Minimization of Visibly Pushdown Automata Using Partial Max-SAT

Matthias Heizmann, Christian Schilling[(⊠)], and Daniel Tischner

University of Freiburg, Freiburg, Germany
`schillic@informatik.uni-freiburg.de`

Abstract. We consider the problem of state-space reduction for nondeterministic weakly-hierarchical visibly pushdown automata (Vpa). Vpa recognize a robust and algorithmically tractable fragment of context-free languages that is natural for modeling programs.

We define an equivalence relation that is sufficient for language-preserving quotienting of Vpa. Our definition allows to merge states that have different behavior, as long as they show the same behavior for reachable equivalent stacks. We encode the existence of such a relation as a Boolean partial maximum satisfiability (PMax-Sat) problem and present an algorithm that quickly finds satisfying assignments. These assignments are sub-optimal solutions to the PMax-Sat problem but can still lead to a significant reduction of states.

We integrated our method in the automata-based software verifier Ultimate Automizer and show performance improvements on benchmarks from the software verification competition SV-COMP.

1 Introduction

The class of visibly pushdown languages (Vpl) [6] lies properly between the regular and the context-free languages. Vpl enjoy most desirable properties of regular languages (closure under Boolean operations and decision procedures for, e.g., the equivalence problem). They are well-suited for representing data that have both a linear and a hierarchical ordering, e.g., procedural programs [4,22,24,37] and XML documents [31,33,34,38].

The corresponding automaton model is called *visibly pushdown automaton* (Vpa). It extends the finite automaton model with a stack of restricted access by requiring that the input symbol specifies the stack action – a call (resp. return) symbol implies a push (resp. pop) operation, and an internal symbol ignores the stack. In this paper, we consider a notion of Vpa where a call always pushes the current state on the stack. These Vpa are called weakly-hierarchical Vpa [7].

Size reduction of automata is an active research topic [2,3,8,9,14,15,32] that is theoretically appealing and has practical relevance: smaller automata require less memory and speed up automata-based tools [21,23,27,29]. In this paper, we present a size reduction technique for a general class of (nondeterministic) Vpa that is different from classes that were considered in previous approaches [5,13,30]. An extended version of this paper is available [26].

It is well-known that for deterministic finite automata the unique minimal automaton can be obtained by quotienting (i.e., merging equivalent states), and there exists an efficient algorithm for this purpose [28]. VPA do not have a canonical minimum [5]. For other automaton classes that lack this property, the usual approach is to find equivalence relations that are sufficient for quotienting [1,3,18]. The main difficulty of a quotienting approach for VPA is that two states may behave similarly given one stack but differently given another stack, and as the number of stacks is usually infinite, one cannot simply compare the behaviors for each of them.

1.1 Motivating Examples

We now present three observations. The first observation is our key insight and shows that VPA have interesting properties that we can exploit. The other observations show that VPA have intricate properties that make quotienting nontrivial. For convenience, we use a for internal, c for call, and r for return symbols, and we omit transitions to the sink state.

Exploiting Unreachable Stacks Allows Merging States. Consider the VPA in Fig. 1(a). The states q_1 and q_2 have the same behavior for the internal symbol a but different behaviors for the return symbol r with stack symbol q_0: Namely, state q_1 leads to the accepting state while q_2 has no respective return transition. However, in q_2 it is generally impossible to take a return transition with stack symbol q_0 since q_2 can only be reached with an empty stack. Thus the behavior for the stack symbol q_0 is "undefined" and we can merge q_1 and q_2 without changing the language. The resulting VPA is depicted in Fig. 1(b).
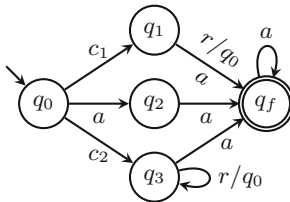
(a) A VPA.
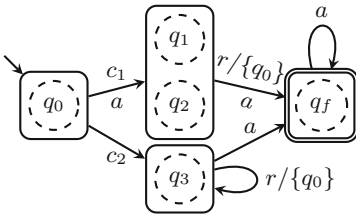
(b) One possible quotienting.
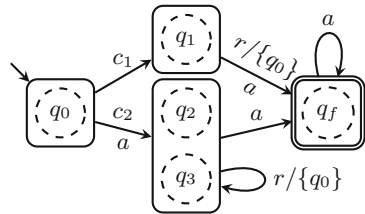
(c) Another possible quotienting.

Fig. 1. A VPA and two possible quotientings due to unreachable stacks.

Merging States Requires a Transitive Relation. Using the same argument as above, we can also merge the states q_2 and q_3; the result is depicted in Fig. 1(c). For finite automata, mergeability of states is transitive. However, here we cannot merge all three states q_1, q_2, and q_3 without changing the language because q_1 and q_3 have different behaviors for stack symbol q_0. For VPA, we have to check compatibility for each pair of states.

Merging States Means Merging Stack Symbols. Consider the VPA in Fig. 2(a). Since for (weakly-hierarchical) VPA, stack symbols are states, merging the states q_1 and q_2 implicitly merges the stack symbols q_1 and q_2 as well. After merging we receive the VPA in Fig. 2(b) which recognizes a different language (e.g., it accepts the word $a_1 c\, r_2$).
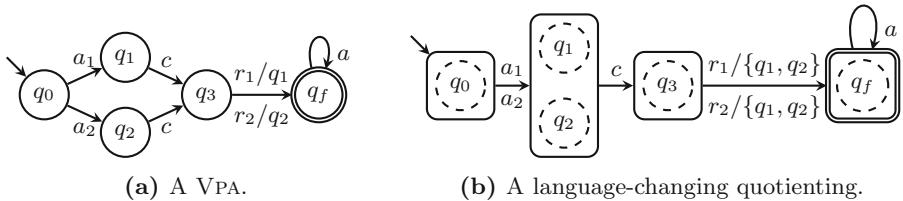

(a) A VPA.   **(b)** A language-changing quotienting.

Fig. 2. A VPA where quotienting of states leads to quotienting of stack symbols.

1.2   Our Approach

We define an equivalence relation over VPA states for quotienting that is language-preserving. This equivalence relation exploits our key observation, namely that we can merge states if they have the same behavior on equivalent reachable stacks, even if they have different behavior in general (Sect. 3). We show an encoding of such a relation as a Boolean partial maximum satisfiability (PMax-SAT) instance (Sect. 4). In order to solve these instances efficiently, we propose a greedy algorithm that finds suboptimal solutions (Sect. 5.1). As a proof of concept, we implemented the algorithm and evaluated it in the context of the automata-based software verifier ULTIMATE AUTOMIZER [23,25] (Sect. 5.2).

2   Visibly Pushdown Automata

In this section, we recall the basic definitions for visibly pushdown automata [6] and quotienting. After that, we characterize when an automaton is *live*.

2.1   Preliminaries

Alphabet. A *(visibly pushdown) alphabet* $\Sigma = \Sigma_i \uplus \Sigma_c \uplus \Sigma_r$ is a partition consisting of three finite sets of *internal* (Σ_i), *call* (Σ_c), and *return* (Σ_r) symbols.

A *word* is a sequence of symbols. We denote the set of finite words over alphabet Σ by Σ^* and the empty word by ε. As a convention we use a for internal, c for call, and r for return symbols, x for any type of symbol, and v, w for words.

The set of *well-matched* words over Σ, $WM(\Sigma)$, is the smallest set satisfying: (1) $\varepsilon \in WM(\Sigma)$; (2) if $w \in WM(\Sigma)$, so is wa for $a \in \Sigma_{\mathrm{i}}$; and if $v, w \in WM(\Sigma)$, so is $vcwr$ for $cr \in \Sigma_{\mathrm{c}} \cdot \Sigma_{\mathrm{r}}$, and we call symbols c and r *matching*. Given a word over Σ, for any return symbol we can uniquely determine whether the symbol is matching. The set of *matched-return* words, $MR(\Sigma)$, consists of all words where each return symbol is matching. Clearly, $WM(\Sigma)$ is a subset of $MR(\Sigma)$.

Visibly Pushdown Automaton. A *visibly pushdown automaton* (VPA) is a tuple $\mathcal{A} = (Q, \Sigma, \bot, \Delta, Q_0, F)$ with a finite set of states Q, a visibly pushdown alphabet Σ, a bottom-of-stack symbol $\bot \notin Q$, a transition relation $\Delta = (\Delta_{\mathrm{i}}, \Delta_{\mathrm{c}}, \Delta_{\mathrm{r}})$ consisting of internal transitions $\Delta_{\mathrm{i}} \subseteq Q \times \Sigma_{\mathrm{i}} \times Q$, call transitions $\Delta_{\mathrm{c}} \subseteq Q \times \Sigma_{\mathrm{c}} \times Q$, and return transitions $\Delta_{\mathrm{r}} \subseteq Q \times \Sigma_{\mathrm{r}} \times Q \times Q$, a nonempty set of initial states $Q_0 \subseteq Q$, and a set of accepting states $F \subseteq Q$.

A *stack* σ is a word over $St \stackrel{\text{def}}{=} \{\bot\} \cdot Q^*$. We write $\sigma[i]$ for the i-th symbol of σ. A *configuration* is a pair $(q, \sigma) \in Q \times St$. A *run* $\rho_{\mathcal{A}}(w)$ of VPA $\mathcal{A}$ on word $w = x_1 x_2 \cdots \in \Sigma^*$ is a sequence of configurations $(q_0, \sigma_0)(q_1, \sigma_1) \cdots$ according to the following rules (for $i \geq 0$):

1. If $x_{i+1} \in \Sigma_{\mathrm{i}}$ then $(q_i, x_{i+1}, q_{i+1}) \in \Delta_{\mathrm{i}}$ and $\sigma_{i+1} = \sigma_i$.
2. If $x_{i+1} \in \Sigma_{\mathrm{c}}$ then $(q_i, x_{i+1}, q_{i+1}) \in \Delta_{\mathrm{c}}$ and $\sigma_{i+1} = \sigma_i \cdot q_i$.
3. If $x_{i+1} \in \Sigma_{\mathrm{r}}$ then $(q_i, x_{i+1}, \hat{q}, q_{i+1}) \in \Delta_{\mathrm{r}}$ and $\sigma_i = \sigma_{i+1} \cdot \hat{q}$.

A run is *initial* if $(q_0, \sigma_0) \in Q_0 \times \{\bot\}$. A configuration (q, σ) is *reachable* if there exists some initial run $\rho = (q_0, \sigma_0)(q_1, \sigma_1) \cdots$ such that $(q_i, \sigma_i) = (q, \sigma)$ for some $i \geq 0$, and *unreachable* otherwise. Similarly, we say that a stack σ is reachable (resp. unreachable) for state q if (q, σ) is reachable (resp. unreachable). A run of length n is *accepting* if $q_n \in F$. A word $w \in \Sigma^*$ is *accepted* if some initial run $\rho_{\mathcal{A}}(w)$ is accepting. The *language* recognized by a VPA $\mathcal{A}$ is defined as $L(\mathcal{A}) \stackrel{\text{def}}{=} \{w \mid w \text{ is accepted by } \mathcal{A}\}$. A VPA is *deterministic* if it has one initial state and the transition relation is functional.

A *finite automaton* (FA) is a VPA where $\Sigma_{\mathrm{c}} = \Sigma_{\mathrm{r}} = \emptyset$.

Remark 1. We use a variant of VPA that deviates from the VPA model by Alur and Madhusudan [6] in two ways: (1) We forbid return transitions when the stack is empty, i.e., the automata accept only matched-return words; this assumption is also used in other works [30,35]. (2) We consider *weakly-hierarchical* VPA where a call transition implicitly pushes the current state on the stack; this assumption is also a common assumption [13,30]; every VPA can be converted to weakly-hierarchical form with $2|Q||\Sigma|$ states [7].

Both assumptions are natural in the context of computer programs: The call stack can never be empty, and return transitions always lead back to the respective program location after the corresponding call.

Quotienting. For an equivalence relation over some set S, we denote the equivalence class of element e by $[e]$; analogously, lifted to sets, let $[T] \overset{\text{def}}{=} \{[e] \mid e \in T\}$.

Given a VPA $\mathcal{A} = (Q, \Sigma, \bot, (\Delta_\text{i}, \Delta_\text{c}, \Delta_\text{r}), Q_0, F)$ and an equivalence relation $\equiv \subseteq Q \times Q$ on states, the *quotient* VPA is the VPA $\mathcal{A}/_\equiv \overset{\text{def}}{=} ([Q], \Sigma, \bot, \Delta', [Q_0], [F])$ with $\Delta' = (\Delta'_\text{i}, \Delta'_\text{c}, \Delta'_\text{r})$ and

- $\Delta'_\text{i} = \{([p], a, [p']) \mid \exists (q, a, q') \in \Delta_\text{i}. \, q \in [p], q' \in [p']\}$,
- $\Delta'_\text{c} = \{([p], c, [p']) \mid \exists (q, c, q') \in \Delta_\text{c}. \, q \in [p], q' \in [p']\}$, and
- $\Delta'_\text{r} = \{([p], r, [\hat{p}], [p']) \mid \exists (q, r, \hat{q}, q') \in \Delta_\text{r}. \, q \in [p], q' \in [p'], \hat{q} \in [\hat{p}]\}$.

Quotienting is the process of merging states from the same equivalence class to obtain the quotient VPA; this implicitly means merging stack symbols, too.

2.2   Live Visibly Pushdown Automata

Let $Q_\bot \overset{\text{def}}{=} Q \cup \{\bot\}$ be the *stack alphabet*. The function $top : St \to Q_\bot$ returns the topmost symbol of a stack:

$$top(\sigma) \overset{\text{def}}{=} \begin{cases} \bot & \sigma = \bot \\ q & \sigma = \sigma' \cdot q \text{ for some } \sigma' \in St \end{cases}$$

Given a state q, the function $tops : Q \to 2^{Q_\bot}$ returns the topmost symbols of all reachable stacks σ for q (i.e., reachable configurations (q, σ)):

$$tops(q) \overset{\text{def}}{=} \{top(\sigma) \mid \exists \sigma \in St. \, (q, \sigma) \text{ is reachable}\}$$

For seeing that *tops* is computable, consider a VPA $\mathcal{A} = (Q, \Sigma, \bot, \Delta, Q_0, F)$. The function *tops* is the smallest function $f : Q \to 2^{Q_\bot}$ satisfying:

1. $q \in Q_0 \implies \bot \in f(q)$
2. $\hat{q} \in f(q), (q, a, q') \in \Delta_\text{i} \implies \hat{q} \in f(q')$
3. (q, σ) reachable for some σ, $(q, c, q') \in \Delta_\text{c} \implies q \in f(q')$
4. $\hat{q} \in f(q), (q, r, \hat{q}, q') \in \Delta_\text{r} \implies f(\hat{q}) \subseteq f(q')$

We call a VPA *live* if the following holds. For each state q and for each internal and call symbol x there is at least one outgoing transition (q, x, q') to some state q'; additionally, for each return symbol r and state $\hat{q}$ there is at least one outgoing return transition $(q, \hat{q}, r, q')$ to some state q' if and only if $\hat{q} \in tops(q)$.

Note that a live VPA has a total transition relation in a weaker sense: There are outgoing return transitions from state q if and only if the respective transition can be taken in at least one run. That is, we forbid return transitions when no corresponding configuration is reachable. Every VPA can be converted to live form by adding one sink state.

Remark 2. For live VPA $\mathcal{A}$, a run $\rho_\mathcal{A}(w)$ on word w can only "get stuck" in an empty-stack configuration, i.e., if $w = v_1 r v_2$ with $r \in \Sigma_\text{r}$ such that $\rho_\mathcal{A}(v_1) = (q_0, \sigma_0) \cdots (q_k, \bot)$ for some $q_k \in Q$. If $w \in MR(\Sigma)$, no run gets stuck.

For the remainder of the paper, we fix a live VPA $\mathcal{A} = (Q, \Sigma, \bot, \Delta, Q_0, F)$. We sometimes refer to this VPA as the *input automaton*.

3   A Quotienting Relation for VPA

In this section, we define an equivalence relation on the states of a VPA that is useful for quotienting, i.e., whose respective quotient VPA is language-preserving.

We first need the notion of *closure under successors* for each kind of symbol.

Let $R \subseteq Q \times Q$ be a binary relation over states and let $p, q, \hat{p}, \hat{q} \in Q$ be states. We say that R is

- *closed under internal successors for* (p, q) if for each internal symbol $a \in \Sigma_i$
    - for all $(p, a, p') \in \Delta_i$ there exists $(q, a, q') \in \Delta_i$ s.t. $(p', q') \in R$ and
    - for all $(q, a, q') \in \Delta_i$ there exists $(p, a, p') \in \Delta_i$ s.t. $(p', q') \in R$,
- *closed under call successors for* (p, q) if for each call symbol $c \in \Sigma_c$
    - for all $(p, c, p') \in \Delta_c$ there exists $(q, c, q') \in \Delta_c$ s.t. $(p', q') \in R$ and
    - for all $(q, c, q') \in \Delta_c$ there exists $(p, c, p') \in \Delta_c$ s.t. $(p', q') \in R$,
- *closed under return successors for* $(p, q, \hat{p}, \hat{q})$ if for each return symbol $r \in \Sigma_r$
    - for all $(p, r, \hat{p}, p') \in \Delta_r$ there exists $(q, r, \hat{q}, q') \in \Delta_r$ s.t. $(p', q') \in R$ and
    - for all $(q, r, \hat{q}, q') \in \Delta_r$ there exists $(p, r, \hat{p}, p') \in \Delta_r$ s.t. $(p', q') \in R$.

We are ready to present an equivalence relation that is useful for quotienting using a fixpoint characterization.

Definition 1 (Reachability-aware quotienting relation). *Let* $\mathcal{A}$ *be a VPA and* $R \subseteq Q \times Q$ *be an equivalence relation over states. We say that* R *is a* RAQ *relation if for each pair of states* $(p, q) \in R$ *the following constraints hold.*

(i) State p *is accepting if and only if state* q *is accepting* $(p \in F \iff q \in F)$.
(ii) R is closed under internal successors for (p, q).
(iii) R is closed under call successors for (p, q).
(iv) For each pair of states (resp. topmost stack symbols) $(\hat{p}, \hat{q}) \in R$,
   - *R is closed under return successors for* $(p, q, \hat{p}, \hat{q})$, *or*
   - *no configuration* (q, σ_q) *with* $\hat{q} = top(\sigma_q)$ *is reachable, or*
   - *no configuration* (p, σ_p) *with* $\hat{p} = top(\sigma_p)$ *is reachable.*

Remark 3. "No configuration (q, σ_q) with $\hat{q} = top(\sigma_q)$ is reachable" is equivalent to "$\hat{q} \notin tops(q)$". The equality relation $\{(q, q) \mid q \in Q\}$ is a RAQ relation for any VPA; the respective quotient VPA is isomorphic to the input automaton.

Example 1. Consider again the VPA from Fig. 1(a). We claim that the relation $R \overset{\text{def}}{=} \{(q, q) \mid q \in Q\} \cup \{(q_1, q_2), (q_2, q_1)\}$ is a RAQ relation. Note that it corresponds to the quotient VPA from Fig. 1(b). First we observe that R is an equivalence relation. We check the remaining constraints only for the two pairs (q_1, q_2) and (q_2, q_1). Both states are not accepting. Relation R is closed under internal (here: a) and call (here: none, i.e., implicitly leading to a sink) successors. The return transition constraint is satisfied because in state q_2 no stack with topmost symbol q_0 is reachable ($q_0 \notin tops(q_2)$).

We want to use a RAQ relation for language-preserving quotienting. For this purpose we need to make sure that unreachable configurations in Definition 1 do not enable accepting runs that are not possible in the original VPA. In the remainder of this section, we show that this is indeed the case.

Given an equivalence relation $R \subseteq Q \times Q$ on states, we call a stack σ the *R-quotienting* of some stack σ' of the same height if either $\sigma = \sigma' = \perp$ or for all $i = 2, \ldots, |\sigma|$ each symbol $\sigma[i]$ is the equivalence class of $\sigma'[i]$, i.e., $\sigma'[i] \in [\sigma[i]]$. We write $\sigma' \in [\sigma]$ in this case. (We compare stacks only for $i \geq 2$ because the first stack symbol is always $\perp$.)

Lemma 1 (Corresponding run). *Let $\mathcal{A}$ be a VPA and $\equiv$ be some RAQ relation for $\mathcal{A}$. Then for any matched-return word w and respective run*

$$\rho_{\mathcal{A}/\equiv}(w) = ([p_0], \perp) \cdots ([p_n], [\sigma_n])$$

with $p_0 \in Q_0$ in $\mathcal{A}/\equiv$ there is some corresponding run

$$\rho_{\mathcal{A}}(w) = (q'_0, \perp) \cdots (q'_n, \sigma'_n)$$

in $\mathcal{A}$ such that $q'_i \in [p_i]$ and $\sigma'_i \in [\sigma_i]$ for all $i \geq 0$, and furthermore $q'_0 \in Q_0$.

Proof. The proof is by induction on the length of w. The case for $w = \varepsilon$ is trivial. Now assume $w' = w \cdot x$ for $x \in \Sigma$ and fix some run $\rho_{\mathcal{A}/\equiv}(w') = ([p_0], \perp) \cdots ([p_n], [\sigma_n]) \cdot ([p_{n+1}], [\sigma_{n+1}])$. The hypothesis ensures that there exists a corresponding run for the prefix $\rho_{\mathcal{A}}(w) = (q'_0, \perp) \cdots (q'_n, \sigma'_n)$ s.t. $q'_n \in [p_n]$ and $\sigma'_n \in [\sigma_n]$. We will extend this run in each of the three cases for symbol x.

(1) If $x \in \Sigma_i$, then, since there is a transition $([p_n], x, [p_{n+1}]) \in \Delta_{i/\equiv}$, there exist some states $q''_n \in [p_n]$ and $q''_{n+1} \in [p_{n+1}]$ s.t. $(q''_n, x, q''_{n+1}) \in \Delta_i$ (from the definition of quotienting). Using that $\equiv$ is closed under internal successors, there also exists a target state $q'_{n+1} \in [p_{n+1}]$ s.t. $(q'_n, x, q'_{n+1}) \in \Delta_i$. Additionally, because $x \in \Sigma_i$, we have that $\sigma'_{n+1} = \sigma'_n \in [\sigma_n] = [\sigma_{n+1}]$ by the hypothesis.

(2) If $x \in \Sigma_c$, a similar argument holds, only this time the stack changes. We have that $\sigma'_{n+1} = \sigma'_n \cdot q'_n \in [\sigma_n \cdot p_n] = [\sigma_{n+1}]$ by the hypothesis.

(3) If $x \in \Sigma_r$, then the configuration (q'_n, σ'_n) is reachable (witnessed by the run $\rho_{\mathcal{A}}(w)$). Since $\equiv$ is closed under return successors for all states in $[p_n]$ (modulo unreachable configurations), for each top-of-stack symbol $\hat{q} \in [top(\sigma'_n)]$ s.t. $(q'_n, \sigma'' \cdot \hat{q})$ is reachable for some stack σ'' there exists a corresponding return transition $(q'_n, x, \hat{q}, q'_{n+1}) \in \Delta_r$ with $q'_{n+1} \in [p_{n+1}]$; in particular, this holds for $\hat{q} = top(\sigma'_n)$. Recall that $\mathcal{A}$ is assumed to be live, which ensures that every return transition that exists in the quotient VPA has such a witness. The stack property $\sigma'_{n+1} \in [\sigma_{n+1}]$ follows from the hypothesis.   □

From the above lemma we can conclude that quotienting with a RAQ relation preserves the language.

Theorem 1 (Language preservation of quotienting). *Let $\mathcal{A}$ be a VPA and let $\equiv$ be a RAQ relation on the states of $\mathcal{A}$. Then $L(\mathcal{A}) = L(\mathcal{A}/\equiv)$.*

Proof. Clearly, $L(\mathcal{A}) \subseteq L(\mathcal{A}/_{\equiv})$ for any equivalence relation $\equiv$. We show the other inclusion by means of a contradiction.

Assume there exists a word w s.t. $w \in L(\mathcal{A}/_{\equiv}) \setminus L(\mathcal{A})$. By assumption, in $\mathcal{A}/_{\equiv}$ there is an initial accepting run $\rho_{\mathcal{A}/_{\equiv}}(w)$. Then, by Lemma 1, there is a corresponding run $\rho_{\mathcal{A}}(w)$, and furthermore this run is initial.

The run $\rho_{\mathcal{A}}(w)$ is also accepting by the property that $[p] \in [F]$ if and only if $q \in F$ for all $q \in [p]$ (cf. Property (i) of a RAQ relation). $\square$

4   Computing Quotienting Relations

In Sect. 3, we introduced the notion of a RAQ relation and showed how we can use it to minimize VPA while preserving the language. In this section, we show how we can compute a RAQ relation. For this purpose, we provide an encoding as a partial maximum satisfiability problem (PMax-SAT). From a (in fact, any) solution, i.e., satisfying assignment, we can synthesize a RAQ relation. While this does not result in the coarsest RAQ relation possible, the relation obtained is locally optimal, i.e., there is no coarser RAQ relation that is a strict superset.

4.1   Computing RAQ Relations

Note that in general there are many possible instantiations of a RAQ relation, e.g., the trivial equality relation which is not helpful for minimization. Since we are interested in reducing the number of states, we prefer coarser relations over finer relations.

To obtain a coarse relation, we describe an encoding of the RAQ relation constraints as an instance of the PMax-SAT problem [12,19]. Such a problem consists of a propositional logic formula in conjunctive normal form with each clause being marked as either *hard* or *soft*. The task is to find a truth assignment such that all hard clauses are satisfied and the number of the satisfied soft clauses is maximal.

SAT Encoding. For the moment, we ignore soft clauses and provide a standard SAT encoding of the constraints. The encoding has the property that any satisfying assignment induces a valid RAQ relation $\equiv$.

Let true and false be the Boolean constants. We need $\mathcal{O}(n^2)$ variables of the form $\mathsf{X}_{\{p,q\}}$ where p and q are states of the input automaton. The idea is that $p \equiv q$ holds if we assign the value true to $\mathsf{X}_{\{p,q\}}$. (We ignore the order of p and q as $\equiv$ must be symmetric.) We express the constraints from Definition 1 as follows.

Consider the constraint (i). For each pair of states (p, q) not satisfying the constraint we introduce the clause

$$\neg\mathsf{X}_{\{p,q\}}. \tag{1}$$

Consider the constraints (ii), (iii), (iv). For each transition $(p, a, p') \in \Delta_i$, $(p, c, p') \in \Delta_c$, and $(p, r, \hat{p}, p') \in \Delta_r$ and all states q and $\hat{q}$ we respectively construct one of the following clauses.

$$\neg X_{\{p,q\}} \vee (X_{\{p',q_1^a\}} \vee \cdots \vee X_{\{p',q_{k_a}^a\}}) \tag{2}$$

$$\neg X_{\{p,q\}} \vee (X_{\{p',q_1^c\}} \vee \cdots \vee X_{\{p',q_{k_c}^c\}}) \tag{3}$$

$$\neg X_{\{p,q\}} \vee \neg X_{\{\hat{p},\hat{q}\}} \vee (X_{\{p',q_1^r\}} \vee \cdots \vee X_{\{p',q_{k_r}^r\}}) \tag{4}$$

Here the q_i^a / q_i^c are the respective a/c-successors of q and the q_i^r are the r-successors of q with stack symbol $\hat{q}$. To account for the unreachable configuration relaxation, we may omit return transition clauses (4) where $\hat{p} \notin tops(p)$ or $\hat{q} \notin tops(q)$.

We also need to express that $\equiv$ is an equivalence relation, i.e., we need additional reflexivity clauses

$$X_{\{q_1,q_1\}} \tag{5}$$

and transitivity clauses

$$\neg X_{\{q_1,q_2\}} \vee \neg X_{\{q_2,q_3\}} \vee X_{\{q_1,q_3\}} \tag{6}$$

for any distinct states q_1, q_2, q_3 (assuming there are least three states). Recall that our variables already ensure symmetry.

Let Φ be the conjunction of all clauses of the form (1), (2), (3), (4), (5), and (6). All assignments satisfying Φ represent valid RAQ relations.

However, we know that the assignment

$$X_{\{p,q\}} \mapsto \begin{cases} \text{true} & p = q \\ \text{false} & \text{otherwise} \end{cases}$$

corresponding to the equality relation is always trivially satisfying. Such an assignment is not suited for our needs. We consider an assignment *optimal* if it represents a RAQ relation with a coarsest partition.

PMax-SAT Encoding. We now describe an extension of the SAT encoding to a PMax-SAT encoding. In this setting, we can enforce that the number of variables that are assigned the value true is maximal.

As an addition to Φ, we add for every two states p, q with $p \neq q$ the clause

$$X_{\{p,q\}} \tag{7}$$

and finally we consider all old clauses, i.e., clauses of the form (1)–(6), as hard clauses and all clauses of the form (7) as soft clauses.

4.2   Locally Maximal RAQ Relation

Note that an assignment obtained from the PMax-SAT encoding does not necessarily give us a coarsest RAQ relation. Consider a VPA with seven states $q_0, \ldots, q_6$ and the partition $\{\{q_0, q_1, q_2, q_3\}, \{q_4\}, \{q_5\}, \{q_6\}\}$. Here we set six variables to true (all pairs of states from the first set). However, the partition $\{\{q_0, q_1, q_2\}, \{q_3, q_4\}, \{q_5, q_6\}\}$ is coarser, and yet we only set five variables to true.

Despite not finding the globally maximal solution, we can establish local maximality.

Theorem 2 (Local maximum). *A satisfying assignment of the* PM*ax-S*AT *instance corresponds to a R*AQ *relation such that no strict superset of the relation is also a R*AQ *relation.*

Proof. It is clear from the construction that in the obtained assignment, no further variable $\mathsf{X}_{\{p,q\}}$ can be assigned the value true. Each such variable determines membership of the symmetric pairs (p, q) and (q, p) in the RAQ relation.   □

5   Experimental Evaluation

In this section, we report on our implementation and its potential in practice.

5.1   Implementation

Initially, we apply the following preprocessing steps for reducing the complexity. First, we remove unreachable and dead states and make the VPA live for return transitions (we do not require that the VPA is total for internal or call transitions). Second, we immediately replace variables $\mathsf{X}_{\{p\}}$ by true (reflexivity). Third, we construct an initial partition of the states and replace variables $\mathsf{X}_{\{p,q\}}$ by false if p and q are not in the same block. This partition is the coarsest fixpoint of a simple partition refinement such that states in the same block have the same acceptance status, the same outgoing internal and call symbols, and, if all states in a block have a unique successor under an internal/call symbol, those successors are in the same block (cf. Definition 1 and Hopcroft's algorithm [28]).

Optimally solving a PMax-SAT instance is an NP-complete problem. Expectedly, a straightforward implementation of the algorithm presented in Sect. 4 using an off-the-shelf PMax-SAT solver does not scale to interesting problems (see also the extended version [26]). Therefore, we implemented a domain-specific greedy PMax-SAT solver that only maximizes the satisfied soft clauses locally.

Our solver is interactive, i.e., clauses are added one after another, and propagation is applied immediately. After adding the last clause, the solver chooses some unset variable and first sets it to true optimistically. Theorem 2 still holds with this strategy. Apart from that, the solver follows the standard DPLL algorithm and uses no further enhancements found in modern SAT solvers.

Remark 4. If the VPA is deterministic, we obtain a Horn clause system. Then the above algorithm never needs to backtrack for more than one level, as the remaining clauses can always be satisfied by assigning false to the variables.

The main limitation of the approach is the memory consumption. Clearly, the majority of clauses are those expressing transitivity. Therefore, we implemented and integrated a solver for the theory of equality: When a variable $X_{\{p,q\}}$ is set to true, this solver returns all variables that must also be set to true for consistency. That allowed us to omit the transitivity clauses (see [26] for details).

5.2   Experiments

Our evaluation consists of three parts. First, we evaluate the impact of our minimization on an verifier ULTIMATE AUTOMIZER. Second, we evaluate the performance of our minimization on automata that were produced by ULTIMATE AUTOMIZER. Third, we evaluate the performance of our minimization on a set of random automata. All experiments are performed on a PC with an Intel i7 3.60 GHz CPU running Linux.

Impact on the software verifier Ultimate Automizer. The software verifier ULTIMATE AUTOMIZER [23] follows an automata-based approach [25] in which sets of program traces are represented by automata. The approach can be seen as a CEGAR-style algorithm in which an abstraction is iteratively refined. This abstraction is represented as a weakly-hierarchical VPA where the automaton stack only keeps track of the states from where function calls were triggered.

For our evaluation, we run ULTIMATE AUTOMIZER on a set of C programs in two different modes. In the mode "No minimization" no automata minimization is applied. In the mode "Minimization" we apply our minimization in each iteration of the CEGAR loop to the abstraction if it has less than 10,000 states. (In cases where the abstraction has more than 10,000 states the minimization can be too slow to pay off on average.)

As benchmarks we took C programs from the repository of the SV-COMP 2016 [10] and let ULTIMATE AUTOMIZER analyze if the error location is reachable. In this repository the folders `systemc` and `eca-rers2012` contain programs that use function calls (hence the VPA contain calls and returns) and in whose analysis the automata sizes are a bottleneck for ULTIMATE AUTOMIZER. We randomly picked 100 files from the `eca-rers2012` folder and took all 65 files from the `systemc` folder. The timeout of ULTIMATE AUTOMIZER was set to 300 s and the available memory was restricted to 4 GiB.

The results are given in Table 1. Our minimization increases the number of programs that are successfully analyzed from 66 to 78. On programs that are successfully analyzed in both modes, the mode using minimization is slightly faster. Hence, the additional cost due to minimization is more than compensated by savings in other operations on the (now smaller) VPA on average.

Evaluation on automata from Ultimate Automizer. To evaluate the performance of our minimization algorithm in more details, we applied it to a benchmark set that consists of 1026 VPA produced by ULTIMATE AUTOMIZER. All

Table 1. Performance of Ultimate Automizer with and without minimization. Column # shows the number of successful reachability analyses (out of 165), average run time is given in milliseconds, average removal shows the states removed for all iterations, and the last column shows the relative number of iterations where minimization was employed. The first two rows contain the data for those programs where both modes succeeded, and the third row contains the data for those programs where only the minimization mode succeeded.

Mode	Set	#	∅ Time total	∅ Time minimization	∅ Removal	% Iterations with minimization
No minimization	Both	66	16085	–	–	–
Minimization			15564	2649	3077	75
Minimization	Exclusive	12	101985	61384	8472	76

Table 2. Performance of our algorithm on automata produced by Ultimate Automizer (see also Fig. 3). We aggregate the data for all automata whose number of states is in a certain interval. Column # shows the number of automata, #nd shows the number of nondeterministic automata, and the other data is reported as average. The next seven columns show information about the input automata. The run time is given in milliseconds. The last two columns show the number of variables and clauses passed to the PMax-Sat solver.

$|Q|$ (interval)	#	#nd	$|Q|$	$|\Sigma_i|$	$|\Sigma_c|$	$|\Sigma_r|$	$|\Delta_i|$	$|\Delta_c|$	$|\Delta_r|$	Time	$|Var|$	$|Cls|$
[22; 250]	102	46	149	29	4	4	131	13	75	130	1440	35375
[250; 1000]	158	64	554	83	11	11	533	43	105	607	8363	53016
[1000; 4000]	161	27	2053	413	34	34	2188	170	345	2536	36865	170256
[4000; 16000]	127	6	8530	1535	152	150	9293	625	889	31481	161214	244007
[16000; 34114]	48	5	21755	2133	203	202	25348	603	1137	32129	361866	813549

automata from this set contain call and return transitions and do not contain any dead ends (states from which no accepting state is reachable). Details on the construction of these automata can be found in the extended version [26]

We ran our implementation on these automata using a timeout of 300 s and a memory limit of 4 GiB. Within the resource bounds we were able to minimize 596 of the automata. Details about these automata and the minimization run are presented in Table 2. In the table we grouped automata according to their size. For instance, the first row aggregates the data of all automata that have up to 250 states. The table shows that we were able to minimize automata up to a five-digit number of states and that automata that have a few thousand states can be minimized within seconds. Figure 3 shows the sizes of the minimization results. The first four graphs compare the sizes of input and output in terms of states and transitions. The fourth graph shows that the (partly) significant size reduction is not only due to "intraprocedural" merges, but that also the number of return transitions is reduced. The last two graphs show that the relative size
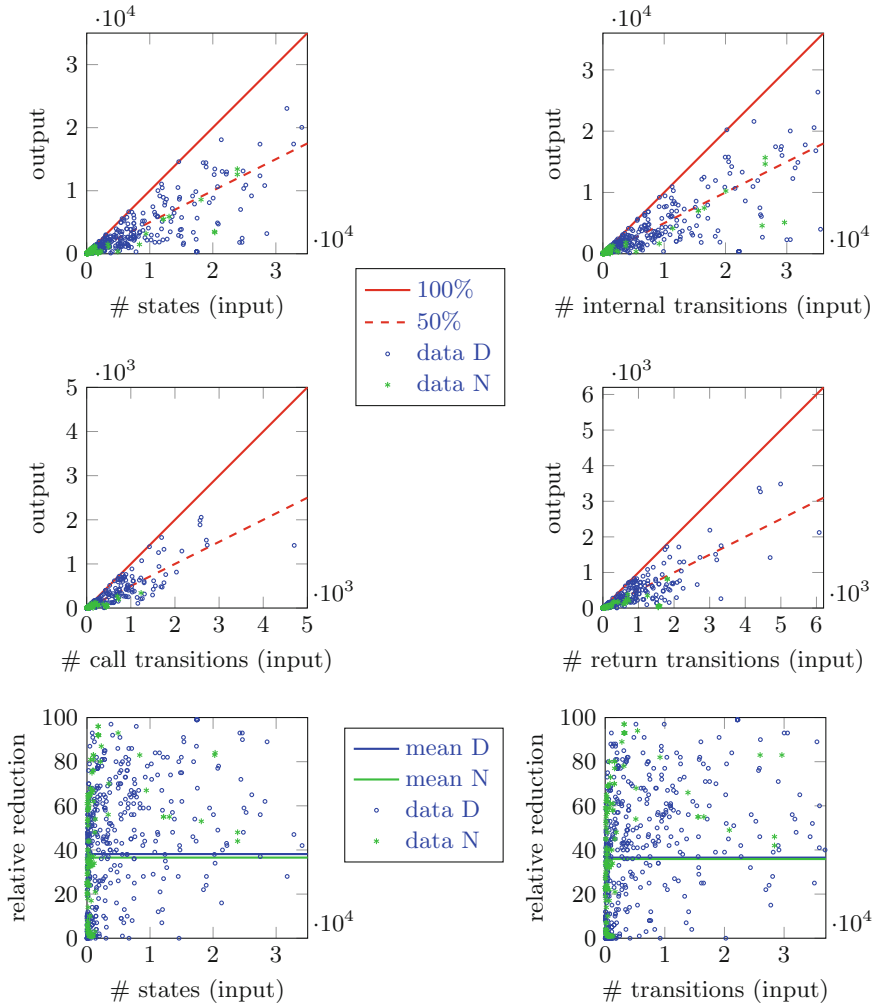
Fig. 3. Minimization results on automata produced by Ultimate Automizer (see also Table 2). D(N) stands for (non-)deterministic automata.

reduction is higher on larger automata. The reason is that small automata in Ultimate Automizer tend to have similarities to the control flow graph of a program, which is usually already minimal.

Evaluation on Random Automata. The automata produced by Ultimate Automizer have relatively large alphabets (according to Table 2 there are on average less than 10 states per symbol) and are extremely sparse (on average less than 1.5 transitions per state). To investigate the applicability of our approach to Vpa without such structure, we also evaluate it on random
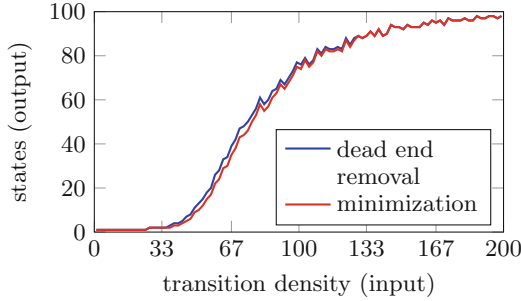
Fig. 4. Minimization results on random VPA with 100 states, of which 50% are accepting, and with one internal, call, and return symbol each. Return transitions are each inserted with 50 random stack symbols. The transition density is increased in steps of 2%. Each data point stems from 500 random automata.

nondeterministic VPA. We use a generalization of the random Büchi automata model by Tabakov and Vardi [36] to VPA (see the extended version [26] for details). Figure 4 shows that our algorithm can remove some states on top of removing dead ends for lower transition densities, but overall it seems more appropriate to automata that have some structure.

6 Related Work

Alur *et al.* [5] show that a canonical minimal VPA does not exist in general. They propose the *single entry*-VPA (SEVPA) model, a special VPA of equivalent expressiveness with the following constraints: Each state and call symbol is assigned to one of k modules, and each module has a unique entry state which is the target of all respective call transitions. This is enough structure to obtain the unique minimal k-SEVPA from any given k-SEVPA by quotienting.

Kumar *et al.* [30] extend the idea to *modular* VPA. Here the requirement of having a unique entry per module is overcome, but more structure must be fixed to preserve a unique minimum – most notably the restriction to weakly-hierarchical VPA and the return alphabet being a singleton.

Chervet and Walukiewicz [13] generalize the above classes to *call driven automata*. They show that general VPA can be exponentially more succinct than the three classes presented. Additionally, they propose another class called *block VPA* for which a unique minimum exists that is at most quadratic in the size of some minimal (general) VPA. However, to find it, the "right" partition into modules must be chosen, for which no efficient algorithm is known.

All above VPA classes have in common that the languages recognized are subsets of $WM(\Sigma)$, the states are partitioned into modules, and the minimal automaton (respecting the partition) can be found by quotienting. While the latter is an enjoyable property from the algorithmic view, the constraints limit practical applicability: Even under the assumption that the input VPA recognizes a well-matched language, if it does not meet the constraints, it must first be

converted to the respective form. This conversion generally introduces an exponential blow-up in the number of states. In contrast, our procedure assumes only weakly-hierarchical VPA accepting matched-return words. In general, a weakly-hierarchical VPA can be obtained with only a linear blow-up. (In ULTIMATE AUTOMIZER the automata already have this form.)

Consider the k-SEVPA in Fig. 5. It has k modules $\{q_1\}, \ldots, \{q_k\}$ (and the default module $\{q_0, q_f\}$). This is the minimal k-SEVPA recognizing the language with the given modules. Our algorithm will (always) merge all singleton modules into one state, resulting in a (minimal) three-state VPA.
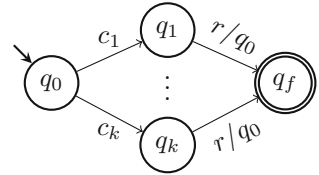
Fig. 5. A parametric k-SEVPA.

Caralp *et al.* [11] present a polynomial trimming procedure for VPA. The task is to ensure that every configuration exhibited in the VPA is both reachable and co-reachable. Such a procedure may add new states. We follow the opposite direction and exploit untrimmed configurations to reduce the number of states.

Ehlers [17] provides a SAT encoding of the question "does there exist an equivalent Büchi automaton (BA) of size $n-1$". Baarir and Duret-Lutz [8,9] extend the idea to so-called *transition-based generalized* BA. Since the search is global, on the one hand, such a query can be used iteratively to obtain a reduced BA after each step and some globally minimal BA upon termination; on the other hand, global search leaves little structure to the solver.

Geldenhuys *et al.* [20] also use a SAT encoding to reduce the state-space of nondeterministic FA. The first step is to construct the minimal deterministic FA $\mathcal{B}$. Then the solver symbolically guesses a candidate FA of a fixed size and checks that the automaton resulting from the subset construction applied to the candidate is isomorphic to $\mathcal{B}$. If the formula is unsatisfiable, the candidate size must be increased. Determinization may incur an exponential blow-up, and the resulting automaton is not always (but often) minimal.

In contrast to the above works, our PMax-SAT encoding consists of constraints about a quotienting relation (which always exists) that is polynomial in the size of the VPA. We do not find a minimal VPA, but our technique can be applied to VPA of practical relevance (the authors report results for automata with less than 20 states), in particular using our greedy algorithm.

Restricted to FA, the definition of a RAQ relation coincides with direct bisimulation [16,18]. This has two consequences. First, for FA, we can omit the transitivity clauses because a direct bisimulation is always transitive. Second, our algorithm always produces the (unique) maximal direct bisimulation. This can be seen as follows. If two states p and q bisimulate each other, then $\mathsf{X}_{\{p,q\}}$ can be assigned true: since we are looking for a maximal assignment, we will assign this value. If p and q do not bisimulate each other, then in any satisfying assignment $\mathsf{X}_{\{p,q\}}$ must be false. Alternatively, one can also say that our algorithm searches for *some* maximal fixpoint, which is unique for direct bisimulation.

For FA, it is well-known that minimization based on direct simulation yields smaller automata compared to direct bisimulation (i.e., the induced equivalence relation is coarser) [18]. Two states can be merged if they simulate each other. Our PMax-SAT encoding can be generalized to direct simulation by making the variables non-symmetric, i.e., using both $X_{p,q}$ and $X_{q,p}$ and adapting the clauses in a straightforward way. This increases the complexity by a polynomial.

References

1. Abdulla, P.A., Chen, Y., Holík, L., Vojnar, T.: Mediating for reduction (on minimizing alternating Büchi automata). In: FSTTCS, LIPIcs, vol. 4, pp. 1–12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2009)
2. Abel, A., Reineke, J.: MeMin: SAT-based exact minimization of incompletely specified mealy machines. In: ICCAD, pp. 94–101. IEEE (2015)
3. Almeida, R., Holík, L., Mayr, R.: Reduction of nondeterministic tree automata. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 717–735. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49674-9_46
4. Alur, R., Bouajjani, A., Esparza, J.: Model checking procedural programs. In: Handbook of Model Checking. Springer, Heidelberg (2017, to appear)
5. Alur, R., Kumar, V., Madhusudan, P., Viswanathan, M.: Congruences for visibly pushdown languages. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1102–1114. Springer, Heidelberg (2005). doi:10.1007/11523468_89
6. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC, pp. 202–211. ACM (2004)
7. Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM 56(3) (2009). Article No. 16
8. Baarir, S., Duret-Lutz, A.: Mechanizing the minimization of deterministic generalized Büchi automata. In: Ábrahám, E., Palamidessi, C. (eds.) FORTE 2014. LNCS, vol. 8461, pp. 266–283. Springer, Heidelberg (2014). doi:10.1007/978-3-662-43613-4_17
9. Baarir, S., Duret-Lutz, A.: SAT-based minimization of deterministic ω-automata. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (eds.) LPAR 2015. LNCS, vol. 9450, pp. 79–87. Springer, Heidelberg (2015). doi:10.1007/978-3-662-48899-7_6
10. Beyer, D.: Reliable and reproducible competition results with BenchExec and witnesses (report on SV-COMP 2016). In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 887–904. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49674-9_55
11. Caralp, M., Reynier, P.-A., Talbot, J.-M.: Trimming visibly pushdown automata. In: Konstantinidis, S. (ed.) CIAA 2013. LNCS, vol. 7982, pp. 84–96. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39274-0_9
12. Cha, B., Iwama, K., Kambayashi, Y., Miyazaki, S.: Local search algorithms for partial MAXSAT. In: AAAI/IAAI, pp. 263–268. AAAI Press/The MIT Press (1997)
13. Chervet, P., Walukiewicz, I.: Minimizing variants of visibly pushdown automata. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 135–146. Springer, Heidelberg (2007). doi:10.1007/978-3-540-74456-6_14
14. Clemente, L.: Büchi automata can have smaller quotients. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6756, pp. 258–270. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22012-8_20

15. D'Antoni, L., Veanes, M.: Minimization of symbolic automata. In: POPL, pp. 541–554. ACM (2014)

16. Dill, D.L., Hu, A.J., Wong-Toi, H.: Checking for language inclusion using simulation preorders. In: Larsen, K.G., Skou, A. (eds.) CAV 1991. LNCS, vol. 575, pp. 255–265. Springer, Heidelberg (1992). doi:10.1007/3-540-55179-4_25

17. Ehlers, R.: Minimising deterministic Büchi automata precisely using SAT solving. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 326–332. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14186-7_28

18. Etessami, K., Wilke, T., Schuller, R.A.: Fair simulation relations, parity games, and state space reduction for Büchi automata. SIAM J. Comput. **34**(5), 1159–1175 (2005)

19. Fu, Z., Malik, S.: On solving the partial MAX-SAT problem. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 252–265. Springer, Heidelberg (2006). doi:10.1007/11814948_25

20. Geldenhuys, J., Merwe, B., Zijl, L.: Reducing nondeterministic finite automata with SAT solvers. In: Yli-Jyrä, A., Kornai, A., Sakarovitch, J., Watson, B. (eds.) FSMNLP 2009. LNCS (LNAI), vol. 6062, pp. 81–92. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14684-8_9

21. Habermehl, P., Holík, L., Rogalewicz, A., Simácek, J., Vojnar, T.: Forest automata for verification of heap manipulation. Formal Methods Syst. Des. **41**(1), 83–106 (2012)

22. Harris, W.R., Jha, S., Reps, T.: Secure programming via visibly pushdown safety games. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 581–598. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31424-7_41

23. Heizmann, M., Dietsch, D., Greitschus, M., Leike, J., Musa, B., Schätzle, C., Podelski, A.: Ultimate automizer with two-track proofs. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 950–953. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49674-9_68

24. Heizmann, M., Hoenicke, J., Podelski, A.: Nested interpolants. In: POPL, pp. 471–482. ACM (2010)

25. Heizmann, M., Hoenicke, J., Podelski, A.: Software model checking for people who love automata. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 36–52. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39799-8_2

26. Heizmann, M., Schilling, C., Tischner, D.: Minimization of visibly pushdown automata using partial Max-SAT, vol. abs/1701.05160 (2017)

27. Holzmann, G.J., Puri, A.: A minimized automaton representation of reachable states. STTT **2**(3), 270–278 (1999)

28. Hopcroft, J.E.: An n log n algorithm for minimizing states in a finite automaton. In: Theory of Machines and Computations, pp. 189–196. Academic Press (1971)

29. Klarlund, N., Møller, A., Schwartzbach, M.I.: MONA implementation secrets. Int. J. Found. Comput. Sci. **13**(4), 571–586 (2002)

30. Kumar, V., Madhusudan, P., Viswanathan, M.: Minimization, learning, and conformance testing of boolean programs. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 203–217. Springer, Heidelberg (2006). doi:10.1007/11817949_14

31. Kumar, V., Madhusudan, P., Viswanathan, M.: Visibly pushdown automata for streaming XML. In: WWW, pp. 1053–1062. ACM (2007)

32. Mayr, R., Clemente, L.: Advanced automata minimization. In: POPL, pp. 63–74. ACM (2013)

33. Mozafari, B., Zeng, K., Zaniolo, C.: High-performance complex event processing over XML streams. In: SIGMOD Conference, pp. 253–264. ACM (2012)

34. Pitcher, C.: Visibly pushdown expression effects for XML stream processing. Programming Language Technologies for XML **1060**, 1–14 (2005)
35. Srba, J.: Beyond language equivalence on visibly pushdown automata. Logical Methods Comput. Sci. **5**(1) (2009)
36. Tabakov, D., Vardi, M.Y.: Experimental evaluation of classical automata constructions. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 396–411. Springer, Heidelberg (2005). doi:10.1007/11591191_28
37. Thakur, A., Lim, J., Lal, A., Burton, A., Driscoll, E., Elder, M., Andersen, T., Reps, T.: Directed proof generation for machine code. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 288–305. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14295-6_27
38. Thomo, A., Venkatesh, S., Ye, Y.Y.: Visibly pushdown transducers for approximate validation of streaming XML. In: Hartmann, S., Kern-Isberner, G. (eds.) FoIKS 2008. LNCS, vol. 4932, pp. 219–238. Springer, Heidelberg (2008). doi:10.1007/978-3-540-77684-0_16