# Efficient Algorithms for Alternating Pushdown Systems with an Application to the Computation of Certificate Chains*

Dejvuth Suwimonteerabuth, Stefan Schwoon, and Javier Esparza

Institut für Formale Methoden der Informatik, Universität Stuttgart,
Universitätsstr. 38, 70569 Stuttgart, Germany
{suwimodh,schwoosn,esparza}@informatik.uni-stuttgart.de

**Abstract.** Motivated by recent applications of pushdown systems to computer security problems, we present an efficient algorithm for the reachability problem of alternating pushdown systems. Although the algorithm is exponential, a careful analysis reveals that the exponent is usually small in typical applications. We show that the algorithm can be used to compute winning regions in pushdown games. In a second contribution, we observe that the algorithm runs in polynomial time for a certain subproblem, and show that the computation of certificate chains with threshold certificates in the SPKI/SDSI authorization framework can be reduced to this subproblem. We present a detailed complexity analysis of the algorithm and its application, and report on experimental results obtained with a prototype implementation.

## 1 Introduction

Pushdown systems are a concept from formal-language theory that has turned out to be useful in computer-aided verification. They naturally model the behaviour of programs with possibly recursive procedures, and therefore model-checking for pushdown systems has been the subject of recent research. Burkhard and Steffen [1] and Walukiewicz [2] have studied the problem for the modal $\mu$-calculus. Other papers [3,4,5] have investigated specialised algorithms for LTL model checking and both forward and backward reachability on pushdown systems. Concrete algorithms for these tasks with a precise complexity analysis were proposed in [5] and subsequently implemented in the Moped tool. Moreover, [3] has shown that a similar approach can be used to solve the *backward* reachability problem in alternating pushdown systems. This can be used to solve the model-checking problem for the alternation-free $\mu$-calculus on (non-alternating) pushdown systems.

More recently, pushdown systems have also been applied in the field of computer security. In the authorization framework SPKI/SDSI [6], certificates are used to assign permissions to groups of principals, which are defined using local,

* This work was partially supported by the DFG project *Algorithms for Software Model Checking* and SFB 627 Nexus, Project A6.

S. Graf and W. Zhang (Eds.): ATVA 2006, LNCS 4218, pp. 141–153, 2006.

hierarchical namespaces. In order to prove that a principal may access a certain resource, he/she needs to produce a chain of certificates that, taken together, provide a proof of authorisation. Jha and Reps [7] showed that a set of certificates can be seen as a pushdown system, and that certificate-chain discovery reduces to pushdown reachability. The SPKI/SDSI specification also provides for so-called *threshold certificates*, allowing specifications whereby a principal can be granted access to a resource if he/she can produce authorisations from multiple sources. We observe that this extension reduces to reachability on *alternating* pushdown systems.

Motivated by the applications in verification and authorisation, we study reachability algorithms for alternating pushdown systems (APDS) in more detail. The algorithm proposed in [3] is abstract (i.e. only the saturation rule is given), and its complexity is given as "exponential", without further details. Here, we provide a concrete algorithm for solving the problem together with a precise complexity analysis. Moreover, inspired by the work of [7], we show that the algorithm is very efficient for a special class of instances. Then, we consider two applications. The first one is straightforward: We show that the algorithm immediately leads to a procedure for computing winning regions in pushdown reachability games, and derive a complexity bound improving a previous analysis by [8]. The second application is perhaps more interesting. In [7], Jha and Reps observed that, for a restricted form of threshold certificates, the certificate-chain-discovery problem can be solved in polynomial, rather than exponential time. We prove this result again by showing that the existence of certificate chains can be reduced to the special class of instances of the reachability problem that we have identified. We perform a detailed complexity analysis, and report on a prototype implementation on top of the Nexus platform for context-aware systems [9].

We proceed as follows: Section 2 introduces alternating pushdown systems and other concepts used in the paper. Section 3 presents an algorithm for solving the reachability problem on APDS and analyzes its complexity. Section 4 studies the special class of instances mentioned above. Section 5 presents new upper bounds for computing winning regions in reachability pushdown games. Section 6 presents our application to certificate-chain discovery, and Section 7 reports experimental results.

Due to lack of space, all proofs have been omitted from this paper. A complete version that contains all the proofs has been published as a technical report [10].

## 2 Preliminaries

An *alternating pushdown system* (APDS) is a triplet $\mathcal{P} = (P, \Gamma, \Delta)$, where $P$ is a finite set of *control locations*, $\Gamma$ is a finite *stack alphabet*, and $\Delta \subseteq (P \times \Gamma) \times 2^{(P \times \Gamma^*)}$ is a set of *transition rules*. A *configuration* of $\mathcal{P}$ is a pair $\langle p, w \rangle$, where $p \in P$ is a control location and $w \in \Gamma^*$ is a *stack content*. If $((p, \gamma), \{(p_1, w_1), \ldots, (p_n, w_n)\}) \in \Delta$, we write $\langle p, \gamma \rangle \hookrightarrow \{\langle p_1, w_1 \rangle, \ldots, \langle p_n, w_n \rangle\}$ instead. We call a rule *alternating* if $n > 1$, or *non-alternating* otherwise. We also write $\langle p, \gamma \rangle \hookrightarrow \langle p_1, w_1 \rangle$ (braces omitted) for a non-alternating rule. Moreover,

for every $w \in \Gamma^*$, the configuration $\langle p, \gamma w\rangle$ is an *immediate predecessor* of the set $\{\langle p_1, w_1 w\rangle, \ldots, \langle p_n, w_n w\rangle\}$.

A *computation tree* of $\mathcal{P}$ is a directed tree whose nodes are labelled by configurations and where every node $n$ is either a leaf or an internal node labelled with $c$ such that $n$ has one outgoing hyperedge whose set of target nodes is labelled with configurations $C = \{c_1, \ldots, c_n\}$, where $c$ is an immediate predecessor of $C$. We define the *reachability relation* $\Rightarrow$ as $c \Rightarrow C$ if there exists a computation tree such that $c$ labels the root and $C$ is the set of labels of the leaves. If $c \Rightarrow C$, then $C$ is *reachable* from $c$. Given a set of configurations $C$, we define the set of *predecessors*, $pre^*(C) = \{c \mid \exists C' \subseteq C\colon c \Rightarrow C'\}$, as the set of configurations that are reachable backwards from subsets of $C$ via the reachability relation.

Let us fix an APDS $\mathcal{P} = (P, \Gamma, \Delta)$. An alternating $\mathcal{P}$-automaton is a quintuple $\mathcal{A} = (Q, \Gamma, \delta, P, F)$, where $Q \supseteq P$ is a finite set of *states*, $F \subseteq Q$ is the set of *final states*, and $\delta \subseteq Q \times \Gamma \times 2^Q$ is a set of *transitions*. The *initial states* of $\mathcal{A}$ are the control locations of $\mathcal{P}$. We define the *transition relation* $\rightarrow \subseteq Q \times \Gamma^* \times 2^Q$ as the smallest relation satisfying:

- $q \xrightarrow{\varepsilon} \{q\}$ for every $q \in Q$,
- if $(q, \gamma, Q') \in \delta$ then $q \xrightarrow{\gamma} Q'$, and
- if $q \xrightarrow{w} \{q_1, \ldots, q_m\}$ and $q_i \xrightarrow{\gamma} Q_i$ for each $1 \leq i \leq m$, then $q \xrightarrow{w\gamma} (Q_1 \cup \ldots \cup Q_m)$.

$\mathcal{A}$ *accepts* or *recognizes* a configuration $\langle p, w\rangle$ if $p \xrightarrow{w} Q'$ for some $Q' \subseteq F$. The set of configurations recognized by $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

In [3], it has been shown that given a set of configurations $C$ of $\mathcal{P}$, recognized by an alternating automaton $\mathcal{A}$, we can construct another automaton $\mathcal{A}_{pre^*}$ such that $L(\mathcal{A}_{pre^*}) = pre^*(C)$.

The procedure of [3] assumes w.l.o.g. that $\mathcal{A}$ has no transition leading to an initial state. $\mathcal{A}_{pre^*}$ is computed by means of a saturation procedure, which adds new transitions to $\mathcal{A}$, according to the following rule:

> If $\langle p, \gamma\rangle \hookrightarrow \{\langle p_1, w_1\rangle, \ldots, \langle p_m, w_m\rangle\} \in \Delta$ and $p_1 \xrightarrow{w_1} P_1, \ldots, p_m \xrightarrow{w_m} P_m$ holds, then add $p \xrightarrow{\gamma} (P_1 \cup \ldots \cup P_m)$.

## 3 An Implementation for *pre**

In this section we present an implementation, as shown in Fig. 1, of the abstract algorithm from Sect. 2. Without loss of generality, the algorithm imposes two restrictions on every rule $\langle p, \gamma\rangle \hookrightarrow R$ in $\Delta$:

> (R1) if $R = \{\langle p', w'\rangle\}$, then $|w'| \leq 2$, and
> (R2) if $|R| > 1$, then $|R| = 2$ and $\forall \langle p', w'\rangle \in R\colon |w'| = 1$.

Note that any APDS can be converted into an equivalent one that satisfies (R1) and (R2) with only a linear increase in size (i.e. the converted automaton executes the same sequences of actions, modulo the fact that one step may be refined into a sequence of steps).

In the rest of the paper we conduct a careful analysis in terms of certain parameters of the input, which are listed below:

- $\Delta_a, \Delta_0, \Delta_1, \Delta_2$ denote the sets of alternating rules and non-alternating rules with $0, 1, 2$ stack symbols in their right-hand side, respectively.
- The set of *pop control locations*, denoted by $P_\varepsilon$, is the set of control locations $p_1 \in P$ such that $\Delta_0$ contains some rule $\langle p, \gamma\rangle \hookrightarrow \langle p_1, \varepsilon\rangle$.
- Given an alternating automaton, we define $Q_{ni}$ as the set of its non-initial states, i.e., $Q_{ni} = Q \setminus P$.

Algorithm 1 computes $\mathcal{A}_{pre^*}$ by implementing the saturation rule. The sets *rel* and *trans* contain the transitions that are known to belong to $\mathcal{A}_{pre^*}$; *rel* contains those that have already been examined. Lines 1–4 initialize the algorithm. The rules $\langle p, \gamma\rangle \hookrightarrow \langle p_1, \epsilon\rangle$ are dealt with first, as in the *pre*$^*$ algorithm of the non-alternating case [5]. All rules are copied to $\Delta'$ (line 3), and the auxiliary function $\mathcal{F}(r)$ is assigned to set of empty set for each rule $r$ (line 4). The algorithm then proceeds by iteratively removing transitions from *trans* (line 6), adding them to *rel* if necessary (lines 7–8), and examining whether they generate other transitions via the saturation rule (lines 9–22). The idea of the algorithm is to avoid unnecessary operations. Imagine that the saturation rule allows to add transition $t$ if transitions $t_1$ and $t_2$ are already present. Now, if $t_1$ is taken from *trans* but $t_2$ has not been added to $\mathcal{A}_{pre^*}$, we do not put $t_1$ back to *trans* but store the following information instead: if $t_2$ is added, then we can also add $t$. It turns out that these implications can be stored in the form of "fake pushdown rules" (like those added in line 18 or 21) and in the form of the auxiliary sets $\mathcal{F}(r)$.

Let us now look at the lines 9–22 in more detail. Lines 9–10 are as in [5]. Push rules (lines 11–19) and alternating rules (lines 20–22), however, require a more delicate treatment. At line 11 we know that $q \xrightarrow{\gamma} Q'$ is a transition of $\mathcal{A}_{pre^*}$ (because it has been popped from *trans*) and that $\langle p_1, \gamma_1\rangle \hookrightarrow \langle q, \gamma\gamma_2\rangle$ is a rule of the APDS. So we divide the states $q' \in Q'$ into those for which there is some rule $q' \xrightarrow{\gamma_2} Q''$ in *rel* and the rest. If there is no rest then we can add new rules to *trans* (lines 14–15). Otherwise we add the "fake rule" of line 18. At line 20 we know that $q \xrightarrow{\gamma} Q'$ is a transition of $\mathcal{A}_{pre^*}$ and $\langle p_1, \gamma_1\rangle \hookrightarrow \{\langle q, \gamma\rangle\} \cup R$ is an alternating rule. So we add the "fake rule" $\langle p_1, \gamma_1\rangle \hookrightarrow R$.

Note that the algorithm obviously runs with exponential time, since the number of transitions of $A_{pre^*}$ can be exponential in the number of states. However, a closer look at the complexity reveals that the algorithm is exponential only in a proper subset of states, which can be small depending on the instance.

**Lemma 1.** *Algorithm 1 takes* $O(|\delta_0| + |\Delta_0| + |\Delta_1|2^n + (|\Delta_2|n + |\Delta_a|)4^n)$ *time, where* $n = |P_\varepsilon| + |Q_{ni}|$.

In typical applications, we start with a small automaton, i.e. $\delta_0$ and $Q_{ni}$ will be small. In that case, $n$ will be dominated by $|P_\varepsilon|$, therefore the complexity can be simplified to $O(|\Delta_0| + |\Delta_1|2^{|P_\varepsilon|} + (|\Delta_2||P_\varepsilon| + |\Delta_a|)4^{|P_\varepsilon|})$

**Theorem 1.** *Let* $\mathcal{P} = (P, \Gamma, \Delta)$ *be an alternating pushdown system and* $\mathcal{A} = (Q, \Gamma, \delta_0, P, F)$ *be an alternating automaton. There exist an alternating automaton* $\mathcal{A}_{pre^*}$ *that recognizes pre*$^*(L(\mathcal{A}))$. *Moreover, if the restrictions R1 and R2 are met,* $\mathcal{A}_{pre^*}$ *can be constructed in* $O(|\delta_0| + |\Delta_0| + |\Delta_1|2^n + (|\Delta_2|n + |\Delta_a|)4^n)$ *time, where* $n = |P_\varepsilon| + |Q_{ni}|$.

**Algorithm 1**
**Input:** an APDS $\mathcal{P} = (P, \Gamma, \Delta)$;
an alternating $\mathcal{P}$-automaton $\mathcal{A} = (Q, \Gamma, \delta_0, P, F)$ without transitions into $P$
**Output:** the set of transitions of $\mathcal{A}_{pre^*}$

```
1  rel := ∅;
2  trans := δ0 ∪ { (p,γ,p') | ⟨p,γ⟩ ↪ ⟨p',ε⟩ ∈ Δ } ∪ { (p,γ,∅) | ⟨p,γ⟩ ↪ ∅ ∈ Δ };
3  Δ' := Δ;
4  F := λx.{∅};
5  while trans ≠ ∅ do
6    pop t := (q,γ,Q') from trans;
7    if t ∉ rel then
8      add t to rel;
9      for all r := ⟨p1,γ1⟩ ↪ ⟨q,γ⟩ ∈ Δ' and Q'' ∈ F(r) do
10       add (p1,γ1,Q' ∪ Q'') to trans;
11     for all ⟨p1,γ1⟩ ↪ ⟨q,γγ2⟩ ∈ Δ' do
12       S := { q' ∈ Q' | ∃Q'' : (q',γ2,Q'') ∈ rel };
13       Q1 := { ⋃_{q'∈S} Q_{q'} | ∀q' ∈ S: (q',γ2,Q_{q'}) ∈ rel };
14       if S = Q' then
15         add {(p1,γ1,Q1) | Q1 ∈ Q1} to trans;
16       else
17         r := ⟨p1,γ1⟩ ↪ {⟨q',γ2⟩ | q' ∈ Q' \ S};
18         add r to Δ';
19         add Q1 to F(r);
20     for all r := ⟨p1,γ1⟩ ↪ {⟨q,γ⟩} ∪ R ∈ Δ' s.t. R ≠ ∅ do
21       add ⟨p1,γ1⟩ ↪ R to Δ';
22       add {Q'' ∪ Q' | Q'' ∈ F(r)} to F(⟨p1,γ1⟩ ↪ R);
23 return rel;
```

**Fig. 1.** An algorithm for computing $pre^*$

Given an APDS $\mathcal{P}$, a configuration $c$ of $\mathcal{P}$, and a set of configurations $C$, the *backward reachability* problem for $\mathcal{P}$, $c$, and $C$ is to check whether $c \in pre^*_{\mathcal{P}}(C)$. By Theorem 1, the problem is in EXPTIME. The following theorem shows a corresponding lower bound. It is a rather straightforward modification of a theorem of [11].

**Theorem 2.** *The backward reachability problem for alternating pushdown systems is* ***EXPTIME****-complete, even if $C$ is a singleton.*

## 4 A Special Case

Recall the saturation rule of the abstract algorithm for the computation of $pre^*$: for every transition rule $\langle p, \gamma\rangle \hookrightarrow \{\langle p_1, w_1\rangle, \ldots, \langle p_m, w_m\rangle\}$ and every set $p_1 \xrightarrow{w_1} P_1, \ldots, p_m \xrightarrow{w_m} P_m$, add a new transition $p \xrightarrow{\gamma} (P_1 \cup \ldots \cup P_m)$. The exponential complexity of the algorithm is due to the fact that the target of the new transition can be an arbitrary set of states, and so we may have to add an exponential number of new rules in the worst case. We now consider a special

class of instances in which a new transition $p \xrightarrow{\gamma} Q$ need only be added if $Q$ is a singleton, and show that a suitable modification of Algorithm 1 has polynomial running time.

**Definition 1.** *Let $\mathcal{P} = (P, \Gamma, \Delta)$ be an APDS, and let $R \subseteq P\Gamma^*$ be a set of configurations. We say that $(\mathcal{P}, R)$ is a* good *instance for the computation of pre$^*$ if for every $\langle p, d\rangle \hookrightarrow \{\langle p_1, w_1\rangle, \ldots, \langle p_n, w_n\rangle\} \in \Delta$ with $n \geq 2$ and for every $i \in \{1, \ldots, n\}$: $p_i w_i w \in pre^*(R)$ implies $w = \varepsilon$.*

I.e., if the set $R$ can be reached from $p_i w_i$, then it cannot be reached from any $p_i w_i w$, where $w$ is a nonempty word. As mentioned above, we introduce the following modification to the saturation rule: a new transition $p \xrightarrow{\gamma} Q$ is added only if $Q$ is a singleton.

**Theorem 3.** *Let $\mathcal{P} = (P, \Gamma, \Delta)$ and $R$ be a good instance, and let $\mathcal{A}$ be a nondeterministic automaton recognizing $R$. Assume w.l.o.g. that $\mathcal{A}$ has one single final state. Then, the modified saturation procedure produces a nondeterministic automaton recognizing the same language as $\mathcal{A}_{pre^*}$.*

Algorithm 1 implements the modified procedure after the following change to line 9: **for all** $r := \langle p_1, \gamma_1\rangle \hookrightarrow \langle q, \gamma\rangle \in \Delta'$ **and** $Q'' \in \mathcal{F}(r) \cap \{\emptyset, Q'\}$ **do**.

**Lemma 2.** *The modified Algorithm 1 takes $O(|\delta_0| + |\Delta_0| + (|\Delta_1| + |\Delta_a|)n + |\Delta_2|n^2)$ time, where $n = |P_\varepsilon| + |Q_{ni}|$, when applied to a good instance.*

Note that Algorithm 1, when applied to a *non-alternating* PDS (i.e. one with $\Delta_a = \emptyset$), has the same complexity as the algorithm from [5] that was specially designed for non-alternating PDS.

## 5 Computing Attractors in Pushdown Games

In [8] Cachat provided an algorithm for computing the winning positions of a player in a pushdown reachability game. It is straightforward to reformulate the algorithm in terms of *pre*$^*$ computations for alternating pushdown automata. We do this, and apply the results of Sect. 3 to provide very precise upper bounds for the complexity of these problems.

A *pushdown game system* (PGS) is a tuple $\mathcal{G} = (P, \Gamma, \Delta_\mathcal{G}, P_0, P_1)$, where $(P, \Gamma, \Delta_\mathcal{G})$ is a PDS and $P_0, P_1$ is a partition of $P$. A PGS defines a pushdown game graph $G_\mathcal{G} = (V, \rightarrow)$ where $V = P\Gamma^*$ is the set of all configurations, and $p\gamma v \rightarrow qwv$ for every $v \in \Gamma^*$ iff $(p, \gamma, q, w) \in \Delta_\mathcal{G}$. $P_0$ and $P_1$ induce a partition $V_0 = P_0\Gamma^*$ and $V_1 = P_1\Gamma^*$ on $V$. Intuititively, $V_0$ and $V_1$ are the nodes at which players 0 and 1 choose a move, repectively. Given a start configuration $\pi_0 \in V$, a play is a maximal (possibly infinite) path $\pi_0\pi_1\pi_2\ldots$ of $G_\mathcal{G}$; the transitions of the path are called *moves*; a move $\pi_i \rightarrow \pi_{i+1}$ is made by player 0 if $\pi_i \in V_0$; otherwise it is made by player 1.

The winning condition of a reachability game is a regular *goal set* of configurations $R \subseteq P\Gamma^*$. Player 0 wins those plays that visit some configuration of the

goal set and also those that reach a deadlock for player 1. Player 1 wins the rest. We wish to compute the winning region for player 0, denoted by $Attr_0(R)$, i.e. the set of nodes from which player 0 can always force a visit to $R$ or a deadlock for player 1. Formally [8]:

$$\begin{aligned} Attr_0^0(R) &= R \ , \\ Attr_0^{i+1}(R) &= Attr_0^i(R) \cup \{u \in V_0 \mid \exists v : u \to v, v \in Attr_0^i(R)\} \\ &\quad \cup \{u \in V_1 \mid \forall v : u \to v \Rightarrow v \in Attr_0^i(R)\} \ , \\ Attr_0(R) &= \bigcup\nolimits_{i \in \mathbb{N}} Attr_0^i(R) \ . \end{aligned}$$

Given a PGS $\mathcal{G} = (P, \Gamma, \Delta_{\mathcal{G}}, P_0, P_1)$, we define an APDS $\mathcal{P} = (P, \Gamma, \Delta)$ as follows. For every $p \in P$ and $\gamma \in \Gamma$: if $p \in P_0$, then for every rule $\langle p, \gamma\rangle \hookrightarrow \langle q, w\rangle$ of $\Delta_{\mathcal{G}}$ add the rule $\langle p, \gamma\rangle \hookrightarrow \{\langle q, w\rangle\}$ to $\Delta$; if $p \in P_1$ and $S$ is the set of right-hand-side configurations of rules with $\langle p, \gamma\rangle$ as left-hand-side, then add $\langle p, \gamma\rangle \hookrightarrow S$ to $\Delta$. It follows immediately from the definitions that $Attr_0(R) = pre^*{}_{\mathcal{P}}(R)$ (intuitively, if $c \in pre^*{}_{\mathcal{P}}(R)$ then $c \Rightarrow C$ for some $C \subseteq R$, and so player 0 can force the play into the set $C$). So we can use Algorithm 1 to compute $Attr_0(R)$. To derive the complexity bound, we apply Lemma 1:

**Theorem 4.** *Let $\mathcal{G} = (P, \Gamma, \Delta_{\mathcal{G}}, P_0, P_1)$ be a PGS and a goal set $R$ recognized by an alternating automaton $\mathcal{A}_R = (Q, \Gamma, \delta_0, P, F)$. An alternating automaton accepting the winning region can be computed in $O(|\delta_0|+|\Delta_0|+|\Delta_1|2^n+(|\Delta_2|n+|\Delta_a|)4^n)$ time, where $n = |P_\varepsilon| + |Q_{ni}|$.*

In [8] an upper bound of $O(|\Delta| \cdot 2^{c \cdot |Q|^2})$ is given. Our algorithm runs in $O(|\Delta| \cdot 2^{c \cdot |Q|})$ time, and in fact Theorem 4 further reduces the exponent $c \cdot |Q|$ to $|P_\varepsilon| + |Q_{ni}|$. Typically, $|P_\varepsilon| + |Q_{ni}|$ is much smaller than $|Q|$. First, recall that, because of the definition of $\mathcal{P}$-automaton, we have $P \subseteq Q$. Moreover, goal sets often take the form $p_1\Gamma^* \cup \ldots \cup p_n\Gamma^*$, i.e., player 0 wins if the play hits one of the control states $p_1, \ldots, p_n$. In this case we can construct $\mathcal{A}_R$ with $|Q_{ni}| = 1$. Since $|P_\varepsilon|$ is typically much smaller than $|P|$, the parameter $n$ is much smaller than $|Q|$.

## 6 Computing Certificate Trees in SPKI/SDSI

In access control of shared resources, authorization systems allow to specify a security policy that assigns permissions to principals in the system. The *authorization problem* is, given a security policy, should a principal be allowed access to a specific resource? In frameworks such as SPKI/SDSI [6] and $RT_0$ [12], the security policy is expressed as a set of certificates, and the authorization problem reduces to discovering a subset of certificates proving that a given principal is allowed to access a given resource.

The SPKI/SDSI standard provides for so-called *threshold certificates*. Jha and Reps already observed in [7] that the authorization problem in the presence of such certificates can be reduced to the APDS reachability problem, and that a special case had polynomial complexity. In this paper, we observe that the special

case corresponds to *good instances* of APDS reachability, as defined in Sect. 4, and provide a detailed complexity analysis. Moreover, we report on experimental results for a prototype implementation of the algorithm as an extension of the Nexus platform [9] with distributed access control.

The expressiveness of $\mathrm{RT}_0$ is very similar to that of SPKI/SDSI and also allows for role intersection. We note, therefore, that the authorization problem for $\mathrm{RT}_0$ also reduces to APDS reachability. In [12], a specialised certificate-chain-discovery algorithm for $\mathrm{RT}_0$ was proposed to which our solution provides an alternative. A comparison between the two algorithms is a little involved, however, and can be found in [10].

We proceed in two steps. First, we consider "simple" SPKI/SDSI, a subset of SPKI/SDSI that has been considered in most of the work on this topic. Simple SPKI/SDSI does not handle threshold certificates, which we present in the second part.

### 6.1 Simple SPKI/SDSI

In this paper, we introduce only the basic notations that are required to understand SPKI/SDSI and its connections with alternating PDS. A more thorough explanation can be found in [7].

In SPKI/SDSI, the principals (individuals, resources, or any other entities) are represented by their public keys. We denote by $\mathcal{K}$ the set of public keys (or principals), specific keys are denoted by $K, K_A, K'$, etc. An *identifier* is a word over some alphabet $\Sigma$ (usually denoted by typewriter font such as $\mathtt{A}$, $\mathtt{B}$, ...). The set of identifiers is denoted by $\mathcal{A}$. A *local name* is of the form $K\ \mathtt{A}$, where $K \in \mathcal{K}$ and $\mathtt{A} \in \mathcal{A}$. For example, $K_X\ \mathtt{Customer}$ is a local name. A *term* is a key followed by zero or more identifiers. For example, $K\ \mathtt{Area}\ \mathtt{Customer}$ is a term. SPKI/SDSI has two types of certificates, or "certs":

*Name Certificates.* A name cert provides a definition of a local name in the issuer's local name space. Simply speaking, it can be understood as a rewrite rule of the form $K\ \mathtt{A} \to S$, where $K\ \mathtt{A}$ is a local name and and $S$ is a term. Intuitively, this defines a meaning for $\mathtt{A}$ in the local name space of principal $K$, and only $K$ may issue and sign such a cert.

Imagine, for instance, that $X$ is a telecommunication company with multiple divisions, including the mobile phone division $Xm$. Alice is a customer with the mobile phone division. Consider the following certificates:

$$K_{Xm}\ \mathtt{customer} \to K_{Alice} \tag{1}$$

$$K_X\ \mathtt{customer} \to K_{Xm}\ \mathtt{customer} \tag{2}$$

Here, (1) intuitively declares Alice to be a customer of $Xm$, while (2) says that customers of $Xm$ are also customers of the company $X$ as a whole.

*Authorization Certificates.* An auth cert grants or delegates a specific authorization from an issuer to a subject. It can be understood as a rewrite rule of the form

$K_R\ \square \rightarrow S\ b$, where $b \in \{\square, \blacksquare\}$. If $K_R$ is the owner of some resource $R$, then this certificate grants access to $R$ to all principals described by term $S$. Only $K_R$ may issue such a certificate. If $b = \square$, then authorized principals may delegate this authorization to other principals, otherwise delegation is not permitted. The following certificate grants access to resource $R$ to all of $X$'s customers, without delegation:

$$K_R\ \square \rightarrow K_X\ \texttt{customers}\ \blacksquare \tag{3}$$

*Certificate Chains.* In order for Alice to prove that she has access to some resource, she needs to provide a list of certificates that lead from the public key to herself by applying left-prefix rewriting. Such a list of certificates is called a *certificate chain*. In the example, Alice is granted authorisation to access $R$ if she can produce the certificate chain (3),(2),(1), because applying them (in this order) shows that:

$$K_R\ \square \overset{(3)}{\rightarrow} K_X\ \texttt{customers}\ \blacksquare \overset{(2)}{\rightarrow} K_{Xm}\ \texttt{customers}\ \blacksquare \overset{(1)}{\rightarrow} K_{Alice}\ \blacksquare$$

Since this chain leads from $K_R\ \square$ to $K_{Alice}\ \blacksquare$, Alice is authorised to access $R$, the "$\blacksquare$" indicating that she is unable to delegate that access further.

It was observed in [7] that a set of name and auth certs can be interpreted as a pushdown system; therefore, the authorization problem reduces to the problem of pushdown reachability and can be solved using the algorithms from [3,5].

### 6.2 SPKI/SDSI with Threshold Certificates

The SPKI/SDSI standard [6] provides for so-called *threshold subjects*. A threshold subject is a pair $(\mathcal{S}, k)$ where $\mathcal{S}$ is a set of terms and $k \leq |\mathcal{S}|$. A threshold certificate is a name or auth cert where the right-hand side is a threshold subject. If threshold certificates are involved, proofs of authorisation can no longer be done purely by certificate chains. Instead, a proof of authorisation for Alice to access resource $R$ becomes a *certificate tree*, where the nodes are labelled with terms and the edges are labelled with rewrite rules that can be applied to the term labelling their source nodes. The root is $K_R\ \square$, and if $K\ \texttt{A} \rightarrow (\mathcal{S}, k)$ is used to rewrite a node $n$, then the children of $n$ are the elements of $\mathcal{S}$. The tree is considered a valid proof of authorisation for Alice if at least $k$ of the children can be rewritten to $K_{Alice}\ b$, where $b \in \{\square, \blacksquare\}$.

We observe that it is sufficient to consider threshold certificates with subject $(\mathcal{S}, k)$ such that $k = |\mathcal{S}|$. (Any certificate where $k < |\mathcal{S}|$ can be simulated by $\binom{|\mathcal{S}|}{k}$ threshold certificates for each subset of $\mathcal{S}$ with exactly $k$ elements.) Therefore, we will omit the number $k$ from now on, silently assuming that it is equal to the cardinality of $\mathcal{S}$.

It can now easily be seen that in the presence of threshold certificates, the certificate set can be interpreted as an *alternating* pushdown system, and that the authorisation problem reduces to APDS reachability. In other words, Alice is granted access to resource $R$ if she can prove that $K_R\ \square \Rightarrow \{K_{Alice}\ \square, K_{Alice}\ \blacksquare\}$.

In [13,7] the use of threshold subjects is restricted to just authorization certificates, claiming that the use of threshold subjects in name certificates would make the semantics "almost surely too convoluted". Moreover, [7] observes that under this restriction the authorisation problem can be solved without incurring (asymptotic) run-time penalties for threshold subjects and gives an informal algorithm. Within our framework, we note that the restriction of threshold subjects to auth certs allows one to obtain a good instance and to apply the algorithm from Sect. 4 to solve the authorisation problem.

**Theorem 5.** *Let $C_t$, $C_0$, $C_1$, and $C_2$ be sets of certificates, where $C_t$ contains the auth certs with threshold subjects, $C_0$ contains the name certs in which terms have zero identifiers, $C_1$ contains the name and auth certs in which terms have one and zero identifiers, respectively, and $C_2$ consists of the rest. Let $n$ be the number of different terms in $C_0$. The authorization problem can be solved in $O(|C_0| + (|C_1| + |C_t|)n + |C_2|n^2)$ time.*

## 7 Implementation and Experiments

We have implemented a prototype of the $pre^*$ algorithm for APDS (in fact, a dedicated version for good instances) inside the Nexus platform [9]. An application can use Nexus "middleware" in order to obtain context data about mobile objects registered at the platform, like the position of an object or whether it enjoys a given relation to another object.

Nexus is based on an *Augmented World Model* (AWM). AWM can contain both real world objects (e.g. rooms or streets) and virtual objects (e.g. websites). Furthermore, Nexus defines a language called *Augmented World Modeling Language* (AWML). This XML-based language is used for exchanging Nexus objects between the platform and data repositories.

Our prototype extends the AWM and AWML with name and authorization relations, which can be viewed as name and authorization certificates in the case of SPKI/SDSI, respectively. In other words, we model relations as virtual objects in the Nexus context. Moreover, we extend the platform so that it can serve applications querying relations between entities. Note that, normally, the base information about objects is contained in a Nexus database (the so-called context server) and returned in the form of AWML documents. Our prototype is not yet connected to such a database; instead, all data is kept directly in AWML.

### 7.1 A Scenario

Consider a scenario where company $X$ takes part in a trade fair. The exhibition center consists of 2 exhibitions. An exhibition's area is a hierarchical structure with 3 exhibition halls, divided into 4 floors with 5 booths each. The structure can be written by pushdown rules as follows, given that $1 \leq i \leq 2, 1 \leq j \leq 3, 1 \leq k \leq 4, 1 \leq l \leq 5$:

$$E_i \texttt{ Area} \rightarrow E_i \texttt{ Hall Floor Booth} \tag{4}$$
$$E_i \texttt{ Hall} \rightarrow H_{[i,j]} \tag{5}$$
$$H_{[i,j]} \texttt{ Floor} \rightarrow F_{[i,j,k]} \tag{6}$$
$$F_{[i,j,k]} \texttt{ Booth} \rightarrow B_{[i,j,k,l]} \tag{7}$$

Now, company $X$ launches a promotion for visitors of the exhibition center to freely download ringtones for their mobile phones. The following visitors are allowed to download: (1) customers of $X$ who are currently in the area of exhibition 1; (2) non-customers to whom the right has been delegated by one of $X$'s customers; (3) customers who are currently not in the area of exhibition 1, but have received delegation from another visitor of exhibition 1. This is expressed by the following rule:

$$K_X \ \Box \rightarrow \{E_1 \texttt{ Area Visitor } \Box, K_X \texttt{ Customer } \Box\} \tag{8}$$

The facts that Alice is visiting a booth in exhibition 1, and that she delegates her right to Bob, who is a customer of $X$, can be written as:

$$B_{[1,j,k,l]} \texttt{ Visitor} \rightarrow K_{Alice}, \qquad \text{for some } j,k,l \tag{9}$$
$$K_{Alice} \ \Box \rightarrow K_{Bob} \ \blacksquare \tag{10}$$
$$K_X \texttt{ customer} \rightarrow K_{Bob} \tag{11}$$

When Bob wants to download a ringtone, we can efficiently compute the set $pre^*(\{\langle K_{Bob}, \Box\rangle, \langle K_{Bob}, \blacksquare\rangle\})$ by noting the fact that the rules (4)–(11) and $\{\langle K_{Bob}, \Box\rangle, \langle K_{Bob}, \blacksquare\rangle\}$ form a good instance. Bob's request is granted in this case because $\langle X, \Box\rangle \in pre^*(\{\langle K_{Bob}, \Box\rangle, \langle K_{Bob}, \blacksquare\rangle\})$. Note that Bob can only download as long as Alice stays in booths in the exhibition 1. As soon as she moves away (i.e. the rule (9) is removed), a request from Bob can no longer be granted even though he is a customer of $X$.

### 7.2 Experiments

The scenario explained above is implemented as an application of the Nexus platform. We report on the running time for some experiments. The experiments should give a rough idea of the size of problems that can be handled in reasonable time.

We randomly add visitors to the exhibition center, and let them randomly issue certificates. We consider a base case with 1000 visitors in the exhibition center, 100 of them are customers of the company $X$, and the visitors issue 1000 authorization certificates. The issuer of a certificate decides randomly whether the right can be further delegated or not. The series were conducted on a 2 GHz PC with 256 MB RAM.

### 7.3 Experiment 1

In the base case, 10 % of visitors are customers of $X$, and a visitor issues one certificate on average. In our first experiment we keep these two ratios constant, and increase the number of visitors (for example, if there are 2000 visitors, there will

be 200 customers that authorize 2000 times). We ran the experiment five times for each set of parameters. In each run 1000 random download requests are made. Table 1 displays the average results for 1000, 2000, 5000, and 10000 visitors (V). The table shows how often the request was granted (G) and rejected (R), the average time of a certificate search (T), and average time for granted (T(G)) and rejected (T(R)) searches. All measurements are in milliseconds.

In a realistic scenario, solving the authorisation problem requires to query databases (e.g. databases containing the positions of objects) and transmit data over a network, which are comparatively expensive operations. We kept relations of various types in different AWML files and whenever a piece of data was needed, we retrieved it from there. Since opening and reading files is also a comparatively expensive operation, this gives some insight as to the overhead such operations would incur in practice. The table shows the number of times AWML files (F) needed to be opened in average. For comparison, the numbers for granted (F(G)) and rejected (F(R)) requests are also displayed.

**Table 1.** Results of Experiment 1

| V | G | R | T | T(G) | T(R) | F | F(G) | F(R) |
|---|---|---|---|---|---|---|---|---|
| 1000 | 229.8 | 770.2 | 18.71 | 29.09 | 15.49 | 13.84 | 22.54 | 11.19 |
| 2000 | 195.6 | 804.4 | 19.23 | 28.76 | 16.92 | 13.14 | 21.25 | 11.16 |
| 5000 | 202.2 | 797.8 | 18.62 | 29.33 | 15.90 | 12.99 | 21.10 | 10.93 |
| 10000 | 199.4 | 800.6 | 24.90 | 38.25 | 21.60 | 13.00 | 22.00 | 10.77 |

This experiment allows to draw a first conclusion: The average time of a search does not depend on the number of visitors per se. When a visitor requests a download, the algorithm has to search for the issuers of its certificates. Since the number of certificates is equal to the number of visitors, each visitor has one certificate in average.

### 7.4 Experiment 2

In this experiment, we kept the number of visitors constant, and increased the number of certificates they issue, shown in column C in Table 2. The other columns are as in Experiment 1. Again, we ran the experiment five times for each value of C. Each run consisted of 100 random requests.

**Table 2.** Results of Experiment 2

| C | G | R | T | T(G) | T(R) | F | F(G) | F(R) |
|---|---|---|---|---|---|---|---|---|
| 1000 | 23.0 | 77.0 | 18.71 | 29.09 | 15.49 | 13.84 | 22.54 | 11.19 |
| 2000 | 56.2 | 43.8 | 120.72 | 193.93 | 21.96 | 74.68 | 118.50 | 15.83 |
| 3000 | 86.4 | 13.6 | 1477.35 | 1704.21 | 33.66 | 625.41 | 721.69 | 12.91 |
| 4000 | 95.2 | 4.8 | 2279.13 | 2393.81 | 13.40 | 898.01 | 942.94 | 9.64 |

We see that the running time grows rapidly with the number of certificates issued. The explanation is the larger number of certificates received by each visitor, which leads to many more certificate chains. Observe also that the number of granted requests increases.

The overall conclusion of the two experiments is that the algorithm scales well to realistic numbers of visitors and certificates. Notice that in the intended application a user will be willing to wait for a few seconds.

## 8 Conclusions

We have provided an efficient implementation of the saturation algorithm of [3] for the computation of $pre^*$ in alternating pushdown systems. Following [8], we have applied the algorithm to the problem of determining the winning region in reachability pushdown games, improving the complexity bound of [8]. We have shown that the algorithm has very low complexity for certain good instances, and provided an application: The computation of certificate chains with threshold subjects in the SPKI/SDSI authorization framework can be reduced to these instances. We have implemented the algorithm within the Nexus platform [9], and shown that it scales up to realistic scenarios.

## References

1. Burkart, O., Steffen, B.: Model checking the full modal mu-calculus for infinite sequential processes. In: Proc. ICALP. LNCS 1256, Springer (1997) 419–429
2. Walukiewicz, I.: Pushdown processes: Games and model checking. In: Proc. CAV. LNCS 1102 (1996) 62–74
3. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Application to model-checking. In: Proc. CONCUR. LNCS 1243 (1997) 135–150
4. Finkel, A., Willems, B., Wolper, P.: A direct symbolic approach to model checking pushdown systems. ENTCS **9** (1997)
5. Esparza, J., Hansel, D., Rossmanith, P., Schwoon, S.: Efficient algorithms for model checking pushdown systems. In: Proc. CAV. LNCS 1855 (2000) 232–247
6. Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylönen, T.: RFC 2693: SPKI Certificate Theory. The Internet Society. (1999)
7. Jha, S., Reps, T.: Model checking SPKI/SDSI. JCS **12**(3–4) (2004) 317–353
8. Cachat, T.: Symbolic strategy synthesis for games on pushdown graphs. In: Proc. ICALP. LNCS 2380 (2002) 704–715
9. Hohl, F., Kubach, U., Leonhardi, A., Rothermel, K., Schwehm, M.: Nexus - an open global infrastructure for spatial-aware applications. Technical Report 1999/02, Universität Stuttgart: SFB 627 (1999)
10. Suwimonteerabuth, D., Schwoon, S., Esparza, J.: Efficient algorithms for alternating pushdown systems: Application to certificate chain discovery with threshold subjects. Technical report, Universität Stuttgart (2006)
11. Chandra, A., Kozen, D., Stockmeyer, L.: Alternation. JACM **28**(1) (1981) 114–133
12. Li, N., Winsborough, W., Mitchell, J.: Distributed credential chain discovery in trust management. In: Proc. CCS, ACM Press (2001) 156–165
13. Clarke, D., Elien, J., Ellison, C., Fredette, M., Morcos, A., Rivest, R.: Certificate chain discovery in SPKI/SDSI. At `http://theory.lcs.mit.edu/~rivest/` (1999)