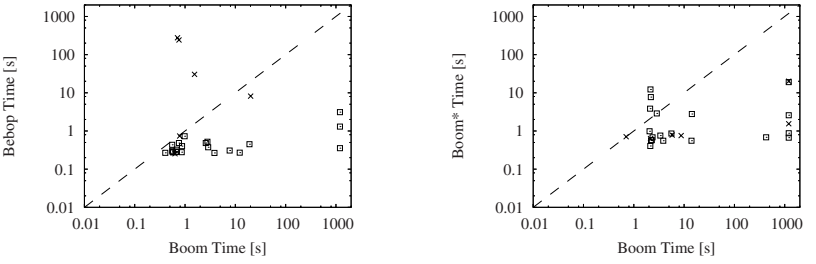
be a summary of the PDS, where $|w_1| = |w_k|$, $|w_i| = |w_j|$, $|w_1| < |w_2| < \ldots < |w_i| < |w_{i+1}|$ and $|w_{j-1}| > |w_j| > \ldots > |w_{k-1}| > |w_k|$, i.e., the summaries $\langle p_1,\gamma_1\rangle \to^* \langle p_k,\gamma_k\rangle$ and $\langle p_i,\gamma_i\rangle \to^* \langle p_j,\gamma_j\rangle$ are nested. Furthermore, let $p_1 = p_i$, $\gamma_1 = \gamma_i$, $p_k = p_j$, and $\gamma_k = \gamma_j$. Then, there exists a summary

$$\begin{aligned}\langle p_0,\gamma_0\rangle \to \langle p_i,\gamma_i w_1\rangle \to \langle p_{i+1},\gamma_{i+1} w_r\rangle \to \ldots \to \\ \langle p_{j-1},\gamma_{j-1} w_t\rangle \to \langle p_j,\gamma_j w_k\rangle \to \langle p_{k+1},\gamma_{k+1}\rangle\end{aligned}$$

that also covers $\langle p_0,\gamma_0\rangle \to^* \langle p_{k+1},\gamma_{k+1}\rangle$, and the nesting depth of this summary is smaller than the nesting depth of the original summary. Thus, any path with a pair of nested summaries can be truncated such that condition 2 holds without changing the set of states reachable via this path. A similar argument can be made for paths that contain a certain state twice (i.e., for paths that are not loop-free). □

The same proof technique has been used by Richard Büchi to show that the set of reachable states of a PDS can be expressed using a finite automaton [8].

There is an obvious similarity between the reachability recurrence diameter presented in Section 3.2 and the bound for the nesting depth introduced in Thm. 1. The reachability recurrence diameter of a PDS is two-dimensional and comprises of a sufficiently large nesting depth r and the length l of the longest loop-free path with a nesting depth at most r. This nesting depth can be computed symbolically using the same SAT-based unrolling technique: The longest loop-free path that contains no nested summaries is of length $|P| \cdot |\Gamma|$. If we increase the depth of the nestings to one, this length increases to $|P| \cdot (|\Gamma| + |\Gamma|^2)$, since each summary in this path may be of length $|P| \cdot |\Gamma|$. For any nesting depth greater than 1, we perform a pairwise comparison of the nested summaries. This can be achieved by means of a SAT-formula that is of quadratic size in the number of nested summaries. By repeatedly increasing the nesting depth r, we can determine the largest r for which the properties in Thm. 1 hold. Note, that if the second condition of Thm. 1 fails for all paths represented by a symbolic path that contains the summary $\Sigma^f_{\star i}$, then it still fails if we replace $\Sigma^f_{\star i}$ with $\Sigma^f_{\star(i+1)}$ (assuming that the first condition still holds after this transformation). In that case, it is not necessary to unroll the summary up to the worst case depth $|P|^2 \cdot |\Gamma|^2$.



(a) Boom vs Bebop (b) Boom vs Boom⋆

Fig. 9. Comparison of Bebop, Boom, and Boom⋆

6 Experiments

We implemented the algorithm in Fig. 7 and evaluated it using 40 PDSs generated by Slam. In the scatter graphs in Fig. 9 we distinguish between PDSs with reachable error locations (indicated by ×) and with unreachable error locations (indicated by □). Fig. 9(a) shows the scatter graph of our comparison of Bebop and the version of our tool that applies universal summaries eagerly. We conclude from these results that our algorithm that applies universal summaries eagerly (Boom) tends to perform better than the BDD based model checker Bebop when the location in question is reachable in the PDS. It performs worse than Bebop when it has to examine the whole state space. Fig. 9(b) compares the effect of the lazy abstraction approach (called Boom⋆) that we presented in Sec. 4 to the eager algorithm (Boom). The situation is less obvious than in Fig. 9(a). To some extent, the algorithm improves the performance for model checking PDSs with reachable error locations. Unfortunately, this cannot be generalized. We observed that about a third of the non-deterministic summaries are not replaced by refined summaries by the algorithm. We believe that we can still improve on these results, since our tool is in a very early state of development. The most recent version of Boom⋆ is available at `http://www.verify.ethz.ch/boom/`.

7 Related Work

The decidability of reachability properties of PDSs was shown by Büchi more than 40 years ago [8]. Efficient automata-based algorithms to construct the regular set of reachable states are presented by Finkel et al. [4] and Esparza et al. [5] (see Sec. 2). Schwoon improved the latter approach using a BDD-based symbolic representation of PDSs [7]. A saturation-based technique for similar models, namely recursive state machines, is presented in [15].

Summarization was introduced by Sharir and Pnueli as part of a dataflow analysis algorithm based on iterative fixpoint detection [3]. Ball and Rajamani's model checker Bebop is based on this work and uses BDDs to represent states

symbolically [1]. An implementation of their algorithm based on satisfiability solvers for propositional logic (SAT) and quantified Boolean formulas (QBF) is presented in [6] (see Sec. 3.1). Universal summaries are introduced in [6] (see Sec. 3). Kroening's model checker Boppo uses SAT-based symbolic simulation and QBF-based fixpoint detection, but does not use summarization [16]. Boppo requires that all function calls can be inlined. Leino combines BDDs and SAT-based techniques in his model checker Dizzy. He does not use summarization and reports that his benchmarks suggest that the approach does not scale [12].

Several attempts have been made to extend the formalism of PDSs with concurrency. In that case, the reachability problem becomes undecidable. Various verification techniques for concurrent PDSs have been proposed, but are either unsound or incomplete: For instance, bounding the number of context switches [17] or bounding communication [18] may miss feasible paths, while over-approximating the set of reachable states [19,20] may report spurious paths. We do not discuss these techniques here, since our approach is inappropriate for asynchronous systems: In general, there is no sufficiently large but finite bound for the sequential depth of concurrent PDSs.

Lal and Reps present a graph-theoretic approach for model checking *weighted* PDSs [21]. Their approach is incomparable to our algorithm, since we do not support weighted PDSs and their approach is not based on satisfiability solving techniques. Boujjani and Esparza survey approaches that use rewriting to solve the reachability problem for sequential as well as for concurrent pushdown systems [22].

BMC was introduced by Biere [23] as a SAT-based alternative to finite-state model checking algorithms that use Binary Decision Diagrams (BDDs). BMC and the recurrence diameter [14] for finite state transition systems is discussed in Sec. 3.2. The Saturn verification tool uses SAT and summarization to detect errors in C programs [24], but handles loops in an unsound manner.

8 Conclusion

BMC is an efficient technique for finding bugs. We showed how it can be applied to PDSs. Our algorithm uses BMC to compute *universal summaries* that relate arbitrary input states to their respective return states according to the transition relation of the function. Universal summaries can be applied in any calling context. We implemented an algorithm that uses SAT to compute universal summaries for functions without recursive calls, and QBF to compute summaries in the presence of recursion. Our benchmarks show that this approach performs better than BDD based algorithms when it comes to detecting bugs, but is less efficient for proving the unreachability of error states. This is a very useful result, since CEGAR-based model checkers generate PDSs with reachable error locations in all but the last iteration. Furthermore, we describe an extension to our algorithm that allows to compute universal summaries for functions with recursion. The implementation and evaluation of this extension is future work.

References

1. Ball, T., Rajamani, S.K.: Bebop: A symbolic model checker for Boolean programs. In: Havelund, K., Penix, J., Visser, W. (eds.) SPIN 2000. LNCS, vol. 1885, pp. 113–130. Springer, Heidelberg (2000)
2. Ball, T., et al.: SLAM and Static Driver Verifier: Technology transfer of formal methods inside Microsoft. In: Boiten, E.A., Derrick, J., Smith, G.P. (eds.) IFM 2004. LNCS, vol. 2999, pp. 1–20. Springer, Heidelberg (2004)
3. Sharir, M., Pnueli, A.: Two approaches to interprocedural dataflow analysis. In: Program Flow Analysis: Theory and Applications, pp. 189–233. Prentice-Hall, Englewood Cliffs (1981)
4. Finkel, A., Willems, B., Wolper, P.: A direct symbolic approach to model checking pushdown systems. In: Workshop on Verification of Infinite State Systems (INFINITY). ENTCS, vol. 9, pp. 27–39 (1997)
5. Esparza, J., Hansel, D., Rossmanith, P., Schwoon, S.: Efficient algorithms for model checking pushdown systems. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 232–247. Springer, Heidelberg (2000)
6. Basler, G., Kroening, D., Weissenbacher, G.: SAT-based summarisation for Boolean programs. In: Bošnački, D., Edelkamp, S. (eds.) SPIN 2007. LNCS, vol. 4595, pp. 131–148. Springer, Heidelberg (2007)
7. Schwoon, S.: Model-Checking Pushdown Systems. PhD thesis, Technische Universität München (2002)
8. Büchi, J.R.: Regular canonical systems. Archive for Mathematical Logic 6, 91 (1964)
9. Ball, T., Rajamani, S.: Boolean programs: A model and process for software analysis. Technical Report 2000-14, Microsoft Research (2000)
10. Ball, T., Majumdar, R., Millstein, T., Rajamani, S.K.: Automatic predicate abstraction of C programs. In: Programming Language Design and Implementation (PLDI), pp. 203–213. ACM Press, New York (2001)
11. Graf, S., Saïdi, H.: Construction of abstract state graphs with PVS. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997)
12. Leino, K.R.M.: A SAT characterization of Boolean-program correctness. In: Ball, T., Rajamani, S.K. (eds.) SPIN 2003. LNCS, vol. 2648, pp. 104–120. Springer, Heidelberg (2003)
13. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
14. Kroening, D., Strichman, O.: Efficient computation of recurrence diameters. In: Zuck, L.D., Attie, P.C., Cortesi, A., Mukhopadhyay, S. (eds.) VMCAI 2003. LNCS, vol. 2575, pp. 298–309. Springer, Heidelberg (2002)
15. Alur, R., Benedikt, M., Etessami, K., Godefroid, P., Reps, T., Yannakakis, M.: Analysis of recursive state machines. ACM Transactions on Programming Languages and Systems (TOPLAS) 27, 786–818 (2005)
16. Cook, B., Kroening, D., Sharygina, N.: Symbolic model checking for asynchronous Boolean programs. In: Godefroid, P. (ed.) SPIN 2005. LNCS, vol. 3639, pp. 75–90. Springer, Heidelberg (2005)
17. Qadeer, S., Rehof, J.: Context-bounded model checking of concurrent software. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 93–107. Springer, Heidelberg (2005)
18. Touili, T., Sighireanu, M.: Bounded communication reachability analysis of process rewrite systems with ordered parallelism. In: Verification of Infinite State Systems (INFINITY). ENTCS, Elsevier, Amsterdam (2006)

19. Cook, B., Kroening, D., Sharygina, N.: Over-Approximating Boolean Programs with unbounded thread creation. In: Formal Methods in Computer-Aided Design FMCAD, pp. 53–59. IEEE Computer Society Press, Los Alamitos (2006)
20. Bouajjani, A., Esparza, J., Touili, T.: A generic approach to the static analysis of concurrent programs with procedures. In: Principles of Programming Languages (POPL), pp. 62–73. ACM Press, New York (2003)
21. Lal, A., Reps, T.: Improving pushdown system model checking. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 343–357. Springer, Heidelberg (2006)
22. Bouajjani, A., Esparza, J.: Rewriting models of Boolean programs. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, Springer, Heidelberg (2006)
23. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: Cleaveland, W.R. (ed.) ETAPS 1999 and TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999)
24. Xie, Y., Aiken, A.: Saturn: A scalable framework for error detection using boolean satisfiability. ACM Transactions on Programming Languages and Systems (TOPLAS) 29 (2007)