Reachability Analysis of Pushdown Systems with an Upper Stack

Adrien Pommellet[1(✉)], Marcio Diaz[1], and Tayssir Touili[2]

[1] LIPN, Université Paris-Diderot, Paris, France
pommellet@lipn.univ-paris13.fr
[2] LIPN, CNRS, Université Paris 13, Villetaneuse, France

Abstract. Pushdown systems (PDSs) are a natural model for sequential programs, but they can fail to accurately represent the way an assembly stack actually operates. Indeed, one may want to access the part of the memory that is below the current stack or base pointer, hence the need for a model that keeps track of this part of the memory. To this end, we introduce pushdown systems with an upper stack (UPDSs), an extension of PDSs where symbols popped from the stack are not destroyed but instead remain just above its top, and may be overwritten by later push rules. We prove that the sets of successors $post^*$ and predecessors pre^* of a regular set of configurations of such a system are not always regular, but that $post^*$ is context-sensitive, so that we can decide whether a single configuration is forward reachable or not. In order to underapproximate pre^* in a regular fashion, we consider a bounded-phase analysis of UPDSs, where a phase is a part of a run during which either push or pop rules are forbidden. We then present a method to overapproximate $post^*$ that relies on regular abstractions of runs of UPDSs. Finally, we show how these approximations can be used to detect stack overflows and stack pointer manipulations with malicious intent.

Keywords: Pushdown systems · Reachability analysis · Stack pointer · Finite automata

1 Introduction

Pushdown systems (PDSs) were introduced to accurately model the *call stack* of a program. A *call stack* is a stack data structure that stores information about the active procedures of a program such as return addresses, passed parameters and local variables. It is usually implemented using a *stack pointer (sp)* register that indicates the head of the stack. Thus, assuming the stack grows downwards, when data is *pushed* onto the stack, *sp* is decremented before the item is placed on the stack. For instance, in $x86$ architecture *sp* is decremented by 4 (pushing 4 bytes). When data is *popped* from the stack, *sp* is incremented. For instance, in $x86$ architecture *sp* is incremented by 4 (popping 4 bytes).

This work was partially funded by the FUI project Freenivi.

F. Drewes et al. (Eds.): LATA 2017, LNCS 10168, pp. 447–459, 2017.
DOI: 10.1007/978-3-319-53733-7_33

However, in a PDS, neither push nor pop rules are truthful to the assembly stack. During an actual pop operation on the stack, the item remains in memory and the stack pointer is increased, as shown in Figs. 1 and 2, whereas a PDS deletes the item on the top of the stack, as shown in Figs. 3 and 4.

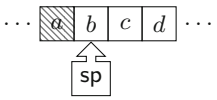

Fig. 1. The original stack

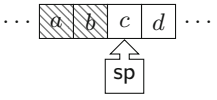

Fig. 2. The stack after one pop

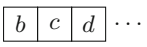

Fig. 3. The original PDS stack

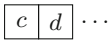

Fig. 4. The PDS stack after one pop

This subtle difference becomes important when we want to analyze programs that directly manipulate the stack pointer and use assembly code. Indeed, in most assembly languages, *sp* can be used like any other register. As an example, the instruction $\mathsf{mov\ eax\ [sp-4]}$ will put the value pointed to at address $\mathsf{sp-4}$ in the register eax (one of the general registers). Since $\mathsf{sp-4}$ is an address above the stack pointer, we do not know what is being copied into the register eax, unless we have a way to record the elements that had previously been popped from the stack and not overwritten yet. Such instructions may happen in malicious assembly programs: malware writers tend to do unusual things in order to obfuscate their payload and thwart static analysis.

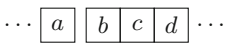

Fig. 5. The original UPDS stacks

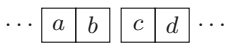

Fig. 6. The UPDS stacks after one pop

Thus, it is important to record the part of the memory that is just above the stack pointer. To this end, we extend PDSs in order to keep track of this *upper stack*: we introduce in this paper a new model called *pushdown system with an upper stack (UPDS)* that extends the semantics of PDSs. In a UPDS, when a letter is popped from the top of the stack (*lower stack* from now on), it is added to the bottom of a write-only *upper stack*, effectively simulating the decrement of the stack pointer. This is shown in Figs. 5 and 6, where after being popped, b is removed from the lower stack (on the right) and added to the upper stack (on the left) instead of being destroyed. The top of the lower stack and the bottom of the upper stack meet at the stack pointer.

Paper Outline. The first contribution of this paper is a more precise model of the stack of a program as outlined above and defined in Sect. 2.

We then investigate the sets of predecessors and successors of a regular set of regular configurations of an UPDS. Unfortunately, in Sect. 3 we prove that neither of them are regular. However, we show that the set of successors is

context-sensitive. As a consequence, we can decide whether a single configuration is forward reachable or not in an UPDS.

Then, in Sect. 4, we prove that the set of predecessors of an UPDS is regular given a limit of k phases, where a phase is a part of a run during which either pop or push rules are forbidden. Bounded phase reachability is an underapproximation of the actual reachability relation on UPDSs that we can use to detect some incorrect behaviours.

In Sect. 5, we give an algorithm to compute an overapproximation of the set of successors. Overapproximation algorithms are often used while proving safety properties since a set of bad configurations not reachable in the overapproximation is not reachable in the original model either. Our idea is to first overapproximate the runs of the UPDS, then compute an overapproximation of the reachable upper stack configuration from this abstraction of runs and consider its product with the regular, accurate and computable set of lower stack configurations.

Finally, in Sect. 6, we use these approximations on programs to detect stack overflow errors and malicious attacks that rely on stack pointer manipulations.

Related Work. In [2,5,6], the pre^* and $post^*$ of a regular set of configurations on a pushdown system are shown to be regular. UPDSs are more expressive than PDSs, since they feature both an upper stack and a lower stack, the latter being equivalent to the single stack of PDSs.

One way to improve the expressiveness of pushdown automata is to change the way transition rules interact with the stack. Ginsburg et al. introduced in [7] *stack automata* that can read the inside of their own stack using a moving stack pointer but can only modify the top. As shown in [8], stack automata are equivalent to linear bounded automata (LBA). A LBA is a non-deterministic Turing machine whose tape is bounded between two end markers that cannot be overwritten. This model cannot simulate a UPDS whose lower stack is of unbounded height.

Uezato et al. defined in [13] *pushdown systems with transductions*: in such a model, a finite transducer is applied to the whole stack after each transition. However, this model is Turing powerful unless the transducers used have a finite closure, in which case it is equivalent to a simple pushdown system. When the set of transducers has a finite closure, this class cannot be used to simulate UPDSs.

Multi-stack automata have two or more stacks that can be read and modified, but are unfortunately Turing powerful. Following the work of Qadeer et al. in [10], La Torre et al. introduced in [12] *multi-stack pushdown systems with bounded phases*: in each phase of a run, there is at most one stack that is popped from. Anil Seth later proved in [11] that the pre^* of a regular set of configurations of a multi-pushdown system with bounded phases is regular; we use this result to perform a bounded phase analysis of our model.

2-visibly pushdown automata (2-VPDA) were defined by Carotenuto et all. in [4] as a variant of two-stack automata where the stack operations are driven by the input word. Reachability is decidable for a subclass of 2-VPDA with an ordering constraint on stack operations. However, these ordered 2-VPDA cannot simulate UPDSs.

2 Pushdown Systems with an Upper Stack

Definition 1 (Pushdown system with an upper stack). *A* pushdown system with an upper stack *(UPDS) is a triplet* $\mathcal{P} = (P, \Gamma, \Delta)$ *where* P *is a finite set of control states,* Γ *is a finite stack alphabet, and* $\Delta \subseteq P \times \Gamma \times P \times \left(\{\varepsilon\} \cup \Gamma \cup \Gamma^2\right)$ *a finite set of transition rules.*

We further note $\Delta_{pop} = \Delta \cap P \times \Gamma \times P \times \{\varepsilon\}$, $\Delta_{switch} = \Delta \cap P \times \Gamma \times P \times \Gamma$, and $\Delta_{push} = \Delta \cap P \times \Gamma \times P \times \Gamma^2$. If $\delta = (p, w, p', w') \in \Delta$, we write $\delta = (p, w) \to (p', w')$. In a UPDS, a write-only *upper stack* is maintained above the stack used for computations (from then on called the *lower stack*), and modified accordingly during a transition.

For $x \in \Gamma$ and $w \in \Gamma^*$, $|w|_x$ stands for the number of times the letter x appears in the word w, and w^R for the mirror image of w. Let $\bar{\Gamma}$ be a disjoint copy (bijection) of the stack alphabet Γ. If $x \in \Gamma$ (resp. Γ^*), then its associated letter (resp. word) in $\bar{\Gamma}$ (resp. $\bar{\Gamma}^*$) is written $\bar{x}$.

A *configuration* of $\mathcal{P}$ is a triplet $\langle p, w_u, w_l \rangle$ where $p \in P$ is a control state, $w_u \in \Gamma^*$ an upper stack content, and $w_l \in \Gamma^*$ a lower stack content. A set of configurations $\mathcal{C}$ of a UPDS $\mathcal{P}$ is said to be *regular* if for all $p \in P$, there exists a finite-state automaton $\mathcal{A}_p$ on the alphabet $\bar{\Gamma} \cup \Gamma$ such that $\mathcal{L}(\mathcal{A}_p) = \{\bar{w_u} w_l \mid \langle p, w_u, w_l \rangle \in \mathcal{C}\}$, where $\mathcal{L}(\mathcal{A})$ stands for the language recognized by an automaton $\mathcal{A}$.

From the set of transition rules Δ, we can infer an *immediate successor relation* $\Rightarrow_{\mathcal{P}} = \left(\bigcup_{\delta \in \Delta} \overset{\delta}{\Rightarrow} \right)$ on configurations of $\mathcal{P}$, which is defined as follows:

Switch rules: if $\delta = (p, a) \to (p', b) \in \Delta_{switch}$, then $\forall w_u \in \Gamma^*$ and $\forall w_l \in \Gamma^*$, $\langle p, w_u, a w_l \rangle \overset{\delta}{\Rightarrow} \langle p', w_u, b w_l \rangle$. The top letter a of the lower stack is replaced by b, but the upper stack is left untouched (the stack pointer doesn't move).

Pop rules: if $\delta = (p, a) \to (p', \varepsilon) \in \Delta_{pop}$, then $\forall w_u \in \Gamma^*$ and $\forall w_l \in \Gamma^*$, $\langle p, w_u, a w_l \rangle \overset{\delta}{\Rightarrow} \langle p', w_u a, w_l \rangle$. The top letter a popped from the lower stack is added to the bottom of the upper stack (the stack pointer moves to the right).

Push rules: if $\delta = (p, a) \to (p', bc) \in \Delta_{push}$, then $\forall w_l \in \Gamma^*$, $\langle p, \varepsilon, a w_l \rangle \overset{\delta}{\Rightarrow} \langle p', \varepsilon, bc w_l \rangle$ and $\forall w_u \in \Gamma^*$, $\forall x \in \Gamma$, $\langle p, w_u x, a w_l \rangle \overset{\delta}{\Rightarrow} \langle p', w_u, bc w_l \rangle$. A new letter b is pushed on the lower stack, and a single letter is deleted from the bottom of the upper stack in order to make room for it, unless the upper stack was empty (the stack pointer moves to the left).

The *reachability* relation $\Rightarrow_{\mathcal{P}}^*$ is the reflexive and transitive closure of the immediate successor relation $\Rightarrow_{\mathcal{P}}$. If $\mathcal{C}$ is a set of configurations, we introduce its set of *successors* $post^*(\mathcal{P}, \mathcal{C}) = \{c \in P \times \Gamma^* \times \Gamma^* \mid \exists c' \in \mathcal{C}, c' \Rightarrow_{\mathcal{P}}^* c\}$ and its set of *predecessors* $pre^*(\mathcal{P}, \mathcal{C}) = \{c \in P \times \Gamma^* \times \Gamma^* \mid \exists c' \in \mathcal{C}, c \Rightarrow_{\mathcal{P}}^* c'\}$. We may omit the variable $\mathcal{P}$ when only a single UPDS is being considered.

For a set of configurations $\mathcal{C}$, let $\mathcal{C}_{low} = \{\langle p, w_l\rangle \mid \exists w_u \in \Gamma^*, \langle p, w_u, w_l\rangle \in \mathcal{C}\}$, $\mathcal{C}_{up} = \{\langle p, w_u\rangle \mid \exists w_l \in \Gamma^*, \langle p, w_u, w_l\rangle \in \mathcal{C}\}$, and $post^*_{up}(\mathcal{P}, \mathcal{C}) = (post^*(\mathcal{P}, \mathcal{C}))_{up}$. We define $post^*_{low}(\mathcal{P}, \mathcal{C})$, $pre^*_{up}(\mathcal{P}, \mathcal{C})$ and $pre^*_{low}(\mathcal{P}, \mathcal{C})$ in a similar fashion.

A *run* r of $\mathcal{P}$ from a configuration c_0 is a sequence $r = (\delta_i)_{i=1,\ldots,n} \in \Delta^*$ such that $c_0 \overset{\delta_1}{\Rightarrow} c_1 \overset{\delta_2}{\Rightarrow} c_2 \ldots \overset{\delta_n}{\Rightarrow} c_n$, where $(c_i)_{i=1,\ldots,n}$ is a sequence of configurations of $\mathcal{P}$. We then write $c_0 \overset{r}{\Rightarrow} c_n$. We say that r is a run of $\mathcal{P}$ from a set of configurations $\mathcal{C}$ if and only if $\exists c \in \mathcal{C}$ such that r is a run of $\mathcal{P}$ from c.

These definitions are related to similar concepts on simple PDSs detailed in [2,6]. A UPDS and a PDS indeed share the same definition, but the semantics of the former expand the latter's. For a set $\mathcal{C} \subseteq P \times \Gamma^*$ of lower stack configurations (the upper stack is ignored) and a UPDS $\mathcal{P}$, let $post^*_{PDS}(\mathcal{P}, \mathcal{C})$ and $pre^*_{PDS}(\mathcal{P}, \mathcal{C})$ be the set of forward and backward reachable configurations from $\mathcal{C}$ using the PDS semantics. The following lemmas hold:

Lemma 2. *For a UPDS $\mathcal{P} = (P, \Gamma, \Delta)$, r in Δ^*, and a set of configurations $\mathcal{C}$, r is a run from $\mathcal{C}$ with respect to the UPDS semantics if and only if r is a run from $\mathcal{C}_{low}$ with respect to the standard PDS semantics.*

Lemma 3. $post^*_{low}(\mathcal{P}, \mathcal{C}) = post^*_{PDS}(\mathcal{P}, \mathcal{C}_{low})$, $pre^*_{low}(\mathcal{P}, \mathcal{C}) = pre^*_{PDS}(\mathcal{P}, \mathcal{C}_{low})$.

3 Reachability Properties

As shown in [2,5,6], we know that pre^*_{PDS} and $post^*_{PDS}$ are regular for a regular set of starting configurations. We prove that these results cannot be extended to UPDSs, but that $post^*$ is still context-sensitive. This implies that reachability of a single configuration is decidable for UPDSs.

3.1 *post** Is Not Regular

The following counterexample proves that, unfortunately, $post^*(\mathcal{P}, \mathcal{C})$ is not always regular for a given regular set of configurations $\mathcal{C}$ and a UPDS $\mathcal{P}$. The intuition behind this statement is that the upper stack can be used to store symbols in a non-regular fashion. The counter-example should be carefully designed in order to prevent later push operations from overwriting these symbols.

Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a UPDS with $P = \{p, p'\}$, $\Gamma = \{a, b, x, y, \bot\}$, and Δ the following set of pushdown transitions:

$$\begin{array}{llll} (S_x) & (p, x) \to (p, a) & (R_a) & (p, a) \to (p, \varepsilon) \\ (S_y) & (p, y) \to (p, b) & (R_b) & (p, b) \to (p, \varepsilon) \\ (C) & (p, a) \to (p, ab) & (E) & (p, \bot) \to (p', \bot) \end{array}$$

Let $\mathcal{C} = \{p\} \times \{\varepsilon\} \times x\,(yx)^* \bot$ be a regular set of configurations. We can compute a relevant subset L of $post^*(\mathcal{C})$:

Lemma 4. $L = \left\{\left\langle p', a^{n+1}b^n, \bot\right\rangle, n \in \mathbb{N}\right\} \subseteq post^*(\mathcal{C})$.

Then, we prove an inequality that holds for any configuration in $post^*$:

Lemma 5. $\forall \langle p, w_u, w_l\rangle \in post^*(\mathcal{C})$, $w = \bar{w_u} w_l$, $|w|_{\bar{b}} + |w|_{\bar{b}} + 1 \geq |w|_a + |w|_{\bar{a}}$.

If we suppose that $post^*(\mathcal{C})$ is regular, then so is the language $L^{p'}$, where $L^{p'} = \{\bar{w_u} w_l \mid \langle p', w_u, w_l\rangle \in post^*(\mathcal{C})\}$, and by the pumping lemma, it admits a pumping length k. We want to apply the pumping lemma to an element of L in order to generate a configuration that should be in $post^*$ but does not comply with the previous inequality.

According to Lemma 4, $L \subseteq post^*(\mathcal{C})$ and as a consequence $w = \overline{a^{k+1}b^k}\perp \in L^{p'}$. Hence, if we apply the pumping lemma to w, there exist $x, y, z \in \left(\Gamma \cup \bar{\Gamma}\right)^*$ such that $w = xyz$, $|xy| \leq k$ and $xy^i z \in post^*(\mathcal{C})$, $\forall i \geq 1$. As a consequence of w's definition, $x, y \in \bar{a}^*$, $|y| \geq 1$, and $z \in \left(\bar{a} + \bar{b}\right)^*$.

Hence, for i large enough, $w_i = xy^i z \in L^{p'}$ and $|w_i|_{\bar{a}} > |w_i|_{\bar{b}} + 1$. By Lemma 5, this cannot happen and therefore neither $L^{p'}$ nor $post^*(\mathcal{C})$ are regular.

It should be noted that $L^{p'}_{up}$ is not regular either. Indeed, from the definition of $\mathcal{P}$ and $\mathcal{C}$, it is clear that $\forall \langle p', w_u, w_l\rangle \in post^*(\mathcal{C}), w_l = \perp$, so $L^{p'}_{up}$ and $L^{p'}$ are in bijection. We have therefore proven the following theorem:

Theorem 6. *There exist a UPDS $\mathcal{P}$ and a regular set of configurations $\mathcal{C}$ for which neither $post^*(\mathcal{C})$ nor $post^*_{up}(\mathcal{C})$ are regular.*

3.2 pre^* Is Not Regular

We now prove that pre^* is not regular either. Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a UPDS with $P = \{p\}, \Gamma = \{a, b, c\}$, and Δ the following set of pushdown transitions:

$$(C_0)\ (p, c) \to (p, ab) \quad (R_a)\ (p, a) \to (p, \varepsilon)$$
$$(C_1)\ (p, c) \to (p, cb) \quad (R_b)\ (p, b) \to (p, \varepsilon)$$

We define the regular set of configurations $\mathcal{C} = \{p\} \times (ab)^* \times \{c\}$ and again, compute a relevant subset of $pre^*(\mathcal{C})$:

Lemma 7. $L = \{\langle p, b^n, c^n c\rangle, n \in \mathbb{N}\} \subseteq pre^*(\mathcal{C})$.

Given the rules of $\mathcal{P}$, the following lemma is verified:

Lemma 8. *If $\langle p, b^m, c^n\rangle \Rightarrow^* \langle p, w_u, w_l\rangle$, then $|w_u|_a + |w_l|_a \leq n$.*

If $pre^*(\mathcal{C})$ is regular, then so is $L^p = \{\bar{w_u} w_l \mid \langle p, w_u, w_l\rangle \in pre^*(\mathcal{C})\}$, and by the pumping lemma, it admits a pumping length k. Moreover, by Lemma 7, $w = \bar{b}^k c^k c \in L^p$.

If we apply the pumping lemma to w, there exist $x, y, z \in \left(\Gamma \cup \bar{\Gamma}\right)^*$ such that $w = xyz$, $|xy| \leq k$ and $w_i = xy^i z \in pre^*(\mathcal{C})$, $\forall i \geq 1$. As a consequence of w's definition, $x, y \in \bar{b}^*$ and $z \in \bar{b}^* c^k c$.

Since $w_i \in L^p$, $\forall i \geq 1$, there exists an integer n_i such that $w_i \Rightarrow^* c_i = \overline{(ab)^{n_i}} c$. Moreover, the size of the stack must grow or remain constant during

a computation, hence $|c_i| \geq |w_i|$ and $n_i \geq \frac{|w_i|-1}{2}$. Since words in the sequence $(w_i)_i$ are unbounded in length, the sequence $(n_i)_i$ must be unbounded as well. However, by Lemma 8, $n_i = |c_i|_{\bar{a}} \leq |w_i|_c = k+1$.

Hence, there is a contradiction and $pre^*(\mathcal{C})$ is not regular.

Theorem 9. *There exist a UPDS $\mathcal{P}$ and a regular set of configurations $\mathcal{C}$ for which $pre^*(\mathcal{C})$ is not regular.*

3.3 *post** Is Context-Sensitive

We prove that, if $\mathcal{C}$ is a regular set of configurations of a UPDS $\mathcal{P}$, then $post^*(\mathcal{P}, \mathcal{C})$ is context-sensitive. This implies that we can decide whether a single configuration is reachable from $\mathcal{C}$ or not.

We first show that the problem of computing $post^*(\mathcal{P}, \mathcal{C})$ can be reduced w.l.o.g. to the case where $\mathcal{C}$ contains a single configuration. To do so, we define a new UPDS $\mathcal{P}'$ by adding new states and rules to $\mathcal{P}$ such that any configuration c in $\mathcal{C}$ can be reached from a single configuration $c_\$ = \langle p_\$, \varepsilon, \$\rangle$. Once a configuration in $\mathcal{C}$ is reached, $\mathcal{P}'$ follow the same behaviour as $\mathcal{P}$.

Theorem 10. *For each UPDS $\mathcal{P} = (P, \Gamma, \Delta)$ and each regular set of configurations $\mathcal{C}$ on $\mathcal{P}$, there exists a UPDS $\mathcal{P}' = \left(P', \Gamma \cup \bar{\Gamma} \cup \{\$\}, \Delta'\right)$, $P \subseteq P'$, and $p_\$ \in P' \setminus P$ such that $post^*(\mathcal{P}, \mathcal{C}) = post^*(\mathcal{P}', \{\langle p_\$, \varepsilon, \$\rangle\}) \cap (P \times \Gamma^* \times \Gamma^*)$.*

We can compute a context-sensitive grammar recognizing $post^*$. Our intuition is to represent a configuration $\langle p, w_u, w_l \rangle$ of $\mathcal{P}$ by a word $\top w_u p w_l \bot$ of a grammar $\mathcal{G}$. We use Theorem 10 so that the single start symbol of $\mathcal{G}$ can be matched to a single configuration $c_\$$. The context-sensitive rules of $\mathcal{G}$ mimic the transitions of the UPDS. As an example, a rule $\delta = (p, a) \to (p', \varepsilon) \in \Delta_{pop}$ can be modelled by three rules $pa \dashrightarrow_{\mathcal{G}} pg_\delta$, $pg_\delta \dashrightarrow_{\mathcal{G}} ag_\delta$, and $ag_\delta \dashrightarrow_{\mathcal{G}} ap'$ such that $pa \dashrightarrow^*_{\mathcal{G}} ap'$, where $\dashrightarrow_{\mathcal{G}}$ stands for the one-step derivation relation and g_δ is a nonterminal symbol of $\mathcal{G}$.

Theorem 11. *Given a UPDS $\mathcal{P}$ and a regular set of configurations $\mathcal{C}$, we can compute a context-sensitive grammar $\mathcal{G}$ such that $\langle p, w_u, w_l \rangle \in post^*(\mathcal{P}, \mathcal{C})$ if and only if $\top w_u p w_l \bot \in \mathcal{L}(\mathcal{G})$*

Since the membership problem is decidable for context-sensitive grammars, the following theorem holds:

Theorem 12. *Given a UPDS $\mathcal{P}$, a regular set of configurations $\mathcal{C}$, and a configuration c of $\mathcal{P}$, we can decide whether $c \in post^*(\mathcal{P}, \mathcal{C})$ or not.*

Unfortunately, this method cannot be extended to pre^*, as the backward reachability relation does not comply with the monotony condition of context-sensitive grammars (a word can only grow or keep the same length during a computation).

4 Underapproximating pre^*

Underapproximations of reachability sets can be used to discover errors in programs: if $\mathcal{X}$ is a regular set of forbidden configurations of a UPDS $\mathcal{P}$, $\mathcal{C}$ a regular set of starting configurations, and $U \subseteq pre^*(\mathcal{X})$ a regular underapproximation, then $U \cap \mathcal{C} \neq \emptyset$ implies that a forbidden configuration can be reached from the starting set. The emptiness of the above intersection has to be decidable, hence, the need for a regular approximation.

In this section, we use results on *multi-stack pushdown automata* to define an underapproximation of pre^* for UPDSs. Multi-stack pushdown systems (MPDSs) are pushdown systems with multiple stacks where, for a given transition, in a given control state, only one stack is read and modified: a rule of the form $(p, w, n) \to (p', w')$ is applied to the n-th stack with semantics similar to those of common pushdown systems.

Multi-stack automata are unfortunately Turing powerful even with only two stacks. Thus, La Torre et al. introduced in [12] a restriction called *phase-bounding*: runs are divided into phases during which only a single stack can be popped from, and only the runs that have a number of phases lower than a chosen bound k are allowed. Let $pre^*_{\text{MPDS}}(\mathcal{M}, \mathcal{C}, k)$ be the set of backward reachable configurations from $\mathcal{C}$ using only runs with k phases. A theorem has been proven in [11]:

Theorem 13. *Given a MPDS $\mathcal{M}$ and a regular set of configurations $\mathcal{C}$, the set $pre^*_{MPDS}(\mathcal{M}, \mathcal{C}, k)$ is regular and effectively computable.*

The notion of bounded-phase computations can be extended to UPDSs. A run r of $\mathcal{P}$ is said to be *k-phased* if it is of the form: $r = r_1 \cdot r_2 \dots r_k$ where $\forall i \in \{1, \dots, k\}$, $r_i \in (\Delta_{Push} \cup \Delta_{Switch})^* \cup (\Delta_{Pop} \cup \Delta_{Switch})^*$. During a phase, one can either push or pop, but can't do both. Such a run has therefore at most k alternations between push and pop rules.

The *k-bounded* reachability relation $\Rightarrow^*_k$ is defined as follows: $c_0 \Rightarrow^*_k c_1$ if there exists a k-phased run r on $\mathcal{P}$ such that $c_0 \overset{r}{\Rightarrow} c_1$. Using this new reachability relation, given a set of configurations $\mathcal{C}$, we can define $pre^*(\mathcal{P}, \mathcal{C}, k)$.

We can show that a UPDS $\mathcal{P}$ can be simulated by a MPDS $\mathcal{M}$ with two stacks, the second stack of $\mathcal{M}$ being equivalent to the lower stack, and the first one, to a mirrored upper stack followed by a symbol $\perp$ that can't be popped and is used to know when the end of the stack has been reached. Elements of $P \times \Gamma^* \times \Gamma^*$ can equally be considered as configurations of $\mathcal{P}$ or $\mathcal{M}$, assuming in the latter case that we consider the mirror of the first stack and add a $\perp$ symbol to its bottom. Thus:

Lemma 14. *For a given UPDS $\mathcal{P} = (P, \Gamma, \Delta)$ and a regular set of configurations $\mathcal{C}$, there exists a MPDS $\mathcal{M}$, a regular set of configurations $\mathcal{C}'$, and $\perp \notin \Gamma$ such that $\langle p, w_u^R \perp, w_l \rangle \in pre^*_{MPDS}(\mathcal{M}, \mathcal{C}', k) \cap (P \times \Gamma^* \times \Gamma^*)$ if and only if $\langle p, w_u, w_l \rangle \in pre^*(\mathcal{P}, \mathcal{C}, k)$.*

From Theorem 13, we get:

Theorem 15. *Given a UPDS $\mathcal{P}$ and a regular set of configurations $\mathcal{C}$, the set $pre^*(\mathcal{P}, \mathcal{C}, k)$ is regular and effectively computable.*

$pre^*(\mathcal{P}, \mathcal{C}, k)$ is obviously an underapproximation of $pre^*(\mathcal{P}, \mathcal{C})$.

5 Overapproximating *post**

While underapproximations of reachability sets can be used to show that an error can occur, *overapproximations* can, on the other hand, prove that a program is safe from a particular error. If $\mathcal{X}$ is a regular set of forbidden configurations on an UPDS $\mathcal{P}$, $\mathcal{C}$ a regular set of starting configurations, and $O \supseteq post^*(\mathcal{C})$ a regular overapproximation, then $O \cap \mathcal{X} = \emptyset$ implies that no forbidden configuration can be reached from the starting set and that the program is therefore safe. The emptiness of the above intersection has to be decidable, hence, the need for a regular approximation.

5.1 A Relationship Between Runs and the Upper Stack

We prove here that from a regular set of runs of a given UPDS, a regular set of corresponding upper stacks can be computed. A subclass of programs whose UPDS model has a regular set of runs are programs with finite recursion (hence, with a stack of finite height).

Theorem 16. *For a UPDS $\mathcal{P} = (P, \Gamma, \Delta)$, a regular set of configurations $\mathcal{C}$, and a regular set of runs R of $\mathcal{P}$ from $\mathcal{C}$, the set of upper stack configurations reachable using runs in R, $\mathcal{T}(R) = \left\{ \langle p, w_u \rangle \mid \exists c \in \mathcal{C}, \exists r \in R, c \overset{r}{\Rightarrow} \langle p, w_u, w_l \rangle \right\}$, is regular and effectively computable.*

Thanks to Theorem 10, we consider the single configuration case where $\mathcal{C} = \{c_\$\}$ w.l.o.g. Let $\mathcal{A}_R = (\Delta, Q, E, I, F)$ be a finite state automaton such that $\mathcal{L}(\mathcal{A}_R) = R$. We can assume that $Q = \bigcup_{p \in P} Q_p$ where $\forall q \in Q_p$, if there is an edge $q' \xrightarrow{\delta}_E q$, then the pushdown rule δ is of the form $(p', a) \rightarrow (p, w)$. We write $F_p = Q_p \cap F$.

We introduce the finite automaton $\mathcal{A}_T = (\Gamma, Q, E', I, F)$ whose set of transitions E' is defined by applying the following rules until saturation:

(S_{pop}) if there is an edge $q_0 \xrightarrow{\delta}_E q_1$ in $\mathcal{A}_R$ and δ is of the form $(p, a) \rightarrow (p', \varepsilon)$, then we add the edge $q_0 \xrightarrow{a} q_1$ to E'.

(S_{switch}) if there is an edge $q_0 \xrightarrow{\delta}_E q_1$ in $\mathcal{A}_R$ and δ is of the form $(p, a) \rightarrow (p', b)$, then we add the edge $q_0 \xrightarrow{\varepsilon} q_1$ to E'.

(S_{push}) if there is an edge $q_0 \xrightarrow{\delta}_E q_1$ in $\mathcal{A}_R$ and δ is of the form $(p, a) \rightarrow (p', bc)$, then for each state q such that either $q \in Q$ and $q \xrightarrow{x}{}^*_{E'} q_0$ for $x \in \Gamma$ or $q \in I$ and $q \xrightarrow{\varepsilon}{}^*_{E'} q_0$, we add an edge $q \xrightarrow{\varepsilon} q_1$ to E'.

Our intuition behind the above construction is to create a new automaton that follows the structure of the run automaton but accepts upper stack words instead: an upper stack word w is accepted by $\mathcal{A}_T$ with the path $q_i \overset{w}{\rightarrow}^*_{E'} q_f$, $q_i \in I$, $q_f \in F_p$, if $\mathcal{A}_R$ accepts a run r with the path $q_i \overset{r}{\rightarrow}^*_E q_f$ and r starts from $c_\$$, ends in state p and produces the upper stack word w. This property is preserved at every step of the saturation procedure.

Consider a run r and its associated upper stack word w. Suppose that r and w satisfy the property above: there is a path $q_i \overset{r}{\rightarrow}^*_E q_0$ in $\mathcal{A}_R$ and a path $q_i \overset{w}{\rightarrow}^*_{E'} q_0$ in $\mathcal{A}_T$. Let $q_0 \overset{\delta}{\rightarrow}_E q_1$ be a transition of $\mathcal{A}_R$, $q_1 \in Q$ and $\delta \in \Delta$. $r\delta$ is also a run of $\mathcal{P}$ with a labelled path $q_i \overset{r\delta}{\rightarrow}^*_E q_1$ in $\mathcal{A}_R$, and in order to satisfy the above property, a path $q_i \overset{w'}{\rightarrow}^*_{E'} q_1$ labelled by its associated upper stack word w' should exist in $\mathcal{A}_T$ as well.

We show that the saturation rules above ensure such a path exists. If $\delta \in \Delta_{pop}$, the run $r\delta$ produces an upper stack word of the form $w' = wa$, $a \in \Gamma$. Rule (S_{pop}) creates an edge $q_0 \overset{a}{\rightarrow} q_1$ to $\mathcal{A}_T$ such that there is a path $q_i \overset{w}{\rightarrow}^*_{E'} q_0 \overset{a}{\rightarrow}_{E'} q_1$ labelled by w'. Rules (S_{switch}) and (S_{push}) follow in a similar fashion.

Following this intuition, we can prove that $\mathcal{A}_T$ only accepts reachable upper stack configurations:

Lemma 17. *At any step of the saturation procedure, if* $q_i \overset{w_u}{\rightarrow}^*_{E'} q'$, $q_i \in I$, $q' \in Q_p$, *then there exists a run* r *of* $\mathcal{P}$ *and* $w_l \in \Gamma^*$ *such that* $q_i \overset{r}{\rightarrow}^*_E q'$ *and* $c_\$ \overset{r}{\Rightarrow} \langle p, w_u, w_l \rangle$. *Moreover, if* $q' \in F_p$, *then* $r \in R$.

On the other hand, $\mathcal{A}_T$ accepts every reachable upper stack configuration:

Lemma 18. *For every run* r *such that* $\exists q_i \in I$, $\exists q \in Q_p$, $q_i \overset{r}{\rightarrow}^*_E q$, *then there exists a path* $q_i \overset{w_u}{\rightarrow}^*_{E'} q$ *in* $\mathcal{A}_T$ *and* $w_l \in \Gamma^*$ *such that* $c_\$ \overset{r}{\Rightarrow} \langle p, w_u, w_l \rangle$. *Moreover, if* $q' \in F_p$, *then* $\mathcal{A}_T$ *accepts* w_u.

Let $\mathcal{L}_p(\mathcal{A}_T) = \left\{ w \mid \exists i \in I, \exists f \in F_p, i \overset{w}{\rightarrow}^*_{E'} f \right\}$ be the set of paths in $\mathcal{A}_T$ ending in a final node related to a state p of $\mathcal{P}$. By Lemmas 18 and 17, $\mathcal{T}(R) = \{\langle p, w_u \rangle \mid w_u \in \mathcal{L}_p(\mathcal{A}_T)\}$. Since the languages $\mathcal{L}_p$ are regular and there is a finite number of them, $\mathcal{T}(R)$ is regular as well and can be computed using $\mathcal{A}_T$.

5.2 Computing an Overapproximation

The set of runs of a UPDS $\mathcal{P} = (P, \Gamma, \Delta)$ from a regular set of configurations $\mathcal{C}$ is not always regular. By Lemma 2, runs of $\mathcal{P}$ are the same for the UPDS and PDS semantics. Thus, we can apply methods originally designed for PDSs to overapproximate runs of a UPDS in a regular fashion, such as [1,3,9].

With one of these methods, we can compute a regular overapproximation $\mathcal{R}(\mathcal{P},\mathcal{C})$ of the set of runs of $\mathcal{P}$ from $\mathcal{C}$. Using the saturation procedure underlying Theorem 16, we can then compute the set $\mathcal{T}(\mathcal{R}(\mathcal{P},\mathcal{C}))$ of upper stack configurations reachable using overapproximated runs of $\mathcal{P}$, hence, an overapproximation of the actual set of reachable upper stack configurations. However, we still lack the lower stack component of the reachability set. As shown in [6], $post^*_{\mathrm{PDS}}(\mathcal{P},\mathcal{C})$ is regular and computable, and we can determine the exact set of reachable lower stack configurations.

With $O = \{\langle p, w_u, w_t\rangle \mid \langle p, w_u\rangle \in \mathcal{T}(\mathcal{R}(\mathcal{P},\mathcal{C})), \langle p, w_t\rangle \in post^*_{\mathrm{PDS}}(\mathcal{P},\mathcal{C})\}$, we get a regular overapproximation of $post^*(\mathcal{P},\mathcal{C})$.

6 Applications

The UPDS model can be used to detect stack behaviours that cannot be found using a simple pushdown system. In this section, we present three such examples.

6.1 Stack Overflow Detection

A stack overflow is a programming malfunction occurring when the call stack pointer exceeds the stack bound. In order to analyze a program's vulnerability to stack overflow errors, we compute its representation as a UPDS $\mathcal{P} = (P, \Gamma, \Delta)$, using the control flow model outlined in [6].

Let $\mathcal{C} = P \times \top\#^m \times L$ be the set of starting configurations, where $\top \in \Gamma$ is a top stack symbol that does not appear in any rule in Δ, $\# \in \Gamma$ a filler symbol, m an integer depending on the maximal size of the stack, and L a regular language of lower stack initial words. Overwriting the top symbol would represent a stack overflow malfunction. Since there is no such thing as an upper stack in a simple pushdown automaton, we need a UPDS to detect this error.

Let $\mathcal{X} = P \times (\Gamma \setminus \{\top\})^* \times \Gamma^*$ be the set of forbidden configurations where the top stack symbol has been overwritten. If the intersection of the underapproximation $\mathcal{U}$ of $pre^*(\mathcal{X})$ with $\mathcal{C}$ is not empty, then a stack overflow does happen in the program. On the other hand, if the intersection of the overapproximation $\mathcal{O}$ of $post^*(\mathcal{C})$ with the set $\mathcal{X}$ of forbidden configurations is empty, then we are sure that a stack overflow will not happen in the program

6.2 Reading the Upper Stack

Let us consider the piece of code 1.1. In line 1, the bottom symbol of the upper stack $\mathsf{sp} - 4$, just above the stack pointer, is copied into the register eax. In line 2, the content of eax is compared to a given value a. In line 3, if the two values are equal, the program jumps to an error state err.

Listing 1.1. Reading the upper stack

```
1    mov eax, [sp - 4]
2    cmp eax, a
3    je err
```

Using a simple PDS model, it is not possible to know what is being read. However, our UPDS model and the previous algorithms provide us with reasonable approximations which can be used to examine possible values stored in eax.

To check whether this program reaches the error state *err* or not, we define the regular set $\mathcal{X} = P \times \Gamma^* a \times \Gamma^*$ of forbidden configurations where a is present on the upper stack just above the stack pointer. If the intersection of the underapproximation of $pre^*(\mathcal{X})$ with the set of starting configurations $\mathcal{C}$ of the program is not empty, then eax can contain a critical value, and the program is unsafe. On the other hand, if the intersection of the overapproximation of $post^*(\mathcal{C})$ with the set $\mathcal{X}$ is empty, then the program can be considered safe.

6.3 Changing the Stack Pointer

Another malicious use of the stack pointer sp would be to change the starting point of the stack. As an example, the instruction mov sp, sp - 12 changes the stack pointer in such a manner that, from the configuration of Fig. 7, the top three elements above it now belong to the stack, as shown in Fig. 8.

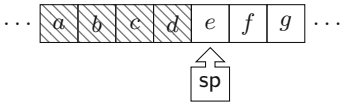

Fig. 7. Original stack

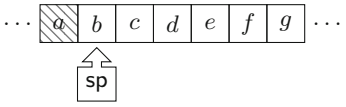

Fig. 8. After changing sp

If we model a program as a UPDS, then using our previous algorithms to compute approximations of the reachability set would allow us to have an approximation of the content of the new stack after the stack pointer change.

References

1. Bermudez, M.E., Schimpf, K.M.: Practical arbitrary lookahead LR parsing. J. Comput. Syst. Sci. **41**, 230–250 (1990)
2. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: application to model-checking. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 135–150. Springer, Heidelberg (1997). doi:10.1007/3-540-63141-0_10
3. Bouajjani, A., Esparza, J., Touili, T.: A generic approach to the static analysis of concurrent programs with procedures. In: POPL 2003 (2003)
4. Carotenuto, D., Murano, A., Peron, A.: 2-visibly pushdown automata. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 132–144. Springer, Heidelberg (2007). doi:10.1007/978-3-540-73208-2_15
5. Caucal, D.: On the regular structure of prefix rewriting. Theor. Comput. Sci. **106**, 61–86 (1992)

6. 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). doi:10.1007/10722167_20
7. Ginsburg, S., Greibach, S.A., Harrison, M.A.: Stack automata and compiling. J. ACM **14**, 172–201 (1967)
8. Hopcroft, J., Ullman, J.: Sets accepted by one-way stack automata are context sensitive. Inf. Control **13**, 114–133 (1968)
9. Pereira, F.C.N., Wright, R.N.: Finite-state approximation of phrase structure grammars. In: ACL 1991 (1991)
10. 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). doi:10.1007/978-3-540-31980-1_7
11. Seth, A.: Global reachability in bounded phase multi-stack pushdown systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 615–628. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14295-6_53
12. Torre, S.L., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: LICS 2007 (2007)
13. Uezato, Y., Minamide, Y.: Pushdown systems with stack manipulation. In: Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 412–426. Springer, Heidelberg (2013). doi:10.1007/978-3-319-02444-8_29