Complexity Results on Register Pushdown Automata

Ryoma Senda
ryoma.private@sqlab.jp
Graduate School of Informatics,
Nagoya University
Nagoya, Japan

Yoshiaki Takata
takata.yoshiaki@kochi-tech.ac.jp
Graduate School of Engineering,
Kochi University of Technology
Kochi, Japan

Hiroyuki Seki
seki@i.nagoya-u.ac.jp
Graduate School of Informatics,
Nagoya University
Nagoya, Japan

ABSTRACT

Register pushdown automata (RPDA) is an extension of classical pushdown automata to handle data values in a restricted way. RPDA attracts attention as a model of a query language for structured documents with data values. The membership and emptiness problems for RPDA are known to be EXPTIME-complete. This paper shows the membership problem becomes PSPACE-complete and NP-complete for nondecreasing and growing RPDA, respectively, while the emptiness problem remains EXPTIME-complete for these subclasses.

CCS CONCEPTS

• **Theory of computation → Automata over infinite objects**; **Grammars and context-free languages**.

KEYWORDS

register pushdown automaton, register context-free grammar, computational complexity, data word, data language

1 INTRODUCTION

There are many computational models having mild power of processing data values, including extensions of finite automata [4, 12, 16, 18], first-order and monadic second-order logics with data equality [2, 3] and linear temporal logics with freeze quantifier [6, 7]. Among them, register automata (abbreviated as RA) [12] is a natural extension of finite automata defined by incorporating registers as well as the equality test between an input data value and the data value kept in a register. Recently, attention has been paid to RA as a computational model of a query language for structured documents such as XML because a query on a document can be specified as the combination of a regular pattern and a condition on data values [15, 17]. For query processing and optimization, the decidability (hopefully in polynomial time) of basic properties of queries is desirable. The most basic problem is the membership problem that asks for a given query q and an element e in a document whether e is in the answer set of q. The satisfiability or (non)emptiness problem asking whether the answer set of a given query is nonempty is also important because if the answer set is empty, the query can be regarded as redundant or meaningless when query optimization is performed. The membership and emptiness problems for RA were shown to be decidable [12] and their computational complexities were also analyzed [6, 19].

While RA have the power of expressing regular patterns on *paths* in a document, it cannot represent patterns over branching paths that can be represented by some query languages such as XPath. Register context-free grammars (RCFG) and register pushdown automata (RPDA) were proposed in [5] as extensions of classical

context-free grammars (CFG) and pushdown automata (PDA), respectively, in a similar way to extending FA to RA. Note that there are related but different extensions, tree automata with data, to deal with patterns over branching paths and data values [8–11, 13, 20]. In [5], properties of RCFG and RPDA were shown including the equivalence in language representation power of RCFG and RPDA, the decidability of the membership and emptiness problems, and the closure properties. In [21], the computational complexity of the above decision problems for RCFG was investigated. In [22], the complexity of these problems for RPDA was also investigated but the analysis for subclasses of RPDA was not enough.

In this paper, we discuss the computational complexity of the decision problems for some subclasses of RPDA. A k-RPDA has a finite-state control, k registers and a pushdown stack (or stack in short) where each cell of the stack can store a data value. A transition of k-RPDA is either a pop, replace or push transition. We introduce subclasses of RPDA called non-decreasing RPDA and growing RPDA and investigate the computational complexity of the membership and emptiness problems for these subclasses.

The main results of [21, 22] and this paper are summarized in Table 1. The results for non-decreasing RPDA and growing RPDA are the contribution of this paper. Note that the complexity of the membership problems is in terms of both the size of a grammar or an automaton and the size of an input word (*combined complexity*). The complexity of the membership problem on the size of an input word only (*data complexity*) is in P for general RCFG and RPDA [21]. It is desirable that the data complexity is small while the combined complexity is rather a criterion of the expressive succinctness of the query language.

Table 1: Complexity results on RCFG and RPDA

	general RCFG / general RPDA	ε-rule free RCFG / non-decreasing RPDA	growing RCFG / growing RPDA
Membership	EXPTIMEc	PSPACEc	NPc
Emptiness	EXPTIMEc	EXPTIMEc	EXPTIMEc

'c' stands for 'complete.'
The results for RCFG and their subclasses were given in [21] and those for general RPDA were given in [22].

2 DEFINITIONS

A register pushdown automaton (abbreviated as RPDA) was originally defined in [5] as a pushdown automaton with registers over an infinite alphabet and the equivalence between RCFG and RPDA was shown in [5]. In this section, we define RPDA as a pushdown

automaton over the product of a finite alphabet and an infinite set of data values, following recent notions [16, 17]. Note that these differences are not essential.

2.1 Preliminaries

Let $\mathbb{N} = \{1, 2, \ldots\}$ and $\mathbb{N}_0 = \{0\} \cup \mathbb{N}$. We assume an infinite set D of data values as well as a finite alphabet Σ. For a given $k \in \mathbb{N}_0$ specifying the number of registers, a mapping $\theta : [k] \to D$ is called an assignment (of data values to k registers) where $[k] = \{1, 2, \ldots, k\}$. We assume that a data value $\perp \in D$ is designated as the initial value of a register and the initial bottom symbol in a stack. We denote by $\theta_\perp$ the register assignment that assigns the initial value $\perp$ to every register. Let Θ_k denote the class of assignments to k registers. For $\theta, \theta' \in \Theta_k$, we write $\theta' = \theta[i \leftarrow d]$ if $\theta'(i) = d$ and $\theta'(j) = \theta(j)$ $(j \neq i)$.

Let F_k denote the set of guard expressions over k registers defined by the following syntax rules:

$$\psi := \text{tt} \mid x_i^= \mid x_{top}^= \mid \psi \vee \psi \mid \neg\psi$$

where $x_i \in \{x_1, \ldots, x_k\}$.

For $d, e \in D$ and $\theta \in \Theta_k$, the satisfaction of $\psi \in F_k$ by (θ, d, e) is recursively defined as follows. Intuitively, d is a current data value in the input, e is the data value at the stack top, θ is a current register assignment, $\theta, d, e \models x_i^=$ means that the data value assigned to the i-th register by θ is equal to the input data value d and $\theta, d, e \models x_{top}^=$ means that the input data value d equals to the data value e at the stack top.

- $\theta, d, e \models x_i^=$ iff $\theta(i) = d$
- $\theta, d, e \models x_{top}^=$ iff $d = e$

The other cases are defined in an ordinary way. In addition, let $\text{ff} = \neg\text{tt}$, $\psi_1 \wedge \psi_2 = \neg(\neg\psi_1 \vee \neg\psi_2)$, $x_i^{\neq} = \neg(x_i^=)$ and $x_{top}^{\neq} = \neg(x_{top}^=)$ for $\psi_1, \psi_2 \in F_k$ and $i \in [k]$.

For a finite alphabet Σ and a set D of data values disjoint from Σ, a *data word* over $\Sigma \times D$ is a finite sequence of elements of $\Sigma \times D$ and a subset of $(\Sigma \times D)^*$ is called a *data language* over $\Sigma \times D$. $|\beta|$ denotes the cardinality of β if β is a set and the length of β if β is a finite sequence.

2.2 Register pushdown automata

For $k \in \mathbb{N}_0$, a k-register pushdown automaton (k-RPDA) over a finite alphabet Σ and a set D of data values is a tuple $\mathcal{A} = (Q, q_0, \delta)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state and δ is a finite set of transition rules having one of the following forms:

- $(p, \psi, i) \xrightarrow{a} (q, \varepsilon)$ (or $(p, \psi) \xrightarrow{a} (q, \varepsilon)$) (pop rule)
- $(p, \psi, i) \xrightarrow{a} (q, j_1)$ (or $(p, \psi) \xrightarrow{a} (q, j_1)$) (replace rule)
- $(p, \psi, i) \xrightarrow{a} (q, j_1 j_2)$ (or $(p, \psi) \xrightarrow{a} (q, j_1 j_2)$) (push rule)

where $p, q \in Q$, $a \in (\Sigma \cup \{\varepsilon\})$, $i, j_1, j_2 \in [k]$, and $\psi \in F_k$. A rule is called an ε-rule if $a = \varepsilon$.

D is used as a stack alphabet. For a state $q \in Q$, a register assignment $\theta \in \Theta_k$, a data word $w \in (\Sigma \times D)^*$, and a stack $u \in D^*$, (q, θ, w, u) is called an *instanteneous description (abbreviated as ID)* of k-RPDA $\mathcal{A}$ and $|u|$ is the stack height of this ID. For two IDs $(q, \theta, w, u), (q', \theta', w', u')$, we say that (q, θ, w, u) can transit to

(q', θ', w', u'), written as $(q, \theta, w, u) \Rightarrow_{\mathcal{A}} (q', \theta', w', u')$ if there exists a rule $(p, \psi, i) \xrightarrow{a} (q', J) \in \delta$ (resp. $(p, \psi) \xrightarrow{a} (q', J)$), data values $d, e \in D$ and $u'' \in D^*$ such that

$$\theta, d, e \models \psi, \ \theta' = \theta[i \leftarrow d] \ (\text{resp. } \theta' = \theta),$$

$$w = \begin{cases} (a, d)w' & a \in \Sigma, \text{ or} \\ w' & a = \varepsilon \end{cases}, u = eu'', \text{ and}$$

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

For an ε-rule, we can choose an arbitrary data value d that satisfies $\theta, d, e \models \psi$ and $\theta' = \theta[i \leftarrow d]$.

Let $\overset{*}{\Rightarrow}_{\mathcal{A}}$ be the reflexive transitive closure of $\Rightarrow_{\mathcal{A}}$. We abbreviate $\Rightarrow_{\mathcal{A}}$ and $\overset{*}{\Rightarrow}_{\mathcal{A}}$ as $\Rightarrow$ and $\overset{*}{\Rightarrow}$ if $\mathcal{A}$ is clear from the context.

For a k-RPDA $\mathcal{A} = (Q, q_0, \delta)$ and $w \in (\Sigma \times D)^*$, if $(q_0, \theta_\perp, w, \perp) \overset{*}{\Rightarrow} (q, \theta, \varepsilon, \varepsilon)$ for some $q \in Q$ and $\theta \in \Theta_k$, then we say $\mathcal{A}$ accepts w, $(q_0, \theta_\perp, w, \perp) \overset{*}{\Rightarrow} (q, \theta, \varepsilon, \varepsilon)$ is an *accepting run* of w in $\mathcal{A}$, and the number of the transitions in the run is called the *length* of the run. Let $L(\mathcal{A}) = \{w \in (\Sigma \times D)^* \mid \mathcal{A} \text{ accepts } w\}$, which is the language recognized by $\mathcal{A}$.

We say that an RPDA $\mathcal{A}$ is *non-decreasing* if every ε-rule of $\mathcal{A}$ is either a replace rule or a push rule. We say that an RPDA $\mathcal{A}$ is *growing* if every ε-rule of $\mathcal{A}$ is a push rule.

EXAMPLE 1. *Let $\mathcal{A}_1 = (Q, q_0, \delta)$ be the 2-RPDA, where*

- $Q = \{q_0, q_1, q_2, q_3\}$, and
- $\delta = \{(q_0, \text{tt}, 1) \xrightarrow{a} (q_1, 11), (q_1, x_1^{\neq}, 2) \xrightarrow{b} (q_1, 22), (q_1, \text{tt}) \xrightarrow{\varepsilon} (q_2, \varepsilon), (q_2, x_{top}^= \wedge x_1^{\neq}) \xrightarrow{b} (q_2, \varepsilon), (q_2, x_{top}^=) \xrightarrow{a} (q_3, \varepsilon)\}$.

In a run of $\mathcal{A}_1$, the first input data value is pushed and loaded to the first register in q_0, and then an arbitrary number of input data values different from the first register are pushed in q_1. After the current state is nondeterministically changed to q_2 by the ε-rule (the third rule in δ), the data values in the stack are popped and compared with the remaining input data values. Hence, $L(\mathcal{A}_1) = \{(a, d_0) \cdots (b, d_n)(b, d_n) \cdots (a, d_0) \mid d_0 \neq d_i \text{ for } i \in [n], n \geq 0\}$.

Stack					Registers		State	Input
				$\perp$	$\perp$	$\perp$	q_0	(a, d_0)
			d_0	d_0	d_0	$\perp$	q_1	(b, d_1)
		d_1	d_1	d_0	d_0	d_1	q_1	(b, d_2)
	d_2	d_2	d_1	d_0	d_0	d_2	q_1	ε
	d_2	d_1	d_0		d_0	d_2	q_2	(b, d_2)
		d_1	d_0		d_0	d_2	q_2	(b, d_1)
			d_0		d_0	d_2	q_2	(a, d_0)
					d_0	d_2	q_3	

Figure 1: The transitions in the accepting run of $(a, d_0)(b, d_1)(b, d_2)(b, d_2)(b, d_1)(a, d_0)$ in $\mathcal{A}_1$ ($d_i \neq d_0$ for $i \neq 0$).

2.3 Turing machine

In this section, we define *Turing machine* (abbreviated as TM) and some notions for TM. We use TM for proving Theorem 2.

A TM is a tuple $M = (Q, \Gamma, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of states,
- Γ is a finite set of tape symbols containing a special symbol representing *blank*, $\sqcup \in \Gamma \backslash \Sigma$,
- $\Sigma \subseteq \Gamma \backslash \{\sqcup\}$ is a set of input symbols,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is a state transition function,
- $q_0 \in Q$ is the initial state, and
- $F \subseteq Q$ is the set of accepting states.

For a state $q \in Q$, a tape content $\alpha \in \Gamma^*$ and a head position $j \in [|\alpha|]$, (q, α, j) is called ID of M. For two IDs $(q, \alpha, j), (q', \alpha', j')$, we say that (q, α, j) can transit to (q', α', j'), written as $(q, \alpha, j) \to (q', \alpha', j')$ if

$\exists a, b \in \Gamma, \exists \beta, \gamma \in \Gamma^*, \exists m \in \{L, R\}$ such that

$|\beta| = j - 1, \ \alpha = \beta a \gamma, \delta(q, a) = (q', b, m)$

$$\alpha' = \begin{cases} \beta b \sqcup & \text{if } |\alpha| = j \ (\text{i.e. } \gamma = \varepsilon) \text{ and } j' = j + 1, \\ \beta b \gamma & \text{otherwise,} \end{cases}$$

$$j' = \begin{cases} \max(1, j - 1) & m = L, \text{ or} \\ j + 1 & m = R. \end{cases}$$

Let $\xrightarrow{*}$ be the reflexive transitive closure of $\to$.

For a TM $M = (Q, \Gamma, \Sigma, \delta, q_0, F)$ and $u \in \Sigma^*$, if $(q_0, u, 1) \xrightarrow{*} (q_f, u', j)$ for some $q_f \in F, u' \in \Gamma^*$ and $j \in \mathbb{N}$, then M accepts u. Let $L(M) = \{u \in \Sigma^* \mid M \text{ accepts } u\}$, which is the language recognized by M. For a function $f : \mathbb{N}_0 \to \mathbb{N}_0$, if for any $u \in \Sigma^*$ and any (q, α, j) such that $(q_0, u, 1) \xrightarrow{*} (q, \alpha, j), |\alpha| \leq f(|u|)$ holds, then we say that M is an $f(n)$-space bounded TM. If M is a $p(n)$-space bounded TM for a polynomial $p(n)$, M is a polynomial space bounded TM.

3 COMPUTATIONAL COMPLEXITY

In [21], it was shown that the membership and emptiness problems for RCFG are EXPTIME-complete. Those problems for RPDA were also shown to be EXPTIME-complete by bidirectional poly-time equivalence transformations between RCFG and RPDA in [22].

THEOREM 1 ([22]). *The membership and emptiness problems for RPDA are EXPTIME-complete.*

In this section, we first show that the membership problems for non-decreasing and growing RPDA are PSPACE-complete and NP-complete, respectively. Although the lower-bound proofs are similar to those for ε-rule free and growing RCFG in [21], we present formal description of the reductions in these proofs to make the paper self-contained.

THEOREM 2. *The membership problem for non-decreasing RPDA is PSPACE-complete.*

Proof If a given RPDA is non-decreasing, we can decrease the length of a stack only by using non-ε-pop rules with reading an input data word. Therefore, for an input data word w, the height of every stack appearing in an accepting run of w is at most $|w|$ because every accepting run must finish with empty stack. Hence, the membership problem is in PSPACE.

To prove PSPACE-hardness, we use a poly-time reduction from the membership problem for polynomial space bounded TM. In the reduction, we simulate tape contents of a given TM M by a register assignment of the RPDA $\mathcal{A}$ constructed from M.

Assume that we are given a $p(n)$-space bounded TM $M = (Q_M, \Gamma, \Sigma, \delta_M, q_0, F)$ where $p(n)$ is a polynomial and an input $u \in \Sigma^*$ to M. Then, we construct $(|\Gamma| + p(|u|))$-RPDA $\mathcal{A} = (Q_{\mathcal{A}}, T_{(1,0)}, \delta_{\mathcal{A}})$ over a singleton alphabet $\{a\}$ and an arbitrary set D of data values that satisfies $u \in L(M) \Leftrightarrow (a, \perp) \in L(\mathcal{A})$, where

$$\begin{aligned} Q_{\mathcal{A}} = \ & \{T_{(i,j)} \mid 1 \leq j < i \leq |\Gamma|\} \cup \{T_{(1,0)}\} \\ & \cup \{W_i \mid 0 \leq i \leq |u|\} \\ & \cup \{A_q^{(i,j)} \mid q \in Q_M, \ i \in [|\Gamma|], \ j \in [p(|u|)]\} \\ & \cup \{B_q^{(i,j)} \mid q \in Q_M, \ i \in [|\Gamma|], \ j \in [p(|u|)]\} \\ & \cup \{E\} \end{aligned}$$

and $\delta_{\mathcal{A}}$ is constructed as follows. Without loss of generality, we assume that $\Gamma = \{1, 2, \dots, |\Gamma|\} \subseteq \mathbb{N}$ and 1 is the blank symbol of M. In the following, we denote the ith element of a sequence α by α_i (i.e., $\alpha = \alpha_1 \alpha_2 \dots \alpha_{|\alpha|}$).

- We construct transition rules that load different data values in the first $|\Gamma|$ registers. Note that we keep the initial value $\perp$ in the first register. To the ith register ($2 \leq i \leq |\Gamma|$), a data value different from $\perp$ is assigned by Rule (1), and that data value is guaranteed to be different from the value of every jth register ($2 \leq j < i$) by Rule (2).

$$(T_{(i-1,i-2)}, x_1^{\neq}, i) \xrightarrow{\varepsilon} (T_{(i,1)}, 1) \quad \text{for } 2 \leq i \leq |\Gamma|, \tag{1}$$

$$(T_{(i,j-1)}, x_i^= \wedge x_j^{\neq}) \xrightarrow{\varepsilon} (T_{(i,j)}, 1) \quad \text{for } 2 \leq j < i \leq |\Gamma|. \tag{2}$$

- To express the initial tape contents u, we construct the following transition rules that load data values corresponding to the symbols in u from left to right into $(|\Gamma| + 1)$th to $(|\Gamma| + |u|)$th registers:

$$(T_{(|\Gamma|, |\Gamma|-1)}, \text{tt}) \xrightarrow{\varepsilon} (W_0, 1), \tag{3}$$

$$(W_{i-1}, x_{u_i}^=, |\Gamma| + i) \xrightarrow{\varepsilon} (W_i, 1) \quad \text{for } i \in [|u|]. \tag{4}$$

- Let $s(m) = -1$ if $m = L$ and $s(m) = 1$ if $m = R$. For encoding the state transition and accepting condition of M by $\mathcal{A}$, we introduce a state $A_q^{(i,j)}$ for $q \in Q_M, i \in [|\Gamma|]$, and $j \in [p(n)]$. $A_q^{(i,j)}$ represents a part of an ID (q, α, j) of M where $i = \alpha_j$, i.e. the tape symbol at the head position. The remaining information about α of (q, α, j) will be represented by a register assignment of $\mathcal{A}$. More precisely, the content of $(|\Gamma| + j)$th register (i.e. $\theta(|\Gamma| + j)$) equals the data value $\theta(\alpha_j)$ representing the tape symbol α_j for $j \in [|\alpha|]$ and $\theta(|\Gamma| + j) = \perp$ for $|\alpha| < j \leq p(|u|)$. Let θ_{α} denote such a register assignment that represents the tape contents α. We illustrate the correspondence between an ID of M and a state and a register assignment of $\mathcal{A}$ in Fig. 2.

  – To derive the states corresponding to the initial ID of M, we construct the following rule:

  $$(W_{|u|}, \text{tt}) \xrightarrow{\varepsilon} (A_{q_0}^{(u_1, 1)}, 1). \tag{5}$$

Figure 2: $\mathcal{A}$'s state and registers that correspond to M's ID (q, α, j).

– Consider $A_q^{(i,j)}$ and let $\delta_M(q, i) = (q', b', m')$. For each $a \in \Gamma$, we construct the following rules:

$$(A_q^{(i,j)}, x_{b'}^=, |\Gamma| + j) \xrightarrow{\varepsilon} (B_{q'}^{(a, \max(1, j + s(m')))}, 1). \tag{6}$$

We also construct the following rule for each $q' \in Q_M$, $a \in \Gamma$, and $j' \in [p(n)]$:

$$(B_{q'}^{(a, j')}, x_a^= \land x_{|\Gamma| + j'}^=) \xrightarrow{\varepsilon} (A_{q'}^{(a, j')}, 1). \tag{7}$$

• Finally, we construct for each $q_f \in F$ the following rules to express accepting IDs:

$$(A_{q_f}^{(i,j)}, x_1^=) \xrightarrow{a} (E, \varepsilon). \tag{8}$$

We can show for each ID (q, α, j),

$$(q, \alpha, j) \xrightarrow{*} (q_f, u', j') \text{ for some } q_f \in F, u' \in \Gamma^* \text{ and } j' \in \mathbb{N}$$

$$\text{iff } (A_q^{(\alpha_j, j)}, \theta_\alpha, (a, \perp), \perp) \overset{*}{\Rightarrow}_{\mathcal{A}} (E, \theta, \varepsilon, \varepsilon) \text{ for some } \theta \in \Theta_k \tag{9}$$

by induction on the length of the run of M for only if part and by induction on the length of the run of $\mathcal{A}$ for if part.

We can easily prove that $(T_{(1,0)}, \theta_\perp, (a, \perp), \perp) \overset{*}{\Rightarrow}_{\mathcal{A}}$ $(A_{q_0}^{(u_1, 1)}, \theta_u, (a, \perp), \perp)$, and moreover, if $(T_{(1,0)}, \theta_\perp, (a, \perp), \perp) \overset{*}{\Rightarrow}_{\mathcal{A}}$ $(E, \theta, \varepsilon, \varepsilon)$ for some $\theta \in \Theta_k$, then this run must be $(T_{(1,0)}, \theta_\perp, (a, \perp), \perp) \overset{*}{\Rightarrow}_{\mathcal{A}} (A_{q_0}^{(u_1, 1)}, \theta_u, (a, \perp), \perp) \overset{*}{\Rightarrow}_{\mathcal{A}} (E, \theta, \varepsilon, \varepsilon)$. By letting $(q, \alpha, j) = (q_0, u, 1)$ in property (9) and by the above-mentioned fact, we obtain $u \in L(M) \Leftrightarrow ((T_{(1,0)}, \theta_\perp, (a, \perp), \perp) \overset{*}{\Rightarrow}_{\mathcal{A}}$ $(E, \theta, \varepsilon, \varepsilon)$ for some $\theta \in \Theta_k) \Leftrightarrow (a, \perp) \in L(\mathcal{A})$.

THEOREM 3. *The membership problem for growing RPDA is NP-complete.*

Proof If a given RPDA is growing, for an input data word w, an accepting run of w applies ε-push rules at most $|w|$ times. Therefore, the length of an accepting run does not exceed $2|w| + 1$. Hence, the membership problem is in NP.

We prove NP-hardness by a polynomial-time reduction from the satisfiability problem for 3-Conjunctive Normal Form (3CNF). Let $\phi = (a_1 \lor b_1 \lor c_1) \ldots (a_m \lor b_m \lor c_m)$ be a 3CNF over Boolean variables $y_1, \ldots, y_n$ where each a_i, b_i, c_i $(i \in [m])$ is a literal y_j or $\overline{y_j}$ for some j $(j \in [n])$. For i $(i \in [m])$, we define register number r_{a_i} as $r_{a_i} = 2j$ if $a_i = y_j$ and $r_{a_i} = 2j + 1$ if $a_i = \overline{y_j}$. We also define the same notation r_{b_i} and r_{c_i} for b_i and c_i. We construct the growing $(2n + 1)$-RPDA $\mathcal{A} = (Q, q_0, \delta)$ over $\Sigma = \{a\}$ where $Q = \{q_0, P_0, \ldots, P_n, C_0, \ldots, C_m, E\}$ and

$$\delta = \{(q_0, \text{tt}, 1) \xrightarrow{a} (P_0, 1)\}$$

$$\cup \{(P_{i-1}, x_1^=, 2i + j) \xrightarrow{a} (P_i, 1) \mid i \in [n], j \in \{0, 1\}\}$$

$$\cup \{(P_n, x_1^=) \xrightarrow{a} (C_0, 1)\}$$

$$\cup \{(C_{i-1}, x_r^=) \xrightarrow{a} (C_i, 1) \mid i \in [m], r \in \{r_{a_i}, r_{b_i}, r_{c_i}\}\}$$

$$\cup \{(C_m, x_1^=) \xrightarrow{a} (E, \varepsilon)\}.$$

The first register of the constructed RPDA $\mathcal{A}$ is used for keeping a data value (possibly) different from $\perp$, and we use that value and $\perp$ for representing tt and ff, respectively. $\mathcal{A}$ nondeterministically loads the value representing tt to exactly one of the $(2i)$th and $(2i + 1)$th registers for each i, to encode a truth value assignment to $y_1, \overline{y_1}, y_2, \overline{y_2}, \ldots, y_n, \overline{y_n}$. Then $\mathcal{A}$ reads the value of one of the literals a_i, b_i, c_i for each clause $a_i \lor b_i \lor c_i$ in ϕ. It is not difficult to show that ϕ is satisfiable if and only if $(a, d)^{n+m+3} \in L(\mathcal{A})$, where d is an arbitrary data value in $D \setminus \{\perp\}$. Since $(a, d_1)^{n+m+3} \in L(\mathcal{A})$ iff $(a, d_2)^{n+m+3} \in L(\mathcal{A})$ for any $d_1, d_2 \in D \setminus \{\perp\}$, we can choose any $d \in D \setminus \{\perp\}$ to make the input data word for the membership problem. Hence, we have shown NP-hardness of the problem.

For both of the RPDA constructed in the proofs of Theorem 2 and 3, the height of the stack appearing in any accepting run is at most one. This fact implies the following property.

COROLLARY 1. *The membership problems for RA with and without ε-transition are PSPACE-complete and NP-complete, respectively.*

Note that NP-completeness of the membership for RA without ε-transition was first proved in [19].

THEOREM 4. *The emptiness problems for non-decreasing RPDA and growing RPDA are both EXPTIME-complete.*

Proof By theorem 1, it suffices to show that the problem is EXPTIME-hard for growing RPDA. We give a poly-time reduction from the emptiness problem for general RPDA. From an arbitrary RPDA $\mathcal{A} = (Q, q_0, \delta)$, we construct the growing RPDA $\mathcal{A}_g = (Q, q_0, \delta_g)$, where

$$\delta_g = \{(q, \psi, i) \xrightarrow{a} (q', J) \mid (q, \psi, i) \xrightarrow{a} (q', J) \in \delta \text{ or } (q, \psi, i) \xrightarrow{\varepsilon} (q', J) \in \delta\}.$$

For this growing RPDA $\mathcal{A}_g$, obviously $L(\mathcal{A}) = \emptyset \Leftrightarrow L(\mathcal{A}_g) = \emptyset$. Hence, the emptiness problem for growing RPDA is EXPTIME-hard.

4 CONCLUSION

We have discussed the computational complexity of the membership and emptiness problems for RPDA. The combined complexity of the membership problem for general RPDA is EXPTIME-complete and decreases when we consider subclasses of RPDA while the data complexity is in P for general RPDA. The emptiness problem for RPDA remains EXPTIME-complete even if we restrict RPDA to be growing. It is an interesting problem left as future study to investigate whether the inclusions among the classes of languages of general, non-decreasing and growing RPDA are proper or not.

Introducing a logic such as FO($\sim$), EMSO($\sim$) and LTL↓ on data trees that corresponds to or subsumes RPDA is a future study. Also, introducing recursive queries such as datalog in relational databases and fixed point logics as related logical foundations [1, 14] would be an interesting topic to be pursued.

REFERENCES

[1] S. Abiteboul, R. Hull and V. Vianu, Foundations of Databases, Part D: Datalog and Recursion, Addison-Wesley, 1995.

[2] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick and L. Segoufin, Two-variable logic on data trees and XML reasoning, Journal of the ACM 56(3), 2009.

[3] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick and L. Segoufin, Two-variable logic on data words, ACM Trans. Computational Logic 12(4), 2011.

[4] P. Bouyer, A logical characterization of data languages, Inf. Process. Lett. 84(2), 75–85, 2002.

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

[6] S. Demri and R. Lazić, LTL with freeze quantifier and register automata, ACM Trans. on Computational Logic 10(3), 2009.

[7] S. Demri, R. Lazić and D. Nowak, On the freeze quantifier in constraint LTL: Decidability and complexity, Inf. Comput. 205(1), 2–24, 2007.

[8] D. Figueira, Forward-XPath and extended register automata on data-trees, International Conference on Database Theory (ICDT 2010).

[9] D. Figueira, Alternating register automata on finite data words and trees, Logical Methods in Computer Science 8(1:22), 1–43, 2012.

[10] D. Figueira and L. Segoufin, Bottom-up automata on data trees and vertical XPath, 28th Symposium on Theoretical Aspects of Computer Science (STACS 2011), 93–104.

[11] M. Jurdziński and R. Lazić, Alternation-free modal mu-calculus for data trees, Logic in Computer Science (LICS 2007).

[12] M. Kaminsky and N. Francez, Finite-memory automata, Theoretical Computer Science 134, 322–363, 1994.

[13] M. Kaminsky and T. Tan, QTree automata over infinite alphabets, Pillars of Computer Science 2008, LNCS 4800, 386–423.

[14] L. Libkin, Elements of Finite Model Theory, Chapter 10: Fixed Point Logics and Complexity Classes, Springer, 2004.

[15] L. Libkin and W. Martens and D. Vrgoč, Querying graphs with data, Journal of the ACM 63(2), 14:1–14:53, 2016.

[16] L. Libkin, T. Tan and D. Vrgoč, Regular expressions for data words, J. Computer and System Sciences 81(7), 1278–1297, 2015.

[17] L. Libkin and D. Vrgoč, Regular path queries on graphs with data, 15th International Conference on Database Theory (ICDT 2012), 74–85.

[18] F. Neven, T. Schwentick and V. Vianu, Finite state machines for strings over infinite alphabets, ACM Trans. on Computational Logic 5(3), 403–435, 2004.

[19] H. Sakamoto and D. Ikeda, Intractability of decision problems for finite-memory automata, Theoretical Computer Science 231, 297–308, 2000.

[20] L. Segoufin, Automata and logics for words and trees over an infinite alphabet, 15th EACSL Annual Conference on Computer Science Logic (CSL 2006), 41–57.

[21] R. Senda, Y. Takata and H. Seki, Complexity results on register context-free grammars and register tree automata, 15th International Colloquium on Theoretical Aspects of Computing (ICTAC 2018), LNCS 11187, 415–434, Oct 2018.

[22] R. Senda, Y. Takata and H. Seki, Complexity results on register context-free grammars and related formalisms, submitted to an international journal.