PAPER

LTL Model Checking for Register Pushdown Systems

Ryoma SENDA[†a)], *Nonmember*, Yoshiaki TAKATA[††b)], *Member*, *and* Hiroyuki SEKI[†c)], *Fellow*

SUMMARY A pushdown system (PDS) is known as an abstract model of recursive programs. For PDS, model checking methods have been studied and applied to various software verification such as interprocedural data flow analysis and malware detection. However, PDS cannot manipulate data values from an infinite domain. A register PDS (RPDS) is an extension of PDS by adding registers to deal with data values in a restricted way. This paper proposes algorithms for LTL model checking problems for RPDS with simple and regular valuations, which are labelings of atomic propositions to configurations with reasonable restriction. First, we introduce RPDS and related models, and then define the LTL model checking problems for RPDS. Second, we give algorithms for solving these problems and also show that the problems are EXPTIME-complete. As practical examples, we show solutions of a malware detection and an XML schema checking in the proposed framework.

key words: *register pushdown system, LTL model checking, regularity preservation, register automaton*

1. Introduction

A pushdown system (PDS) is a pushdown automaton without input, which is useful for an abstraction of recursive programs [6], [28]. A configuration of a PDS is a pair (p, w) where p and w are a finite control and stack contents, respectively. The model checking problem for a PDS $\mathcal{P}$ is to check whether all runs of $\mathcal{P}$ reachable from a given configuration satisfy a specification φ or not. The model checking problem for PDS has been studied for various logics such as linear temporal logic (LTL) and branching-time temporal logics. In [6], [15], a simple valuation that labels each configuration (p, w) with atomic propositions depending only on p and the stack top symbol of w is assumed and the LTL model checking problem for PDS is shown to be decidable. In [16], the problem is shown to be still decidable for more general valuations called regular valuations. A regular valuation can label a configuration (p, w) with atomic propositions depending on whether the whole stack contents w match a given regular pattern as well as the finite control p. For both kinds of valuations, the problem was shown to be EXPTIME-complete.

Manuscript received December 24, 2020.
Manuscript revised June 23, 2021.
Manuscript publicized August 31, 2021.

[†]The authors are with Graduate School of Informatics, Nagoya University, Nagoya-shi, 464–8601 Japan.
[††]The author is with Graduate School of Engineering, Kochi University of Technology, Kami-shi, 782–8502 Japan.
a) E-mail: ryoma.private@sqlab.jp
b) E-mail: takata.yoshiaki@kochi-tech.ac.jp
c) E-mail: seki@i.nagoya-u.ac.jp

DOI: 10.1587/transinf.2020EDP7265

Although PDS is useful for modeling recursive programs, it cannot directly deal with data values in the programs. When we apply an existing PDS model checking method to those programs, we have to ignore data values in the programs or statically expand each variable in the program to abstract data values. In order to handle data values more accurately, extended models for PDS have been expected.

Register automata (RA) were introduced as an extension of finite automata (FA) by adding the capability of dealing with data values in a restricted way [18]. Recently, RA has attracted attention as a formal model of navigational queries asking both patterns and data values to structured data such as XML documents [19]. Pushdown systems were also extended to pushdown register systems (PDRS) and register pushdown systems (RPDS), both of them are equivalent each other, and the reachability has been shown to be EXPTIME-complete [20]. However, mathematically plausible properties of RPDS including the decidability of reachability have not been applied to model checking recursive programs against general classes of temporal properties such as LTL.

In this paper, we introduce LTL model checking problem for register pushdown systems (RPDS), show the problem is decidable and investigate its computational complexity. In Sect. 2, we define RPDS where the guard condition of a transition rule is concisely defined by an equivalence relation among registers and the stack top. In Sect. 3, we also define register automata (RA) and Büchi RPDS (BRPDS) as related classes of RPDS. An RA is equipped with initial states and accepting conditions and has only pop rules, regarding its pushdown stack as the input tape. A Büchi RPDS is equipped with accepting states, and it has a notion of accepting run with reference to that of Büchi automata. We define the LTL model checking problem for RPDS in Sect. 4. The problem takes an RPDS $\mathcal{P}$ with a start configuration c_0, an LTL formula φ and a valuation Λ, and asks whether every run of $\mathcal{P}$ reachable from c_0 satisfies φ under Λ. In Sect. 5, we give two practical examples. One is about malware detection problem and the other is about XML schema checking, which can be represented as model checking problems with simple and regular valuations, respectively. In Sect. 6, we show the decidability of LTL model checking problem for RPDS with simple and regular valuations. In the case of simple valuation, we reduce the model checking problem to the membership problem for RA. The size of the constructed RA is exponential to the input size of the problem and the problem is shown to be EXPTIME-complete. We show the problem with regular valuation is also EXPTIME-complete by reducing the problem to the one with simple valuation.

The contributions of the paper are as follows.

- We define simple and regular valuations for RPDS as extensions of those for PDS [6], [15], [16]. Also, we extend RPDS to Büchi RPDS in a similar way to PDS to Büchi PDS. These definitions are essential to the proposed model checking algorithm for RPDS.
- We propose an algorithm of LTL model checking for RPDS. Though the basic flow of the algorithm is similar to that of [16], the termination of the algorithm is nontrivial because RPDS deals with data values from an infinite set. We show that the behavior of an RPDS depends only on equivalence relation over contents of its registers and stack. The finiteness of such equivalence relations is crucial to the termination.
- While the extension from PDS to RPDS does not increase the complexity, which is EXPTIME-complete, the proposed model checking method can be applied to recursive programs or data structures without getting rid of the information on data values. This advantage realizes more accurate analysis in various applications. For example, the malware detection problem in Sect. 5 requires to distinguish contents of variables of the same name in beneign and malicious programs. This distinction is not possible in the previous methods such as [27], but our method can.

In the algorithm for regular valuations, we assume backward-deterministic RAs are available for representing the regular pattern of the stack contents. This constraint is essential because unlike NFAs, determinization is not possible for general RAs. Thus, regular valuations for RPDS can only handle deterministic cases, not as [16]. However, we do not often need backward nondeterminism to represent the regular pattern of stack contents as the second example in Sect. 5 illustrates.

Related work *Automata with data values*: As extensions of finite automata, RA [18], data automata [7], pebble automata (PA) are known. Libkin and Vrgoč,LV12 argue that RA is the only model that has efficient data complexity for membership among the above extensions. Neven, et al. consider variations of RA and PA, which are either one way or two ways, deterministic, nondeterministic or alternating. They show inclusion and separation relationships among these automata, FO($\sim, <$) and EMSO($\sim, <$) [21]. Nominal automata (NA) is defined by an infinite set of data values with orbit-finite symmetry and finite support, and properties of NA are investigated including Myhill-Nerode theorem, closure and a sufficient condition for finite representation in [4]. (Usual) RA with equality and RA with total order can be regarded as NA where the data sets have equality symmetry and total order symmetry, respectively. There are other extensions of RA such as first-order NFA (FO-NFA) [11], RA with linear arithmetic [8] and symbolic RA [12].

PDS and RPDS: Regularity preservation is an essential property for showing the decidability of basic problems, which was shown by Büchi in 1964 (also see [17]) for PDS. Later, this property of PDS was widely used for PDS model checking, one of the most successful approaches to infinite-state model checking [6], [15], [28]. For example, PDS model checking was applied to security verification of stack inspection, which is a security enforcement mechanism in Java runtime environment [16], [22].

For RPDS, backward and forward regularity preservation properties were shown in [20] and [26], respectively. As a related model, register context-free grammar is also studied [9], [24], [25]. First-order definable PDS (FO-PDS) is an extension of both of RPDS and FO-NFA. Reachability of FO-PDS is decidable when the structure of data values is oligomorphic and solvable in EXPTIME with stronger restriction called homogeneity [11]. In [29], well-structured PDS (WS-PDS) is defined as an PDS with well quasi-ordered control states and stack alphabet, and its coverability is shown to be decidable if the set of control states is finite.

LTL with freeze quantifier: Linear temporal logic (LTL) was extended to LTL$\downarrow$ with freeze quantifier [13], [14]. The relationship among subclasses of LTL$\downarrow$ and RA as well as the decidability and complexity of the satisfiability (nonemptiness) problems are investigated [13]. They especially showed that the emptiness problem for (both nondeterministic and deterministic) RA are PSPACE-complete.

Application to infinite model checking: As mentioned, the infinity of data sets causes a difficulty in analyzing programs that deal with data values. In previous studies such as [2], [3], [23], this problem was solved by expressing a program by a computational model with finite elements. In [2], a pushdown automaton with gap-order constraints is defined, which can check whether data values in two registers have a gap larger than a given natural number or not, and it is proved that the reachability of the model with such gap constraints is decidable. In [23], an extension of PDS is introduced, which can encode global and local variables as well as call and return of procedure and create an unbounded number of objects. They also showed such behavior of objects can be simulated by a finite set of natural numbers. The reachability problem of multithreaded PDS dealing with pointer values is addressed in [3] and shown to be decidable on some assumptions such as bounded depth heap and finite context switch. Multi-pushdown models with data values have been also studied under certain constraints (e.g., on context switch) [1], [5]. However, none of these studies discuss the computational complexity and they deal with only the reachability, i.e., runs (or traces) of finite length.

2. Preliminaries

Let $\mathbb{N} = \{1, 2, \ldots\}$ and $\mathbb{N}_0 = \{0\} \cup \mathbb{N}$. We assume a countable set D of *data values*. For a given $k \in \mathbb{N}_0$, let D^k be the set of all words over D whose lengths are k. A mapping $\theta : [k] \to D$ is called an *assignment* (of data values to k registers) where $[k] = \{1, 2, \ldots, k\}$. Let Θ_k denote the collection of assignments to k registers. For a set A, let A^* and A^ω be the sets of finite and infinite words over A, and let $A^\infty = A^* \cup A^\omega$. For a word $\alpha \in A^\infty$ over a set A, let $\alpha(i) \in A$ be the i-th element of α $(i \geq 0)$. We write $\exists^\omega i.P(i)$ if there exist infinitely many i that satisfy the condition $P(i)$. By $|\beta|$, we mean the cardinality of β if β is a set and the length of β if β is a finite sequence.

2.1 Equivalence Relation and Quotient Set

We define the equivalence relation $\equiv$ over D^k as follows. For $d_1, \ldots, d_k, d'_1, \ldots, d'_k \in D$, $d_1 \cdots d_k \equiv d'_1 \cdots d'_k$ iff $d_i = d_j \Leftrightarrow d'_i = d'_j$ for all $i, j \in [k]$. For example, $35523 \equiv 84418$ while $12 \not\equiv 33$. Let $\Phi_k = D^k/\equiv$ denote the quotient set of D^k by $\equiv$. Let $\Phi = \bigcup_{k \in \mathbb{N}} \Phi_k$. For $u \in D^k$, let $\langle u \rangle \in \Phi_k$ denote the equivalence class containing u. To relate an assignment with an equivalence class in Φ, we regard an assignment $\theta \in \Theta_k$ as a word $\theta(1) \cdots \theta(k) \in D^k$ and thus $\langle \theta \rangle$ means $\langle \theta(1) \cdots \theta(k) \rangle$. For $u_1, \ldots, u_m \in D^*$, we let $\langle u_1, \ldots, u_m \rangle$ denote $\langle u_1 \cdots u_m \rangle$, and thus for $\theta, \theta' \in \Theta_k$ and $d \in D$, $\langle \theta, \theta', d \rangle$ means $\langle \theta(1) \cdots \theta(k) \theta'(1) \cdots \theta'(k) d \rangle \in \Phi_{2k+1}$.

We will use an equivalence class ϕ in Φ_{2k+1} for defining the meaning of a transition rule of an RPDS. When we apply a transition rule labeled with ϕ, we require $\langle \theta, \theta', d \rangle = \phi$ to be satisfied, where $\theta, \theta' \in \Theta_k$ are the assignments to the registers before and after the transition, respectively, and d is the value at the stack top before the transition. In other words, a transition rule labeled with ϕ can be applied when the current assignment θ and the current stack top d and some assignment θ' satisfy $\langle \theta, \theta', d \rangle = \phi$, and by this transition, the assignment to the registers is updated to θ'.

Note that Φ_k is isomorphic to the set of equivalence relations over $[k]$. That is, for each $\phi \in \Phi_k$, we can define a unique equivalence relation $\sim_\phi$ over $[k]$ as $i \sim_\phi j$ iff every data word $d_1 \cdots d_k \in D^k$ such that $\langle d_1 \cdots d_k \rangle = \phi$ satisfies $d_i = d_j$. For example, for $\phi = \langle 35523 \rangle$, $i \sim_\phi i$ $(i \in [5])$, $1 \sim_\phi 5$, $2 \sim_\phi 3$ and $i \nsim_\phi j$ for every other $i, j \in [5]$. Conversely, each equivalence relation $\sim$ over $[k]$ determines a unique element ϕ in Φ_k, which is the equivalence class containing every data word $d_1 \cdots d_k \in D^k$ such that $d_i = d_j \Leftrightarrow i \sim j$. By this correspondence between Φ_k and the set of equivalence relations over $[k]$, $|\Phi_k| \leq 2^{\frac{(k-1)k}{2}}$ holds because the number of reflexive symmetric binary relations over k elements is $2^{\frac{(k-1)k}{2}}$. ($|\Phi_k|$ is known as *Bell number* B_k, which is still exponential to k^2.)

We use the above-mentioned equivalence relation $\sim_\phi$ for specifying $\phi \in \Phi_{2k+1}$ in a transition rule. Moreover, for readability, we write $1, 2, \ldots, k, k+1, \ldots, 2k, 2k+1$ appearing in the notation $i \sim_\phi j$ as $\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_k, \mathtt{x}'_1, \ldots, \mathtt{x}'_k, \mathtt{top}$, respectively. For example, $\mathtt{x}_i \sim_\phi \mathtt{x}'_j$ means $\theta(i) = \theta'(j)$ for every $\theta, \theta' \in \Theta_k$ that satisfy $\langle \theta, \theta', d \rangle = \phi$ for some $d \in D$. In other words, $\mathtt{x}_i \sim_\phi \mathtt{x}'_j$ means that the contents of the i-th register before the transition is equal to the contents of the j-th register after the transition. Intuitively, $\mathtt{x}_i \sim_\phi \mathtt{x}'_j$ and $\mathtt{top} \sim_\phi \mathtt{x}'_j$ represent how the registers are updated. On the other hand, $\mathtt{x}_i \sim_\phi \mathtt{x}_j$ and $\mathtt{x}_i \sim_\phi \mathtt{top}$ represent the guard condition of the transition, which means that this transition rule can be applied if and only if the current assignment θ and the value d at the stack top satisfy $\theta(i) = \theta(j)$ and $\theta(i) = d$, respectively.

We define the *projection* of an equivalence class $proj : \Phi_{2k+1} \to \Phi_{k+1}$ as $proj(\langle \theta, \theta', d \rangle) = \langle \theta, d \rangle$ for $\theta, \theta' \in \Theta_k$ and $d \in D$.

2.2 Linear Temporal Logic (LTL)

Let At be a finite set of atomic propositions, and let $\Sigma = 2^{At}$. An LTL formula over At is given by the following syntax:

$$\varphi := \mathrm{tt} \mid A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathcal{X}\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2$$

where $A \in At$. For an infinite word $w = w(0)w(1) \cdots \in \Sigma^\omega$, the satisfaction relation $\models$ is defined as follows.

$$\begin{array}{ll}
w \models \mathrm{tt} & \\
w \models A & \Longleftrightarrow A \in w(0) \\
w \models \neg\varphi & \Longleftrightarrow w \not\models \varphi \\
w \models \varphi_1 \wedge \varphi_2 & \Longleftrightarrow w \models \varphi_1 \text{ and } w \models \varphi_2 \\
w \models \mathcal{X}\varphi & \Longleftrightarrow w(1)w(2) \cdots \models \varphi \\
w \models \varphi_1 \,\mathcal{U}\, \varphi_2 & \Longleftrightarrow \text{there exists } j \in \mathbb{N}_0 \text{ such that} \\
 & \qquad w(j)w(j+1) \cdots \models \varphi_2 \text{ and} \\
 & \qquad w(i)w(i+1) \cdots \models \varphi_1 \text{ for all } 0 \leq i < j
\end{array}$$

We also define $\mathrm{ff} \equiv \neg\mathrm{tt}$, $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \Rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$, $\mathcal{F}\varphi \equiv \mathrm{tt}\,\mathcal{U}\,\varphi$ and $\mathcal{G}\varphi \equiv \neg\mathcal{F}(\neg\varphi)$.

3. Register Pushdown Systems and Register Automata

3.1 Definitions

Definition 3.1: A k-register pushdown system (k-RPDS) over an infinite set D of data values is a pair $\mathcal{P} = (P, \Delta)$ where P is a finite set of states and Δ is a finite set of transition rules having one of the following forms:

- $(p, \phi) \to (q, \varepsilon)$ (pop rule)
- $(p, \phi) \to (q, j_1)$ (replace rule)
- $(p, \phi) \to (q, j_1 j_2)$ (push rule)

where $p, q \in P$, $j_1, j_2 \in [k]$, and $\phi \in \Phi_{2k+1}$.

For a state $p \in P$, an assignment $\theta \in \Theta_k$ and a stack $w \in D^*$, (p, θ, w) is called a *configuration* or *instantaneous description (abbreviated as ID)* of k-RPDS $\mathcal{P}$. Let $ID_\mathcal{P}$ denote the set of all IDs of $\mathcal{P}$. For two IDs $(p, \theta, w), (q, \theta', w') \in ID_\mathcal{P}$, we say that (q, θ', w') is a successor of (p, θ, w), written as $(p, \theta, w) \Rightarrow_\mathcal{P} (q, \theta', w')$, if there exist a rule $r = (p, \phi) \to (q, J) \in \Delta$, a data value $d \in D$, and a sequence of data values $u \in D^*$ such that $\langle \theta, \theta', d \rangle = \phi$, $w = du$ and

$$w' = \begin{cases} u & \text{if } J = \varepsilon, \\ \theta'(j_1)u & \text{if } J = j_1, \text{ or} \\ \theta'(j_1)\theta'(j_2)u & \text{if } J = j_1 j_2. \end{cases}$$

Let $\Rightarrow^*_{\mathcal{P}}$ and $\Rightarrow^+_{\mathcal{P}}$ be the reflexive transitive closure and the transitive closure of $\Rightarrow_{\mathcal{P}}$, respectively. For $n \in \mathbb{N}_0$, let $\Rightarrow^n_{\mathcal{P}}$ be the n-times composition of $\Rightarrow_{\mathcal{P}}$; that is, $c \Rightarrow^0_{\mathcal{P}} c' \Leftrightarrow c = c'$ and for every $n > 0$, $c \Rightarrow^n_{\mathcal{P}} c' \Leftrightarrow c \Rightarrow^{n-1}_{\mathcal{P}} c''$ and $c'' \Rightarrow_{\mathcal{P}} c'$ for some c''. If $\mathcal{P}$ is clear from the context, we abbreviate $\Rightarrow_{\mathcal{P}}$, $\Rightarrow^*_{\mathcal{P}}$, $\Rightarrow^+_{\mathcal{P}}$, etc. as $\Rightarrow$, $\Rightarrow^*$, $\Rightarrow^+$, etc., respectively. If we emphasize the rule r and the data value d, we write $\Rightarrow^r_d$. By definition, any ID $(p, \theta, \varepsilon) \in ID_{\mathcal{P}}$ has no successor. That is, there is no transition from an ID with empty stack. A *run* of k-RPDS $\mathcal{P}$ is a finite or infinite sequence of IDs $\rho \in (ID_{\mathcal{P}})^\infty$ that satisfy $\rho(i) \Rightarrow_{\mathcal{P}} \rho(i+1)$ for $i \geq 0$.

Example 3.1: Let us consider 2-RPDS $\mathcal{P} = (\{p_0, p, p_f\}, \{(p_0, \phi) \to (p, 12) \mid \mathtt{x}_2 \sim_\phi \mathtt{x}'_2 \nsim_\phi \mathtt{x}'_1, \phi \in \Phi_5\} \cup \{(p, \phi) \to (p, J) \mid \mathtt{x}_2 \sim_\phi \mathtt{x}'_2 \nsim_\phi \mathtt{x}'_1, \phi \in \Phi_5, J \in \{11, 1, \varepsilon\}\} \cup \{(p, \phi) \to (p_f, 2) \mid \mathtt{top} \sim_\phi \mathtt{x}_2, \phi \in \Phi_5\})$. In the example, we write $\theta \in \Theta_2$ as $[d, d']$ where $\theta(1) = d$ and $\theta(2) = d'$. In the following, $d_0, d_1, \ldots, d_7 \in D$ represent eight distinct data values. We show a run of $\mathcal{P}$ from $(p_0, [d_0, d_1], d_2) \in ID_{\mathcal{P}}$. First, we can apply only the rule of the form $(p_0, \phi) \to (p, 12)$ where $\phi \in \Phi_5$ satisfies $\mathtt{x}_2 \sim_\phi \mathtt{x}'_2 \nsim_\phi \mathtt{x}'_1$, which requires the 2nd register is not changed by the transition and the contents of the 1st and 2nd registers after the transition are not the same. Therefore, one of the possible transitions is $(p_0, [d_0, d_1], d_2) \Rightarrow (p, [d_3, d_1], d_3 d_1)$ by a rule $(p_0, \phi) \to (p, 12)$ where ϕ is uniquely given by the restrictions $\mathtt{x}_2 \sim_\phi \mathtt{x}'_2 \nsim_\phi \mathtt{x}'_1$, $\mathtt{x}_1 \nsim_\phi \mathtt{x}_2$, $\mathtt{x}_1 \nsim_\phi \mathtt{top}$, $\mathtt{x}_1 \nsim_\phi \mathtt{x}'_1$, $\mathtt{x}_2 \nsim_\phi \mathtt{top}$, $\mathtt{x}_2 \nsim_\phi \mathtt{x}'_1$ and $\mathtt{top} \nsim_\phi \mathtt{x}'_1$. Note that the stack contents after the transition are $d_3 d_1$, i.e., the contents of the 1st and the 2nd registers after the transition. We cannot apply a rule of the form $(p, \phi) \to (p_f, 2)$ to $(p, [d_3, d_1], d_3 d_1)$ because this ID does not satisfy the restriction $\mathtt{top} \sim_\phi \mathtt{x}_2$ required by ϕ. On the other hand, we can apply some rules of the form $(p, \phi) \to (p, J)$ where $J \in \{11, 1, \varepsilon\}$, and obtain a run $(p, [d_3, d_1], d_3 d_1) \Rightarrow (p, [d_4, d_1], d_4 d_4 d_1) \Rightarrow (p, [d_5, d_1], d_5 d_4 d_1) \Rightarrow (p, [d_5, d_1], d_4 d_1) \Rightarrow (p, [d_6, d_1], d_1)$ for example (Fig. 1). Note that every transition in this run satisfies the restriction $\mathtt{x}_2 \sim_\phi \mathtt{x}'_2 \nsim_\phi \mathtt{x}'_1$. Finally, we can apply a rule of the form $(p, \phi) \to (p_f, 2)$ to $(p, [d_6, d_1], d_1)$ because this ID satisfies $\mathtt{top} \sim_\phi \mathtt{x}_2$, and thus we obtain $(p, [d_6, d_1], d_1) \Rightarrow (p_f, [d_6, d_7], d_7)$ for example.

3.2 Register Automata

We define a register automaton (RA) as an RPDS equipped with initial states and accepting conditions that has only pop rules. While an instantaneous description of RA has the same form (p, θ, w) as an ID of RPDS, the third component w is regarded as an input string to the RA.

Definition 3.2: A k-register automaton (k-RA) over an infinite set D of data values is $\mathcal{A} = (\mathcal{P}, I, \xi)$, where

- $\mathcal{P} = (P, \Delta)$ is a k-RPDS where Δ consists of pop rules only,
- $I \subseteq P$ is a set of initial states,
- $\xi \subseteq P \times \Phi_k$ is a set of *accepting conditions*.

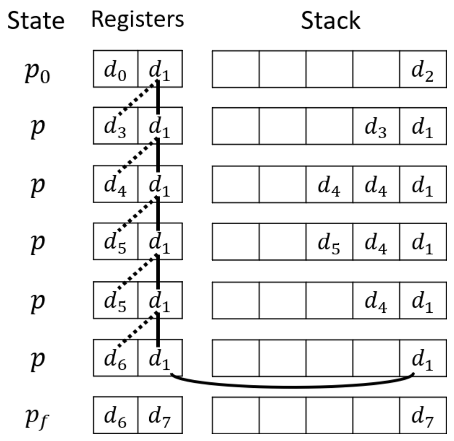


Fig. 1 A run of $\mathcal{P}$ from an ID $(p_0, [d_0, d_1], d_2)$. (The equality restrictions given by $\sim$ and the disequality restrictions given by $\nsim$ are shown by solid and dotted lines, respectively.)

We call $(p, \theta, \varepsilon) \in ID_{\mathcal{P}}$ an *accepting ID* of $\mathcal{A}$ if there exists $(p, \psi) \in \xi$ such that $\langle\theta\rangle = \psi$. We denote the set of all accepting ID as $Acc_{\mathcal{A}}$. We assume that any $p \in I$ does not appear in the right-hand side of any rule in Δ.

We define the set of IDs of $\mathcal{A}$ as that of $\mathcal{P}$, i.e., $ID_{\mathcal{A}} = ID_{\mathcal{P}}$. We also define the transition relation $\vdash_{\mathcal{A}}$ of $\mathcal{A}$ as that of $\mathcal{P}$, i.e., $\vdash_{\mathcal{A}} = \Rightarrow_{\mathcal{P}}$. Let $\vdash^*_{\mathcal{A}}$ be the reflexive transitive closure of $\vdash_{\mathcal{A}}$. We abbreviate $\vdash_{\mathcal{A}}$ and $\vdash^*_{\mathcal{A}}$ as $\vdash$ and $\vdash^*$ if $\mathcal{A}$ is obvious.

In the rest of the paper, we write $\mathcal{A} = (\mathcal{P}, I, \xi)$ with $\mathcal{P} = (Q, \delta)$ as (Q, I, ξ, δ) and $(p, \phi) \to (q, \varepsilon) \in \delta$ as $(p, \phi) \to q$. The language $L(\mathcal{A})$ recognized by a k-RA $\mathcal{A} = (Q, I, \xi, \delta)$ is defined as $L(\mathcal{A}) = \{(p, \theta, w) \in ID_{\mathcal{A}} \mid p \in I, (p, \theta, w) \vdash^* (q, \theta', \varepsilon)$ for some $(q, \theta', \varepsilon) \in Acc_{\mathcal{A}}\}$. We call a set $C \subseteq I \times \Theta_k \times D^*$ *regular* if there exists a k-RA $\mathcal{A}$ that satisfies $C = L(\mathcal{A})$. (The number k of registers of $\mathcal{A}$ equals the number of registers the set C of IDs assumes.)

As a related class of RA, we define backward-deterministic RA.

Definition 3.3: A *backward-deterministic* k-RA $\mathcal{A} = (Q, I, (q_f, \theta_f), \delta)$ is defined as an RA whose accepting condition is given as a pair $(q_f, \theta_f) \in Q \times \Theta_k$ and every pair of ID of the form $(q, \theta, \varepsilon) \in Q \times \Theta_k \times \{\varepsilon\}$ and $d \in D$ determine a unique predecessor ID $(q', \theta', d) \in Q \times \Theta_k \times D$ such that $(q', \theta', d) \vdash (q, \theta, \varepsilon)$.

We call a set $C \subseteq I \times \Theta_k \times D^*$ a *backward-deterministic regular set* if there exists a backward-deterministic k-RA $\mathcal{A}$ that satisfies $C = L(\mathcal{A})$.

3.3 Properties of RPDS and RA

Consider the relation $\sim$ over IDs defined by $(p_1, \theta_1, u_1) \sim (p_2, \theta_2, u_2)$ iff $p_1 = p_2$ and $\langle\theta_1, u_1\rangle = \langle\theta_2, u_2\rangle$. The following lemma states that $\sim$ is a bisimulation that preserves the number of transition steps, which will be used for proving Proposition 3.2. This lemma also applies to RA and Büchi RPDS, which will be described later.

Lemma 3.1: For a k-RPDS $\mathcal{P} = (P, \Delta)$, IDs (p, θ_0, u_0), (p, θ_1, u_1) of $\mathcal{P}$ such that $\langle\theta_0, u_0\rangle = \langle\theta_1, u_1\rangle$, and $q \in P$, if there exists an ID $(q, \theta_2, u_2) \in ID_{\mathcal{P}}$ that satisfies $(p, \theta_0, u_0) \Rightarrow^n (q, \theta_2, u_2)$ for some $n \in \mathbb{N}_0$, then there exists an ID $(q, \theta_3, u_3) \in ID_{\mathcal{P}}$ that satisfies $(p, \theta_1, u_1) \Rightarrow^n (q, \theta_3, u_3)$ and $\langle\theta_2, u_2\rangle = \langle\theta_3, u_3\rangle$.

The following lemma is a property of RA similar to, but stronger than the property of RPDS stated in the previous lemma in the sense that any run of a k-RA can be simulated by a run that uses only $2k$ fresh data values that do not appear in the input data word of the original run. This lemma will be used for proving Proposition 3.1 described below.

Lemma 3.2: Let $\mathcal{A} = (Q, I, \xi, \delta)$ be a k-RA and $(p_0, \theta_0, w_0) \vdash (p_1, \theta_1, w_1) \vdash \cdots \vdash (p_n, \theta_n, w_n)$ be a run of $\mathcal{A}$, where $w_0 = d_1 d_2 \cdots d_m$. Let $D_{w_0} = \{d_1, \ldots, d_m\}$ and let $D' \subseteq D$ be a finite subset of D such that $D_{w_0} \subseteq D'$ and $|D' \setminus D_{w_0}| \geq 2k$. Let $\Theta'_k \subseteq \Theta_k$ be the set of assignments whose range is D', and let $\theta'_0 \in \Theta'_k$ be an assignment such that $\langle\theta_0, w_0\rangle = \langle\theta'_0, w_0\rangle$. Then, there exist $\theta'_1, \ldots, \theta'_n \in \Theta'_k$ such that $(p_0, \theta'_0, w_0) \vdash (p_1, \theta'_1, w_1) \vdash \cdots \vdash (p_n, \theta'_n, w_n)$ and $\langle\theta_j, w_j\rangle = \langle\theta'_j, w_j\rangle$ for each $j \in [n]$.

The following proposition is essentially the same as [18, Proposition 1], and used for showing decidability of the membership problem of an RA.

Proposition 3.1: For a k-RA $\mathcal{A} = (Q, I, \xi, \delta)$, a finite subset D' of D, $p \in I$, and $\theta \in \Theta_k$ such that $\theta(i) \in D'$ for $i \in [k]$, we can construct a nondeterministic finite automaton (NFA) $\mathcal{A}^p_\theta$ over D' such that for any $w \in (D')^*$, $w \in L(\mathcal{A}^p_\theta)$ iff $(p, \theta, w) \in L(\mathcal{A})$. Moreover, the number of states of $\mathcal{A}^p_\theta$ is at most $|Q| \cdot (2k + |D'|)^k$.

3.4 Büchi Register Pushdown Systems

We define an extended model of RPDS called Büchi register pushdown systems (abbreviated as BRPDS) by adding a notion of Büchi acceptance. In the case of LTL model checking for PDS described in [16], Büchi PDS is defined and used in the algorithm for model checking. Similarly, BRPDS will be used in the proposed model checking algorithm.

Definition 3.4: A k-BRPDS $\mathcal{BP} = (P, \Delta, G)$ is defined as a k-RPDS accompanied with the acceptance condition given by a set of *repeated states* G $(\subseteq P)$. The set $ID_{\mathcal{BP}}$ of all IDs of $\mathcal{BP}$ and the transition relation $\Rightarrow_{\mathcal{BP}}$ of $\mathcal{BP}$ are defined in the same way as those of k-RPDS. An *accepting run* of $\mathcal{BP}$ is a run $\rho \in (ID_{\mathcal{BP}})^\omega$ of $\mathcal{BP}$ that satisfies $\exists^\omega i.\ \rho(i) \in G \times \Theta_k \times D^*$. We say that $c \in ID_{\mathcal{BP}}$ is accepted by $\mathcal{BP}$ if there exists an accepting run ρ such that $\rho(0) = c$.

The following proposition describes a necessary and sufficient condition for an ID to be accepted by a BRPDS.

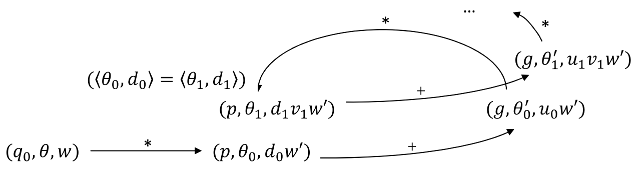

*
...
*
$(g, \theta'_1, u_1 v_1 w')$
$(\langle\theta_0, d_0\rangle = \langle\theta_1, d_1\rangle)$
+
$(p, \theta_1, d_1 v_1 w')$
$(g, \theta'_0, u_0 w')$
(q_0, θ, w)
*
$(p, \theta_0, d_0 w')$
+

Fig. 2 An accepting run

This proposition is a natural but non-trivial extension of [6, Proposition 3.1] to BRPDS.

Proposition 3.2: For a k-BRPDS $\mathcal{BP} = (P, \Delta, G)$ and an ID (q_0, θ, w) of $\mathcal{BP}$, (q_0, θ, w) is accepted by $\mathcal{BP}$ if and only if there exist IDs (p, θ_0, d_0), (g, θ'_0, u_0), $(p, \theta_1, d_1 v_1) \in ID_{\mathcal{BP}}$ where $g \in G$ and $d_0, d_1 \in D$ and $u_0, v_1 \in D^*$ such that the following conditions hold:

(1) $(q_0, \theta, w) \Rightarrow^* (p, \theta_0, d_0 w')$ for some $w' \in D^*$,

(2) $(p, \theta_0, d_0) \Rightarrow^+ (g, \theta'_0, u_0) \Rightarrow^* (p, \theta_1, d_1 v_1)$, and

(3) $\langle\theta_0, d_0\rangle = \langle\theta_1, d_1\rangle$.

Proof. In this proof, for an ID (p, θ, du) of $\mathcal{BP}$ where $d \in D$ and $u \in D^*$, we define the *type* of (p, θ, du) as $(p, \langle\theta, d\rangle)$. We prove the only-if part of the proposition here. The if-part can be proved easily.

(Only-if part) Assume that ρ is an accepting run of $\mathcal{BP}$ such that $\rho(0) = (q_0, \theta, w)$. Let g be an element of G that appears in ρ infinitely many times.

Let $i_0 \in \mathbb{N}_0$ be an index such that the length of the stack in $\rho(i_0)$ is not larger than the stack in $\rho(j)$ for every $j \geq i_0$. Such i_0 must exist because the lengths of stacks are non-negative. There is also an index $i_1 > i_0$ such that the length of the stack in $\rho(i_1)$ is not larger than the stack in $\rho(j)$ for every $j \geq i_1$ for the same reason. Repeating this selection of indices, we obtain a subsequence $\rho' = \rho(i_0)\rho(i_1)\rho(i_2)\cdots$ of ρ where $0 \leq i_0 < i_1 < i_2 < \cdots$ such that for every $n \geq 0$, the length of the stack in $\rho(i_n)$ is not larger than the stack in $\rho(j)$ for every $j \geq i_n$. Note that if the stack in $\rho(i_n)$ is dw' for some $d \in D$ and $w' \in D^*$, then w' is the suffix of the stack in $\rho(j)$ for every $j \geq i_n$, because any $d \in D$ in that common suffix w' never becomes the stack top in $\rho(i_n)\rho(i_n+1)\rho(i_n+2)\cdots$. Moreover, even if we remove the common suffix w' of every stack in $\rho(i_n)\rho(i_n+1)\rho(i_n+2)\cdots$, the resultant sequence is also a run of $\mathcal{BP}$.

Consider the types of IDs in ρ'. Since P and Φ_{k+1} are finite, there must be a type (p, ϕ) that appears in ρ' infinitely many times. Choose some $\rho(i_m) = (p, \theta_0, d_0 w')$ in ρ' whose type $(p, \langle\theta_0, d_0\rangle)$ equals (p, ϕ). Then, choose some $j > i_m$ such that the state in $\rho(j) = (g, \theta'_0, u_0 w')$ is g. And finally, choose $\rho(i_n) = (p, \theta_1, d_1 v_1 w')$ in ρ' such that $i_n \geq j$ and the type of $\rho(i_n)$ (i.e. $(p, \langle\theta_1, d_1\rangle)$) equals (p, ϕ). Since $\rho(0) \Rightarrow^* \rho(i_m) \Rightarrow^+ \rho(j) \Rightarrow^* \rho(i_n)$, conditions (1) and (2) hold (by removing the common suffix w' of every stack in $\rho(i_m)\rho(i_m+1)\cdots\rho(i_n)$). Condition (3) also holds because $\langle\theta_0, d_0\rangle = \langle\theta_1, d_1\rangle = \phi$. □

3.5 Acceptance Checking for BRPDS

Let $\mathcal{P} = (P, \Delta)$ be a k-RPDS (or a k-BRPDS) and let $C \subseteq ID_{\mathcal{P}}$ be a subset of IDs of $\mathcal{P}$. We define

$$pre_{\mathcal{P}}(C) := \{c \mid c \Rightarrow_{\mathcal{P}} c' \text{ for some } c' \in C\}.$$

Let $pre^*_{\mathcal{P}}$ and $pre^+_{\mathcal{P}}$ be the reflexive transitive closure and the transitive closure of $pre_{\mathcal{P}}$ (as a relation), respectively. It is known that $pre^*_{\mathcal{P}}$ and $pre^+_{\mathcal{P}}$ effectively preserve regularity [20]; that is, if $C \subseteq ID_{\mathcal{P}}$ is regular (i.e. there is an RA $\mathcal{A}$ such that $C = L(\mathcal{A})$), then we can construct an RA $\widetilde{\mathcal{A}}$ such that $pre^*_{\mathcal{P}}(C) = L(\widetilde{\mathcal{A}})$ (or $pre^+_{\mathcal{P}}(C) = L(\widetilde{\mathcal{A}})$).

For $p \in P$ and $\phi \in \Phi_{k+1}$, we let $L^p_\phi = \{(p, \theta, dw) \mid \langle\theta, d\rangle = \phi\}$. We can show that L^p_ϕ is regular but omit the proof here. We also define $L^p_{\phi,1} = \{(p, \theta, d) \mid \langle\theta, d\rangle = \phi, d \in D\}$, which is also regular.

Using pre^*, pre^+, L^p_ϕ and $L^p_{\phi,1}$, Proposition 3.2 can be rephrased as follows, which is a BRPDS version of (1') and (2') in [6, Proposition 3.1]: k-BRPDS $\mathcal{BP} = (P, \Delta, G)$ accepts (q_0, θ, w) iff there exist $p \in P$ and $\phi \in \Phi_{k+1}$ (which is $\langle\theta_0, d_0\rangle$ for θ_0 and d_0 in Proposition 3.2) that satisfy the followings:

(1′) $(q_0, \theta, w) \in pre^*_{\mathcal{BP}}(L^p_\phi)$, and

(2′) $(p, \theta_0, d_0) \in pre^+_{\mathcal{BP}}((G \times \Theta_k \times D^*) \cap pre^*_{\mathcal{BP}}(L^p_\phi))$ for some $(p, \theta_0, d_0) \in L^p_{\phi,1}$.

Obviously, (2′) is equivalent to $pre^+_{\mathcal{BP}}((G \times \Theta_k \times D^*) \cap pre^*_{\mathcal{BP}}(L^p_\phi)) \cap L^p_{\phi,1} \neq \emptyset$. Note that although Proposition 3.2 requires that (p, θ_0, d_0) mentioned in (2′) should satisfy $(q_0, \theta, w) \Rightarrow^* (p, \theta_0, d_0 w')$ for some $w' \in D^*$, we do not need to directly test it, because by Lemma 3.1, if $(p, \theta_0, d_0) \Rightarrow^+ (g, \theta'_0, u_0) \Rightarrow^* (p, \theta_1, d_1 v_1)$ for some $(g, \theta'_0, u_0) \in G \times \Theta_k \times D^*$ and $(p, \theta_1, d_1 v_1) \in L^p_\phi$, then for *every* $(p, \theta_2, d_2) \in L^p_{\phi,1}$, there exist $(g, \theta'_2, u_2) \in G \times \Theta_k \times D^*$ and $(p, \theta_3, d_3 v_3) \in L^p_\phi$ such that $(p, \theta_2, d_2) \Rightarrow^+ (g, \theta'_2, u_2) \Rightarrow^* (p, \theta_3, d_3 v_3)$.

It is known that the class of languages recognized by RA is closed under intersection, and the membership problem for RA is decidable [18]. Hence, conditions (1′) and (2′) are decidable. Because both P and Φ_{k+1} are finite, we can decide whether there exist p and ϕ that satisfy (1′) and (2′). We can conclude the above discussion as follows.

Proposition 3.3: For a k-BRPDS $\mathcal{BP} = (P, \Delta, G)$ and an ID (q_0, θ, w) of $\mathcal{BP}$, it is decidable whether (q_0, θ, w) is accepted by $\mathcal{BP}$ in $O((|\Delta| + |w|^{O(k)}) \cdot |P|^{O(1)} \cdot 2^{O(k^2)})$ time.

4. LTL Model Checking Problem and Valuations

In the rest of the paper, we fix a finite set At of atomic propositions and also let $\Sigma = 2^{At}$.

A *valuation* for a k-RPDS $\mathcal{P} = (P, \Delta)$ is $\Lambda : ID_{\mathcal{P}} \to \Sigma$, which labels each ID of $\mathcal{P}$ with atomic propositions. We extend the domain of Λ to $(ID_{\mathcal{P}})^\infty$ pointwise; that is, $\Lambda(c_1 c_2 \cdots) = \Lambda(c_1)\Lambda(c_2)\cdots$ for $c_1, c_2, \ldots \in ID_{\mathcal{P}}$. We define the model checking problem as follows.

Definition 4.1:
Input: A k-RPDS $\mathcal{P} = (P, \Delta)$, an LTL formula φ over At, a valuation $\Lambda : ID_{\mathcal{P}} \to \Sigma$ and an ID $c \in ID_{\mathcal{P}}$.
Model checking problem: Check whether $\Lambda(\rho) \models \varphi$ for all runs $\rho \in (ID_{\mathcal{P}})^\omega$ of $\mathcal{P}$ such that $\rho(0) = c$.

For general valuations, LTL model checking problem is undecidable even for PDS [16]. Therefore, we focus on two restricted cases, *simple valuations* (Definition 4.2) and *regular valuations* (Definition 4.3), which are reasonable extensions of simple and regular valuations for finite-alphabet models [16].

Example 4.1: Let us consider a model checking problem for the 2-RPDS $\mathcal{P} = (P, \Delta)$ defined in Example 3.1 and an LTL formula $\varphi = \mathcal{X}(\mathcal{F}C \Rightarrow (A\ \mathcal{U}\ B))$. Let $\Lambda : ID_{\mathcal{P}} \to \Sigma$ ($\Sigma = 2^{\{A,B,C\}}$) be the valuation satisfying the followings:

- $A \in \Lambda(c)$ iff $c \in \{(p, \theta, du) \in \{p\} \times \Theta_k \times D^+ \mid \theta(1) \neq \theta(2) \neq d, d \in D, u \in D^*\}$
- $B \in \Lambda(c)$ iff $c \in \{(p, \theta, du) \in \{p\} \times \Theta_k \times D^+ \mid \theta(1) \neq \theta(2) = d, d \in D, u \in D^*\}$
- $C \in \Lambda(c)$ iff $c \in \{(p_f, \theta, u) \in \{p_f\} \times \Theta_k \times D^+\}$

for all $c \in ID_{\mathcal{P}}$.

For the run ρ in Fig. 1, $\Lambda(\rho) = \{\}\{A\}\{A\}\{A\}\{A\}\{B\}\{C\}$ holds, and thus $\Lambda(\rho) \models \varphi$. In fact, $\Lambda(\rho') \models \varphi$ holds for all $\rho' \in (ID_{\mathcal{P}})^\omega$ such that $\rho'(0) = (p_0, [d_0, d_1], d_2)$. We can solve this model checking problem because the valuation Λ is a simple valuation, which will be defined in Definition 4.2.

Next, we define simple and regular valuations. Let $\Lambda^{-1} : \Sigma \to 2^{ID_{\mathcal{P}}}$ be the inverse of a valuation $\Lambda : ID_{\mathcal{P}} \to \Sigma$, that is, $\Lambda^{-1}(\sigma) = \{(p, \theta, u) \in ID_{\mathcal{P}} \mid \Lambda(p, \theta, u) = \sigma\}$ for every $\sigma \in \Sigma$.

Recall $L^p_\phi = \{(p, \theta, dw) \mid \langle\theta, d\rangle = \phi, d \in D, w \in D^*\}$ for $p \in P$ and $\phi \in \Phi_{k+1}$ defined in Sect. 3.5.

Definition 4.2: For a k-RPDS $\mathcal{P} = (P, \Delta)$, a subset $C \subseteq ID_{\mathcal{P}}$ is *simple* if $C = L^p_\phi$ for some $p \in P$ and $\phi \in \Phi_{k+1}$. A valuation $\Lambda : ID_{\mathcal{P}} \to \Sigma$ is *simple* if $\Lambda^{-1}(\sigma)$ is a finite union of simple sets of $ID_{\mathcal{P}}$ for every $\sigma \in \Sigma$.

By the definition, a simple valuation determines $\Lambda(p, \theta, dw)$ only by $p \in P$ and $\phi = \langle\theta, d\rangle \in \Phi_{k+1}$. We write a simple valuation as $\Lambda : P \times \Phi_{k+1} \to \Sigma$ instead of $\Lambda : ID_{\mathcal{P}} \to \Sigma$.

Definition 4.3: For a k-RPDS $\mathcal{P} = (P, \Delta)$, a valuation $\Lambda : ID_{\mathcal{P}} \to \Sigma$ is *backward-deterministic regular* if the set $\{c \in ID_{\mathcal{P}} \mid A \in \Lambda(c)\}$ is a backward-deterministic regular set of $ID_{\mathcal{P}}$ for every $A \in At$.

In this paper, we call such Λ a *regular valuation*. The definitions of simple and regular valuations are extensions of those of [18] and [16], respectively. We define a regular valuation for RPDS based on backward-deterministic regular sets of $ID_{\mathcal{P}}$, which corresponds to the regular valuation for PDS in [16] based on deterministic finite automata that decide the acceptance of the reverse of stack contents.

5. Examples

Example 5.1 (malware detection): This example is taken from [27]. Let $\mathcal{P}_{1a}$ and $\mathcal{P}_{1b}$ be the programs shown in Fig. 3 (a) and (b), respectively. $\mathcal{P}_{1a}$ and $\mathcal{P}_{1b}$ push a data value assigned to *eax* in l_1 and l'_1, respectively, and call the function *GetModuleHandleA* with reference to the stack top data value as an argument. Assume that if a program calls *GetModuleHandleA* with 0 as an (unexpected) argument, the latter returns a security sensitive information z to the register *eax*. If the program calls *UseA* with this z as an argument, the program accomplishes its attack. Hence, $\mathcal{P}_{1a}$ is a benign program, but $\mathcal{P}_{1b}$ is considered as a malicious program because $\mathcal{P}_{1b}$ tries to give 0 to *GetModuleHandleA* as an argument and later $\mathcal{P}_{1b}$ pushes the contents of *eax* to the stack and calls *UseA*.

Whichever we use traditional signature analysis or traditional PDS model checking, it is difficult to distinguish the malicious behaviors from the benign ones. Actually, signature analysis does not work if some meaningless instructions (like l'_3 and l'_4 in Fig. 3 (b)) are inserted into the code, and PDS model checking fails to distinguish $\mathcal{P}_{1a}$ and $\mathcal{P}_{1b}$, as discussed in [27], because the analysis assumes data values 0 and 1 are the same. (The example given in [27] assumes the stack contents to be the names of registers, instead of data values in the registers. Therefore, we cannot distinguish benign and malicious programs if the name of register used as an argument of *GetModuleHandleA* is the same as in the case of *eax* appearing in $\mathcal{P}_{1a}$ and $\mathcal{P}_{1b}$.) In [27], PDS model checking was extended so that it can deal with a finite number of data values.

We construct $\mathcal{P}_{1a}$ and $\mathcal{P}_{1b}$ as 3-RPDS whose 1st and 2nd registers are used for keeping the contents of *eax* and *ebx*, respectively, and the data value in the 3rd register stands for 0 in the original program. Let $\mathcal{P}_1 = (P_{1a}, \Delta_{1a})$ be 3-RPDS where $P_{1a} = \{l_j \mid j \in [5]\} \cup \{l_{end}\}$ and $\Delta_{1a} = \{(l_1, \phi_1) \to (l_2, 2) \mid \mathtt{x}'_1 \nsim_{\phi_1} \mathtt{x}_3 \sim_{\phi_1} \mathtt{x}'_3,\ \mathtt{x}_2 \sim_{\phi_1} \mathtt{x}'_2\} \cup \{(l_2, \phi_{keep}) \to (l_3, 12), (l_4, \phi_{keep}) \to (l_5, 12), (l_5, \phi_{keep}) \to (l_{end}, \varepsilon) \mid \mathtt{x}_1 \sim_{\phi_{keep}} \mathtt{x}'_1, \mathtt{x}_2 \sim_{\phi_{keep}} \mathtt{x}'_2, \mathtt{x}_3 \sim_{\phi_{keep}} \mathtt{x}'_3\} \cup \{(l_3, \phi_{gh}) \to (l_4, \varepsilon) \mid \mathtt{x}_3 \sim_{\phi_{gh}} \mathtt{x}'_3\}$. $\mathcal{P}_{1a}$ behaves as Fig. 3 (a) by using its registers and stack to restore adequate data values. (Note that $\mathcal{P}_{1a}$ never pushes the data value $\theta(3)$ to the stack at l_3.) We can define 3-RPDS $\mathcal{P}_{1b} = (P_{1b}, \Delta_{1b})$ in the same way as $\mathcal{P}_{1a}$ except that the transition $(l'_1, \phi'_1) \to (l'_2, 2) \in \Delta_{1b}$ should transfer the data value $\theta(3)$ to the first register, and thus we require $\mathtt{x}'_1 \sim_{\phi'_1} \mathtt{x}_3 \sim_{\phi'_1} \mathtt{x}'_3$, and $\mathtt{x}_2 \sim_{\phi'_1} \mathtt{x}'_2$. Let

$$At = \{call_{GMHA}, call_{UseA}, \mathtt{top} = \mathtt{x}_1, \mathtt{top} = \mathtt{x}_3\}$$

be the set of atomic propositions and let $\Sigma = 2^{At}$. For $\mathcal{P}_1$, define the simple valuation $\Lambda_{1a} : P_{1a} \times \Phi_4 \to \Sigma$ as

- $call_{GMHA} \in \Lambda_{1a}(p, \phi)$ iff $p = l_3$,
- $call_{UseA} \in \Lambda_{1a}(p, \phi)$ iff $p = l_5$, and
- for $j = 1, 3$, $\mathtt{top} = \mathtt{x}_j \in \Lambda_{1a}(p, \phi)$ iff $\phi = \langle \theta, d \rangle$ for some $\theta \in \Theta_3$ and $d \in D$ such that $d = \theta(j)$.

Define the simple valuation Λ_{1b} to be the same as Λ_{1a} except that we use l'_5 and l'_7 instead of l_3 and l_5, respectively. We can express the malicious behaviors as

$$\psi_1 = \mathcal{F}[(\mathtt{top} = \mathtt{x}_3) \wedge call_{GMHA} \wedge \mathcal{F}\{(\mathtt{top} = \mathtt{x}_1) \wedge call_{UseA}\}].$$

The answer to the model checking problem for $\mathcal{P}_1$, $\neg\psi_1$, Λ_{1a} and $(l_1, [d_1, d_2, d_3], d_4)$ is 'YES', and the answer is 'NO' for $\mathcal{P}_2$, $\neg\psi_1$, Λ_{1b} and $(l'_1, [d_1, d_2, d_3], d_4)$. As shown in Fig. 4, no run of $\mathcal{P}_1$ from $(l_1, [d_1, d_2, d_3], d_4)$ satisfies ψ_1 because $\mathtt{top} = \mathtt{x}_3$ never hold in l_3 due to ϕ_1. On the other hand, a run of $\mathcal{P}_2$ from $(l'_1, [d_1, d_2, d_3], d_4)$ may satisfy ψ_1 because ϕ'_1 forces $\mathtt{x}'_1 \sim_{\phi'_1} \mathtt{x}_3$.

Example 5.2 (XML processing): We consider an XML document as valid if and only if an element name at the top level does not appear inside the contents surrounded by the start and end tags of the top level. For example,

```
<a> <b> <c> </c> </b> </a> <b> </b>
```

is valid while

```
<c> </c> <a> <b> <a> </a> </b> </a>
```

is not valid because $\langle a \rangle$ and $\langle /a \rangle$ appear inside the contents

```
l1: move eax, 1
l2: push eax
l3: call GMHA
l3r: add esp, 4
l4: push eax
l5: call UseA
```

```
l1': move eax, 0
l2': push eax
l3': push ebx
l4': pop ebx
l5': call GMHA
l5r': add esp, 4
l6': push eax
l7': call UseA
```

Fig. 3 **(a)** benign program and **(b)** malware program, where GMHA stands for GetModuleHandleA

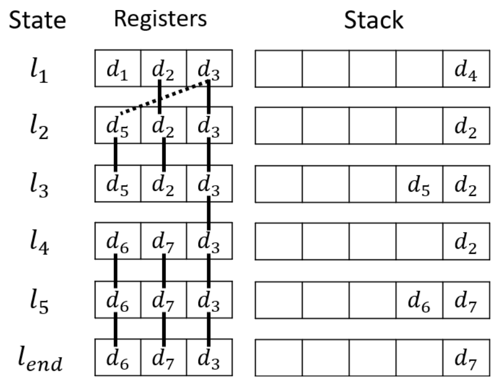


Fig. 4 A run of $\mathcal{P}_1$ from an ID $(l_1, [d_1, d_2, d_3], d_4)$. (The equality restrictions given by $\sim$ and the disequality restrictions given by $\nsim$ are shown by solid and dotted lines, respectively.)

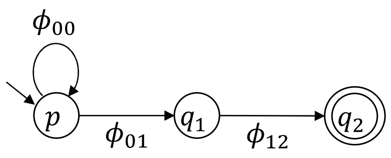


Fig. 5 XML processing

surrounded by the top level $\langle a\rangle$ and $\langle /a\rangle$. A program $\mathcal{P}_2$ takes a non-empty XML document, which is a sequence of start and end tags. First, $\mathcal{P}_2$ pushes $\perp$ as a bottom marker, which is different from any element name. Each time $\mathcal{P}_2$ reads a start tag, $\mathcal{P}_2$ pushes the element name of the tag to the stack. When $\mathcal{P}_2$ reads an end tag, $\mathcal{P}_2$ pops the stack and checks whether the popped data value matches the end tag. Also $\mathcal{P}_2$ must check whether a tag with the same element name as that of the top level never appears (called the occur-check).

Let $At = \{ok\}$ and $\Sigma = 2^{At}$. The following LTL formula states that $\mathcal{P}_2$ correctly performs the occur-check:

$$\psi_2 = \mathcal{G}(ok)$$

where ok is true for an ID of $\mathcal{P}_2$ if and only if the stack contents of the ID has the pattern informally described by a regular expression $(any\backslash d_1)^* \cdot d_1 \cdot \perp$ for some data value d_1. Formally, let $\mathcal{P}_2 = (P_2, \Delta_2)$ be an RPDS, $\Lambda_2 : ID_{\mathcal{P}_2} \to \Sigma$ be the regular valuation defined by $\Lambda_2^{-1}(\{ok\}) = L(\mathcal{A}_2)$ where $\mathcal{A}_2 = (P_2 \cup \{q_1, q_2\}, P_2, (q_2, [\perp]), \Delta_2)$ is the backward-deterministic 1-RA with $P_2 \cap \{q_1, q_2\} = \emptyset$, $\Delta_2 = \{(p, \phi_{00}) \to p \mid p \in P_2,\ \mathtt{top} \nsim_{\phi_{00}} \mathtt{x}_1 \sim_{\phi_{00}} \mathtt{x}_1'\} \cup \{(p, \phi_{01}) \to q_1 \mid \mathtt{x}_1 \sim_{\phi_{01}} \mathtt{top} \nsim_{\phi_{01}} \mathtt{x}_1'\} \cup \{(q_1, \phi_{12}) \to q_2 \mid \mathtt{top} \sim_{\phi_{12}} \mathtt{x}_1 \sim_{\phi_{12}} \mathtt{x}_1'\}$ (see Fig. 5). $\mathcal{A}_2$ guesses the element name at the top level in the XML document given to $\mathcal{P}_2$ and stores it in the register in the initial ID. $\mathcal{A}_2$ verifies whether the guess was correct by ϕ_{01}. An example of a run reaching the accepting ID is: $(p, [d_1], d_2d_1\perp) \vdash (p, [d_1], d_1\perp) \vdash (q_1, [\perp], \perp) \vdash (q_2, [\perp], \varepsilon)$ where $d_1 \neq d_2$.

6. Model Checking LTL with Simple and Regular Valuations

We assume that a simple valuation $\Lambda : P \times \Phi_{k+1} \to \Sigma$ is given as a part of an input in Definition 4.1 and provide an algorithm for the problem. We first construct a Büchi automaton $\mathcal{B}$ over Σ such that $L(\mathcal{B}) = \{\alpha \in \Sigma^\omega \mid \alpha \models \neg\varphi\}$ in the standard way. After that, we construct the *product* of $\mathcal{P}$ and $\mathcal{B}$ as k-BRPDS $\mathcal{BP}$ defined below.

Definition 6.1: Let $\mathcal{P} = (P, \Delta)$ be a k-RPDS, $\mathcal{B} = (Q, q_0, F, \delta)$ be a Büchi automaton over Σ, and $\Lambda : P \times \Phi_{k+1} \to \Sigma$ be a simple valuation function. The product of $\mathcal{P}$ and $\mathcal{B}$ is k-BRPDS $\mathcal{BP} = (P \times Q, \Delta', P \times F)$, where $\Delta' = \{((p,q), \phi) \to ((p', q'), J) \mid (p, \phi) \to (p', J) \in \Delta, (q, \sigma) \to q' \in \delta, \sigma = \Lambda(p, proj(\phi))\}$.

Then, the model checking problem can be reduced to the problem of checking the acceptance of the ID $((p_0, q_0), \theta_0, u_0)$ by $\mathcal{BP}$ as stated in the following proposition.

Proposition 6.1: Let $\mathcal{B} = (Q, q_0, F, \delta)$ be a Büchi automaton over Σ that satisfies $L(\mathcal{B}) = \{\alpha \in \Sigma^\omega \mid \alpha \models \neg\varphi\}$ for a given LTL formula φ over At. Let $\Lambda : ID_{\mathcal{P}} \to \Sigma$ be a simple valuation function. Also, let $\mathcal{BP} = (P \times Q, \Delta', P \times F)$ be the product k-BRPDS of a k-RPDS $\mathcal{P} = (P, \Delta)$ and $\mathcal{B}$ defined in Definition 4.1. Then, $\mathcal{BP}$ does not accept an ID $((p_0, q_0), \theta_0, u_0)$ if and only if $\Lambda(\rho) \models \varphi$ for all run $\rho \in (ID_{\mathcal{P}})^\omega$ such that $\rho(0) = (p_0, \theta_0, u_0)$.

Since the acceptance problem for BRPDS is decidable by Proposition 3.3, the model checking problem with simple valuation is decidable.

Theorem 6.1: The LTL model checking problem for RPDS with simple valuation is EXPTIME-complete.

Next, we show the decidability of the model checking problem with regular valuation. We prove this by a reduction to the model checking problem with simple valuation as shown in Proposition 6.2.

The idea behind the proof is based on [16], but we need to extend the idea of [16] to deal with RPDS as follows. To simulate a given RPDS $\mathcal{P}$ with a regular valuation by an RPDS $\mathcal{P}'$ with a simple valuation, the current IDs of backward-deterministic RAs for all atomic propositions are stored in additional registers and the remaining IDs are stored in the stack of $\mathcal{P}'$. The purpose of this transformation is that we can determine which atomic propositions hold under the regular valuation of $\mathcal{P}$ by observing only the finite state, the (extended) register assignments and the stack top of $\mathcal{P}'$. For example, assume $At = \{A\}$ and let $\mathcal{A} = (Q, q_0, F, \delta)$ be a backward-deterministic RA for the proposition A. For stack contents $d_n \cdots d_1 \in D^*$ of $\mathcal{P}$, we associate IDs of $\mathcal{A}$ as $d_n(p_{n-1}, \theta_{n-1}, d_{n-1}) \cdots (p_1, \theta_1, d_1)$ such that $(p_{i+1}, \theta_{i+1}, d_i) \vdash_{\mathcal{A}} (p_i, \theta_i, \varepsilon)$ holds for each $i \in [n-1]$. For a technical reason, (p_n, θ_n) is stored in additonal registers of $\mathcal{P}'$. More accurately, we introduce the encoding table θ_Q that keeps distinct data values $d_{p_1}, \dots, d_{p_{|Q|}}$ and we store the constant data value d_{p_i} instead of the state p_i of $\mathcal{A}$ to registers and stack of $\mathcal{P}'$ because we cannot store a finite symbol p_i without extending the definition of RPDS.

Proposition 6.2: For a given k-RPDS $\mathcal{P} = (P, \Delta)$, a regular valuation function $\Lambda : ID_{\mathcal{P}} \to \Sigma$ $(\Sigma = 2^{At})$, an ID $c \in P \times \Theta_k \times D$ whose stack is a single data value, and an LTL formula φ over At, we can construct an RPDS $\mathcal{P}' = (P', \Delta')$, a simple valuation function $\Lambda_s : ID_{\mathcal{P}'} \to \Sigma \cup \{\{\sqcup\}\}$, an ID $c' \in ID_{\mathcal{P}'}$ of $\mathcal{P}'$, and an LTL formula φ_s over $At \cup \{\sqcup\}$, where $\sqcup \notin At$, that satisfy the following conditions: for every run $\rho \in (ID_{\mathcal{P}})^\omega$ of $\mathcal{P}$ starting with c and every run $\rho' \in (ID_{\mathcal{P}'})^\omega$ of $\mathcal{P}'$ starting with c', $\Lambda(\rho) \models \varphi \iff \Lambda_s(\rho') \models \varphi_s$.

Proof For a given k-RPDS $\mathcal{P} = (P, \Delta)$ over D and a regular valuation function $\Lambda : ID_{\mathcal{P}} \to \Sigma$, there are backward-deterministic k-RAs $\mathcal{A}_A = (Q_A, P, (f_A, \vartheta_A), \delta_A)$ such that $A \in \Lambda(p, \theta, w)$ iff $(p, \theta, w) \in L(\mathcal{A}_A)$ for $A \in At$. We use a bold symbol to represent a tuple consisting of the corresponding symbols representing states, assignments, etc. of RA indexed by At. Let $\boldsymbol{Q} = \prod_{A\in At} Q_A$, $\boldsymbol{\Theta}_k = (\Theta_k)^{|At|}$, $\boldsymbol{f} = \prod_{A\in At} f_A$ and $\boldsymbol{\vartheta} = \prod_{A\in At} \vartheta_A$. For every bold symbol $\boldsymbol{B}$, let $\boldsymbol{B}_A$ be the component of the tuple $\boldsymbol{B}$ indexed by $A \in At$. For $\boldsymbol{p}, \boldsymbol{p}' \in \boldsymbol{Q}, \boldsymbol{\theta}, \boldsymbol{\theta}' \in \boldsymbol{\Theta}_k$, we write $(\boldsymbol{p}', \boldsymbol{\theta}', d) \vdash (\boldsymbol{p}, \boldsymbol{\theta}, \varepsilon)$ if $(\boldsymbol{p}'_A, \boldsymbol{\theta}'_A, d) \vdash (\boldsymbol{p}_A, \boldsymbol{\theta}_A, \varepsilon)$ for all $A \in At$. Note that $\boldsymbol{p}'$ and $\boldsymbol{\theta}'$ are uniquely determined by $\boldsymbol{p}, \boldsymbol{\theta}$ and d because every $\mathcal{A}_A$ is backward-deterministic.

For each word $d_{n-1} \cdots d_1 \in D^*$, there exist unique sequences $\boldsymbol{p}_n, \ldots, \boldsymbol{p}_1 \in \boldsymbol{Q}$ and $\boldsymbol{\theta}_n, \ldots, \boldsymbol{\theta}_1 \in \boldsymbol{\Theta}_k$ such that $\boldsymbol{p}_1 = \boldsymbol{f}$, $\boldsymbol{\theta}_1 = \boldsymbol{\vartheta}$, and $(\boldsymbol{p}_{i+1}, \boldsymbol{\theta}_{i+1}, d_i) \vdash (\boldsymbol{p}_i, \boldsymbol{\theta}_i, \varepsilon)$ for all $i \in [n-1]$. We call them the *consistent sequences* for any ID $(p, \theta, d_n d_{n-1} \cdots d_1) \in ID_{\mathcal{P}}$. By the definition of Λ and $\mathcal{A}_A$ for $A \in At$, $A \in \Lambda(p, \theta, d_n \cdots d_1)$ iff $(p, \theta, d_n \cdots d_1) \in L(\mathcal{A}_A)$ iff $(p, \theta, d_n) \vdash_{\mathcal{A}_A} ((\boldsymbol{p}_n)_A, (\boldsymbol{\theta}_n)_A, \varepsilon)$ holds where $\boldsymbol{p}_n$ and $\boldsymbol{\theta}_n$ are the first elements of the consistent sequences $\boldsymbol{p}_n, \ldots, \boldsymbol{p}_1$ and $\boldsymbol{\theta}_n, \ldots, \boldsymbol{\theta}_1$ of $(p, \theta, d_n \cdots d_1)$ and $(\boldsymbol{p}_n)_A$ and $(\boldsymbol{\theta}_n)_A$ are the components of $\boldsymbol{p}_n$ and $\boldsymbol{\theta}_n$ indexed by A, respectively.

Let $t = \sum_{A\in At} |Q_A|$ and $K = k + t + |At| + |At| \times k$. For $\theta_1 \in \Theta_{k_1}, \theta_2 \in \Theta_{k_2}$, let $\theta_1 \circ \theta_2 \in \Theta_{k_1+k_2}$ be the concatenation of assignments defined as $(\theta_1 \circ \theta_2)(i) = \theta_1(i)$ for $1 \le i \le k_1$ and $(\theta_1 \circ \theta_2)(i) = \theta_2(i - k_1)$ for $k_1 < i \le k_1 + k_2$, and $str : \Theta_k \to D^k$ be the decomposition of assignments that satisfies $str(\theta) = \theta(1) \cdots \theta(k)$ for $\theta \in \Theta_k$. We construct a K-RPDS $\mathcal{P}' = (P', \Delta')$ over D that simulates a transition of $\mathcal{P}$ and transitions of $\mathcal{A}_A$ for every $A \in At$. For an ID $c = (p, \theta, d_n \cdots d_1) \in ID_{\mathcal{P}}$ of $\mathcal{P}$, we define a corresponding ID $(p, \theta \circ \theta_{\boldsymbol{Q}} \circ \theta_{\boldsymbol{p}_n} \circ \boldsymbol{\theta}_n, d_n str(\theta_{\boldsymbol{p}_{n-1}}) str(\boldsymbol{\theta}_{n-1}) d_{n-1} \cdots str(\theta_{\boldsymbol{p}_1}) str(\boldsymbol{\theta}_1) d_1)$ of $\mathcal{P}'$, denoted by $cor(c)$, where

- $\boldsymbol{p}_n, \ldots, \boldsymbol{p}_1 \in \boldsymbol{Q}$ and $\boldsymbol{\theta}_n, \ldots, \boldsymbol{\theta}_1 \in \boldsymbol{\Theta}_k$ are the consistent sequences for $(p, \theta, d_n \cdots d_1)$.
- $\theta_{\boldsymbol{Q}} \in \boldsymbol{\Theta}_t$ is the *encoding table* of $\boldsymbol{Q}$ such that $\theta_{\boldsymbol{Q}}(i) \neq \theta_{\boldsymbol{Q}}(j)$ for all distinct $i, j \in [t]$. For each $q \in \boldsymbol{Q}$, we assign a unique index number $\gamma(q)$ between $k+1$ to $k+t$. (In the K-RPDS $\mathcal{P}'$, the $\gamma(q)$-th register is in the encoding table and its value is used for the code of q.)
- $\theta_{\boldsymbol{p}_n}, \ldots, \theta_{\boldsymbol{p}_1} \in \Theta_{|At|}$ are the *encoded states* of $\boldsymbol{p}_n, \ldots, \boldsymbol{p}_1$, respectively, such that $\theta_{\boldsymbol{p}_i}(j) = \theta_{\boldsymbol{Q}}(\gamma((\boldsymbol{p}_i)_{A_j}) - k)$ for $i \in [n]$ and $j \in [m]$ where $m = |At|$ and $At = \{A_1, \ldots, A_m\}$.

As the method of [16], we need to

For a fixed encoding table $\theta_{\boldsymbol{Q}}$, $cor : ID_{\mathcal{P}} \to ID_{\mathcal{P}'}$ defined above satisfies $cor(c) = cor(c')$ iff $c = c'$ for any $c, c' \in ID_{\mathcal{P}}$. For readability, we write $str(\theta_{\boldsymbol{p}}) str(\boldsymbol{\theta}) d \in D_{|At|+|At|\times k+1}$ as $(\boldsymbol{p}, \boldsymbol{\theta}, d)$.

As shown in the following claim, we will define Δ' so that the following conditions hold:

- Pop condition: $(p, \theta \circ \theta_{\boldsymbol{Q}} \circ \theta_{\boldsymbol{p}} \circ \boldsymbol{\theta}, d(\boldsymbol{p}', \boldsymbol{\theta}', d')) \Rightarrow^*_{\mathcal{P}'} (q, \theta' \circ \theta_{\boldsymbol{Q}} \circ \theta_{\boldsymbol{p}'} \circ \boldsymbol{\theta}', d')$ iff $(p, \theta, d) \Rightarrow_{\mathcal{P}} (q, \theta', \varepsilon)$
- Replace condition: $(p, \theta \circ \theta_{\boldsymbol{Q}} \circ \theta_{\boldsymbol{p}} \circ \boldsymbol{\theta}, d) \Rightarrow^*_{\mathcal{P}'} (q, \theta' \circ \theta_{\boldsymbol{Q}} \circ \theta_{\boldsymbol{p}} \circ \boldsymbol{\theta}, d_1)$ iff $(p, \theta, d) \Rightarrow_{\mathcal{P}} (q, \theta', d_1)$
- Push condition: $(p, \theta \circ \theta_{\boldsymbol{Q}} \circ \theta_{\boldsymbol{p}} \circ \boldsymbol{\theta}, d) \Rightarrow^*_{\mathcal{P}'} (q, \theta' \circ \theta_{\boldsymbol{Q}} \circ \theta_{\boldsymbol{p}'} \circ \boldsymbol{\theta}', d_2(\boldsymbol{p}, \boldsymbol{\theta}, d_1))$ iff $(p, \theta, d) \Rightarrow_{\mathcal{P}} (q, \theta', d_2 d_1)$ and $(\boldsymbol{p}', \boldsymbol{\theta}', d_1) \vdash (\boldsymbol{p}, \boldsymbol{\theta}, \varepsilon)$.

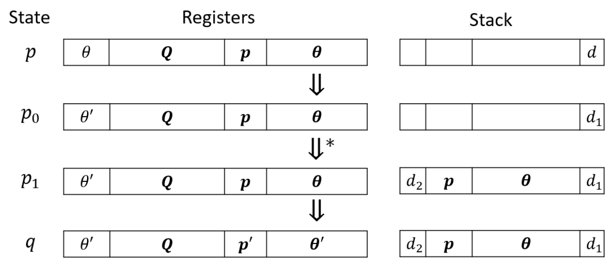


Fig. 6 An outline of transitions of the push condition

Claim 1. We can construct $\mathcal{P}'$ that satisfies pop, replace and push conditions.

(Proof of Claim 1) As mentioned in the above three conditions, an assignment for the K registers of $\mathcal{P}'$ can be decomposed as $\theta \circ \theta_{\boldsymbol{Q}} \circ \theta_{\boldsymbol{p}} \circ \boldsymbol{\theta}$, and the set of registers of $\mathcal{P}'$ consists of the following four parts: the first k registers simulate the registers of $\mathcal{P}$, the next t registers form the encoding table, the next $|At|$ registers simulate the states of $\mathcal{A}_A$ for all $A \in At$, and the final $k|At|$ registers simulate the registers of $\mathcal{A}_A$ for all $A \in At$. Let $s(A)$ and $r(A, l)$ be the indices of registers of $\mathcal{P}'$ that store the state and the contents of the l-th register of $\mathcal{A}_A$. That is, if the elements of At are ordered as $A_1, \ldots, A_{|At|}$, then $s(A_i) = k + t + i$ and $r(A_i, l) = k + t + |At| + (i-1)k + l$.

We first consider the push condition. For a push rule $r_1 = (p, \phi) \to (q, j_2 j_1) \in \Delta$, at first we add a new state p_0 to P' and add the following replace rule to Δ':

$$r'_1 = (p, \phi_1) \to (p_0, j_1)$$

for every $\phi_1 \in \Phi_{2k+1}$ that satisfies

$$\begin{aligned}
&\mathrm{x}_i \sim_{\phi_1} \mathrm{x}_j \text{ iff } \mathrm{x}_i \sim_{\phi} \mathrm{x}_j,\\
&\mathrm{x}_i \sim_{\phi_1} \mathrm{x}'_j \text{ iff } \mathrm{x}_i \sim_{\phi} \mathrm{x}'_j,\\
&\mathrm{x}'_i \sim_{\phi_1} \mathrm{x}'_j \text{ iff } \mathrm{x}'_i \sim_{\phi} \mathrm{x}'_j,\\
&top \sim_{\phi_1} \mathrm{x}_j \text{ iff } top \sim_{\phi} \mathrm{x}_j,\\
&top \sim_{\phi_1} \mathrm{x}'_j \text{ iff } top \sim_{\phi} \mathrm{x}'_j,\\
&\mathrm{x}_l \sim_{\phi_1} \mathrm{x}'_l, \text{ and}\\
&\mathrm{x}_{l_1} \sim_{\phi_1} \mathrm{x}_{l_2} \text{ iff } l_1 = l_2
\end{aligned}$$

for all $i, j \in [k]$, $k < l \le K$, and $k+1 \le l_1, l_2 \le k+t$. By definition, $(p, \theta, d) \Rightarrow^{r_1}_{d} (q, \theta', d_2 d_1)$ in $\mathcal{P}$ iff $(p, \theta \circ \theta_{\boldsymbol{Q}} \circ \theta_{\boldsymbol{p}} \circ \boldsymbol{\theta}, d) \Rightarrow^{r'_1}_{d} (p_0, \theta' \circ \theta_{\boldsymbol{Q}} \circ \theta_{\boldsymbol{p}} \circ \boldsymbol{\theta}, d_1)$ in $\mathcal{P}'$.

We construct $(k+1)|At| + 1$ push rules for transferring $(k+1)|At| + 1$ data values $\theta_{\boldsymbol{p}}$, $\boldsymbol{\theta}$ and d_2 to the stack. The construction is easy but tedious and we omit it here. Let $p_1 \in P'$ be the (newly introduced) state immediately after transitions by the omitted rules. By the rules, we obtain the transition $(p_0, \theta' \circ \theta_{\boldsymbol{Q}} \circ \theta_{\boldsymbol{p}} \circ \boldsymbol{\theta}, d_1) \Rightarrow^*_{\mathcal{P}'} (p_1, \theta' \circ \theta_{\boldsymbol{Q}} \circ \theta_{\boldsymbol{p}} \circ \boldsymbol{\theta}, d_2(\boldsymbol{p}, \boldsymbol{\theta}, d_1))$ of $\mathcal{P}'$.

Finally, we add replace rules to Δ' for simulating transitions of $\mathcal{A}_A$ for $A \in At$. For each $A \in At$, choose a transition rule $(p'_A, \phi_A) \to p_A \in \delta_A$ of $\mathcal{A}_A$. For every combination of these $|At|$ rules, we add the following replace rule to Δ':

$$r_1'' = (p_1, \phi_2) \to (q, j_2)$$

for every $\phi_2 \in \Phi_{2k+1}$ that satisfies

$$\begin{array}{rl}
\mathtt{x}_{s(A)} \sim_{\phi_2} \mathtt{x}_{\gamma(p_A)}, & \\
\mathtt{x}'_{s(A)} \sim_{\phi_2} \mathtt{x}_{\gamma(p'_A)}, & \\
\mathtt{x}'_{r(A,i)} \sim_{\phi_2} \mathtt{x}'_{r(A,j)} & \text{iff } \mathtt{x}_i \sim_{\phi_A} \mathtt{x}_j, \\
\mathtt{x}'_{r(A,i)} \sim_{\phi_2} \mathtt{x}_{r(A,j)} & \text{iff } \mathtt{x}_i \sim_{\phi_A} \mathtt{x}'_j, \\
\mathtt{x}_{r(A,i)} \sim_{\phi_2} \mathtt{x}_{r(A,j)} & \text{iff } \mathtt{x}'_i \sim_{\phi_A} \mathtt{x}'_j, \\
\mathtt{x}'_{r(A,i)} \sim_{\phi_2} \mathtt{x}_{j_2} & \text{iff } \mathtt{x}_i \sim_{\phi_A} top, \\
\mathtt{x}_{r(A,i)} \sim_{\phi_2} \mathtt{x}_{j_2} & \text{iff } \mathtt{x}'_i \sim_{\phi_A} top, \\
\mathtt{x}_l \sim_{\phi_2} \mathtt{x}'_l, & \text{and} \\
\mathtt{x}_{l_1} \sim_{\phi_2} \mathtt{x}_{l_2} & \text{iff } l_1 = l_2
\end{array}$$

for all $A \in At$, $i, j \in [k]$, $1 \le l \le k+t$, and $k+1 \le l_1, l_2 \le k+t$. By definition, $(p, \theta, d) \Rightarrow_d^{r_1} (q, \theta', d_2 d_1)$ in $\mathcal{P}$ for some θ and d and $(\boldsymbol{p}', \boldsymbol{\theta}', d_1) \vdash (\boldsymbol{p}, \boldsymbol{\theta}, \varepsilon)$ iff $(p_1, \theta' \circ \theta_{\boldsymbol{Q}} \circ \theta_{\boldsymbol{p}} \circ \boldsymbol{\theta}, d_2(\boldsymbol{p}, \boldsymbol{\theta}, d_1)) \Rightarrow_{d_2}^{r_1''} (q, \theta' \circ \theta_{\boldsymbol{Q}} \circ \theta_{\boldsymbol{p}'} \circ \boldsymbol{\theta}', d_2(\boldsymbol{p}, \boldsymbol{\theta}, d_1))$ in $\mathcal{P}'$. Thus, the claim holds for the push condition.

Replace condition can be realized by the rule $(p, \phi_1) \to (q, j_1)$. Pop condition can also be realized by the rule $(p, \phi_1) \to (p_0, \varepsilon)$ and $(k+1)|At|$ pop rules from p_0 to q for transferring topmost $(k+1)|At|$ data values $\theta_{\boldsymbol{p}}, \boldsymbol{\theta}$ to the $(k+t)$-th to K-th registers.

(end of the proof of the Claim 1)

Let $b = (k+1)|At| + 2$. By Claim 1, for every IDs $c, c' \in ID_{\mathcal{P}}$ such that $c \Rightarrow_{\mathcal{P}} c'$, $cor(c) \Rightarrow_{\mathcal{P}'}^{n} cor(c')$ holds for some $n \le b$. We can assume that $cor(c) \Rightarrow_{\mathcal{P}'}^{b} cor(c')$ if $c \Rightarrow_{\mathcal{P}} c'$ for $c, c' \in ID_{\mathcal{P}}$. If not, we can alter $\mathcal{P}'$ by inserting an appropriate number of dummy states and transitions for rules of replace and pop conditions. Then, for every run $\rho \in (ID_{\mathcal{P}})^\omega$ and a fixed encoding table, there exists a unique run $\rho' \in (ID_{\mathcal{P}'})^\omega$ such that $\rho'(bi) = cor(\rho(i))$ for every $i \in \mathbb{N}_0$.

We define a simple valuation Λ_s as satisfying

$$\begin{aligned}\Lambda_s^{-1}(\sigma) = \{&(p, \theta \circ \theta_{\boldsymbol{Q}} \circ \theta_{\boldsymbol{p}} \circ \boldsymbol{\theta}, dw) \mid d \in D, w \in D^*, \\ &((p, \theta, d) \vdash (\boldsymbol{p}_A, \boldsymbol{\theta}_A, \varepsilon) \text{ iff } A \in \sigma)\}\end{aligned}$$

for every $\sigma \in \Sigma$.

By the definition, $\Lambda_s(p', \theta, w) = \{\sqcup\}$ for all $p' \in P' \setminus P$. Also, $\Lambda(c) = \Lambda_s(cor(c))$ holds for $c \in ID_{\mathcal{P}}$. Thus, for every run $\rho \in (ID_{\mathcal{P}})^\omega$ and a fixed encoding table, there exists a unique run $\rho' \in (ID_{\mathcal{P}'})^\omega$ such that $\Lambda(\rho(i)) = \Lambda_s(\rho'(bi))$ for every $i \in \mathbb{N}_0$.

From a given regular valuation Λ, we have defined the simple valuation Λ_s such that $\Lambda(\rho(i)) = \Lambda_s(\rho'(bi))$ where the i-th element of ρ corresponds to the bi-th element of ρ' in terms of valuation. By this difference of positions over these runs, we cannot evaluate the LTL formula φ under $\Lambda_s(\rho')$ directly but apply a new LTL formula φ_s over $At \cup \{\sqcup\}$ to fill this gap of positions. From φ, we construct an LTL formula φ_s by replacing all $\mathcal{X}\phi$ appearing in φ to $\mathcal{X}^b\phi$ and $\phi\ \mathcal{U}\ \psi$ to $(\phi \vee \sqcup)\ \mathcal{U}\ (\psi \wedge \neg\sqcup)$ for all LTL formulae ϕ and ψ, where $\mathcal{X}^b$ is the concatenation of b numbers of $\mathcal{X}$. Then, $\Lambda(\rho) \models \varphi \Leftrightarrow \Lambda_s(\rho') \models \varphi_s$ holds for every run ρ of $\mathcal{P}$ and a corresponding run ρ' of $\mathcal{P}'$. $\square$

Theorem 6.2: The RPDS model checking problem of LTL with regular valuation is EXPTIME-complete.

7. Conclusion

We have defined Büchi RPDS and shown some properties of RPDS, RA and Büchi RPDS in Sect. 3. We have also defined simple and regular valuations for RPDS and shown EXPTIME-completeness of the LTL model checking for RPDS with simple valuation and regular valuations.

We have implemented a tool for reachability analysis of RPDS. Implementation of the other parts and application to practical problems are left as future work. We are also interested in an extension of the model checking problem in terms of specification logic. We are considering a class of temporal logic which can directly deal with data values but has good properties on computational complexity of its model checking problems.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Numbers 21J14332, JP19H04083 and 20K20625.

References

[1] P.A. Abdulla, C. Aiswarya, and M.F. Atig, "Data multi-pushdown automata," CONCUR 2017, pp.38:1–38:7, 2017.

[2] P.A. Abdulla, M.F. Atig, G. Delzanno, and A. Podelski, "Push-down automata with gap-order constraints," FSEN 2013, pp.199–216, 2013.

[3] A. Bouajjani, S. Fratani, and S. Qadeer, "Context-bounded analysis of multithreaded programs with dynamic linked structures," CAV 2007, pp.207–220, 2007.

[4] M. Bojańczyk, B. Klin, and S. Lasota, "Automata theory in nominal sets," Logical Methods in Computer Science, vol.10, no.3:4, pp.1–44, Aug. 2014.

[5] B. Bollig, A. Cyriac, P. Gastin, and K.N. Kumar, "Model checking languages of data words," FoSSaCS 2012, LNCS, vol.7213, pp.391–405, 2012.

[6] A. Bouajjani, J. Esparza, and O. Maler, "Reachability analysis of pushdown automata: Application to model-checking," CONCUR 1997, LNCS, vol.1243, pp.135–150, 1997.

[7] P. Bouyer, "A logical characterization of data languages," Inf. Process. Lett. vol.84, no.2, pp.75–85, Oct. 2002.

[8] Y-F. Chen, O. Lengál, T. Tan, and Z. Wu, "Register automata with linear arithmetic," 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, 2017.

[9] E.Y.C. Cheng and M. Kaminski, "Context-free languages over infinite alphabets," Acta Informatica 35, pp.245–267, 1998.

[10] E.M. Clarke, O. Grumberg, and D.A. Peled, Model checking, MIT Press, 2001.

[11] L. Clemente and S Lasota, "Reachability analysis of first-order definable pushdown systems," CSL 2015, pp.244–259, 2015.

[12] L. D'Antoni, T. Ferreira, M. Sammartino, and A. Silva, "Symbolic register automata," 31th Int. Conf. Computer Aided Verification, LNCS, vol.11561, 2019.

[13] S. Demri and R. Lazić, "LTL with the freeze quantifier and register automata," ACM TOCS, vol.10, no.3, pp.1–30, April 2009.
[14] S. Demri, R. Lazić and D. Nowak, "On the freeze quantifier in constraint LTL: Decidability and complexity," Inf. Comput. vol.205, no.1, pp.2–24, Jan. 2007.
[15] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon, "Efficient algorithms for model checking pushdown systems," CAV 2000, LNCS, vol.1855, pp.232–247, 2000.
[16] J. Esparza, A. Kučera, and S. Schwoon, "Model checking LTL with regular valuations for pushdown systems," Inform. and Comput, vol.186, no.2, pp.355–376, Nov. 2003.
[17] S.A. Greibach, "A note on pushdown store automata and regular systems," Proc. the American Mathematical Society, vol.18, pp.263–268, 1967.
[18] M. Kaminski and N. Francez, "Finite-memory automata," TCS, vol.134, pp.322–363, Nov. 1994.
[19] L. Libkin and D. Vrgoč, "Regular path queries on graphs with data," 15th Int. Conf. Database Theory, pp.74–85, March 2012.
[20] A.S. Murawski, S.J. Ramsay, and N. Tzevelekos, "Reachability in pushdown register automata," JCSS, vol.87, pp.58–83, Aug. 2017.
[21] F. Neven, T. Schwentick, and V. Vianu, "Finite state machines for strings over infinite alphabets," ACM Trans. Computational Logic vol.5, no.3, pp.403–435, July 2004.
[22] N. Nitta, Y. Takata, and H. Seki, "An efficient security verification method for programs with stack inspection," ACM CCS, pp.68–77, Nov. 2001.
[23] J. Rot, F. de Boer, and M. Bonsangue, "Pushdown system representation for unbounded object creation," Tech. Rep. KIT-13, Karlsruhe Institute of Technology, 38–52, 2010.
[24] R. Senda, Y. Takata, and H. Seki, "Complexity results on register context-free grammars and register tree automata," ICTAC 2018, LNCS, vol.11187, pp.415–434, 2018.
[25] R. Senda, Y. Takata, and H. Seki, "Generalized register context-free grammars," LATA 2019, LNCS, vol.11417, pp.259–271, revised version: IEICE Trans. Inf. & Syst., vol.E103-D, no.3, pp.540–548, March 2020.
[26] R. Senda, Y. Takata, and H. Seki, "Forward regularity preservation property of register pushdown systems," IEICE Trans. Inf. & Syst., vol.E104-D, no.3, pp.370–380, March 2021 (to appear).
[27] F. Song and T. Touili, "Pushdown model checking for malware detection," TACAS 2012, extended version, IJSTTT, vol.16, pp.147–173, 2014.
[28] I. Walukiewicz, "Pushdown processes: Games and model checking," CAV 1996, LNCS, vol.1102, pp.62–74, 1996.
[29] C. Xiaojuan and M. Ogawa, "Well-structured extensions of pushdown systems," CONCUR 2013, pp.121–136, 2013.

Appendix: Omitted Proofs

Before giving omitted proofs, we introduce the following lemma, which is given by the definition of equivalence class.

Lemma Appendix .1: Let $u_1, u_2, u_3, v_1, v_2, v_3 \in D^*$ be any words over D and $d, d' \in D$ be any data values.

(i) If $\langle u_1, u_2\rangle = \langle v_1, v_2\rangle$ and $|u_1| = |v_1|$, then for any $x \in D^*$, there exists $y \in D^*$ such that $|x| = |y|$ and $\langle u_1, x, u_2\rangle = \langle v_1, y, v_2\rangle$.

(ii) If $\langle u_1, u_2, u_3\rangle = \langle v_1, v_2, v_3\rangle$, $|u_1| = |v_1|$ and $|u_2| = |v_2|$, then $\langle u_1, u_3\rangle = \langle v_1, v_3\rangle$.

(iii) If $\langle u_1, d, u_2, u_3\rangle = \langle v_1, d', v_2, v_3\rangle$, $|u_1| = |v_1|$ and $|u_2| = |v_2|$, then $\langle u_1, d, u_2, d, u_3\rangle = \langle v_1, d', v_2, d', v_3\rangle$.

Also, we define $proj' : \Phi_{2k+1} \to \Phi_k$ as $proj'(\langle\theta, \theta', d\rangle) = \langle\theta'\rangle$ for $\theta, \theta' \in \Theta_k$ and $d \in D$.

We show all of omitted proofs below.

Lemma 3.1: For a k-RPDS $\mathcal{P} = (P, \Delta)$, IDs (p, θ_0, u_0), (p, θ_1, u_1) of $\mathcal{P}$ such that $\langle\theta_0, u_0\rangle = \langle\theta_1, u_1\rangle$, and $q \in P$, if there exists an ID $(q, \theta_2, u_2) \in ID_{\mathcal{P}}$ that satisfies $(p, \theta_0, u_0) \Rightarrow^n (q, \theta_2, u_2)$ for some $n \in \mathbb{N}_0$, then there exists an ID $(q, \theta_3, u_3) \in ID_{\mathcal{P}}$ that satisfies $(p, \theta_1, u_1) \Rightarrow^n (q, \theta_3, u_3)$ and $\langle\theta_2, u_2\rangle = \langle\theta_3, u_3\rangle$.

Proof. We prove the lemma by the induction on n. The case of $n = 0$ is obvious.

Assume that $n > 0$. Then the transition sequence must be $(p, \theta_0, u_0) \Rightarrow^{n-1} (q', \theta_4, u_4) \Rightarrow^r_d (q, \theta_2, u_2)$ for some $(q', \theta_4, u_4) \in ID_{\mathcal{P}}$, $r = (q', \phi) \to (q, J) \in \Delta$ and $d \in D$, and $u_4 = du'_4$ for some $u'_4 \in D^*$ and $\langle\theta_4, \theta_2, d\rangle = \phi$. By the induction hypothesis, there is an ID $(q', \theta_5, u_5) \in ID_{\mathcal{P}}$ such that $(p, \theta_1, u_1) \Rightarrow^{n-1} (q', \theta_5, u_5)$ and $\langle\theta_4, u_4\rangle = \langle\theta_5, u_5\rangle$. Since $\langle\theta_4, u_4\rangle = \langle\theta_5, u_5\rangle$ implies $|u_4| = |u_5|$ and since $u_4 = du'_4$, it holds that $u_5 = d'u'_5$ for some $d' \in D$ and $u'_5 \in D^*$. By Lemma Appendix .1 (i), $\langle\theta_4, du'_4\rangle = \langle\theta_5, d'u'_5\rangle$ implies that there exists $\theta_3 \in \Theta_k$ such that $\langle\theta_4, \theta_2, du'_4\rangle = \langle\theta_5, \theta_3, d'u'_5\rangle$. Hence, $\langle\theta_4, \theta_2, d\rangle = \langle\theta_5, \theta_3, d'\rangle = \phi$ by Lemma Appendix .1 (ii), and thus $(q', \theta_5, u_5) \Rightarrow^r_{d'} (q, \theta_3, u_3)$ where $u_3 = \theta_3(j_1)\theta_3(j_2)u'_5$ if $J = j_1j_2$, $u_3 = \theta_3(j_1)u'_5$ if $J = j_1$, and $u_3 = u'_5$ if $J = \varepsilon$. Note that $\langle\theta_4, \theta_2, du'_4\rangle = \langle\theta_5, \theta_3, d'u'_5\rangle$ also implies $\langle\theta_2, u'_4\rangle = \langle\theta_3, u'_5\rangle$ by Lemma Appendix .1 (ii). If $J = j_1j_2$, then $u_2 = \theta_2(j_1)\theta_2(j_2)u'_4$, and thus $\langle\theta_2, u_2\rangle = \langle\theta_3, u_3\rangle$ holds because $\langle\theta_2, u'_4\rangle = \langle\theta_3, u'_5\rangle$ implies $\langle\theta_2, \theta_2(j_1)\theta_2(j_2)u'_4\rangle = \langle\theta_3, \theta_3(j_1)\theta_3(j_2)u'_5\rangle$ by applying Lemma Appendix .1 (iii) twice. In the same way, we can show $\langle\theta_2, u_2\rangle = \langle\theta_3, u_3\rangle$ in the cases of $J = j_1$ and $J = \varepsilon$. □

Lemma 3.2: Let $\mathcal{A} = (Q, I, \xi, \delta)$ be a k-RA and $(p_0, \theta_0, w_0) \vdash (p_1, \theta_1, w_1) \vdash \cdots \vdash (p_n, \theta_n, w_n)$ be a run of $\mathcal{A}$, where $w_0 = d_1d_2\cdots d_m$. Let $D_{w_0} = \{d_1, \ldots, d_m\}$ and let $D' \subseteq D$ be a finite subset of D such that $D_{w_0} \subseteq D'$ and $|D' \setminus D_{w_0}| \geq 2k$. Let $\Theta'_k \subseteq \Theta_k$ be the set of assignments whose range is D', and let $\theta'_0 \in \Theta'_k$ be an assignment such that $\langle\theta_0, w_0\rangle = \langle\theta'_0, w_0\rangle$. Then, there exist $\theta'_1, \ldots, \theta'_n \in \Theta'_k$ such that $(p_0, \theta'_0, w_0) \vdash (p_1, \theta'_1, w_1) \vdash \cdots \vdash (p_n, \theta'_n, w_n)$ and $\langle\theta_j, w_j\rangle = \langle\theta'_j, w_j\rangle$ for each $j \in [n]$.

Proof. We prove the lemma by the induction on n. The case of $n = 0$ is obvious.

Assume that $n > 0$. Then, $(p_0, \theta_0, w_0) \vdash^r_{d_1} (p_1, \theta_1, w_1)$ for some $r = (p_0, \phi) \to p_1 \in \delta$ and $\langle\theta_0, \theta_1, d_1\rangle = \phi$. There exists $\theta'_1 \in \Theta'_k$ that satisfies $\langle\theta_0, \theta_1, w_0\rangle = \langle\theta'_0, \theta'_1, w_0\rangle$, because $\langle\theta_0, w_0\rangle = \langle\theta'_0, w_0\rangle$ implies $\langle\theta_0, \theta_1, w_0\rangle = \langle\theta'_0, \theta', w_0\rangle$ for some $\theta' \in \Theta_k$ by Lemma Appendix .1 (i), and we can replace every $\theta'(i) \notin D'$ with an element in $D' \setminus (D_{w_0} \cup \{\theta'_0(1), \ldots, \theta'_0(k)\})$ without changing the equivalence class.

By Lemma Appendix .1 (ii), $\langle\theta_0, \theta_1, w_0\rangle = \langle\theta'_0, \theta'_1, w_0\rangle$ implies $\langle\theta_0, \theta_1, d_1\rangle = \langle\theta'_0, \theta'_1, d_1\rangle = \phi$ and $\langle\theta_1, w_1\rangle = \langle\theta'_1, w_1\rangle$, and the former implies $(p_0, \theta'_0, w_0) \vdash^r_{d_1} (p_1, \theta'_1, w_1)$. By the induction hypothesis, the lemma holds. □

Proposition 3.1: For a k-RA $\mathcal{A} = (Q, I, \xi, \delta)$, a finite subset D' of D, $p \in I$, and $\theta \in \Theta_k$ such that $\theta(i) \in D'$ for $i \in [k]$, we can construct a nondeterministic finite automaton (NFA) $\mathcal{A}^p_\theta$ over D' such that for any $w \in (D')^*$, $w \in L(\mathcal{A}^p_\theta)$ iff $(p, \theta, w) \in L(\mathcal{A})$. Moreover, the number of states of $\mathcal{A}^p_\theta$ is at most $|Q| \cdot (2k + |D'|)^k$.

Proof. First, we construct a finite set D'' such that $D' \subseteq D'' \subset D$ and $|D''| = |D'|+2k$, by adding arbitrary $2k$ elements of $D \backslash D'$ to D'. Let $\Theta'_k \subseteq \Theta_k$ be the set of assignments whose range is D''. Then, we construct an NFA $\mathcal{A}^p_\theta$ as follows:

- The set of states is $Q \times \Theta'_k$.
- The initial state is (p, θ).
- The set of final states is $\{(q, \theta_f) \mid (q, \langle\theta_f\rangle) \in \xi\}$.
- The set of transition rules is $\{((q_1, \theta_1), d) \rightarrow (q_2, \theta_2) \mid (q_1, \langle\theta_1, \theta_2, d\rangle) \rightarrow q_2 \in \delta\}$.

By the construction, if $((p, \theta), w) \vdash^*_{\mathcal{A}^p_\theta} ((q, \theta'), w')$, then $(p, \theta, w) \vdash^*_{\mathcal{A}} (q, \theta', w')$, obviously. By Lemma 3.2, if $(p, \theta, w) \vdash^*_{\mathcal{A}} (q, \theta', w')$ for $w \in (D')^*$, then $((p, \theta), w) \vdash^*_{\mathcal{A}^p_\theta} ((q, \theta''), w')$ for some $\theta'' \in \Theta'_k$ such that $\langle\theta'\rangle = \langle\theta''\rangle$. Thus for any $w \in (D')^*$, $w \in L(\mathcal{A}^p_\theta)$ iff $(p, \theta, w) \in L(\mathcal{A})$. The number of states of $\mathcal{A}^p_\theta$ is $|Q| \cdot |D''|^k = |Q| \cdot (2k + |D'|)^k$. □

Proposition 3.2: For a k-BRPDS $\mathcal{BP} = (P, \Delta, G)$ and an ID (q_0, θ, w) of $\mathcal{BP}$, (q_0, θ, w) is accepted by $\mathcal{BP}$ if and only if there exist IDs (p, θ_0, d_0), (g, θ'_0, u_0), $(p, \theta_1, d_1 v_1) \in ID_{\mathcal{BP}}$ where $g \in G$ and $d_0, d_1 \in D$ and $u_0, v_1 \in D^*$ such that the following conditions hold:

(1) $(q_0, \theta, w) \Rightarrow^* (p, \theta_0, d_0 w')$ for some $w' \in D^*$,

(2) $(p, \theta_0, d_0) \Rightarrow^+ (g, \theta'_0, u_0) \Rightarrow^* (p, \theta_1, d_1 v_1)$, and

(3) $\langle\theta_0, d_0\rangle = \langle\theta_1, d_1\rangle$.

Rest of the proof.
(If part) Since $\langle\theta_0, d_0\rangle = \langle\theta_1, d_1\rangle$ by condition (3), by applying Lemma 3.1 to condition (2), there exist $\theta'_1, \theta_2 \in \Theta_k$, $u_1, v_2 \in D^*$ and $d_2 \in D$ such that $(p, \theta_1, d_1) \Rightarrow^+ (g, \theta'_1, u_1) \Rightarrow^* (p, \theta_2, d_2 v_2)$ and $\langle\theta'_0, u_0\rangle = \langle\theta'_1, u_1\rangle$ and $\langle\theta_1, d_1 v_1\rangle = \langle\theta_2, d_2 v_2\rangle$. By Lemma Appendix .1 (ii), $\langle\theta_1, d_1 v_1\rangle = \langle\theta_2, d_2 v_2\rangle$ implies $\langle\theta_1, d_1\rangle = \langle\theta_2, d_2\rangle$. Repeating this selection of IDs, we obtain $\theta'_i, \theta_{i+1} \in \Theta_k$, $u_i, v_{i+1} \in D^*$ and $d_{i+1} \in D$ for $i \geq 1$ such that $(p, \theta_i, d_i) \Rightarrow^+ (g, \theta'_i, u_i) \Rightarrow^* (p, \theta_{i+1}, d_{i+1} v_{i+1})$. Concatenating these transition sequences, we obtain a run $(q_0, \theta, w) \Rightarrow^* (p, \theta_0, d_0 w') \Rightarrow^+ (g, \theta'_0, u_0 w') \Rightarrow^* (p, \theta_1, d_1 v_1 w') \Rightarrow^+ (g, \theta'_1, u_1 v_1 w') \Rightarrow^* (p, \theta_2, d_2 v_2 v_1 w') \Rightarrow^+ \cdots$, where g appears infinitely many times. Therefore this run is an accepting run of $\mathcal{BP}$, and thus $\mathcal{BP}$ accepts (q_0, θ, w). □

Proposition 3.3: For a k-BRPDS $\mathcal{BP} = (P, \Delta, G)$ and an ID (q_0, θ, w) of $\mathcal{BP}$, it is decidable whether (q_0, θ, w) is accepted by $\mathcal{BP}$ in $O((|\Delta| + |w|^{O(k)}) \cdot |P|^{O(1)} \cdot 2^{O(k^2)})$ time.

Proof. We investigate the complexity of the algorithm described in Sect. 3.5 that decides whether $(q_0, \theta, w) \in pre^*_{\mathcal{BP}}(L^p_\phi)$ and $(p, \theta_0, d_0) \in pre^+_{\mathcal{BP}}((G \times \Theta_k \times D^*) \cap pre^*_{\mathcal{BP}}(L^p_\phi))$ for some $(p, \theta_0, d_0) \in L^p_{\phi,1}$ for some $p \in P$ and $\phi \in \Phi_{k+1}$. The algorithm checks these conditions for every $p \in P$ and $\phi \in \Phi_{k+1}$. Below we consider the complexity for a fixed pair of p and ϕ.

As mentioned above, we can construct an RA $\mathcal{A}^p_\phi$ that recognizes L^p_ϕ. Note that the set of states of $\mathcal{A}^p_\phi$ is $P \cup \{q_f\}$ where q_f is a new state.

From $\mathcal{A}^p_\phi$, we construct RAs that recognize $pre^*_{\mathcal{BP}}(L^p_\phi)$ and $pre^+_{\mathcal{BP}}((G \times \Theta_k \times D^*) \cap pre^*_{\mathcal{BP}}(L^p_\phi))$, respectively. The algorithm given in [20] constructs an RA $\widetilde{\mathcal{A}}$ such that $L(\widetilde{\mathcal{A}}) = pre^*_{\mathcal{BP}}(L(\mathcal{A}))$ for a given RA $\mathcal{A}$ whose set of initial states equals the set P of states of $\mathcal{BP}$. This construction does not increase the number of states of RA, and the construction takes $O(|\Delta| \cdot |Q|^{O(1)} \cdot |\Phi_{2k+1}|^{O(1)}) = O(|\Delta| \cdot |Q|^{O(1)} \cdot 2^{O(k^2)})$ time, where Q is the set of states of $\mathcal{A}$. An RA recognizing $pre^+_{\mathcal{BP}}(L(\mathcal{A}))$ can be constructed in a similar way. An RA $\mathcal{A}'$ such that $L(\mathcal{A}') = (G \times \Theta_k \times D^*) \cap L(\mathcal{A})$ for a given $\mathcal{A}$ (whose set of initial states equals P) can be obtained simply by removing all transition rules whose left-hand side contains a state in $P \setminus G$. (Note that by the definition of RA, every state in the set P of initial states does not appear in the right-hand side of any transition rule.) Hence, the construction of RAs that recognize $pre^*_{\mathcal{BP}}(L^p_\phi)$ and $pre^+_{\mathcal{BP}}((G \times \Theta_k \times D^*) \cap pre^*_{\mathcal{BP}}(L^p_\phi))$, respectively, can be performed in $O(|\Delta| \cdot |P|^{O(1)} \cdot 2^{O(k^2)})$ time. The set of states of each obtained RA is the same as $\mathcal{A}^p_\phi$.

In the final step, we check the membership $(q_0, \theta, w) \in pre^*_{\mathcal{BP}}(L^p_\phi)$ for given (q_0, θ, w) and $(p, \theta_0, d_0) \in pre^+_{\mathcal{BP}}((G \times \Theta_k \times D^*) \cap pre^*(L^p_\phi))$ for an arbitrary ID $(p, \theta_0, d_0) \in L^p_{\phi,1}$. The latter problem, which is the membership problem of (p, θ_0, d_0) for a k-RA $\mathcal{A} = (P \cup \{q_f\}, P, \xi, \delta)$, can be reduced to a problem to find a transition rule $(p, \phi') \rightarrow q \in \delta$ such that $proj(\phi') = \phi$ and $(q, proj'(\phi')) \in \xi$. This can be performed in $O(|\delta| \cdot |\xi|) = O(|P|^3 \cdot |\Phi_{2k+1}| \cdot |\Phi_k|) = O(|P|^3 \cdot 2^{O(k^2)})$ time. To check $(q_0, \theta, w) \in pre^*_{\mathcal{BP}}(L^p_\phi)$, we let D' be the finite set that consists of $\theta(i)$ for $i \in [k]$ and every element of w, and then we construct an NFA $\mathcal{A}^{q_0}_\theta$ over D' that satisfies $w \in L(\mathcal{A}^{q_0}_\theta)$ iff $(q_0, \theta, w) \in pre^*_{\mathcal{BP}}(L^p_\phi)$, according to Proposition 3.1. Whether $w \in L(\mathcal{A})$ for an NFA $\mathcal{A}$ over Σ can be decided in $O(|w| \cdot |Q| \cdot |\delta|) = O(|w| \cdot |Q|^3 \cdot |\Sigma|)$ time where Q and δ are the sets of states and transition rules of $\mathcal{A}$, respectively. In the case of $\mathcal{A}^{q_0}_\theta$, the number of states is at most $(|P| + 1)(2k + |D'|)^k$, and thus this step can be performed in $O(|w| \cdot |P|^3 \cdot (2k + |D'|)^{O(k)} \cdot |D'|) = O(|w| \cdot |P|^3 \cdot (3k + |w|)^{O(k)}) = O(|w|^{O(k)} \cdot |P|^3 \cdot 2^{O(k \log k)})$ time.

Therefore the algorithm described in Sect. 3.5 for a fixed pair of p and ϕ is solvable in $O((|\Delta| + |w|^{O(k)}) \cdot |P|^{O(1)} \cdot 2^{O(k^2)})$ time. The complexity for all $p \in P$ and $\phi \in \Phi_{k+1}$ is also $O((|\Delta| + |w|^{O(k)}) \cdot |P|^{O(1)} \cdot 2^{O(k^2)})$ time. □

Proposition 6.1: Let $\mathcal{B} = (Q, q_0, F, \delta)$ be a Büchi automaton over Σ that satisfies $L(\mathcal{B}) = \{\alpha \in \Sigma^\omega \mid \alpha \models \neg\varphi\}$ for a given LTL formula φ over At. Let $\Lambda : ID_{\mathcal{P}} \rightarrow \Sigma$ be a simple valuation function. Also, let $\mathcal{BP} = (P \times Q, \Delta', P \times F)$ be the product k-BRPDS of a k-RPDS $\mathcal{P} = (P, \Delta)$ and $\mathcal{B}$ defined in Definition 4.1. Then, $\mathcal{BP}$ does not accept an ID $((p_0, q_0), \theta_0, u_0)$ if and only if $\Lambda(\rho) \models \varphi$ for all run $\rho \in (ID_{\mathcal{P}})^\omega$ such that $\rho(0) = (p_0, \theta_0, u_0)$.

Proof. By the definition of $\mathcal{B}$, $\Lambda(\rho) \models \varphi$ iff $\Lambda(\rho) \notin L(\mathcal{B})$ for any run ρ of $\mathcal{P}$. Thus the proposition can be rephrased as follows: $\mathcal{BP}$ accepts $((p_0, q_0), \theta_0, u_0)$ if and only if there is a run ρ of $\mathcal{P}$ such that $\Lambda(\rho) \in L(\mathcal{B})$ and $\rho(0) = (p_0, \theta_0, u_0)$.

Below we prove this statement.

(Only-if part) Assume that there is an accepting run $\rho' = ((p_0, q_0), \theta_0, u_0)\,((p_1, q_1), \theta_1, u_1) \cdots$ of $\mathcal{BP}$. By removing the elements of Q in ρ', we obtain a sequence $\rho = (p_0, \theta_0, u_0)\,(p_1, \theta_1, u_1), \ldots \in (ID_{\mathcal{P}})^{\omega}$. By the definition of Δ', ρ is a run of $\mathcal{P}$. Moreover, by the definition of Δ', $(q_i, \sigma_i) \to q_{i+1} \in \delta$ for $i \geq 0$ where $\sigma_i = \Lambda(p_i, \theta_i, u_i)$. (Note that because Λ is a simple valuation, $\Lambda(p_i, \theta_i, u_i) = \Lambda(p_i, \langle\theta_i, d_i\rangle)$ where d_i is the first element of u_i, and $\langle\theta_i, d_i\rangle$ should equal $proj(\phi)$ where ϕ is the label of the transition rule of $\mathcal{P}$ used in the transition $(p_i, \theta_i, u_i) \Rightarrow_{\mathcal{P}} (p_{i+1}, \theta_{i+1}, u_{i+1})$.) Because ρ' is an accepting run of $\mathcal{BP}$, there are infinitely many $q_i \in F$ in the sequence $q_0, q_1, \ldots$, and thus $\Lambda(\rho) = \sigma_0\sigma_1 \cdots \in L(\mathcal{B})$.

(If part) Assume that there is a run $\rho = (p_0, \theta_0, u_0)\,(p_1, \theta_1, u_1), \ldots$ of $\mathcal{P}$ such that $\Lambda(\rho) \in L(\mathcal{B})$. Let $\sigma_i = \Lambda(p_i, \theta_i, u_i)$ for $i \geq 0$. Because $\sigma_0\sigma_1 \cdots \in L(\mathcal{B})$, there is a sequence $q_0, q_1, \ldots$ over Q such that $(q_i, \sigma_i) \to q_{i+1} \in \delta$ for $i \geq 0$, and there are infinitely many $q_i \in F$ in this sequence. By the definition of Δ', there is a run $\rho' = ((p_0, q_0), \theta_0, u_0)\,((p_1, q_1), \theta_1, u_1), \ldots$ of $\mathcal{BP}$. Since $\exists^{\omega} i.\, q_i \in F$, ρ' is an accepting run of $\mathcal{BP}$. □

Theorem 6.1: The LTL model checking problem for RPDS with simple valuation is EXPTIME-complete.

Proof. Since the LTL model checking problem for PDS with simple valuation is EXPTIME-complete [6], it is enough to show EXPTIME solvability. Let $\mathcal{P} = (P, \Delta)$, φ, and $\Lambda : ID_{\mathcal{P}} \to \Sigma$ be a given k-RPDS, an LTL formula over At, and a simple valuation, respectively. Without loss of generality, we assume every $p \in P$ appears in some rule in Δ, and thus $|P| \leq 2|\Delta|$. We define the description length of $\mathcal{P}$ as $\|\mathcal{P}\| = |\Delta|(\log|P| + k^2)$, because each rule $(p, \phi) \to (q, J) \in \Delta$ where $J \in [k]^2 \cup [k] \cup \{\varepsilon\}$ can be described in $2\log|P| + \log|\Phi_{2k+1}| + \log(k^2 + k + 1) = O(\log|P| + k^2)$ bits. We define the length $|\varphi|$ of φ as the number of operators and atomic propositions appearing in φ. We assume that Λ is given as a list of $\Lambda(p, \phi) \in \Sigma$ for all $p \in P$ and $\phi \in \Phi_{k+1}$, and thus the description length of Λ is $\|\Lambda\| = |P| \cdot |\Phi_{k+1}| \cdot \log|\Sigma| = |P| \cdot 2^{O(k^2)} \cdot |At|$.

In the first step of the model-checing algorithm, we construct the Büchi automaton $\mathcal{B} = (Q, q_0, F, \delta)$ over $\Sigma = 2^{At}$ whose language is the set of infinite words that satisfy the negation of φ. In the standard way of the construction of $\mathcal{B}$, $|Q| \leq 2^{O(|\varphi|)}$ and $|\delta| \leq 2^{O(|\varphi|)}$, and the construction takes at most $2^{O(|\varphi|)}$ time [10].

In the second step, we construct the product k-BRPDS $\mathcal{BP} = (P \times Q, \Delta', P \times F)$. By the definition of Δ', each rule $((p, q), \phi) \to ((p', q'), J) \in \Delta'$ can be regarded as a pair of rules $(p, \phi) \to (p', J) \in \Delta$ and $(q, \sigma) \to q' \in \delta$ such that $\sigma = \Lambda(p, proj(\phi))$. Thus, $|\Delta'| \leq |\Delta| \cdot |\delta|$. It takes at most $O(|P| \cdot |\Phi_{k+1}|)$ time to check whether $\sigma = \Lambda(p, proj(\phi))$, if Λ is given as a list of $\Lambda(p', \phi')$ for all $p' \in P$ and $\phi' \in \Phi_{k+1}$. Thus, Δ' can be constructed in $O(|P| \cdot |\Phi_{k+1}| \cdot |\Delta| \cdot |\delta|) = O(|P| \cdot |\Delta| \cdot 2^{O(k^2 + |\varphi|)})$ time.

In the final step, we solve the acceptance problem of $((p_0, q_0), \theta_0, u_0)$ for $\mathcal{BP}$. By Proposition 3.2, it takes $O((|\Delta'| + |u_0|^{O(k)}) \cdot |P \times Q|^{O(1)} \cdot 2^{O(k^2)}) = O((|\Delta| + |u_0|^{O(k)}) \cdot |P|^{O(1)} \cdot 2^{O(k^2 + |\varphi|)})$ time. □

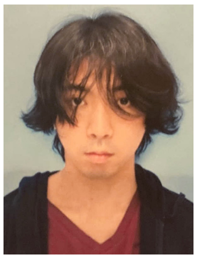
Theorem 6.2: The RPDS model checking problem of LTL with regular valuation is EXPTIME-complete.

Proof. By Proposition 6.2, a model checking problem for a k-RPDS $\mathcal{P} = (P, \Delta)$, an LTL formula φ and a regular valuation $\Lambda : ID_{\mathcal{P}} \to \Sigma$ can be reduced to a model checking problem for a K-RPDS $\mathcal{P}' = (P', \Delta')$ over $\Sigma \cup \{\{\sqcup\}\}$, the LTL formula φ_s over $At \cup \{\sqcup\}$ and the simple valuation $\Lambda_s : ID_{\mathcal{P}'} \to \Sigma \cup \{\{\sqcup\}\}$, which is also written as $\Lambda_s : P' \times \Phi_K \to \Sigma \cup \{\{\sqcup\}\}$. For K and b appearing in Proposition 6.2, $K = k + \sum_{A \in At} |Q_A| + |At| + |At| \cdot k = O(|\Lambda| + k \cdot |\varphi|)$ and $b = (1 + k)|At| + 2 = O(k \cdot |\varphi|)$ hold. By the construction of φ_s, $|\varphi_s| \leq |\varphi| \times b = O(k \cdot |\varphi|^2)$ holds.

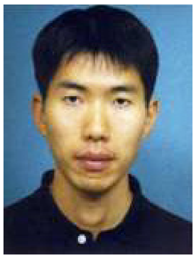
By Theorem 6.1, a model checking for simple valuation with the inputs $\mathcal{P}' = (P', \Delta')$, Λ_s and φ_s can be solved in exponential to K and $|\varphi_s|$, and polynomial to $|P'|$ and $|\Delta'|$. Because both K and $|\varphi_s|$ are polynomial to k, $|\Lambda|$ and $|\varphi|$, it is enough to prove $|P'|$ and $|\Delta'|$ are at most single-exponential to the input size $k + |\Lambda| + |\varphi|$.

For constructing each rules for the push, replace and pop in Proposition 6.2, we add $b - 1 = O(k \cdot |\varphi|)$ states to Δ'. Thus, $|P'| = O(k \cdot |\varphi| \cdot |\Delta|)$ holds. By the definition of Δ', $|\Delta'| \leq |P'| \cdot |\Phi_K| \cdot |P'| \cdot 3 = O((k \cdot |\varphi| \cdot |\Delta|)^2 \cdot 2^{O((|\Lambda| + k \cdot |\varphi|)^2)})$. Hence, $|P'|$ and $|\Delta'|$ are exponential to the input size $k + |\Lambda| + |\varphi|$, and thus the EXPTIME solvability holds.

Since the problem for PDS is EXPTIME-complete [6] we obtain the EXPTIME-hardness of the problem. □



Ryoma Senda received his Master's degree from Nagoya University in 2020. He has pursued Doctor's degree in the university. His research interests include formal language theory and its application to software engineering.



Yoshiaki Takata received the Ph.D. degree in information and computer science from Osaka University in 1997. He was with Nara Institute of Science and Technology as an Assistant Professor in 1997–2007. In 2007, he joined the faculty of Kochi University of Technology, where he has been an Associate Professor since 2009. His current research interests include formal specification and verification of software systems.

Hiroyuki Seki received his Ph.D. degree from Osaka University in 1987. He was an Assistant Professor, and later, an Associate Professor in Osaka University from 1987 to 1994. In 1994, he joined Nara Institute of Science and Technology, where he was a Professor during 1996 to 2013. Currently, he is a Professor in Nagoya University. His current research interests include formal language theory and formal approach to software development.