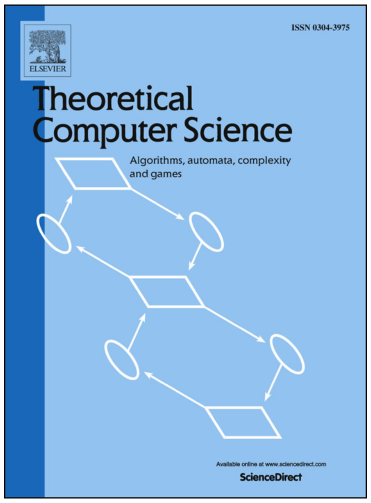
Complexity Results on Register Context-Free Grammars and Related Formalisms

Ryoma Senda, Yoshiaki Takata and Hiroyuki Seki



PII: S0304-3975(22)00283-3
DOI: https://doi.org/10.1016/j.tcs.2022.04.055
Reference: TCS 13387

To appear in: *Theoretical Computer Science*

Received date: 2 April 2019
Revised date: 6 April 2022
Accepted date: 27 April 2022

Complexity Results on Register Context-Free Grammars and Related Formalisms[⋆,⋆⋆]

Ryoma Senda[a,*], Yoshiaki Takata[b], Hiroyuki Seki[a]

[a]*Graduate School of Informatics, Nagoya University, Furo-cho, Chikusa, Nagoya 464-8601, Japan*
[b]*Graduate School of Engineering, Kochi University of Technology, Tosayamada, Kami City, Kochi 782-8502, Japan*

Abstract

Register context-free grammars (RCFG), register pushdown automata (RPDA) and register tree automata (RTA) are extensions of their classical counterparts to handle data values in a restricted way. These extended models are paid attention as models of query languages for structured documents such as XML with data values. This paper investigates the computational complexity of the basic decision problems for the models. We show that the membership and emptiness problems for RCFG are EXPTIME-complete and also show the membership problem becomes PSPACE-complete and NP-complete for ε-rule free RCFG and growing RCFG, respectively while the emptiness problem remains EXPTIME-complete for these subclasses. The complexity of these problems for RPDA and RTA as well as their language expressive powers are also investigated.

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

[⋆]This paper is an extended version of the conference paper [1] and [2].
[⋆⋆]This work was supported by JSPS KAKENHI Grant Number JP19H04083.
[*]Corresponding author
Email addresses: `ryoma.private@sqlab.jp` (Ryoma Senda), `takata.yoshiaki@kochi-tech.ac.jp` (Yoshiaki Takata), `seki@i.nagoya-u.ac.jp` (Hiroyuki Seki)

1. Introduction

An extension of finite automata by adding an ability of processing data values easily becomes Turing equivalent even if very simple operations are allowed such as in two counter machines. There have been studies on defining computational models having mild powers of processing data values by extending classical models. Some of them are shown to have decidability on basic problems and closure properties, including first-order and monadic second-order logics with data equality [3], linear temporal logic with freeze quantifier [4] and register automata [5]. Among them, register automata (abbreviated as RA) are a natural extension of finite automata defined by incorporating registers that can keep data values as well as the equality test between an input data value and the data value kept in a register. Regular expressions were extended to regular expressions with memory (REM), which have the same expressive power as RA [6].

Recently, attention has been paid to RA as a computational model of a query language for structured documents such as XML because a structured document can be modeled as a tree or a graph where data values are associated with nodes and a query on a document can be specified as the combination of a regular pattern and a condition on data values [7, 8]. For query processing and optimization, the decidability (hopefully in polynomial time) of basic properties of queries is desirable. 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 is the most basic problem. 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 considered to be redundant or meaningless when query optimization is performed. The membership and emptiness problems for RA were already shown to be decidable [5] and their computational complexities were also analyzed [4, 9].

While RA have a power sufficient for expressing regular patterns on *paths* of a tree or a graph, it cannot represent tree patterns (or 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 [10] as extensions of classical context-free grammars (CFG) and pushdown automata (PDA), respectively, in a similar way to extending FA to RA. In [10], properties of RCFG and RPDA were shown including the decidability of the membership and emptiness problems, the equivalence of RCFG and RPDA in their language expressive powers and the closure properties. However, the computational complexity of the above decision problems has not been reported yet. In parallel with this, RA were extended to a model dealing with trees, called tree automata over an infinite alphabet [11, 12]. For uniformity, we will call the latter register tree automata (abbreviated as RTA).

In this paper, we analyze the computational complexity of the membership and emptiness problems for general RCFG, RPDA, RTA and some subclasses of them. In the original definition of RCFG [10], an infinite alphabet is assumed and an RCFG is defined as a formal system that generates finite strings over the infinite alphabet as in the original definition of RPDA [10] and RA [5]. In a derivation of RCFG, a symbol can be loaded to a register only when the symbol is different from any symbol stored in the other registers, by which the equality checking is implicitly incorporated. In recent studies on RA [7, 6], more concrete notions suitable for modeling a query language are adopted, namely, a word is a finite string over the product of a finite alphabet and an infinite set of data values (called *data word*), and the equality check between an input data value and the data value kept in a register can be specified as the guard condition of a transition rule. Also, different registers can keep an identical value in general.

Following those modern notions, we first define an RCFG as a grammar that derives a data word. In a derivation of a k-RCFG, k data values are associated with each occurrence of a nonterminal symbol (called *an assignment of data values to registers* or *assignment* in short) and a production rule can be applied only when the guard condition of the rule, which is a Boolean combination of the equality check between an input data value and the data value in a register, is satisfied. We introduce subclasses of RCFG, including ε-rule free RCFG, growing RCFG, and RCFG with bounded register numbers.

We then show that the membership problems for general RCFG, ε-rule free RCFG, growing RCFG, and RCFG with bounded register numbers are EXPTIME-complete, PSPACE-complete, NP-complete, and solvable in P, respectively. For example, to show the upper bound for general RCFG, we use the property that any RCFG can be translated into a classical CFG when the number of different data values used in the derivation is finite, which was shown in [10]. EXPTIME-hardness is proved by a polynomial time reduction from the membership problem for polynomial space-bounded alternating Turing machines. We also show that the emptiness problem for general RCFG is EXPTIME-complete and the complexity does not decrease even if we restrict RCFG to be growing.

Next, we define an RPDA as a pushdown automaton with registers that takes a data word as an input. 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 also introduce subclasses of RPDA called non-decreasing RPDA, growing RPDA, quasi non-decreasing RPDA and quasi growing RPDA and analyze the computational complexity of the membership and emptiness problems for RPDA and these subclasses. Also, we show the equivalence between the classes of languages generated by ε-rule free RCFG and quasi non-decreasing RPDA.

Finally, we define an RTA as a tree automaton over a data tree where a data value as well as a symbol is labelled with each node and analyze the computational complexity of the above two problems for RTA. It is well-known that the class of the yields of tree languages accepted by tree automata coincides with the class of CFL. The difference of RCFG and RTA in the membership problem is that a derivation tree is not specified as an input in the former case while a data tree is given as an input to the problem in the latter case.

Main results of this paper are summarized in Table 1. Note that the complexity of the membership problems is in terms of both the size of a grammar or an automaton and that of an input word (*combined complexity*). The complexity of the membership problem on the size of an input word only (*data complexity*) is P for general RCFG, RPDA and RTA. In application, the size of a query (a grammar or an automaton) is usually much smaller than that of a data (an input word). 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.

Related work Early studies on query optimization and static analysis for structured documents used traditional models such as tree automata, two variable logic and LTL. While those studies were successful, most of them neglected data values associated with documents. Later, researchers developed richer models that can be applied to structured documents with data values, including extensions of automata (register automata, pebble automata, data automata) and extensions of logics (two-variable logics with data equality, LTL with freeze quantifier). We review each of them in the following.

Table 1: Complexity results on RCFG, RPDA and RTA

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

Register automata and register context-free grammars: As already mentioned, register automata (RA) was first introduced in [5] as finite-memory automata where they show that the membership and emptiness problems are decidable, and the class of data languages recognized by RA is closed under union, concatenation and Kleene-star. Later, the computational complexity of the former two problems are analyzed in [9, 4]. In [10], register context-free grammars (RCFG) as well as pushdown automata over an infinite alphabet were introduced and the equivalence of the two models as well as the decidability results and closure properties similar to RA were shown.
Other automata for data words: There are extensions of automata to deal with data in a restricted way other than RA, namely, data automata [13] and pebble automata (PA) [14, 15]. It is desirable for a query language to have an efficient data complexity for the membership problem. Libkin and Vrgoč [7] argue that register automata (RA) is the only model that has this property among the above mentioned formalisms and adopt RA as the core computational model of their queries on graphs with data. Neven [15] considers variations of RA and PA, either they are one way or two ways, deterministic, nondeterministic or alternating, shows inclusion and separation relationships among these automata, FO($\sim, <$) and EMSO($\sim, <$), and gives the answer to some open problems including the undecidability of the universality problem for RA.
LTL with freeze quantifier: Linear temporal logic (LTL) was extended to LTL with freeze quantifier (LTL$\downarrow$) [16, 4]. A data value is bound to a variable in a formula and is referred to later in the scope of a freeze quantifier of that variable. The relationship among subclasses of LTL$\downarrow$ and RA as well as the decidability and complexity of the satisfiability (nonemptiness) problems are investigated [4]. Especially, they showed that the emptiness problem for

(both nondeterministic and deterministic) RA are PSPACE-complete.
Two-variable logics with data equality: First-order logic (FO) and monadic second-order logic (MSO) are major logics for finite model theory. It is known that two-variable $\mathrm{FO}^2(<,+1)$ where $<$ is the ancestor-descendant relation and $+1$ is the parent-child relation is decidable and corresponds to Core XPath. The logic was extended to those with data equality. It was shown in [3] that $\mathrm{FO}^2(\sim,<,+1)$ with data equality $\sim$ is decidable on data words. Note that $\mathrm{FO}^2(\sim,<,+1)$ is incomparable with $\mathrm{LTL}_1^{\downarrow}(X,U)$ of [4]. Also it was shown in [17] that $\mathrm{FO}^2(\sim,+1)$ and existential $\mathrm{MSO}^2(\sim,+1)$ are decidable on unranked data trees.
Tree automata and data XPath: In [11], tree automata over infinite alphabets are introduced as a natural extension of RA. We call them register tree automata (RTA) in this paper. They showed that the membership and emptiness problems for RTA are decidable and the the universality and inclusion problems are undecidable, and also showed that a data language L is generated by an RCFG if and only if there is an RTA that accepts a data tree language whose yield is L. However, the complexity of those decidable problems was not shown. In connection with XPath, top-down tree automata for data trees called alternating tree register automata (ATRA) [18] which correspond to forward XPath were introduced, and the decidability of the emptiness problems for single-register cases was shown. While RTA work on ranked data trees, ATRA work on unranked data trees. Later, Figueira [19, 20] extended ATRA so that (1) the extended ATRA can guess a data value to store it in a register, and also (2) they can universally quantify the data values encountered in a given run. The emptiness for the extended ATRA was also shown in [19, 20] to be decidable by identifying ATRA as well-structured transition systems. Also, bottom-up tree automata for unranked data trees, which correspond to vertical XPath were introduced and the decidability of the emptiness was shown in [21]. Since XML documents are usually modeled as unranked trees, ATRA may be a better model for XPath than RTA. However, the complexity of the emptiness for ATRA is not elementary and an appropriate subclass would be needed for broader applications.

This paper is an extended version of [1] and [2] by (1) adding non-trivial formal proofs, (2) showing some results on the relation between the classes of languages generated by restricted RCFG, (3) showing the linear-time convertibleness between RCFG and RPDA and (4) introducing quasi non-decreasing

RPDA and quasi growing RPDA. In addition, we prove the equivalence between the classes of data languages accepted by quasi non-decreasing RPDA and generated by ε-rule free RCFG.

2. Definitions

2.1. Preliminaries

Register context-free grammars were introduced in [10] as grammars over an infinite alphabet. We define them as grammars over the product of a finite alphabet and an infinite set of data values, following recent notions [7, 6]. Note that these differences are not essential.

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. Let Θ_k denote the class of assignments to k registers and x_i is a variable that represents the contents of the i-th register. We write $X_k = \{x_1, \ldots, x_k\}$. For $\theta, \theta' \in \Theta_k$, we write $\theta' = \theta[x_i \leftarrow d]$ 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:

$$\psi := \mathrm{tt} \mid \mathrm{ff} \mid x_i^{=} \mid x_i^{\neq} \mid \psi \wedge \psi \mid \psi \vee \psi \mid \neg\psi$$

where $x_i \in X_k$. The description length of guard expression ψ is defined as

$$\|\psi\| = \begin{cases} 1 & \text{if } \psi = \mathrm{tt} \text{ or } \mathrm{ff}, \\ 1 + \log k & \text{if } \psi = x_i^{=} \text{ or } x_i^{\neq}, \\ 1 + \|\psi_1\| + \|\psi_2\| & \text{if } \psi = \psi_1 \wedge \psi_2 \text{ or } \psi_1 \vee \psi_2, \\ 1 + \|\psi_1\| & \text{if } \psi = \neg\, \psi_1. \end{cases}$$

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

- $\theta, d \models \mathrm{tt}$
- $\theta, d \not\models \mathrm{ff}$
- $\theta, d \models x_i^{=}$ iff $\theta(i) = d$

- $\theta, d \models x_i^{\neq}$ iff $\theta(i) \neq d$
- $\theta, d \models \psi_1 \wedge \psi_2$ iff $\theta, d \models \psi_1$ and $\theta, d \models \psi_2$
- $\theta, d \models \psi_1 \vee \psi_2$ iff $\theta, d \models \psi_1$ or $\theta, d \models \psi_2$
- $\theta, d \models \neg\psi$ iff $\theta, d \not\models \psi$

where $\theta, d \not\models \psi$ holds iff $\theta, d \models \psi$ does not hold.

For a finite alphabet Σ and a set D of data values, 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 $\Sigma \times D$. For a data word $w = (a_1, d_1)(a_2, d_2) \dots (a_n, d_n)$, $a_1 a_2 \dots a_n$ is the label of w and $d_1 d_2 \dots d_n$ is the data part of w. $|\beta|$ denotes the cardinality of β if β is a set and the length of β if β is a finite sequence.

2.2. Register context-free grammars

Let Σ be a finite alphabet, D be a set of data values and $k \in \mathbb{N}$. A k-register context-free grammar (k-RCFG) is a triple $G = (V, R, S)$ where

- V is a finite set of nonterminal symbols (abbreviated as nonterminals),
- R is a finite set of production rules (abbreviated as rules) having either of the following forms:

$$(A, \psi, x_i) \to \alpha, \quad (A, \psi) \to \alpha$$

  where $A \in V$, $\psi \in F_k$, $x_i \in X_k$ and $\alpha \in (V \cup (\Sigma \times X_k))^*$; we call (A, ψ, x_i) (or (A, ψ)) the left-hand side and α the right-hand side of the rule, and,
- $S \in V$ is the start symbol.

A rule whose right-hand side is ε is an *ε-rule* and a rule whose right-hand side is a single nonterminal symbol is a *unit rule*. If R contains no ε-rule, G is called *ε-rule free*. If R contains neither ε-rule nor unit rule, G is called *growing*. The description length of a k-RCFG $G = (V, R, S)$ is defined as $\|G\| = |V| + |R| \max\{(|\alpha| + 1)(\log |V| + \log k) + \|\psi\| \mid (A, \psi, x_i) \to \alpha \in R$ or $(A, \psi) \to \alpha \in R\}$, where $\|\psi\|$ is the description length of ψ. In this definition, we assume that the description length of $\alpha \in (V \cup (\Sigma \times X_k))^*$ is $|\alpha|(\log |V| + \log k)$ because the description length of each element of α is $O(\log |V| + \log k)$ bits if we consider $|\Sigma|$ is a constant. Since the description length of the left-hand side of a rule $(A, \psi, x_i) \to \alpha$ is $\log |V| + \|\psi\| + \log k$,

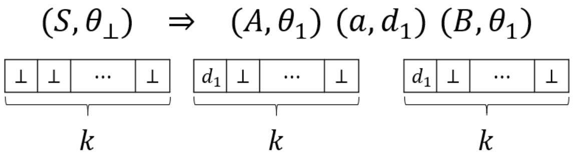

$(S,\theta_\perp) \Rightarrow (A,\theta_1)\ (a,d_1)\ (B,\theta_1)$
$\perp$ $\perp$ ⋯ $\perp$
d_1 $\perp$ ⋯ $\perp$
d_1 $\perp$ ⋯ $\perp$
k
k
k

Figure 1: An example of derivation from $(S,\theta_\perp)$ using $(S,\mathrm{tt},x_1) \to A(a,x_1)B$

we let the description length of this rule be $(|\alpha|+1)(\log|V|+\log k)+\|\psi\|$. We assume $k \le \|G\|$ without loss of generality.

We define $\Rightarrow_G$ as the smallest relation containing the instantiations of rules in R and closed under the context as follows. For $A \in V$, $\theta \in \Theta_k$ and $X \in ((V \times \Theta_k) \cup (\Sigma \times D))^*$, we say (A,θ) directly derives X, written as $(A,\theta) \Rightarrow_G X$ if there exist $d \in D$ (regarded as an input data value) and $(A,\psi,x_i) \to c_1 \dots c_n \in R$ (resp. $(A,\psi) \to c_1 \dots c_n \in R$) such that

$$\theta, d \models \psi,\ X = c'_1 \dots c'_n,\ \theta' = \theta[x_i \leftarrow d] \text{ (resp. } \theta' = \theta) \text{ where}$$
$$c'_j = \begin{cases} (B,\theta') & \text{if } c_j = B \in V, \\ (b,\theta'(l)) & \text{if } c_j = (b,x_l) \in \Sigma \times X_k. \end{cases}$$

For $X, Y \in ((V \times \Theta_k) \cup (\Sigma \times D))^*$, we also write $X \Rightarrow_G Y$ if there are $Z_1, Z_2, Z_3 \in ((V \times \Theta_k) \cup (\Sigma \times D))^*$ such that $X = Z_1(A,\theta)Z_2$, $Y = Z_1Z_3Z_2$ and $(A,\theta) \Rightarrow_G Z_3$.

Let $\overset{*}{\Rightarrow}_G$ and $\overset{+}{\Rightarrow}_G$ be the reflexive transitive closure and the transitive closure of $\Rightarrow_G$, respectively, meaning the derivation of zero or more steps (resp. the derivation of one or more steps). We abbreviate $\Rightarrow_G$, $\overset{*}{\Rightarrow}_G$ and $\overset{+}{\Rightarrow}_G$ as $\Rightarrow$, $\overset{*}{\Rightarrow}$ and $\overset{+}{\Rightarrow}$ if G is clear from the context.

We denote by $\theta_\perp$ the assignment that assigns the initial value $\perp$ to every register. Fig.1 shows an example of a direct derivation from $(S,\theta_\perp)$ using $(S,\mathrm{tt},x_1) \to A(a,x_1)B$. In the figure, d_1 is an arbitrary data value in D and $\theta_1 = \theta_\perp[x_1 \leftarrow d_1]$, because the guard of the rule is tt.

We let
$$L(G) = \{w \mid (S,\theta_\perp) \overset{+}{\Rightarrow} w \in (\Sigma \times D)^*\}.$$

$L(G)$ is called the data language generated by G. For example, if G is a 1-RCFG having two rules $(S,\mathrm{tt},x_1) \to (a,x_1)S(a,x_1)$ and $(S,\mathrm{tt},x_1) \to \varepsilon$, $L(G) = \{(a,d_1)\dots(a,d_n)(a,d_n)\dots(a,d_1) \mid n \ge 0\}$.

0-RCFG coincide with classical context-free grammars and we call a 0-RCFG a *context-free grammar (CFG)*.

Example 1. *For a CFG G, it is well-known that if $L(G) \neq \emptyset$, then there exists a derivation tree of some word $w \in L(G)$ whose height is $O(\|G\|)$. However, this property does not hold for RCFG. Let $\Sigma = \{a\}$ and D be an arbitrary infinite set. Consider the following k-RCFG $G = (V, R, S)$, which satisfies $L(G) = \{(a, \perp)\}$. While $\|G\| = O(k \log k)$, the height of the derivation tree of $(a, \perp)$ is $\Omega(2^k)$.*

$$\begin{aligned} V &= \{A_{(i,b)} \mid 1 \leq i \leq k,\ b \in \{0,1\}\} \\ &\quad \cup \{B_{(i,b)} \mid 1 \leq i < k,\ b \in \{0,1\}\}, \\ S &= A_{(1,0)}, \end{aligned}$$

and R consists of the following rules:

$$(A_{(k,0)}, tt) \to (a, x_k),$$

and for $1 \leq i < k$,

$$\begin{aligned} (A_{(i,0)}, x_k^{=} \wedge x_i^{=}) &\to B_{(i,0)}, \\ (A_{(i,1)}, x_k^{=} \wedge x_i^{\neq}) &\to B_{(i,1)}, \\ (B_{(i,0)}, x_k^{\neq}, x_i) &\to A_{(1,0)} \mid A_{(1,1)}, \\ (B_{(i,1)}, x_k^{=}, x_i) &\to A_{(i+1,0)} \mid A_{(i+1,1)}. \end{aligned}$$

The derivation of G from $(S, \theta_\perp)$ to $(a, \perp)$ simulates a $(k-1)$-bit binary counter. We consider that the ith bit of the binary counter is "0" (resp. "1") if the value of ith register is $\perp$ (resp. not $\perp$). The derivation keeps the value of the kth register being $\perp$. For $1 \leq i < k$ and $b \in \{0,1\}$, derivation $(A_{(i,b)}, \theta) \Rightarrow_G (B_{(i,b)}, \theta)$ can exist if only if the ith bit of the binary counter represented by θ equals b. After this derivation, the ith bit is flipped by the derivation from $(B_{(i,b)}, \theta)$, and it derives $(A_{(j,0)}, \theta')$ and $(A_{(j,1)}, \theta')$ where θ' is the updated assignment, and $j = i+1$ if "the carry to the next bit" exists, and $j = 1$ otherwise. Because $(A_{(k,0)}, \theta)$ for some θ (and $(a, \perp)$) can be derived only when every bit of the binary counter becomes "1", the derivation from $(S, \theta_\perp)$ to $(a, \perp)$ must pass through the elements of $\{A_{(1,0)}, A_{(1,1)}\} \times \Theta_k$ 2^{k-1} times.

3. Basic Properties of RCFG

The properties of RCFG in this section were first shown in [10]. We will give sketches of proofs of the properties to make this paper self-contained. We fix a finite alphabet Σ and a set D of data values.

Proposition 1. *Let G be an RCFG and $D' \subseteq D$ be a finite set. We can construct a CFG G' from G and D' that satisfies $L(G') = L(G) \cap (\Sigma \times D')^*$.*

Proof Let $G = (V, R, S)$ be a k-RCFG and D' be a finite subset of D. We construct a CFG $G' = (V', R', S')$ from G and D' as follows. A nonterminal of G' is a nonterminal of G with k data values that represent an assignment. The rules of G' are constructed accordingly. Note that whether a rule can be applied does not depend on data values themselves but depends on the equality among the data values given as an input or assigned to registers. By this fact, if $|D'| \geq k + 1$, then it suffices to consider data values in D' to simulate the derivations of G. Otherwise, i.e., $|D'| \leq k$, we need $k + 1$ different data values including those in D'.

- $V' = V \times D''^k$ where $D'' = D'$ if $|D'| \geq k + 1$ and D'' is a set such that $D'' \supseteq D'$ and $|D''| = k + 1$ otherwise. Note that we can consider an assignment θ in Θ_k to be a k-tuple over D, i.e. an element of D^k, and thus $V' \subseteq V \times \Theta_k$.
- $R' = \{(A, \theta) \to X \mid (A, \theta) \in V',\ X \in (V' \cup (\Sigma \times D'))^*,$ and $(A, \theta) \Rightarrow_G X\}$.
- $S' = (S, \theta_\perp)$.

We can show by induction on the length of derivations that for any $X \in (V' \cup (\Sigma \times D'))^*$, $S' \overset{*}{\Rightarrow}_{G'} X$ if and only if $(S, \theta_\perp) \overset{*}{\Rightarrow}_G X$. This establishes $L(G') = L(G) \cap (\Sigma \times D')^*$.

Proposition 2. *If a k-RCFG G generates a data word of length n, G generates a data word of length n that contains only data values in an arbitrary subset $D' \subseteq D$ where $|D'| = k + 1$ and $\perp \in D'$.*

Proof Let $G = (V, R, S)$ be a k-RCFG and $w \in L(G)$ with $|w| = n$. Also, let $d_1, \ldots, d_{k+1} \in D$ be arbitrary data values that are mutually different and contain $\perp$. Consider a direct derivation of G:

$$(A, \theta) \Rightarrow_G X$$

where $A \in V$, $\theta \in \Theta_k$, $X \in ((V \times \Theta_k) \cup (\Sigma \times D))^*$ and $\theta(j) \in \{d_1, \ldots, d_{k+1}\}$ for every j $(1 \leq j \leq k)$. We alter the direct derivation as follows: Assume that an applied rule is $(A, \psi, x_i) \to \alpha$ and the content of i-th register of θ is updated as d, i.e., the assignment appearing in X is $\theta' = \theta[x_i \leftarrow d]$. Let θ'' be the assignment as follows:

- If there is j with $1 \leq j \leq k$ and $j \neq i$ such that $\theta(j) = d$, let $\theta'' = \theta'$.
- Otherwise, there is a data value $d' \in \{d_1, \dots, d_{k+1}\}$ such that $\theta(j) \neq d'$ for every j ($1 \leq j \leq k$). Let $\theta'' = \theta[x_i \leftarrow d']$.

Let X' be obtained from X by replacing every θ' in X with θ''. Then, $(A, \theta) \Rightarrow_G X'$ with X' containing at most $k+1$ different data values. For a given derivation $(S, \theta_\perp) \overset{*}{\Rightarrow}_G w$, by starting with $(S, \theta_\perp)$ and repeating the above transformation to each direct derivation in $(S, \theta_\perp) \overset{*}{\Rightarrow}_G w$, we obtain a desired derivation $(S, \theta_\perp) \overset{*}{\Rightarrow}_G w'$ with $|w'| = n$ and at most $k+1$ different data values.

Proposition 3. *The membership problem for RCFG is decidable.*

Proof Let G be an RCFG. It holds that $w = (a_1, d_1) \dots (a_n, d_n) \in L(G)$ if and only if $w \in L(G) \cap (\Sigma \times \{d_1, \dots, d_n\})^*$. By Proposition 1, we can construct a CFG G' from G and w that generates $L(G) \cap (\Sigma \times \{d_1, \dots, d_n\})$. This implies the decidability of the membership problem of RCFG because the membership problem of CFG is decidable.

Proposition 4. *The emptiness problem for RCFG is decidable.*

Proof For a given k-RCFG G, let $D_k = \{d_1, \dots, d_{k+1}\} \subseteq D$ be an arbitrary subset of D consisting of $k+1$ different data values. By Proposition 2, $L(G) = \emptyset$ if and only if $L(G) \cap (\Sigma \times D_k)^* = \emptyset$. By Proposition 1, we can construct a CFG G' from G and D_k such that $L(G') = L(G) \cap (\Sigma \times D_k)^*$. Hence, the emptiness for RCFG is decidable because the emptiness for CFG is decidable [22, Section 10.3].

RCFG has the following closure properties [10].

Proposition 5. *The class of data languages generated by RCFG are closed under union, concatenation and Kleene-star.*

Proof For given k-RCFG $G_1 = (V_1, R_1, S_1), G_2 = (V_2, R_2, S_2)$ where $V_1 \cap V_2 = \emptyset$, we can construct k-RCFG $G_U = (V_U, R_U, S_U), G_C = (V_C, R_C, S_C), G_K = (V_K, R_K, S_K)$ such that $L(G_U) = L(G_1) \cup L(G_2), L(G_C) = L(G_1) \cdot L(G_2), L(G_K) = (L(G_1))^*$ as follows.
Union: $V_U = V_1 \cup V_2 \cup \{S_U\}, R_U = R_1 \cup R_2 \cup \{(S_U, \mathrm{tt}) \to S_1, (S_U, \mathrm{tt}) \to S_2\}$.
Concatenation: $V_C = V_1 \cup V_2 \cup \{S_C\}, R_U = R_1 \cup R_2 \cup \{(S_C, \mathrm{tt}) \to S_1 S_2\}$.
Kleene-star: $V_K = V_1 \cup \{S_K\}, R_K = R_1 \cup \{(S_K, \mathrm{tt}) \to S_1 S_K, (S_K, \mathrm{tt}) \to \varepsilon\}$.

Relation between the classes of languages of restricted RCFG

In this section, we investigate inclusion relations among the classes of data languages of a few subclasses of RCFG. In the following, we fix a finite nonempty alphabet Σ and an infinite set D of data values. Let $a \in \Sigma$ be an arbitrary terminal symbol.

Let $\mathcal{L}_{k\text{-RCFG}}$ denote the class of data languages that can be generated by a k-RCFG over Σ and D. We can show that $\mathcal{L}_{k\text{-RCFG}} \subsetneq \mathcal{L}_{(k+1)\text{-RCFG}}$ as a corollary of Proposition 2.

Corollary 1. $\mathcal{L}_{k\text{-RCFG}} \subsetneq \mathcal{L}_{(k+1)\text{-RCFG}}$.

Proof We can construct a $(k+1)$-RCFG G over Σ and D such that $L(G) = \{(a, d_1)(a, d_2) \dots (a, d_{k+2}) \mid d_i \neq d_j$ for any $i, j \in [k+2]$ such that $i \neq j\}$. In fact, we can let $G = (V, R, S)$ where $V = \{T_0, T_1, \dots, T_{k+1}\}$, $S = T_0$, and R consists of the following rules:

$$\begin{aligned} &(T_{i-1}, x_1^{\neq} \wedge \dots \wedge x_{i-1}^{\neq}, x_i) \to (a, x_i)\, T_i \qquad \text{for } 1 \leq i \leq k+1, \\ &(T_{k+1}, x_1^{\neq} \wedge \dots \wedge x_{k+1}^{\neq}, x_1) \to (a, x_1). \end{aligned}$$

Because every data word in $L(G)$ contains $k+2$ different data values, by Proposition 2, there is no k-RCFG that generates $L(G)$. $\square$

A k-RCFG G is called an *r-bounded-guard k-RCFG* if every guard expression of G refers to at most r registers. Let $\mathcal{L}_{r\text{-bounded-guard-}k\text{-RCFG}}$ denote the class of data languages that can be generated by an r-bounded-guard k-RCFG. As shown below, we can prove that

$$\begin{aligned} \mathcal{L}_{0\text{-bounded-guard-}k\text{-RCFG}} &\subsetneq \mathcal{L}_{1\text{-bounded-guard-}k\text{-RCFG}} \\ &\subsetneq \mathcal{L}_{2\text{-bounded-guard-}k\text{-RCFG}} = \mathcal{L}_{k\text{-RCFG}}. \end{aligned}$$

Proposition 6. *For a k-RCFG G, there exists a 2-bounded-guard k-RCFG G' such that $L(G) = L(G')$ and $\|G'\|$ is a polynomial of $\|G\|$.*

Proof We can transform G into a 2-bounded-guard k-RCFG G' without changing its language as follows. In the following, for a guard expression ψ, let $\psi|_{x_i^{=} \leftarrow \mathrm{tt}}$ be the guard expression obtained from ψ by replacing $x_i^{=}$ and $x_i^{\neq}$ with tt and ff, respectively. We define $\psi|_{x_i^{=} \leftarrow \mathrm{ff}}$ in the same way. By the Shannon expansion, we can transform ψ into $(x_i^{=} \wedge \psi|_{x_i^{=} \leftarrow \mathrm{tt}}) \vee (x_i^{\neq} \wedge \psi|_{x_i^{=} \leftarrow \mathrm{ff}})$ without changing its meaning.

First, we apply the following equivalence transformations to G:

1. Apply de Morgan's law to each guard expression for moving negation to the innermost position.

2. Replace a rule $(A, \psi, x_i) \to \alpha$ with $(A, x_i^= \wedge \psi|_{x_i^= \leftarrow \mathrm{tt}}, x_i) \to \alpha$ and $(A, x_i^{\neq} \wedge \psi|_{x_i^= \leftarrow \mathrm{ff}}, x_i) \to \alpha$. Then, transform the rule $(A, \xi \wedge \psi', x_i) \to \alpha$ where ξ is either $x_i^=$ or $x_i^{\neq}$ and ψ' does not refer to x_i as follows: Introduce a new nonterminal symbol A'. Then, replace the rule with $(A, \xi, x_i) \to A'$ and $(A', x_i^= \wedge \psi') \to \alpha$.

After the above transformations, the guard expression of every rule updating a register only refers to a single register. Hence, in the following, we only consider the rules without updating registers.

The guard expression of a rule obtained in the above step 2 has the form of $x_j^= \wedge \psi$ for some $j \in [k]$. We can assume that every guard expression of the grammar that refers to more than two registers has this form without loss of generality, because we can replace a rule $(A, \psi) \to \alpha$ where ψ is not in this form with the following rules:

$$\begin{aligned} (A, x_j^= \wedge \psi|_{x_j^= \leftarrow \mathrm{tt}}) &\to \alpha \quad \text{for } 1 \leq j \leq k, \\ (A, \psi|_{x_1^= \leftarrow \mathrm{ff}, \ldots, x_k^= \leftarrow \mathrm{ff}}) &\to \alpha. \end{aligned}$$

Note that $\psi|_{x_1^= \leftarrow \mathrm{ff}, \ldots, x_k^= \leftarrow \mathrm{ff}}$ refers to no register and is either tt or ff. This transformation is based on the expansion of ψ into $(x_1^= \wedge \psi|_{x_1^= \leftarrow \mathrm{tt}}) \vee \ldots \vee (x_k^= \wedge \psi|_{x_k^= \leftarrow \mathrm{tt}}) \vee (x_1^{\neq} \wedge \ldots \wedge x_k^{\neq} \wedge \psi|_{x_1^= \leftarrow \mathrm{ff}, \ldots, x_k^= \leftarrow \mathrm{ff}})$. We can replace the last conjunction with $\psi|_{x_1^= \leftarrow \mathrm{ff}, \ldots, x_k^= \leftarrow \mathrm{ff}}$ because for any $\theta \in \Theta_k$, there must exist $d \in D$ that satisfies $\theta, d \models x_1^{\neq} \wedge \ldots \wedge x_k^{\neq}$, and the rule does not keep d in a register.

Then, for each rule whose guard expression refers to more than two registers, we repeatedly apply the following equivalence transformations to the rule. Without loss of generality, we assume that the guard expression of the rule is either $x_j^= \wedge (\psi_1 \wedge \psi_2)$ or $x_j^= \wedge (\psi_1 \vee \psi_2)$ for some $j \in [k]$.

- Transform a rule $(A, x_j^= \wedge (\psi_1 \wedge \psi_2)) \to \alpha$ as follows: Introduce a new nonterminal symbol A'. Then, replace the above rule with $(A, x_j^= \wedge \psi_1) \to A'$ and $(A', x_j^= \wedge \psi_2) \to \alpha$.

- Replace a rule $(A, x_j^= \wedge (\psi_1 \vee \psi_2)) \to \alpha$ with $(A, x_j^= \wedge \psi_1) \to \alpha$ and $(A, x_j^= \wedge \psi_2) \to \alpha$.

The above transformation does not change the language generated by the grammar. Moreover, the transformation polynomially preserves the description size of the grammar.

Corollary 2. $\mathcal{L}_{\text{2-bounded-guard-}k\text{-RCFG}} = \mathcal{L}_{k\text{-RCFG}}$ *for any* $k \geq 2$.

Proof Obvious from Proposition 6.

Proposition 7. *If a* 0*-bounded-guard* k*-RCFG* G *generates a data word of length* n*, then* G *generates a data word of length* n *whose data part contains only* $\perp$.

Proof Let G be a 0-bounded-guard k-RCFG and $w \in L(G)$ with $|w| = n$. We can assume that every guard expression of the rules of G is tt without loss of generality. Because every guard expression is tt, we can choose $\perp$ as an input data value at every derivation step of G. Hence, for a given derivation $(S, \theta_\perp) \overset{*}{\Rightarrow}_G w$, by replacing the input data value with $\perp$ at every derivation step, we obtain another derivation in which every assignment is $\theta_\perp$ and the derived data word w' only contains $\perp$ in its data part.

Corollary 3. $\mathcal{L}_{\text{0-bounded-guard-}k\text{-RCFG}} \subsetneq \mathcal{L}_{\text{1-bounded-guard-}k\text{-RCFG}}$ *for any* $k \geq 1$.

Proof We can construct a 1-bounded-guard k-RCFG G such that $L(G) = \{(a, d) \mid d \neq \perp\}$. In fact, we can let $G = (V, R, S)$ where $V = \{S\}$ and $R = \{(S, x_1^{\neq}, x_1) \to (a, x_1)\}$. Because every data word in $L(G)$ contains a data value different from $\perp$, by Proposition 7, there is no 0-bounded-guard k-RCFG that generates $L(G)$.

Proposition 8. *If a* 1*-bounded-guard* k*-RCFG* G *generates a data word of length* n*, then* G *generates a data word of length* n *that contains at most two different data values.*

Proof Let $G = (V, R, S)$ be a 1-bounded-guard k-RCFG and $w \in L(G)$ with $|w| = n$. Also let $d_1 \in D$ be an arbitrary data value that is different from $\perp$. Without loss of generality, we can assume that every guard expression of the rules of G is either tt, $x_j^{=}$ or $x_j^{\neq}$ for some $j \in [k]$. Consider a direct derivation of G:

$$(A, \theta) \Rightarrow_G X$$

where $A \in V$, $\theta \in \Theta_k$, $X \in ((V \times \Theta_k) \cup (\Sigma \times D))^*$ and $\theta(j) \in \{\perp, d_1\}$ for every j $(1 \leq j \leq k)$. Assume that a rule $(A, \psi, x_i) \to \alpha$ and an input data value d are applied in this derivation. We alter the derivation by choosing an input data value d' as follows:

- If $\psi = \text{tt}$, then $d' = \bot$.
- If $\psi = x_j^=$, then $d' = \theta(j)$.
- If $\psi = x_j^{\neq}$, then $d' \in \{\bot, d_1\} \setminus \theta(j)$.

Since the above d' satisfies $\theta, d' \models \psi$, we obtain another direct derivation $(A, \theta) \Rightarrow_G X'$ and X' contains at most two different data values. For a given derivation $(S, \theta_\bot) \overset{*}{\Rightarrow}_G w$, by repeating the above transformation to each derivation step, we obtain another derivation $(S, \theta_\bot) \overset{*}{\Rightarrow}_G w'$ with $|w'| = n$ and w' contains at most two different data values.

Corollary 4. $\mathcal{L}_{\text{1-bounded-guard-}k\text{-RCFG}} \subsetneq \mathcal{L}_{\text{2-bounded-guard-}k\text{-RCFG}}$ *for any* $k \geq 2$.

Proof We can construct a 2-bounded-guard k-RCFG G such that $L(G) = \{(a, d_1)(a, d_2)(a, d_3) \mid d_i \neq d_j$ for any $i, j \in [3]$ such that $i \neq j\}$. In fact, we can let $G = (V, R, S)$ where $V = \{S, T_1, T_2\}$ and R consists of the following rules:

$$\begin{aligned}(S, \text{tt}, x_1) &\to (a, x_1)\, T_1,\\ (T_1, x_1^{\neq}, x_2) &\to (a, x_2)\, T_2,\\ (T_2, x_1^{\neq} \wedge x_2^{\neq}, x_1) &\to (a, x_1).\end{aligned}$$

Because every data word in $L(G)$ contains three different data values, by Proposition 8, there is no 1-bounded-guard k-RCFG that generates $L(G)$.

4. Complexities of RCFG and Its Subclasses

In this section, we show the complexities of membership and emptiness problems for general, ε-rule free, growing and register-bounded RCFG. For readability, we divide this section into three subsections that discuss upper bounds, lower bounds and completeness in this order.

4.1. Upper Bounds

Lemma 1. *For the CFG G' constructed from a given k-RCFG G and a finite set $D' \subseteq D$ of data values by the construction of Proposition 1, we have $\|G'\| \in O(\|G\| \cdot |D''|^{k+1}k)$ where D'' is a subset of data values defined in the proof of Proposition 1.*

Proof Let $G = (V, R, S)$ be a k-RCFG and $D' \subseteq D$ be a finite subset of data values. The following properties hold on the size of the CFG $G' = (V', R', S')$ constructed in the proof of Proposition 1.

$$\begin{aligned}
&|V'| = |V \times D''^k| \in O(\|G\| \cdot |D''|^k).\\
&|R'| \leq |R| \cdot |D''|^k.\\
&\|R'\| = |R'| \max\{(|X| + 1)(\log |V| + k \log |D''| + \log |D'|) \mid (A, \theta) \to X \in R'\}\\
&\qquad \in O(\|G\| \cdot |D''|^k \cdot k \log |D''|).\\
&\|S'\| \in O(1).
\end{aligned}$$

Hence, $\|G'\| \in O(\|G\| \cdot |D''|^{k+1}k)$ holds.

Lemma 2. *The membership problem for RCFG is decidable in EXPTIME.*

Proof Assume we are given a k-RCFG G and a data word $w = (a_1, d_1) \dots (a_n, d_n)$ as an input. Consider the CFG G' such that $L(G) \cap (\Sigma \times \{d_1, \dots, d_n\})^*$, which is constructed in the proof of Proposition 3. By Lemma 1, $\|G'\| \in O(\|G\| c^{k+1} k)$ where $c = \max\{|w|, k + 1\}$. Since the membership problem for CFG is decidable in deterministic polynomial time, the problem $w \in L(G)$ for k-RCFG G is decidable in deterministic time exponential to k.

Lemma 3. *The membership problem for ε-rule free RCFG is decidable in PSPACE.*

Proof Let $G = (V, R, S)$ be an ε-rule free k-RCFG. Because G is ε-rule free, for any α such that $(S, \theta_\perp) \overset{*}{\Rightarrow}_G \alpha \overset{*}{\Rightarrow}_G w$, $|\alpha| \leq |w|$ holds. The space needed for representing α is at most $\|G\| \cdot k(\log c) \cdot |w|$ where $c = \max\{|w|, k+1\}$. We can check whether $w \in L(G)$ by nondeterministically guessing a derivation from S to w and checking step by step whether $(S, \theta_\perp) \overset{*}{\Rightarrow}_G w$ in polynomial space in $\|G\|$, $|w|$, and k.

Lemma 4. *The membership problem for growing RCFG is decidable in NP.*

Proof Let $G = (V, R, S)$ be a growing k-RCFG. Because G is a growing RCFG, for any α, α' such that $(S, \theta_\perp) \overset{*}{\Rightarrow}_G \alpha \Rightarrow_G \alpha' \overset{*}{\Rightarrow}_G w$, $1 \leq |\alpha| + \sigma(\alpha) < |\alpha'| + \sigma(\alpha') \leq 2|w|$ holds where $\sigma(\alpha)$ and $\sigma(\alpha')$ are the numbers of occurrence of terminal symbols in α and α', respectively. Therefore we can check whether $w \in L(G)$ by nondeterministically guessing a derivation from S to w, where the length of derivation is less than $2|w|$, and checking step by step whether $(S, \theta_\perp) \overset{*}{\Rightarrow}_G w$ in nondeterministical polynomial time in $\|G\|$, $|w|$, and k.

Lemma 5. *The emptiness problem for RCFG is decidable in EXPTIME.*

Proof Let G be a k-RCFG and $D_k = \{d_1, ..., d_{k+1}\} \subseteq D$. Assume that we construct the CFG G' from G and D_k such that $L(G) \cap (\Sigma \times D_k)^*$ according to the proof of Proposition 1. By Lemma 1, $\|G'\| \in O(\|G\| \cdot |D_k|^{k+1}k)$ holds. Because the emptiness problem for CFG is decidable in linear time, the problem of $L(G) = \emptyset$ for a k-RCFG is decidable in deterministic time exponential to k.

4.2. Lower Bounds

In this subsection, we first define *Alternating Turing machine* [23, 22] (abbreviated as ATM) and some notions for ATM for proving lower bounds.

An *alternating Turing machine* is a tuple $M = (Q, Q_e, Q_a, \Gamma, \Sigma, \delta, q_0, q_{acc}, q_{rej})$ where

- Q is a finite set of states, Q_e and Q_a are the set of *existential states* and the set of *universal states*, respectively, such that $Q_e \cup Q_a \cup \{q_{acc}, q_{rej}\} = Q$ and $Q_e, Q_a, \{q_{acc}, q_{rej}\}$ are mutually disjoint,
- Γ is a finite set of tape symbols containing a special symbol representing *blank*, $\sqcup \in \Gamma \backslash \Sigma$,
- $\Sigma \subseteq \Gamma$ a set of input symbols,
- $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ is a state transition function where $\mathcal{P}(A)$ denotes the powerset of a set A, and
- $q_0, q_{acc}, q_{rej} \in Q$ are the initial state, the accepting state and the rejecting state, respectively.

For a state $q \in Q$, a tape content $\alpha \in \Gamma^*$ and a head position j $(1 \leq j \leq |\alpha|)$, (q, α, j) is called an *instantaneous description* (abbreviated as ID) of M. For two IDs $(q, \alpha, j), (q', \alpha', j')$, we say that (q, α, j) can transit to (q', α', j') or (q', α', j') is a successor of (q, α, j), written as $(q, \alpha, j) \to (q', \alpha', j')$ if

$$\exists a, b \in \Gamma,\ \exists \beta, \gamma \in \Gamma^*,\ |\beta| = j - 1,\ \alpha = \beta a \gamma \text{ such that}$$

$$j' = \begin{cases} j - 1 & \delta(q, a) \ni (q', b, L), \text{ or} \\ j + 1 & \delta(q, a) \ni (q', b, R). \end{cases}$$

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

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

We define the set of accepting IDs $\mathrm{Acc} \subseteq Q \times \Gamma^* \times N$ of M as the smallest set that satisfies the following conditions.

- $(q_{acc}, \alpha, j) \in \mathrm{Acc}$ for any $\alpha \in \Gamma^*$ and $j \in N$
- For $q \in Q_e$, $(q, \alpha, j) \in \mathrm{Acc}$ if $g(q', \alpha', j') \in \mathrm{Acc}$ for some successor (q', α', j') of (q, α, j)
- For $q \in Q_a$, $(q, \alpha, j) \in \mathrm{Acc}$ if $g(q', \alpha', j') \in \mathrm{Acc}$ for all successor (q', α', j') of (q, α, j)

For an ATM $M = (Q, Q_e, Q_a, \Gamma, \Sigma, \delta, q_0, q_{acc}, q_{rej})$ and $u \in \Sigma^*$, M accepts u if and only if $(q_0, u, 1) \in \mathrm{Acc}$. Let $L(M) = \{u \in \Sigma^* \mid (q_0, u, 1) \in \mathrm{Acc}\}$, which is the language recognized by M. Let $s : \mathbb{N}_0 \to \mathbb{N}_0$ be a function. If for any $u \in \Sigma^*$ and any (q, α, j) such that $(q_0, u, 1) \xrightarrow{*} (q, \alpha, j)$, $|\alpha| \leq s(|u|)$ holds, then we say that M is an $s(n)$-space bounded ATM. If M is a $p(n)$-space bounded ATM for a polynomial $p(n)$, M is a polynomial space bounded ATM. It is well-known that APSPACE = EXPTIME where APSPACE is the class of languages accepted by polynomial space bounded ATM.

Lemma 6. *The membership problem for RCFG is EXPTIME-hard.*

Proof We show the EXPTIME-hardness by a polynomial-time reduction from the membership problem for polynomial-space bounded ATM. In the reduction, we simulate tape contents of a given ATM M by an assignment of the RCFG G constructed from M. For this purpose, we encode the state transition function of M by production rules of G.

Assume that we are given a $p(n)$-space bounded ATM $M = (Q, Q_e, Q_a, \Gamma, \Sigma, \delta, q_0, q_{acc}, q_{rej})$ where $p(n)$ is a polynomial and an input $u \in \Sigma^*$ to M. Then, we construct $(|\Gamma| + p(|u|))$-RCFG $G = (V, R, S)$ that satisfy $u \in L(M) \Leftrightarrow \varepsilon \in L(G)$, where

$$\begin{aligned} V = {} & \{T_{(i,j)} \mid 1 \leq j < i \leq |\Gamma|\} \cup \{T_{(1,0)}\} \\ & \cup \{W_i \mid 0 \leq i \leq |u|\} \\ & \cup \{A_q^{(i,j)} \mid q \in Q,\ 1 \leq i \leq |\Gamma|,\ 1 \leq j \leq p(|u|)\} \\ & \cup \{B_q^{(i,j)} \mid q \in Q,\ 1 \leq i \leq |\Gamma|,\ 1 \leq j \leq p(|u|)\} \\ & \cup \{C_q^{(i,j,k)} \mid q \in Q,\ 1 \leq i \leq |\Gamma|,\ 1 \leq j \leq p(|u|), \\ & \qquad 0 \leq k \leq \max_{q \in Q,\, a \in \Gamma} |\delta(q, a)|\}, \\ S = {} & T_{(1,0)}, \end{aligned}$$

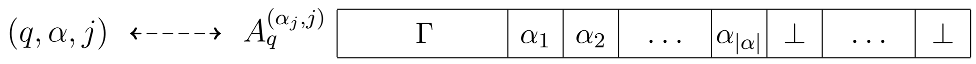


Figure 2: The correspondence between M's ID and G's nonterminal symbol and registers.

and R 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 production rules that load pairwise different data values in the first $|\Gamma|$ registers. Note that we keep the initial value $\perp$ in the first register. To the ith register ($i \geq 2$), 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).

$$(T_{(i-1,i-2)}, x_1^{\neq}, x_i) \to T_{(i,1)} \quad \text{for } 2 \leq i \leq |\Gamma|, \tag{1}$$
$$(T_{(i,j-1)}, x_i^{=} \wedge x_j^{\neq}) \to T_{(i,j)} \quad \text{for } 2 \leq j < i \leq |\Gamma|. \tag{2}$$

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

$$(T_{(|\Gamma|,|\Gamma|-1)}, \mathrm{tt}) \to W_0, \tag{3}$$
$$(W_{i-1}, x_{u_i}^{=}, x_{|\Gamma|+i}) \to W_i \quad \text{for } 1 \leq i \leq |u|. \tag{4}$$

- 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 G, we introduce a nonterminal symbol $A_q^{(i,j)}$ for $q \in Q$, $1 \leq i \leq |\Gamma|$, and $1 \leq j \leq p(n)$. $A_q^{(i,j)}$ 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 assignment of G. 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 $1 \leq j \leq |\alpha|$ and $\theta(|\Gamma| + j) = \perp$ for $|\alpha| < j \leq p(|u|)$. Let θ_α denote such an assignment that represents the tape contents α. We illustrate the correspondence between an ID of M and a nonterminal symbol and an assignment of G in Fig. 2.

- To derive the nonterminal symbol corresponding to the initial ID of M, we construct the following rule:

$$(W_{|u|}, \mathrm{tt}) \to A_{q_0}^{(u_1,1)}. \tag{5}$$

- Consider $A_q^{(i,j)}$ and let $\{(q_1, b_1, m_1), \dots, (q_t, b_t, m_t)\} = \delta(q, i)$. For each $a \in \Gamma$ and $1 \le k \le t$, we construct the following rules. If $q \in Q_e$, then:

$$(A_q^{(i,j)}, x_{b_k}^{=}, x_{|\Gamma|+j}) \to B_{q_k}^{(a,j+s(m_k))}. \tag{6}$$

If $q \in Q_a$, then:

$$(A_q^{(i,j)}, \mathrm{tt}) \to C_q^{(i,j,1)} \dots C_q^{(i,j,t)}, \tag{7}$$

$$(C_q^{(i,j,k)}, x_{b_k}^{=}, x_{|\Gamma|+j}) \to B_{q_k}^{(a,j+s(m_k))}. \tag{8}$$

Note that if $t = 0$, then the right-hand side of Rule (7) is ε. We also construct the following rule for each $q' \in Q$, $a \in \Gamma$, and $1 \le j' \le p(n)$:

$$(B_{q'}^{(a,j')}, x_a^{=} \wedge x_{|\Gamma|+j'}^{=}) \to A_{q'}^{(a,j')}. \tag{9}$$

- Finally, we construct the following rules to express accepting IDs.

$$(A_{q_{acc}}^{(i,j)}, \mathrm{tt}) \to \varepsilon \tag{10}$$

Next, we prove $u \in L(M) \Leftrightarrow \varepsilon \in L(G)$. For this purpose, we prove a more general property that for each ID (q, α, j),

$$(q, \alpha, j) \in \mathrm{Acc} \;\text{ iff }\; (A_q^{(\alpha_j,j)}, \theta_\alpha) \overset{*}{\Rightarrow}_G \varepsilon. \tag{11}$$

First we prove $(q, \alpha, j) \in \mathrm{Acc}$ implies $(A_q^{(\alpha_j,j)}, \theta_\alpha) \overset{*}{\Rightarrow}_G \varepsilon$ by induction on the application number n of the recursive definition of Acc.

- If $n = 0$, then by definition of Acc, we obtain $q = q_{acc}$ or ($q \in Q_a$ and $|\delta(q, \alpha_j)| = 0$). In this case, R contains $(A_q^{(\alpha_j,j)}, \mathrm{tt}) \to \varepsilon$ as Rule (10) or (7), and thus we can derive $(A_q^{(\alpha_j,j)}, \theta_\alpha) \Rightarrow_G \varepsilon$. Therefore, $(q, \alpha, j) \in$ Acc implies $(A_q^{(\alpha_j,j)}, \theta_\alpha) \overset{*}{\Rightarrow} \varepsilon$.

- We assume the implication holds when $n = h$ ($h \geq 0$) and consider the case of $n = h + 1$.

  - If $q \in Q_e$, then $(q, \alpha, j) \in \text{Acc}$ iff there exists (q', α', j') such that $(q, \alpha, j) \to (q', \alpha', j')$ and $(q', \alpha', j') \in \text{Acc}$. By $(q, \alpha, j) \to (q', \alpha', j')$ and the definitions of Rules (6) and (9), we obtain

$$(A_q^{(\alpha_j, j)}, \theta_\alpha) \Rightarrow_G (B_{q'}^{(\alpha'_{j'}, j')}, \theta_{\alpha'}) \Rightarrow_G (A_{q'}^{(\alpha'_{j'}, j')}, \theta_{\alpha'}).$$

    By the inductive hypothesis, we also obtain

$$(A_{q'}^{(\alpha'_{j'}, j')}, \theta_{\alpha'}) \overset{*}{\Rightarrow}_G \varepsilon.$$

    Therefore, $(q, \alpha, j) \in \text{Acc}$ implies $(A_q^{(\alpha_j, j)}, \theta_\alpha) \overset{*}{\Rightarrow}_G \varepsilon$.

  - If $q \in Q_a$, then $(q, \alpha, j) \in \text{Acc}$ iff $(q', \alpha', j') \in \text{Acc}$ for every (q', α', j') that satisfies $(q, \alpha, j) \to (q', \alpha', j')$. Let $\{(q_1, b_1, m_1), \ldots, (q_t, b_t, m_t)\} = \delta(q, \alpha_j)$ and let (q_i, α^i, j_i) be the successor of (q, α, j) defined by the tuple $(q_i, b_i, m_i) \in \delta(q, \alpha_j)$. Then, by the definitions of Rules (7) to (9), we obtain

$$(A_q^{(\alpha_j, j)}, \theta_\alpha) \overset{*}{\Rightarrow}_G (A_{q_1}^{(\alpha^1_{j_1}, j_1)}, \theta_{\alpha^1}) \ldots (A_{q_t}^{(\alpha^t_{j_t}, j_t)}, \theta_{\alpha^t}).$$

    By the inductive hypothesis, we also obtain

$$(A_{q_i}^{(\alpha^i_{j_i}, j_i)}, \theta_{\alpha^i}) \overset{*}{\Rightarrow}_G \varepsilon$$

    for $1 \leq i \leq t$. Therefore, $(q, \alpha, j) \in \text{Acc}$ implies $(A_q^{(\alpha_j, j)}, \theta_\alpha) \overset{*}{\Rightarrow}_G \varepsilon$.

The converse direction can be proved in a similar way. Now we prove $(A_q^{(\alpha_j, j)}, \theta_\alpha) \overset{*}{\Rightarrow}_G \varepsilon$ implies $(q, \alpha, j) \in \text{Acc}$ by induction on the length n of the derivation from $(A_q^{(\alpha_j, j)}, \theta_\alpha)$ to ε.

- Consider the case of $n = 1$. Then the applied rule must be either Rule (10) or (7), because there is no other rule whose right-hand side is ε. If the applied rule was Rule (10), then $q = q_{acc}$, and thus $(q, \alpha, j) \in \text{Acc}$. If the applied rule was Rule (7), then $q \in Q_a$ and $|\delta(q, \alpha_j)| = 0$, and also thus $(q, \alpha, j) \in \text{Acc}$.

- We assume the implication holds when $n = h$ ($h \geq 1$) and consider the case of $n = h + 1$.
  - If $q \in Q_e$, then by the definition of R, the derivation must be
  $$(A_q^{(\alpha_j,j)}, \theta_\alpha) \Rightarrow_G (B_{q'}^{(a,j')}, \theta') \Rightarrow_G (A_{q'}^{(a,j')}, \theta') \overset{*}{\Rightarrow}_G \varepsilon$$
  for some q', a, j', and θ'. By the definitions of Rules (6) and (9), there is an ID (q', α', j') such that $(q, \alpha, j) \to (q', \alpha', j')$, $\theta' = \theta_{\alpha'}$ and $a = \alpha'_{j'}$. Thus, by the inductive hypothesis, $(q', \alpha', j') \in \mathrm{Acc}$. By the definition of Acc, $(q, \alpha, j) \in \mathrm{Acc}$.
  - If $q \in Q_a$, then by the definition of R, the derivation must be
  $$(A_q^{(\alpha_j,j)}, \theta_\alpha) \Rightarrow_G (C_q^{(\alpha_j,j,1)}, \theta_\alpha) \dots (C_q^{(\alpha_j,j,t)}, \theta_\alpha) \overset{*}{\Rightarrow}_G \varepsilon$$
  where $t = |\delta(q, \alpha_j)|$, and thus $(C_q^{(\alpha_j,j,i)}, \theta_\alpha) \overset{*}{\Rightarrow}_G \varepsilon$ for $1 \leq i \leq t$. Let $\{(q_1, b_1, m_1), \dots, (q_t, b_t, m_t)\} = \delta(q, \alpha_j)$ and let (q_i, α^i, j_i) be the successor of (q, α, j) defined by the tuple $(q_i, b_i, m_i) \in \delta(q, \alpha_j)$. By the definition of R, the derivation from $(C_q^{(\alpha_j,j,i)}, \theta_\alpha)$ to ε must be
  $$(C_q^{(\alpha_j,j,i)}, \theta_\alpha) \Rightarrow_G (B_{q_i}^{(a,j_i)}, \theta') \Rightarrow_G (A_{q_i}^{(a,j_i)}, \theta') \overset{*}{\Rightarrow}_G \varepsilon$$
  for some a and θ'. By the definitions of Rules (8) and (9), $\theta' = \theta_{\alpha^i}$ and $a = \alpha^i_{j_i}$. By the inductive hypothesis, $(q_i, \alpha^i, j_i) \in \mathrm{Acc}$ for every (q_i, α^i, j_i) such that $(q, \alpha, j) \to (q_i, \alpha^i, j_i)$. Therefore, by the definition of Acc, $(q, \alpha, j) \in \mathrm{Acc}$.

By the construction of G, we can prove that $(S, \theta_\perp) \overset{*}{\Rightarrow}_G (A_{q_0}^{(u_1,1)}, \theta_u)$, and moreover, if $(S, \theta_\perp) \overset{*}{\Rightarrow}_G \varepsilon$, then this derivation must be $(S, \theta_\perp) \overset{*}{\Rightarrow}_G (A_{q_0}^{(u_1,1)}, \theta_u) \overset{*}{\Rightarrow}_G \varepsilon$. By letting $(q, \alpha, j) = (q_0, u, 1)$ in property (11) and by the above-mentioned fact, we obtain $u \in L(M) \Leftrightarrow (q_0, u, 1) \in \mathrm{Acc} \Leftrightarrow ((S, \theta_\perp) \overset{*}{\Rightarrow} \varepsilon) \Leftrightarrow \varepsilon \in L(G)$.

Lemma 7. *The membership problem for ε-rule free RCFG is PSPACE-hard.*

Proof We prove the PSPACE-hardness by a polynomial-time reduction from the membership problem for polynomial-space bounded ordinary Turing machines (TM), in a similar way to the proof of Lemma 6. A TM can be regarded as an ATM that has no universal states, and hence we do not need to construct ε-rules for a universal state that has no successor (i.e. we do not need Rule (7), whose right-hand side is ε if $t = 0$). We modify Rule (10), the ε-rule for the accepting state, as $(A^{(i,j)}_{q_{acc}}, \mathrm{tt}) \to (a, x_1)$. The resulting RCFG G is ε-rule free, and $u \in L(M) \Leftrightarrow (a, \perp) \in L(G)$ (because the data value in the first register is always $\perp$).

Lemma 8. *The membership problem for growing RCFG is NP-hard. This holds even for growing RCFG in which every guards is either tt or* $x_1^{=}$.

Proof We prove the NP-hardness by a polynomial-time reduction from the satisfiability problem for 3-Conjunctive Normal Form (3CNF). Let $\phi = (a_1 \vee b_1 \vee c_1) \dots (a_m \vee b_m \vee c_m)$ be a 3CNF over Boolean variables $y_1, \dots, y_n$ where each a_i, b_i, c_i $(1 \le i \le m)$ is a literal y_j or $\overline{y_j}$ for some j $(1 \le j \le n)$. For i $(1 \le i \le m)$, 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 $(2n+1)$-RCFG $G = (V, S, R)$ over $\Sigma = \{a\}$ where $V = \{S, A_{P_0}, \dots, A_{P_n}, A_{C_0}, \dots, A_{C_m}\}$ and

$$\begin{aligned} R = \{&(S, \mathrm{tt}, x_1) \to (a, x_1)A_{P_0}\} \\ &\cup \{(A_{P_{i-1}}, x_1^{=}, x_{2i+j}) \to (a, x_1)A_{P_i} \mid 1 \le i \le n,\ j \in \{0,1\}\} \\ &\cup \{(A_{P_n}, \mathrm{tt}) \to (a, x_1)A_{C_0}\} \\ &\cup \{(A_{C_{i-1}}, \mathrm{tt}) \to (a, x_r)A_{C_i} \mid 1 \le i \le m,\ r \in \{r_{a_i}, r_{b_i}, r_{c_i}\}\} \\ &\cup \{(A_{C_m}, \mathrm{tt}) \to (a, x_1)\}. \end{aligned}$$

The first register of the constructed RCFG G is used for keeping a data value (possibly) different from $\perp$, and we use that value and $\perp$ for representing tt and ff, respectively. G 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 $y_1, \dots, y_n$. Then G outputs the value of one of the literals a_i, b_i, c_i for each clause $a_i \vee b_i \vee c_i$ in ϕ.

We can show ϕ is satisfiable iff $(a, d)^{n+m+3} \in L(G)$, where d is an arbitrary data value in $D \setminus \{\perp\}$.

(If part) Assume $(S, \theta_\perp) \overset{*}{\Rightarrow} (a, d)^{n+m+3}$. Then there exists $\theta \in \Theta_k$ that satisfies $(S, \theta_\perp) \overset{*}{\Rightarrow} (a, d)^{n+2}(A_{C_0}, \theta) \overset{*}{\Rightarrow} (a, d)^{n+m+3}$. Let $I = \{i \in [2n+1] \setminus \{1\} \mid \theta(i) = d\}$. Note that $|I| = n$ always holds. Because $(A_{C_0}, \theta) \overset{*}{\Rightarrow} (a, d)^{m+1}$, at least one of $r_{a_i} \in I$, $r_{b_i} \in I$ and $r_{c_i} \in I$ hold for all $i \in [m]$. By the definition of r_{a_i}, r_{b_i} and r_{c_i}, we can obtain the truth value assignment

$\nu : \{y_1, \ldots, y_n\} \to \{\text{tt}, \text{ff}\}$ that satisfies ϕ as follows: for all i, $\nu(y_i) = \text{tt}$ if $2i \in I$ and $\nu(y_i) = \text{ff}$ if $2i + 1 \in I$.
(Only if part) Assume ϕ is satisfiable. Then there exists a truth value assignment $\nu : \{y_1, \ldots, y_n\} \to \{\text{tt}, \text{ff}\}$. Let $I = \{2i \mid i \in [n], \nu(y_i) = \text{tt}\} \cup \{2i + 1 \mid i \in [n],\ \nu(y_i) = \text{ff}\}$. Because $2i \in I \Leftrightarrow 2i + 1 \notin I$ for all $i \in [n]$, there exists $\theta \in \Theta_k$ and $d \in D$ such that $(S, \theta_\perp) \overset{*}{\Rightarrow} (a, d)^{n+2}(A_{C_0}, \theta)$ holds. By the definition of r_{a_i}, r_{b_i} and r_{c_i}, at least one of $\theta(r_{a_i}) = d$, $\theta(r_{b_i}) = d$ and $\theta(r_{c_i}) = d$ hold for all $i \in [m]$. Hence, $(A_{C_0}, \theta) \overset{*}{\Rightarrow} (a, d)^{m+1}$ holds, and thus we obtain $(S, \theta_\perp) \overset{*}{\Rightarrow} (a, d)^{n+m+3}$.

Since $(a, d_1)^{n+m+3} \in L(G)$ iff $(a, d_2)^{n+m+3} \in L(G)$ for any $d_1, d_2 \in D \setminus \{\perp\}$, we can choose any $d \in D \setminus \{\perp\}$ for constructing the input data word $(a, d)^{n+m+3}$ for the membership problem.

Hence, we have shown the NP-hardness of the problem.

Lemma 9. *The emptiness problem for RCFG is EXPTIME-hard, even if RCFG are restricted to be growing.*

Proof Let $a \in \Sigma$ be an arbitrary terminal symbol. In the proof of Lemma 6, we construct ε-rules when the state q under consideration is a universal state that has no successor (see Rule (7)) or q is the accepting state (see Rule (10)). We modify those production rules (7) and (10) as follows:

$$\begin{aligned} &(A_q^{(i,j)}, \text{tt}) \to (a, x_1) \quad \text{if } \delta(q, i) = \emptyset \text{ and } q \in Q_a, \\ &(A_{q_{acc}}^{(i,j)}, \text{tt}) \to (a, x_1). \end{aligned}$$

Also, a unit rule, say $A \to B$ ($A, B \in V$) can be replaced with $A \to (a, x_1)B$. With this modification, the constructed RCFG G has neither ε-rule nor unit rule, and $L(G)$ is nonempty iff the given ATM M accepts the input u. Therefore, we reduced the membership problem for polynomial-space bounded ATM to the emptiness problem for growing RCFG in polynomial time. Note that we cannot use this construction for proving Lemma 6 because the length of a (shortest) word in $L(G)$ is not guaranteed to be a polynomial of the sizes of M and u. □

As shown in Lemma 8, the membership problem for RCFG is NP-hard even if RCFG is restricted to have guards referring to at most one register. In contrast, the emptiness problem for RCFG with guards referring to at most one register is in P, as shown in the next theorem.

Theorem 1. *The emptiness problem for RCFG with guards referring to at most one register is decidable in linear time.*

Proof If a guard expression ψ refers to at most one register, then for every assignment θ, there must be a data value d that satisfies $\theta, d \models \psi$. That is, regardless of θ, rule $(A, \psi, x_i) \to \alpha$ or $(A, \psi) \to \alpha$ can be applied to expanding (A, θ).

Now, for a given RCFG G, let G' be the CFG obtained from G by removing the guard expression and register numbers in each production rule (e.g., replacing $(A, \psi, x_i) \to B(a, x_j)$ with $A \to Ba$). For a string $X \in ((V \times \Theta_k) \cup (\Sigma \times D))^*$, let $r(X) \in (V \cup \Sigma)^*$ be the string obtained from X by removing the assignments and the data values. By the discussion in the previous paragraph, if $X \Rightarrow_G Y$, then $r(X) \Rightarrow_{G'} r(Y)$, and if $r(X) \Rightarrow_{G'} Y'$ for some Y', then $X \Rightarrow_G Y$ for some Y such that $r(Y) = Y'$. Therefore $L(G) \neq \emptyset$ iff $L(G') \neq \emptyset$. Since $\|G'\| = O(\|G\|)$ and the emptiness problem for CFG is decidable in linear time, the emptiness problem for RCFG with guards referring to at most one register is also decidable in linear time.

4.3. Complexity Results

Theorem 2. *The membership problem for RCFG is EXPTIME-complete.*

Proof By Lemmas 2 and 6.

Theorem 3. *The membership problem for ε-rule free RCFG is PSPACE-complete.*

Proof By Lemmas 3 and 7.

Theorem 4. *The membership problem for growing RCFG is NP-complete. This holds even for growing RCFG in which every guards is either tt or $x_1^=$.*

Proof By Lemmas 4 and 8.

Theorem 5. *The emptiness problems for general, ε-rule free and growing RCFG are EXPTIME-complete.*

Proof By Lemmas 5 and 9.

Theorem 6. *The membership problem and emptiness problem for RCFG with bounded registers are in P. The data complexity of the membership problem for general RCFG is in P.*

Proof Let G be a k-RCFG over Σ and D and G' be the CFG constructed as in the proof of Proposition 1 from G and a finite set $D' \subseteq D$. Then $\|G'\| = O(\|G\| \cdot |D''|^{k+1}k)$ holds by Lemma 1. If k is a constant independent of the choice of G, then $\|G'\|$ is a polynomial of $\|G\|$ and $|D''|$. Hence, by Lemma 2 and Lemma 5, both of the membership problem and the emptiness problem for RCFG with bounded registers are in P. By the same reason, the data complexity of the membership problem for general RCFG is in P.

5. Register Pushdown Automata

5.1. Definitions

Register pushdown automata (abbreviated as RPDA) were originally defined in [10] as pushdown automata with registers over an infinite alphabet, and the equivalence between RCFG and RPDA was shown in [10]. In this section, we define RPDA as pushdown automata that take a data word as an input for the same reason as RCFG. We also present equivalence translations between RCFG and RPDA in this section for making the paper self-contained and for proving the computational complexity of the decision problems for RPDA and its subclasses defined later by reducing to/from RCFG and its subclasses. Due to guard expressions, which enable us to directly specify the equality among an input data value and data values in registers, the translation from RPDA to RCFG in this paper is simpler than the original one in [10] where the translation is divided in three steps by using M-grammar and simple M-grammar as intermediate models.

Definition 1. *A k-register pushdown automaton (k-RPDA) over a finite alphabet Σ and a set D of data values is a tuple $P = (Q, q_0, \delta)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, $\perp \in D$ is the initial data value at the bottom of a stack as well as the initial value of registers (note that D is used as a stack alphabet), and δ is a finite set of transition rules having one of the following forms:*

- $(p, \psi, x_i) \xrightarrow{a} (q, \varepsilon)$ *(or* $(p, \psi) \xrightarrow{a} (q, \varepsilon)$*) (pop rule)*
- $(p, \psi, x_i) \xrightarrow{a} (q, x_{j_1})$ *(or* $(p, \psi) \xrightarrow{a} (q, x_{j_1})$*) (replace rule)*

- $(p, \psi, x_i) \xrightarrow{a} (q, x_{j_1} x_{j_2})$ *(or* $(p, \psi) \xrightarrow{a} (q, x_{j_1} x_{j_2})$*) (push rule)*

where $p, q \in Q$, $a \in (\Sigma \cup \{\varepsilon\})$, $x_i, x_{j_1}, x_{j_2} \in X_k$, *and* $\psi \in F'_k$ *where* F'_k *is the set of guard expressions defined in the same way as* F_k, *except adding* $top^=$ *as an atomic proposition.*

For a state $q \in Q$, *an 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 P. 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') *or* (q', θ', w', u') *is a successor of* (q, θ, w, u), *written as* $(q, \theta, w, u) \Rightarrow_P (q', \theta', w', u')$ *if there exists a rule* $(q, \psi, x_i) \xrightarrow{a} (q', J) \in \delta$ *(resp.* $(q, \psi) \xrightarrow{a} (q', J)$*), data values* $d, e \in D$ *and* $\gamma \in D^*$ *such that*

$$\begin{aligned} &\theta, d, e \models \psi, \quad \theta' = \theta[x_i \leftarrow d] \text{ (resp. } \theta' = \theta\text{)},\\ &w = (a, d)w' \text{ if } a \in \Sigma,\ w = w' \text{ otherwise},\ u = ey,\ \text{and}\\ &u' = \begin{cases} y & J = \varepsilon,\\ \theta'(j_1)y & J = x_{j_1},\ \text{or}\\ \theta'(j_1)\theta'(j_2)y & J = x_{j_1}x_{j_2} \end{cases}\end{aligned}$$

where $\theta, d, e \models top^=$ *iff* $d = e$ *(d and e are supposed to be the input data value and the data value at the stack top, respectively).*

Let $\overset{*}{\Rightarrow}_P$ *be the reflexive transitive closure of* $\Rightarrow_P$. *We abbreviate* $\Rightarrow_P$ *and* $\overset{*}{\Rightarrow}_P$ *as* $\Rightarrow$ *and* $\overset{*}{\Rightarrow}$ *if P is clear from the context.*

A rule is called an ε*-rule if* $a = \varepsilon$. *The description length of a k-RPDA* $P = (Q, q_0, \delta)$ *is defined as* $\|P\| = |Q| + |\delta|(2 \log |Q| + \|\psi\| + 3 \log(k+1) + \log(|\Sigma| + 1))$, *where* $\|\psi\|$ *is the description length of* ψ, *defined in the same way as in Section 2.1 except adding the definition* $\|top^=\| = 1$. *In this definition, we assume that the description length of* $r = (p, \psi, x_i) \xrightarrow{a} (q, J) \in \delta$ $(0 \le |J| \le 2)$ *is* $2 \log |Q| + \|\psi\| + 3 \log(k+1) + \log(|\Sigma| + 1)$ *because there are two states* $p, q \in Q$, *a guard expression* $\psi \in F'_k$, *at most three register indexes i and those in J, and* $a \in \Sigma \cup \{\varepsilon\}$ *in r.*

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

5.2. Polynomial-time convertibility with RCFG

In this subsection, we show polynomial-time convertibility between RCFG and RPDA (Theorem 7). The complexities of membership and emptiness problems for RPDA are obtained straightforwardly from the theorem.

Theorem 7. *RCFG and RPDA are convertible each other, in polynomial time.*

Proof It suffices to prove the following two properties.

- For a given k-RCFG G, we can construct an $O(k)$-RPDA P_G such that $L(G) = L(P_G)$ in polynomial time.
- For a given k-RPDA P, we can construct an $O(k)$-RCFG G_P such that $L(P) = L(G_P)$ in polynomial time.

Let $G = (V, R, S)$ be a k-RCFG over Σ and D. We construct $(k + |\Sigma| + |V|)$-RPDA $P_G = (Q, q_0, \delta)$ where

$$Q = \{q_0\} \cup \{q_i \mid i \in [|\Sigma| + |V|]\} \cup \{q_{read}\} \cup \\ \{q_a \mid a \in \Sigma\} \cup \{q_r^i \mid r \in R, i \in [k]\}$$

and δ is constructed as follows.

- We construct transition rules that load pairwise different data values in the last $|\Sigma| + |V|$ registers:

  $$(q_{i-1}, \bigwedge_{j=1}^{i-1} x_j^{\neq}, x_{k+i}) \xrightarrow{\varepsilon} (q_i, x_1) \quad i \in [|\Sigma| + |V|]. \tag{12}$$

  The content of the $(k+j)$th register encodes either the j-th element of Σ (if $j \in [|\Sigma|]$) or the $(j - |\Sigma|)$-th element of V (if $|\Sigma| < j \leq (|\Sigma| + |V|)$) for some total orderings on Σ and V. We write the index of the register whose content encodes $a \in \Sigma$ as $\gamma(a)$ (and $A \in V$ as $\gamma(A)$ respectively).

- We construct P_G that simulates leftmost derivations in G. More concretely, the stack has information on the remaining string that does not yet match the scanned prefix of an input word in the leftmost derivation (including assignments associated with nonterminals), and P_G decides how it should move according to the stack top value.

  In the following, we will use a rule in the form of

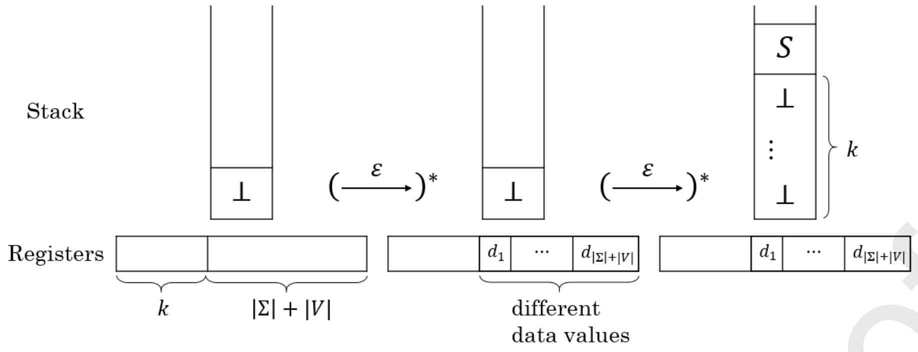


Figure 3: The changes of the stack and registers of P_G by applying the transition rules (12) and (13) in this order.

- $(p, \psi, x_i) \xrightarrow{a} (q, x_{j_1} \cdots x_{j_n})$ $(n \geq 3)$ (push$^+$ rule)

because this rule can be decomposed into the following two rules:

- $(p, \psi, x_i) \xrightarrow{a} (p', x_{j_2} \cdots x_{j_n})$,
- $(p', tt) \xrightarrow{\varepsilon} (q, x_{j_1} x_{j_2})$.

By repeating this decomposition, finally we can obtain push rules that have the same behavior as the push$^+$ rule.

First, P_G changes its state to q_{read} by the rule below and pushes the data value that encodes S and k initial data values as the initial assignment associated with S (Figure 3).

$$(q_{(|\Sigma|+|V|)}, tt) \xrightarrow{\varepsilon} (q_{read}, x_{\gamma(S)} x_1 \cdots x_k). \tag{13}$$

After that, P_G simulates a leftmost derivation from S step by step. The behavior of P_G is decided by the stack top value as follows. (An example of the simulation is in Figure 4.)

- If the stack top is equal to $x_{\gamma(A)}$ $(A \in V)$ and $r = (A, \psi, x_i) \to c_1 \cdots c_n \in R$, then P_G transfers k data values from the stack top to the first k registers, and pushes one dummy value x_1 that will be popped in the subsequent rule. Next, P_G pushes the symbols and data values in the registers corresponding to the right-hand side of r:

$$(q_{read}, x^{=}_{\gamma(A)} \wedge top^{=}) \xrightarrow{\varepsilon} (q_r^0, \varepsilon), \tag{14}$$

$$(q_r^{j-1}, top^{=}, x_j) \xrightarrow{\varepsilon} (q_r^j, \varepsilon) \quad j \in [k-1] \tag{15}$$

$$(q_r^{k-1}, top^{=}, x_k) \xrightarrow{\varepsilon} (q_r^k, x_1). \tag{16}$$

$$(q_r^k, \psi, x_i) \xrightarrow{\varepsilon} (q_{read}, J_1 \cdots J_n). \tag{17}$$

where $J_s = x_{\gamma(a)}x_h$ if $c_s = (a, x_h) \in (\Sigma \times X_k)$ and $J_s = x_{\gamma(c_s)}x_1 \cdots x_k$ if $c_s \in V$ for $s \in [n]$.

– If the stack top is equal to $x_{\gamma(a)}$ ($a \in \Sigma$) followed by $d \in D$, then P_G reads the next symbol and data value, say (a', d'), in the input word and if $a = a'$ and $d = d'$, it pops a and d from the stack:

$$(q_{read}, x^{=}_{\gamma(a)} \wedge top^{=}) \xrightarrow{\varepsilon} (q_a, \varepsilon), \tag{18}$$

$$(q_a, top^{=}) \xrightarrow{a} (q_{read}, \varepsilon). \tag{19}$$

We can show $L(G) \subseteq L(P_G)$ by induction on the length of a derivation of G and $L(P_G) \subseteq L(G)$ by induction on the length of an accepting run of P_G as shown in Lemma 10 described later.

Next, we give a converse translation, from RPDA to RCFG. Let $K = k+1$. Let $P = (Q, q_0, \delta)$ be a k-RPDA over Σ and D. Note that any replace rule can be simulated by a pair of a push rule and a pop rule in this order. Also we can remove any transition rule whose third component in its left-hand is missing

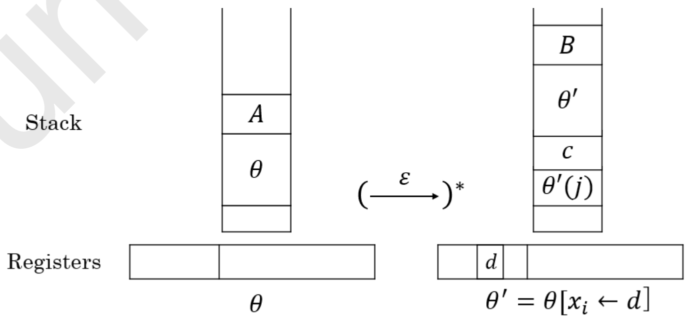

Stack
Registers
A
θ
$(\xrightarrow{\varepsilon})^*$
B
θ'
c
$\theta'(j)$
d
θ
$\theta' = \theta[x_i \leftarrow d]$

Figure 4: The simulation of the production rule $(A, \psi, x_i) \to B(c, x_j)$ of G by the transition rules (14) to (17) of P_G

(i.e., a rule that does not load an input data value into any register) by using one dummy register to store an input data value. Therefore, we assume P has no replace rule and every transition rule has the third component in its left-hand. We construct $3K$-RCFG $G_P = (V, R, S)$, where

$$\begin{aligned} V = &\{S\} \cup \{S_i \mid i \in [k-1]\} \cup \\ &\{[p,q], [p,q]'_r, [p,q]''_r, [p,q]^i_{new}, [p,q]^i_{left}, [p,q]^i_{right} \mid p,q \in Q, r \in \delta, 0 \le i \le K\} \\ &\cup \{E_i \mid i \in [k]\} \end{aligned}$$

and R is constructed as follows.

We intend that each nonterminal $[p,q]$ $(p,q \in Q)$ together with an assignment to $3K$ registers θ generates a data word w appearing in transitions $(p, \theta_p, w, d) \overset{*}{\Rightarrow} (q, \theta_q, \varepsilon, \varepsilon)$ of P where $\theta_p, \theta_q \in \Theta_k, w \in (\Sigma \times D)^*, d \in D$. In order to simulate these transitions, we construct production rules so that the first k registers represent θ_p, the K-th register stores the stack top d and the next k registers represent θ_q. (The last $(k+2)$ registers are used for temporary space for register transfer described later.)

First, we construct production rules that load arbitrary data values into the $(K+1)$-th to the $(K+k)$-th registers:

$$(S, \mathrm{tt}, x_{K+1}) \to S_1, \tag{20}$$
$$(S_{i-1}, \mathrm{tt}, x_{K+i}) \to S_i \quad (2 \le i \le k-1), \tag{21}$$
$$(S_{k-1}, \mathrm{tt}, x_{K+k}) \to [q_0, q] \quad (q \in Q). \tag{22}$$

For $\psi \in F'_k$, let $m(\psi)$ denote the guard expression obtained from ψ by replacing $top^=$ with $x^=_K$.

- For each $[p,q] \in V$ and a push rule $r = (p, \psi, x_j) \xrightarrow{a} (p', x_{j_1} x_{j_2}) \in \delta$, we construct the rule that divides the range of P's simulation by using an arbitrary state $t \in Q$ and corresponding registers, as $[p,q]$ to $[p',t][t,q]$. First, we construct the following rules that verify whether the guard expression ψ holds by using the data value in the K-th register as the stack top data value, and set a new stack top data value in the K-th register and the next stack data value in the $3K$-th register:

$$([p,q], m(\psi), x_j) \to (a, x_j)[p',q]'_r \text{ (if } a \in \Sigma) \text{ or } [p',q]'_r \text{ (if } a = \varepsilon), \tag{23}$$
$$([p',q]'_r, x^=_{j_1}, x_K) \to [p',q]''_r, \tag{24}$$
$$([p',q]''_r, x^=_{j_2}, x_{3K}) \to [p',q]^0_{new}. \tag{25}$$

Next, we construct the following rules to divide the range of P's simulation.

– We load k arbitrary data values into the $(2K+i)$-th registers $(i \in [k])$ that encode k registers of P associated with an arbitrary state $t \in Q$ and divide the nonterminal by the following rules:

$$([p', q]^{i-1}_{new}, \mathrm{tt}, x_{2K+i}) \to [p', q]^{i}_{new} \quad (i \in [k]), \tag{26}$$

$$([p', q]^{k}_{new}, \mathrm{tt}) \to [p', t]^{0}_{\mathit{left}}[t, q]^{0}_{\mathit{right}}. \tag{27}$$

– Next, we transfer K data values stored in the $(2K+i)$-th registers $(i \in [K])$ to the $(K+i)$-th registers associated with nonterminal $[p', t]$ that will simulate the left part. Also, we transfer the same K data values to the first K registers associated with nonterminal $[t, q]$ that will simulate the right part:

$$([p', t]^{i-1}_{\mathit{left}}, x^{=}_{2K+i}, x_{K+i}) \to [p', t]^{i}_{\mathit{left}} \quad (i \in [k]), \tag{28}$$

$$([p', t]^{k}_{\mathit{left}}, x^{=}_{3K}, x_{2K}) \to [p', t], \tag{29}$$

$$([t, q]^{i-1}_{\mathit{right}}, x^{=}_{2K+i}, x_{i}) \to [t, q]^{i}_{\mathit{right}} \quad (i \in [k]), \tag{30}$$

$$([t, q]^{k}_{\mathit{right}}, x^{=}_{3K}, x_{K}) \to [t, q]. \tag{31}$$

• For each $[p, q] \in V$ and a pop rule $r = (p, \psi, x_j) \xrightarrow{a} (q, \varepsilon) \in \delta$, we construct the following rules:

$$([p, q], m(\psi), x_j) \to (a, x_j)E_0 \text{ (if } a \in \Sigma) \text{ or } E_0 \text{ (if } a = \varepsilon), \tag{32}$$

$$(E_{i-1}, x^{=}_{i} \wedge x^{=}_{K+i}) \to E_i \quad (i \in [k-1]), \tag{33}$$

$$(E_{k-1}, x^{=}_{k} \wedge x^{=}_{K+k}) \to \varepsilon. \tag{34}$$

These production rules simulate the above pop rule r by guessing the assignment when the state becomes q. If the guessed assignment for the first k registers and the assignment associated with q in the $(K+1)$-th to the $(K+k)$-th registers are the same, (a, j) is generated.

We can show $L(P) \subseteq L(G_P)$ by induction on the length of an accepting run of P, and $L(G_P) \subseteq L(P)$ by induction on the length of a derivation of G_P as shown in Lemma 11 described later.

Lemma 10. *Let $G = (V, R, S)$ be a given k-RCFG and $P_G = (Q, q_0, \delta)$ be the κ-RPDA constructed from G in the proof of Theorem 7 where $\kappa = k + |\Sigma| + |V|$. Then, $L(G) = L(P_G)$ holds.*

Proof. By the construction of P_G, the contents of $(k+1)$-th to κ-th registers do not change once after the state becomes q_{read} in any accepting run of P_G. Hence, we write $\iota(A)$ and $\iota(a)$ to denote $\theta(\gamma(A))$ and $\theta(\gamma(a))$ for any $\theta \in \Theta_\kappa$ and $A \in V$, $a \in \Sigma$.

Let $\sigma_\perp, \sigma_S \in \Theta_\kappa$ such that $\sigma_\perp(i) = \perp$ $(i \in [\kappa])$ and $\sigma_S(i) = \perp$ $(i \in [k])$, $\sigma_S(i) \neq \sigma_S(j)$ $(k+1 \leq i, j \leq \kappa,\ i \neq j)$.

By (12) and (13),

$$(q_0, \sigma_\perp, \varepsilon, \perp) \overset{*}{\Rightarrow}_{P_G} (q_{read}, \sigma_S, \varepsilon, \iota(S)\perp^k). \tag{35}$$

We will show the following claim holds. Let $\Rightarrow_{G,lm}$ and $\overset{*}{\Rightarrow}_{G,lm}$ denote the one-step leftmost derivation relation defined in the same way as that of CFG and its transitive closure, respectively.

Claim 1. *$(A, \theta_A) \overset{*}{\Rightarrow}_{G,lm} (a_1, d_1) \cdots (a_h, d_h)X$ iff*

$$(q_{read}, \sigma_A, (a_1, d_1) \cdots (a_h, d_h), f(A, \sigma_A)) \overset{*}{\Rightarrow}_{P_G} (q_{read}, \sigma'_A, \varepsilon, f(X))$$

for some $\sigma_A, \sigma'_A \in \Theta_\kappa$ where $\sigma_A(i) = \theta_A(i)$ $(i \in [k])$ and $\sigma_A(i) \neq \sigma_A(j)$ $(k+1 \leq i, j \leq \kappa, i \neq j)$ and f is the homomorphism $f : ((\Sigma \times D) \cup (V \times \Theta_k))^* \to D^*$ defined by $f(a, d) = \iota(a)d$ $(a \in \Sigma,\ d \in D)$, $f(A, \theta) = \iota(A)\theta(1) \cdots \theta(k)$ $(A \in V,\ \theta \in \Theta_k)$, $f(\varepsilon) = \varepsilon$ and $f(\alpha\beta) = f(\alpha)f(\beta)$.

Proof of Claim 1. We prove the claim by double induction on h, and for the same h, on the length of the leftmost derivation in G.

(Case 1) Assume

$$(A, \theta_A) \overset{*}{\Rightarrow}_{G,lm} (a_1, d_1) \cdots (a_h, d_h)(B, \theta_B)X \Rightarrow_{G,lm} (a_1, d_1) \cdots (a_h, d_h)YX$$

whose last step is obtained by $d \in D$ and $(B, \psi, x_j) \to c_1 \cdots c_N \in R$ such that $\theta_B, d \models \psi$, $\theta'_B = \theta_B[x_j \leftarrow d]$, $Y = c'_1 \cdots c'_N$ where

$$c'_s = \begin{cases} (c_s, \theta'_B) & c_s \in V \\ (b, \theta'_B(l)) & c_s = (b, l) \in \Sigma \times [k] \end{cases} \quad (s \in [N]).$$

By the inductive hypothesis on the length of the leftmost derivation in G,

$$(q_{read}, \sigma_A, (a_1, d_1) \cdots (a_h, d_h), f(A, \sigma_A)) \overset{*}{\Rightarrow}_{P_G} (q_{read}, \sigma'_A, \varepsilon, f(B, \theta_B)f(X))$$

where $\sigma_A, \sigma'_A \in \Theta_\kappa$ are assignments declared in the claim. Note that $f(B, \theta_B) = \iota(B)\theta_B(1)\cdots\theta_B(k)$. By (14) – (17),

$$(q_{read}, \sigma'_A, \varepsilon, f(B, \theta_B)) \overset{*}{\Rightarrow}_{P_G} (q_{read}, \sigma''_A, \varepsilon, J'_1 \cdots J'_N)$$

where

$$J'_s = \begin{cases} \iota(c_s)d'_1 \cdots d'_k & c_s \in V \\ \iota(b)d'_l & c_s = (b, l) \in \Sigma \times [k] \end{cases} \quad (s \in [N]),$$

$$d'_i = \begin{cases} \theta_B(i) & i \neq j \\ d & i = j \end{cases} \quad \text{i.e., } d'_i = \theta'_B(i) \quad (i \in [k]).$$

Hence,

$$f(c'_s) = \begin{cases} f(c_s, \theta'_B) = \iota(c_s)\theta'_B(1)\cdots\theta'_B(k) = J'_s & c_s \in V \\ f(b, \theta'_B(l)) = \iota(b)\theta'_B(l) = J'_s & c_s = (b, l) \in \Sigma \times [k] \end{cases}$$

$(s \in [N])$. Therefore,

$$(q_{read}, \sigma_A, (a_1, d_1)\cdots(a_h, d_h), f(A, \sigma_A)) \overset{*}{\Rightarrow}_{P_G} (q_{read}, \sigma''_A, \varepsilon, f(YX))$$

because

$$f(YX) = f(Y)f(X) = f(c'_1)\cdots f(c'_N)f(X) = J'_1 \cdots J'_N f(X).$$

(Case 2) Assume

$$(A, \theta_A) \overset{*}{\Rightarrow}_{G,lm} (a_1, d_1)\cdots(a_h, d_h)(a_{h+1}, d_{h+1})X$$

with the same side condition as (Case 1). By the inductive hypothesis on h,

$$(q_{read}, \sigma_A, (a_1, d_1)\cdots(a_h, d_h), f(A, \sigma_A)) \overset{*}{\Rightarrow}_{P_G} (q_{read}, \sigma'_A, \varepsilon, f(a_{h+1}, d_{h+1})f(X)).$$

Since $f(a_{h+1}, d_{h+1}) = \iota(a_{h+1})d_{h+1}$,

$$(q_{read}, \sigma'_A, (a_{h+1}, d_{h+1}), f(a_{h+1}, d_{h+1})) \overset{*}{\Rightarrow}_{P_G} (q_{read}, \sigma'_A, \varepsilon, \varepsilon)$$

by (18),(19). Hence,

$$(q_{read}, \sigma_A, (a_1, d_1)\cdots(a_h, d_h)(a_{h+1}, d_{h+1}), f(A, \sigma_A)) \overset{*}{\Rightarrow}_{P_G} (q_{read}, \sigma'_A, \varepsilon, f(X))$$

and the claim holds, which completes the inductive proof. The inverse direction can be shown by induction on the length of an accepting run in P_G by traversing the above arguments in the reverse way.

(end of the proof of Claim 1)

By letting (A, θ_A), h and X be $(S, \theta_\perp)$, n and ε, respectively, in the claim, we obtain

$(S, \theta_\perp) \overset{*}{\Rightarrow}_{G,lm} (a_1, d_1) \cdots (a_n, d_n)$ iff

$$(q_{read}, \sigma_S, (a_1, d_1) \cdots (a_n, d_n), f(S, \sigma_S)) \overset{*}{\Rightarrow}_{P_G} (q_{read}, \sigma'_S, \varepsilon, \varepsilon).$$

By combining (35) and the left-hand side of the above statement,

$$\begin{aligned} &(q_0, \sigma_\perp, (a_1, d_1) \cdots (a_n, d_n), \perp) \\ &\quad \overset{*}{\Rightarrow}_{P_G} \quad (q_{read}, \sigma_S, (a_1, d_1) \cdots (a_n, d_n), f(S, \sigma_S)) \\ &\quad \overset{*}{\Rightarrow}_{P_G} \quad (q_{read}, \sigma'_S, \varepsilon, \varepsilon). \end{aligned}$$

Note that $f(S, \sigma_S) = \iota(S)\theta_\perp(1) \cdots \theta_\perp(k) = \iota(S)\perp^k$, which is the last component of the right-hand side of (35). Therefore,

$$w \in L(G) \text{ iff } w \in L(P_G)$$

and the lemma holds.

Lemma 11. *Let $P = (Q, q_0, \delta)$ be a given k-RPDA and $G_P = (V, R, S)$ be the $3K$-RCFG constructed from P in the proof of Theorem 7 where $K = k+1$. Then, $L(P) = L(G_P)$ holds.*

Proof. By (20) – (22),

$$(S, \theta_\perp) \overset{*}{\Rightarrow}_{G_P} ([q_0, q], \sigma) \tag{36}$$

for any $q \in Q$ and $\sigma \in \Theta_{3K}$ such that $\sigma(i) = \perp$ ($i \in [K]$ or $2K \le i \le 3K$). Furthermore, all the derivations from S to a data word must start with (36).

Claim 2. *$(p, \theta, w, d_0) \overset{*}{\Rightarrow}_P (q, \theta', \varepsilon, \varepsilon)$ iff $([p, q], \sigma) \overset{*}{\Rightarrow}_{G_P} w$ where $\sigma(i) = \theta(i)$ $(i \in [k])$, $\sigma(K) = d_0$, $\sigma(K + i) = \theta'(i)$ $(i \in [k])$.*

Proof of Claim 2. We prove the only if part by induction on the length of the accepting run in P.

Basis. Assume

$$(p, \theta, (a, d), d_0) \Rightarrow_P (q, \theta', \varepsilon, \varepsilon)$$

with a pop rule $(p, \psi, x_j) \xrightarrow{a} (q, \varepsilon) \in \delta$. By (32) – (34),

$$([p, q], \sigma) \overset{*}{\Rightarrow}_{G_P} (a, d)$$

where $\sigma(i) = \sigma(K + i)$ $(i \in [k],\ i \neq j)$ by (33), (34) and $\sigma(K + j) = d$ by (32). Hence, if we choose

$$\sigma(i) = \theta(i) \ \ (i \in [k]) \ \text{ and } \ \sigma(K) = d_0,$$

then we have

$$\sigma(K + i) = \sigma(i) = \theta(i) = \theta'(i)\ (i \in [k], i \neq j), \sigma(K + j) = d = \theta'(j)$$

and the claim holds.

Induction. Assume

$$(p, \theta, (a, d)w_1w_2, d_0) \Rightarrow_P (p', \theta_1, w_1w_2, \theta_1(j_1)\theta_1(j_2)) \overset{*}{\Rightarrow}_P (q, \theta_3, \varepsilon, \varepsilon)$$

where

$$\theta_1(i) = \theta(i)\ (i \in [k], i \neq j), \theta_1(j) = d \tag{37}$$

with a push rule $(p, \psi, x_j) \xrightarrow{a} (p', j_1j_2) \in \delta$, followed by

$$(p', \theta_1, w_1, \theta_1(j_1)) \overset{*}{\Rightarrow}_P (t, \theta_2, \varepsilon, \varepsilon)$$

and

$$(t, \theta_2, w_2, \theta_1(j_2)) \overset{*}{\Rightarrow}_P (q, \theta_3, \varepsilon, \varepsilon).$$

By the inductive hypothesis,

$$([p', t], \sigma_1) \overset{*}{\Rightarrow}_{G_P} w_1 \ \text{ and } \ ([t, q], \sigma_2) \overset{*}{\Rightarrow}_{G_P} w_2 \tag{38}$$

where

$$\sigma_1(i) = \theta_1(i)\ (i \in [k]), \sigma_1(K) = \theta_1(j_1), \sigma_1(K + i) = \theta_2(i)\ (i \in [k]), \tag{39}$$

$$\sigma_2(i) = \theta_2(i)\ (i \in [k]), \sigma_2(K) = \theta_1(j_2), \sigma_2(K + i) = \theta_3(i)\ (i \in [k]) \tag{40}$$

and therefore, $\sigma_1(K + i) = \sigma_2(i)$.

Let $\sigma \in \Theta_{3K}$ such that

$$\sigma(i) = \theta(i)\ (i \in [k]), \sigma(K) = d_0, \sigma(K + i) = \theta_3(i)\ (i \in [k]). \tag{41}$$

By (23) – (31),

$$([p,q],\sigma) \overset{*}{\Rightarrow}_{G_P} (a,d)([p',t],\sigma'_1)([t,q],\sigma'_2) \tag{42}$$

where

$$\begin{aligned}
\sigma'_1(i) &= \sigma(i)\ (i \in [k], i \neq j) \text{ by (23),(27)}, & (43)\\
\sigma'_1(j) &= d \text{ by (23),(27)}, & (44)\\
\sigma'_1(K) &= \sigma[x_j \leftarrow d](j_1)\ \text{ by (24),(27)}, & (45)\\
\sigma'_1(K+i) &= \sigma'_2(i)\ (i \in [k]) \text{ by (28),(30)}, & (46)\\
\sigma'_1(2K) &= \sigma'_2(K) = \sigma[x_j \leftarrow d](j_2) \text{ by (25),(29),(31)}, & (47)\\
\sigma'_2(K+i) &= \sigma(K+i)\ (i \in [k]) \text{ by (23),(27)}. & (48)
\end{aligned}$$

Note that the values of $\sigma'_1(K+i)$ and $\sigma'_2(i)$ $(i \in [k])$ are arbitrary as long as (46) is satisfied, and hence we choose

$$\sigma'_1(K+i) = \sigma'_2(i) = \theta_2(i)\ (i \in [k]). \tag{49}$$

Comparing (37), (39) – (41), (43) – (49), we obtain $\sigma_1(i) = \sigma'_1(i)$ and $\sigma_2(i) = \sigma'_2(i)$ $(i \in [2k+1])$. Hence, by (38) and (42),

$$([p,q],\sigma) \overset{*}{\Rightarrow}_{G_P} (a,d)w_1w_2$$

and the claim holds. By the induction, we complete the proof of the only if part. We can prove the if part by induction on the length of the derivation in G_P by traversing the above arguments backwards.

(end of the proof of Claim 2)

By letting p, θ and d_0 be q_0, $\theta_\perp$ and $\perp$, respectively, in Claim 2, we can obtain

$$(q_0,\theta_\perp,w,\perp) \overset{*}{\Rightarrow}_P (q,\theta',\varepsilon,\varepsilon) \text{ iff } ([q_0,q],\sigma) \overset{*}{\Rightarrow}_{G_P} w. \tag{50}$$

By combining (36) and (50),

$$(q_0,\theta_\perp,w,\perp) \overset{*}{\Rightarrow}_P (q,\theta',\varepsilon,\varepsilon) \text{ iff } (S,\theta_\perp) \overset{*}{\Rightarrow}_{G_P} w.$$

Thus, we obtain

$$w \in L(P) \text{ iff } w \in L(G_P)$$

and the lemma holds.

Corollary 5. *Membership and emptiness problems for RPDA are EXPTIME-complete.*

Proof By Theorem 7, we can use the reductions to and from corresponding problems for RCFG, which is EXPTIME-complete.

5.3. Non-decreasing and growing RPDA

We say that an RPDA P is *non-decreasing* if every ε-rule of P is either a replace rule or a push rule. We say that an RPDA P is *growing* if every ε-rule of P is a push rule.

Proposition 9. *Let $P = (Q, q_0, \delta)$ be a k-RPDA and $\rho = (q_0, \theta_\perp, w_0, \perp) \overset{*}{\Rightarrow} (q, \theta, w, u) \overset{*}{\Rightarrow} (q', \theta', \varepsilon, \varepsilon)$ be an accepting run of P. If P is non-decreasing, $|u| \le |w_0|$. If P is growing, $|\rho| \le 2|w_0| - 1$.*

Proof Consider k-RPDA P and the accepting run ρ of P given above.

Assume P is non-decreasing. Since every pop move in ρ is a non-ε move, the number of pop moves applied in ρ is at most $|w_0|$. This implies that $|u| \le |w_0|$ since otherwise the run cannot make the stack empty.

Assume P is growing. We see that $|\rho| = n_1 + n_2 + n_3$ where n_1, n_2, n_3 are the numbers of push, non-ε-replace and non-ε-pop moves, respectively. Since $n_2 + n_3 \le |w_0|$, we have $n_1 = n_3 - 1 \le |w_0| - 1$ and hence $|\rho| \le 2|w_0| - 1$. □

Theorem 8. *The membership problem for non-decreasing RPDA is PSPACE-complete.*

Proof By Proposition 9, it is enough to show PSPACE-hardness. We prove it by using a polynomial-time reduction from the membership problem for polynomial space bounded TM. For the ε-rule free RCFG $G = (V, R, S)$ constructed in the proof of Lemma 7, every production rule has a form either $(A, \psi, x_i) \to B$ or $(A, \mathrm{tt}) \to (a, x_1)$ for $A, B \in V$, $\psi \in F_k$ and $x_i \in X_k$. From such an ε-rule free RCFG G, we can construct an equivalent non-decreasing RPDA $A_G = (Q, q_0, \delta)$, where

- $Q = \{q_A \mid A \in V\}$,
- $q_0 = q_S$, and
- $\delta = \{(q_A, \psi, x_i) \xrightarrow{\varepsilon} (q_B, x_1) \mid (A, \psi, x_i) \to B \in R\}$
  $\cup \{(q_A, x_1^{=}) \xrightarrow{a} (q_S, \varepsilon) \mid (A, \mathrm{tt}) \to (a, x_1) \in R\}$.

Note that every stack data value, which is always equal to x_1, does not matter to the behavior of A_G. By the construction of A_G, we can show $L(A_G) = L(G)$. Hence, we obtain the polynomial-time reduction from such restricted ε-rule free RCFG G and also the polynomial-time reduction from the membership problem for polynomial space bounded TM.

Theorem 9. *The membership problem for growing RPDA is NP-complete.*

Proof By Proposition 9, it is enough to show NP-hardness. We prove it by using a polynomial-time reduction from the satisfiability problem for 3CNF. For the growing RCFG $G = (V, R, S)$ constructed in the proof of Lemma 8, every production rule has one of the following forms $(A, \mathrm{tt}, x_i) \to (a, x_i)B$, $(A, x_i^=, x_j) \to (a, x_i)B$, $(A, \mathrm{tt}) \to (a, x_i)B$ or $(A, \mathrm{tt}) \to (a, x_i)$ for $A, B \in V$ and $x_i, x_j \in X_k$. From such a growing RCFG G, we can construct an equivalent growing RPDA $\mathcal{A}_G = (Q, q_0, \delta)$, where

- $Q = \{q_A \mid A \in V\}$,
- $q_0 = q_S$, and
- $\delta = \{(q_A, \mathrm{tt}, x_i) \xrightarrow{a} (q_B, x_1) \mid (A, \mathrm{tt}, x_i) \to (a, x_i)B \in R\}$
  $\cup \{(q_A, x_i^=, x_j) \xrightarrow{a} (q_B, x_1) \mid (A, x_i^=, x_j) \to (a, x_i)B \in R\}$
  $\cup \{(q_A, x_i^=) \xrightarrow{a} (q_B, x_1) \mid (A, \mathrm{tt}) \to (a, x_i)B \in R\}$
  $\cup \{(q_A, x_i^=) \xrightarrow{a} (q_S, \varepsilon) \mid (A, \mathrm{tt}) \to (a, x_i) \in R\}$.

Note that as the same as the RPDA constructed in Theorem 8, every stack data value, which is always equal to x_1, does not matter to the behavior of $\mathcal{A}_G$. By the construction of $\mathcal{A}_G$, we can show $L(\mathcal{A}_G) = L(G)$. Hence, we obtain the polynomial-time reduction from such restricted growing RCFG G and also the polynomial-time reduction from the satisfiability problem for 3CNF.

For both of the RPDA constructed in the proofs of Theorem 8 and 9, the height of the stack appearing in any accepting run is at most one. Such RPDA are equivalent to register automata (RA) [5], and thus this fact implies the following property.

Corollary 6. *The membership problems for RA with and without ε-transition are PSPACE-complete and NP-complete, respectively.*

Note that NP-completeness of the membership problem for RA without ε-transition was first proved in [9].

Theorem 10. *The emptiness problems for non-decreasing RPDA and growing RPDA are both EXPTIME-complete.*

Proof By Corollary 5, 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

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

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.

5.4. Quasi non-decreasing and quasi growing RPDA

We have shown that computational complexities of the membership problems for ε-rule free RCFG and growing RCFG are of the same order as that for non-decreasing RPDA and growing RPDA, namely, PSPACE-complete and NP-complete, respectively. However, it is unknown whether the class of data languages generated by ε-rule free RCFG is the same as the class of data languages recognized by non-decreasing RPDA. Growing RCFG and growing RPDA are in the same situation. For example, it seems difficult to translate a given ε-rule free RCFG to a language-equivalent non-decreasing RPDA by the following reason: While each nonterminal A is associated with an assignment θ and each terminal a is associated with a data value d in a derivation, an RPDA cannot push or pop (A, θ) or (a, d) by a single transition; instead, the RPDA has to first pop a block of data values from its stack and then push another (sequence of) data values to the stack. Thus, it is unlikely to be able to simulate a derivation of RCFG, even if it is ε-rule free, without using ε-pop rules such as (14), (15) and (18). Remember that we use the homomorphism $f : ((\Sigma \times D) \cup (V \times \Theta_k))^* \to D^*$ defined by $f(a, d) = \iota(a)d$ and $f(A, \theta) = \iota(A)\theta(1) \cdots \theta(k)$. If we regard $f(A, \theta)$ and $f(a, d)$ as a single data value and consider the sequence of moves (called *block move* as defined later) that operates on $f(A, \theta)$ or $f(a, d)$ as a single move, then we can introduce subclasses of RPDA that directly correspond to ε-rule free RCFG and growing RCFG.

In the following, we will slightly generalize non-decreasing and growing RPDA. We assume there is no replace rule without loss of generality as already mentioned. Informally, we call a string over D a *block* if the string has some specified pattern. This pattern is formally represented by a code, which is a prefix-free injective function $f : \Delta \to D^+$ where Δ is a finite or an infinite alphabet. We assume that every accepting run ρ can be divided into sub-runs $\rho_i = (p_{i-1}, \theta_{i-1}, w_{i-1}, u_{i-1}) \overset{*}{\Rightarrow} (p_i, \theta_i, w_i, u_i)$ $(i \in [N])$ as $\rho = \rho_1; \rho_2; \cdots; \rho_N$ where the contents of the stack u_i of the boundary ID $(p_i, \theta_i, w_i, u_i)$ can be segmented into blocks. We regard the moves in the sub-run as a kind of macro move, called a *block move* and apply the constraints of non-decreasing and growing RPDA to block moves. We give the formal definitions as follows.

Let Δ_1 and Δ_2 be finite or infinite alphabets. A function $f : \Delta_1 \to \Delta_2^+$ is called a uniquely decodable prefix-free code (or simply *code*) if f is an injection and f satisfies the prefix condition such that for every $v, v' \in \Delta_1$ $(v \neq v')$, $f(v)$ is not a proper prefix of $f(v')$ and there is a constant $c \in \mathbb{N}$ such that for every $v \in \Delta_1$, $|f(v)| \leq c$. We call $|f(v)|$ the length of the code word for v in f. f is extended to $f : \Delta_1^* \to \Delta_2^*$ by $f(\varepsilon) = \varepsilon$ and $f(x_1x_2) = f(x_1)f(x_2)$ for $x_1, x_2 \in \Delta_1^*$. A string $y \in \Delta_2^*$ is called a code string of f if there is $x \in \Delta_1^*$ such that $y = f(x)$. Note that if y is a code string of f, y is uniquely decomposed as $y = y_1 \cdots y_n$ where $y_j = f(x_j)$ for some $x_j \in \Delta_1$ $(j \in [n])$. Each y_j is called a *block* of y and we write $\|y\|_f = n$ or $\|y\| = n$ if f is understood.

Let $P = (Q, q_0, \delta)$ be a k-RPDA over Σ and D, and let Δ be an alphabet such that $\perp \in \Delta$. Also let $f : \Delta \to D^+$ be a code and $b \in \mathbb{N}_0$. P is *blocked* with respect to f and b if every accepting run of P can be written as

$$\rho = (p_0, \theta_0, w_0, u_0) \overset{*}{\Rightarrow} (p_1, \theta_1, w_1, u_1) \overset{*}{\Rightarrow} \cdots \overset{*}{\Rightarrow} (p_N, \theta_N, w_N, u_N)$$

where $p_0 = q_0$, $\theta_0 = \theta_\perp$, $u_0 = \perp$, $w_N = u_N = \varepsilon$ and for each $i \in [N]$, u_i is a code string of f and the i-th sub-run $\rho_i = (p_{i-1}, \theta_{i-1}, w_{i-1}, u_{i-1}) \overset{*}{\Rightarrow} (p_i, \theta_i, w_i, u_i)$ satisfies one of the following conditions. We call N the block length of ρ and write $\langle\langle \rho \rangle\rangle = N$.

- There is $v \in \Delta$ such that $u_{i-1} = f(v)u_i$ and one of the following two conditions is satisfied.
  - All the moves in ρ_i are ε-pop moves. This sequence of moves is called a *block ε-pop move.*
  - One or more non-ε-pop moves exist in ρ_i and the other moves in ρ_i are ε-pop moves. This sequence of moves is called a *block non-ε-pop move.*
- There are $v, v' \in \Delta$ and $u'' \in D^*$ such that $u_{i-1} = f(v)u''$ and $u_i = f(v')u''$, and $|\rho_i| \leq b$. This sequence of moves in ρ_i is called a *block replace move.*

- There is $v \in \Delta$ such that $f(v)u_{i-1} = u_i$ and all the moves in ρ_i are push moves. This sequence of moves is called a *block push move.*

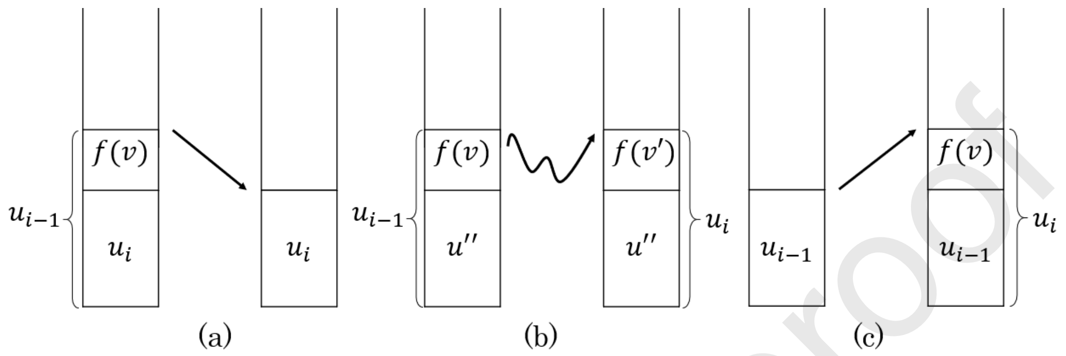


Figure 5: Illustrations of (a) block pop move, (b) block replace move and (c) block push move.

Definition 2. *A k-RPDA P is* quasi non-decreasing *if there exist $b \in \mathbb{N}_0$ and a code f such that for every accepting run ρ of P, P is blocked with respect to f and b and there is no block ε-pop move in ρ.*

A k-RPDA P is quasi growing *if P is quasi non-decreasing and in every accepting run of P, each of all the block replace moves except $O(n)$ moves is followed by a block push move where n is the length of an input data word. This pair of a block replace move and a block push move is called a* derived block push move.

Lemma 12. *Let $P = (Q, q_0, \delta)$ be a k-RPDA over Σ and D, blocked with respect to $f : \Delta \to D^+$ and $b \in \mathbb{N}_0$. Assume*

$$\rho = (q_0, \theta_\perp, w_0, \perp) \overset{*}{\Rightarrow} \cdots (p_i, \theta_i, w_i, u_i) \cdots \overset{*}{\Rightarrow} (p_N, \theta_N, \varepsilon, \varepsilon) \qquad (51)$$

is an accepting run of P and let $n = |w_0|$.

If P is quasi non-decreasing, $\|u_i\| \leq n$ and $|u_i| \in O(n)$ for $i \in [N]$. Also, the membership problem for quasi non-decreasing RPDA is in PSPACE.

If P is quasi growing, $\langle\langle\rho\rangle\rangle, |\rho| \in O(n)$. Also, the membership problem for quasi growing RPDA is in NP.

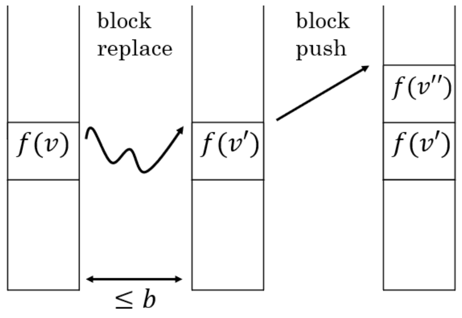


Figure 6: An illustration of derived block push move.

Proof Assume P is quasi non-decreasing. Then,

$$\|u_i\| \leq \text{ (the number of block non-}\varepsilon\text{-pop moves in } \rho \text{ (51))} \ \leq n$$

for $i \in [n]$. This is because there is no ε-pop move. (This property is similar to the first part of Proposition 9.) Next, we show that the membership problem for quasi non-decreasing RPDA is solvable in PSPACE. Assume we are given an input data word $w_0 \in (\Sigma \times D)^*$ and let $n = |w_0|$. As we just showed, if there is an accepting run ρ of w_0, then it is blocked as in (51) above with respect to a code $f : \Delta \to D^+$ and $b \in \mathbb{N}_0$, and $\|u_i\| \leq n$ for $i \in [n]$. Remember that by the definition of quasi non-decreasing RPDA, there is a constant $c \in \mathbb{N}$ such that for every $v \in \Delta$, $|f(v)| \leq c$. Hence, for some $v_1, \ldots, v_m \in \Delta$ ($m \in O(n)$), $u_i = f(v_1) \cdots f(v_m)$ and $|u_i| \leq cm \in O(n)$. Therefore, the accepting run ρ can be simulated in $O(n)$ space by a similar way to the one in the proof of Lemma 3.

Assume P is quasi growing and for a given data word w_0, there is an accepting run ρ of w_0, which is blocked with respect to a code f and $b \in \mathbb{N}_0$. Also assume that there is at most $c_1 n + c_2$ block replace moves that are not followed by a block push move in ρ. Let

$$\begin{aligned}
n_1 &= \text{the number of derived block push moves,} \\
n_2 &= \text{the number of block push moves not included in} \\
&\quad\ \text{any derived block push move,} \\
n_3 &= \text{the number of block non-}\varepsilon\text{-pop moves}
\end{aligned}$$

in ρ. Then, $\langle\langle\rho\rangle\rangle \leq (2n_1 + n_2) + (c_1 n + c_2) + n_3 = 2n_3 + n_1 + (c_1 n + c_2 - 1) \leq (3 + c_1)n + c_2 - 1$ because $n_3 = n_1 + n_2 + 1$. (This property is similar to the second part of Proposition 9.) Let $c \in \mathbb{N}$ be a constant such that for every $v \in \Delta$, $|f(v)| \leq c$. Then, $|\rho| = \max\{b, c\}\langle\langle\rho\rangle\rangle \in O(n)$. As in the proof of Lemma 4, the accepting run ρ can be simulated in NP.

Theorem 11. *The membership problem for quasi non-decreasing RPDA is PSPACE-complete.*

Proof PSPACE solvability holds by Lemma 12. Since the membership problem for ε-rule free RCFG is PSPACE-complete, it suffices to show that if we restrict RCFG to be ε-rule free in the translation from RCFG to RPDA in Theorem 7, the constructed RPDA is quasi non-decreasing. Remember that RPDA P_G constructed in the translation simulates a derivation of the given k-RCFG $G = (V, R, S)$ as follows.

- Represent a pair (a, d) ($a \in \Sigma$, $d \in D$) by two consecutive data values $x_{\gamma(a)}$ and d on the stack.
- Represent a pair (A, θ) ($A \in V$, $\theta \in \Theta_k$) by $(k+1)$ consecutive data values $x_{\gamma(A)}$ and $\theta(1), \ldots, \theta(k)$ on the stack.

We regard these representations as the encoding function $f : \Delta \to D^+$ where $\Delta = \{\bot\} \cup (\Sigma \times D) \cup (V \times \Theta_k)$ and $f(\bot) = \bot$, $f(a, d) = \iota(a)d$ ($a \in \Sigma$, $d \in D$) and $f(A, \theta) = \iota(A)\theta(1)\cdots\theta(k)$ ($A \in V$, $\theta \in \Theta_k$). We verify that RPDA P_G is quasi non-decreasing by examining the transition rules constructed in the translation as follows.

- The $(|\Sigma| + |V|)$ transition rules (12) for the register initialization contribute to block replace moves.
- The transition rule (13) for the start symbol S is a push rule.
- The transition rules for a rule $r = (A, \psi, i) \to c_1 \cdots c_n \in R$ are divided into two groups where the first group ((14) to push J_n of (17)) contributes to a block replace move and the second group (push $J_1 \cdots J_{n-1}$) of (17)) contributes to zero or more block push moves because G is ε-rule free and hence $n \geq 1$.
- The transition rules (18) and (19) for $a \in \Sigma$ contribute to a non-ε-pop move.

We can alternatively prove this lower-bound by the facts that non-decreasing RPDA is a subclass of quasi non-decreasing RPDA, and that the membership problem for the former class is PSPACE-complete (Theorem 8). The reason why we show the lower-bound by giving an equivalent transformation from ε-rule free RCFG to quasi non-decreasing RPDA is that we want to show the language equivalence given in the next corollary.

Let $\mathcal{L}_{RCFG}$, $\mathcal{L}_{RCFG\backslash\varepsilon}$ denote the classes of data languages generated by RCFG and by ε-rule free RCFG, respectively. Let $\mathcal{L}_{RPDA}$, $\mathcal{L}_{qndRPDA}$ denote the classes of data languages recognized by RPDA and by quasi non-decreasing RPDA, respectively.

Corollary 7. $\mathcal{L}_{RCFG} = \mathcal{L}_{RCFG\backslash\varepsilon} = \mathcal{L}_{qndRPDA} = \mathcal{L}_{RPDA}$

Proof. $\mathcal{L}_{RCFG} = \mathcal{L}_{RCFG\backslash\varepsilon}$ is by Theorem 4 of [24]. $\mathcal{L}_{RCFG\backslash\varepsilon} \subseteq \mathcal{L}_{qndRPDA}$ is by Theorem 11. $\mathcal{L}_{qndRPDA} \subseteq \mathcal{L}_{RPDA}$ is trivial. $\mathcal{L}_{RPDA} = \mathcal{L}_{RCFG}$ is by Theorem 7.

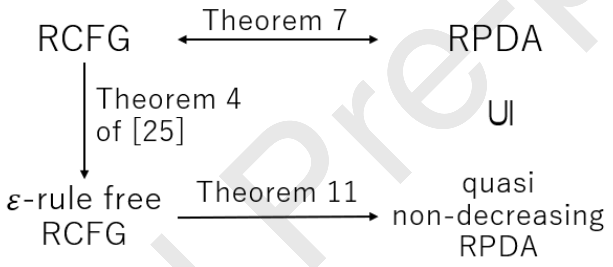


Figure 7: The relation among general RCFG, ε-rule free RCFG, RPDA and quasi non-decreasing RPDA

Theorem 12. *The membership problem for quasi growing RPDA is NP-complete.*

Proof NP solvability holds by Lemma 12. By the definition of quasi growing RPDA, growing RPDA is a subclass of quasi growing RPDA. Since the membership of growing RPDA is NP-hard by Theorem 9, the membership of quasi growing RPDA is also NP-hard.

Theorem 13. *The membership and emptiness problems for RPDA with a bounded number of registers are both in P.*

Proof By Theorem 7, we can convert k-RPDA to $(3k+3)$-RCFG in polynomial time. If k is bounded, then the number of registers in the obtained $(3k+3)$-RCFG is also bounded. By Theorem 5, the membership and emptiness problems for RCFG with a bounded number of registers are in P. Therefore, these problems for RPDA with a bounded number of registers are also in P.

6. Register Tree Automata

6.1. Definitions

A *ranked alphabet* Σ is a finite set of symbols, each of which is associated with a nonnegative integer called a rank. Let Σ_n be the set of symbols having rank n in Σ. Let T_Σ be the smallest set satisfying $f \in \Sigma_n$ and $t_j \in T_\Sigma (1 \le j \le n)$ imply $f(t_1, \ldots, t_n) \in T_\Sigma$. A member $t \in T_\Sigma$ is called a *tree* over Σ. The set of positions $\mathrm{Pos}(t)$ of a tree $t = f(t_1, \ldots, f_n)$ $(f \in \Sigma_n)$ is defined by $\mathrm{Pos}(t) = \{\varepsilon\} \cup \{jp \mid p \in \mathrm{Pos}(t_j), 1 \le j \le n\}$. For $t = f(t_1, \ldots, t_n)$ $(f \in \Sigma_n)$ and $p \in \mathrm{Pos}(t)$, the label $\mathrm{lab}(t,p)$ and the subtree $t|_p$ of t at p is defined as $\mathrm{lab}(t,\varepsilon) = f$, $t|_\varepsilon = t$, $\mathrm{lab}(t, jp) = \mathrm{lab}(t_j, p)$ and $t|_{jp} = t_j|_p$ $(1 \le j \le n)$. Let D be an infinite set of data values. A *data tree* over Σ and D is a pair $\tau = (t, \delta)$ where $t \in T_\Sigma$ and δ is a mapping $\delta : \mathrm{Pos}(t) \to D$. We let $Pos(\tau) = Pos(t)$. The set of all data trees over Σ and D is denoted as $T_{\Sigma \times D}$.

Definition 3. *A k-register tree automaton (k-RTA) over a ranked alphabet Σ and a set D of data values is a tuple $A = (Q, q_0, T)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state and T is a set of transition rules having one of the following forms:*

$$f(q, \psi, x_i) \to (q_1, \ldots, q_n) \quad \textit{or} \quad f(q, \psi) \to (q_1, \ldots, q_n)$$

where $f \in \Sigma_n$, $q \in Q$, $\psi \in F_k$, $1 \le i \le k$ and q_j $(1 \le j \le n)$. When $n = 1$, we omit the parentheses in the right-hand side. When $n = 0$, we write only the left-hand side $f(q, \psi, x_i)$ or $f(q, \psi)$ to denote the rule. A run *of A on a data tree $\tau = (t, \delta)$ is a mapping $\rho : Pos(t) \to Q \times \Theta_k$ satisfying the following condition: For $p \in Pos(t)$, if $\rho(p) = (q, \theta)$, $lab(t,p) = f$, and $\rho(pj) = (q_j, \theta_j)$ for $1 \le j \le n$, then there is $f(q, \psi, x_i) \to (q_1, \ldots, q_n) \in T$ (resp. $f(q, \psi) \to (q_1, \ldots, q_n) \in T$) such that $\theta, \delta(p) \models \psi$ and $\theta_j = \theta[x_i \leftarrow \delta(p)]$ (resp. $\theta_j = \theta$) $(1 \le j \le n)$. A run ρ is* accepting *if $\rho(\varepsilon) = (q_0, \bot)$. The data tree language recognized by A is $L(A) = \{\tau \in T_{\Sigma \times D} \mid$ there is an accepting run of A on $\tau\}$.*

6.2. Computational complexity

Theorem 14. *The membership problem for RTA is NP-complete. This holds even if Σ is monadic, i.e., $\Sigma = \Sigma_0 \cup \Sigma_1$.*

Proof To prove an upper bound, assume we are given an RTA A and a data tree $\tau = (t, \delta) \in T_{\Sigma\times D}$ and simulate a run of A on τ. For one step transition, A reads the data value at a node of τ and moves down. Therefore, the number of transitions of A is exactly $|t|$ and A will not read the data value of any node more than once. Hence, we can decide whether $\tau \in L(A)$ by nondeterministically assign a state of A to each node of τ and verify that the guessed assignment of states constitutes an accepting run of A on τ in polynomial time.

The NP-hardness can be proved in a similar way to the proof of Theorem 4. (The NP-hardness is first proved in [12].)

Theorem 15. *The emptiness problem for RTA is EXPTIME-complete.*

Proof To prove the theorem, it suffices to prove the following two properties because the emptiness problem for ε-rule free RCFG is EXPTIME-complete (Theorem 3).

- For a given ε-rule free k-RCFG G, we can construct a k-RTA A_G such that $L(G) = \emptyset \Leftrightarrow L(A_G) = \emptyset$ in polynomial time.
- For a given k-RTA A, we can construct an ε-rule free k-RCFG G_A such that $L(A) = \emptyset \Leftrightarrow L(G_A) = \emptyset$ in polynomial time.

Let $G = (V, R, S)$ be a k-RCFG over Σ and D. We first translate G to a k-RCFG $G' = (V', R', S)$ such that $L(G') = L(G)$ and R consists of production rules having one of the following forms:

$$(A, \varphi, x_i) \to \alpha, \quad (A, \varphi) \to \alpha \quad (\alpha \in V^+)$$
$$(A, \mathrm{tt}) \to (a, x_j) \quad ((a, x_j) \in \Sigma \times X_k)$$

by replacing $(a, x_j) \in \Sigma \times X_k$ in the right-hand side of a production rule in R with a new nonterminal, say X, and adding a rule $(X, \mathrm{tt}) \to (a, x_j)$. From G', we construct the following k-RTA $A_G = (Q, q_S, T)$ over Σ' and D where $\Sigma'_n = \{f_n \mid \text{there is a production rule in } R' \text{ such that the length of its right-hand side is } n\}$ and

- $Q = \{q_c \mid c \in V \cup \Sigma_0\}$, and
- $T = \{f_n(q_A, \varphi, x_i) \to (q_{B_1}, q_{B_2}, \ldots, q_{B_n}) \mid (A, \varphi, x_i) \to B_1 B_2 \ldots B_n \in R'\}$
  $\cup \{f_n(q_A, \varphi) \to (q_{B_1}, q_{B_2}, \ldots, q_{B_n}) \mid (A, \varphi) \to B_1 B_2 \ldots B_n \in R'\}$
  $\cup \{f_0(q_A, \mathrm{tt}) \mid (A, \mathrm{tt}) \to (a, x_j) \in R'\}$.

It is straightforward to check that this RTA A_G has the desired property.

Next, let $A = (Q, q_0, T)$ be a k-RTA over a ranked alphabet Σ and D and we construct k-register RCFG $G_A = (V, R, A_{q_0})$ where

- $V = \{A_c \mid c \in Q \cup \Sigma_0\}$ and
- $R = \{(A_q, \varphi, x_i) \to A_{q_1} A_{q_2} \ldots A_{q_n} \mid$
  $f(q, \varphi, x_i) \to (q_1, q_2, \ldots, q_n) \in T,\ f \in \Sigma_n\ (n \geq 1)\}$
  $\cup \{(A_q, \varphi) \to A_{q_1} A_{q_2} \ldots A_{q_n} \mid$
  $f(q, \varphi) \to (q_1, q_2, \ldots, q_n) \in T,\ f \in \Sigma_n\ (n \geq 1)\}$
  $\cup \{(A_q, \varphi, x_i) \to c \mid c(q, \varphi, x_i) \in T\} \cup \{(A_q, \varphi) \to c \mid c(q, \varphi) \in T\}$.

It is also straightforward to check that this RCFG G_A has the desired property and we are done.

7. Conclusion

We have discussed the computational complexity of the membership and emptiness problems for RCFG, RPDA and RTA. The combined complexity of the membership problem for general RCFG is EXPTIME-complete and decreases when we consider subclasses of RCFG while the data complexity is in P for general RCFG. There is an interesting similarity of the computational hierarchies between RCFG and multiple context-free grammars (MCFG) [25] where an MCFG is another natural extension of CFG generating tuples of strings. The emptiness problem for RCFG remains EXPTIME-complete even if we restrict RCFG to be growing. We also analyze how the complexity reduces when we restrict the number of registers occurring in the guard of a production rule.

Introducing a logic such as FO($\sim$), EMSO($\sim$) and LTL$\downarrow$ on data trees that corresponds to or subsumes RCFG, RPDA and RTA is a future study. Also, introducing recursive queries such as datalog in relational databases and fixed point logics as related logical foundations [26, Part D][27, Chapter 10] would be an interesting topic to be pursued.

References

[1] R. Senda, Y. Takata, H. Seki, Complexity results on register context-free grammars and register tree automata, in: B. Fischer, T. Uustalu (Eds.), 15th International Colloquium on Theoretical Aspects of Computing (ICTAC 2018), Vol. 11187 of Lecture Notes in Computer Science, Springer, 2018, pp. 415–434. doi:10.1007/978-3-030-02508-3_22.

[2] R. Senda, Y. Takata, H. Seki, Complexity results on register pushdown automata, in: S. Hidaka, Y. Ishihara, Z. G. Ives (Eds.), Third Workshop on Software Foundations for Data Interoperability (SFDI2019+), arXiv e-prints, 2019, pp. 3:1–3:5.
URL `http://arxiv.org/abs/1910.10357`

[3] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, L. Segoufin, Two-variable logic on data words, ACM Transactions on Computational Logic 12 (4) (2011) 27:1–27:26. doi:10.1145/1970398.1970403.

[4] S. Demri, R. Lazić, LTL with the freeze quantifier and register automata, ACM Transactions on Computational Logic 10 (3) (2009) 16:1–16:30. doi:10.1145/1507244.1507246.

[5] M. Kaminski, N. Francez, Finite-memory automata, Theoretical Computer Science 134 (2) (1994) 329–363. doi:10.1016/0304-3975(94)90242-9.

[6] L. Libkin, T. Tan, D. Vrgoč, Regular expressions for data words, J. Computer and System Sciences 81 (7) (2015) 1278–1297. doi:10.1016/j.jcss.2015.03.005.

[7] L. Libkin, D. Vrgoč, Regular path queries on graphs with data, in: A. Deutsch (Ed.), 15th International Conference on Database Theory (ICDT '12), ACM, 2012, pp. 74–85. doi:10.1145/2274576.2274585.

[8] L. Libkin, W. Martens, D. Vrgoč, Querying graphs with data, J. ACM 63 (2) (2016) 14:1–14:53. doi:10.1145/2850413.

[9] H. Sakamoto, D. Ikeda, Intractability of decision problems for finite-memory automata, Theoretical Computer Science 231 (2) (2000) 297–308. doi:10.1016/S0304-3975(99)00105-X.

[10] E. Y. Cheng, M. Kaminski, Context-free languages over infinite alphabets, Acta Informatica 35 (3) (1998) 245–267. doi:10.1007/s002360050120.

[11] M. Kaminski, T. Tan, Tree automata over infinite alphabets, in: A. Avron, N. Dershowitz, A. Rabinovich (Eds.), Pillars of Computer Science, Vol. 4800 of Lecture Notes in Computer Science, Springer-Verlag, 2008, pp. 386–423. doi:10.1007/978-3-540-78127-1_21.

[12] L. Segoufin, Automata and logics for words and trees over an infinite alphabet, in: Z. Ésik (Ed.), Computer Science Logic, CSL 2006, Vol. 4207 of Lecture Notes in Computer Science, Springer, 2006, pp. 41–57. doi:10.1007/11874683_3.

[13] P. Bouyer, A logical characterization of data languages, Information Processing Letters 84 (2) (2002) 75–85. doi:10.1016/S0020-0190(02)00229-6.

[14] T. Milo, D. Suciu, V. Vianu, Typechecking for XML transformers, in: V. Vianu, G. Gottlob (Eds.), 19th ACM Symposium on Principles of Database Systems (PODS 2000), ACM, 2000, pp. 11–22. doi:10.1145/335168.335171.

[15] F. Neven, T. Schwentick, V. Vianu, Finite state machines for strings over infinite alphabets, ACM Transactions on Computational Logic 5 (3) (2004) 403–435. doi:10.1145/1013560.1013562.

[16] S. Demri, R. Lazić, D. Nowak, On the freeze quantifier in constraint LTL: Decidability and complexity, Information and Computation 205 (1) (2007) 2–24. doi:10.1016/j.ic.2006.08.003.

[17] M. Bojańczyk, A. Muscholl, T. Schwentick, L. Segoufin, Two-variable logic on data trees and XML reasoning, J. ACM 56 (3) (2009) 13:1–13:48. doi:10.1145/1516512.1516515.

[18] M. Jurdziński, R. Lazić, Alternation-free modal mu-calculus for data trees, in: 22nd IEEE Symposium on Logic in Computer Science (LICS 2007), IEEE Computer Society, 2007, pp. 131–140. doi:10.1109/LICS.2007.11.

[19] D. Figueira, Forward-xpath and extended register automata on data-trees, in: L. Segoufin (Ed.), Database Theory - ICDT 2010, 13th International Conference, ACM International Conference Proceeding Series, ACM, 2010, pp. 231–241. doi:10.1145/1804669.1804699.

[20] D. Figueira, Alternating register automata on finite words and trees, Logical Methods in Computer Science 8 (1:22) (2012) 1–44. doi:10.2168/LMCS-8(1:22)2012.

[21] D. Figueira, L. Segoufin, Bottom-up automata on data trees and vertical xpath, in: T. Schwentick, C. Dürr (Eds.), 28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011, Vol. 9 of LIPIcs, Schloss Dagstuhl, 2011, pp. 93–104. doi:10.4230/LIPIcs.STACS.2011.93.

[22] M. Sipser, Introduction to the theory of computation, 3rd Edition, Course Technology, 2013.

[23] A. K. Chandra, D. C. Kozen, L. J. Stockmeyer, Alternation, J. ACM 28 (1) (1981) 114–133. doi:10.1145/322234.322243.

[24] R. Senda, Y. Takata, H. Seki, Generalized register context-free grammars, in: C. Martín-Vide, A. Okhotin, D. Shapira (Eds.), Language and Automata Theory and Applications (LATA 2019), Vol. 11417 of Lecture Notes in Computer Science, Springer, 2019, pp. 259–271. doi:10.1007/978-3-030-02508-3_22.

[25] Y. Kaji, R. Nakanishi, H. Seki, T. Kasami, The computational complexity of the universal recognition problem for parallel multiple context-free grammars, Computational Intelligence 10 (1994) 440–452.

[26] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995.

[27] L. Libkin, Elements of Finite Model Theory, Texts in Theoretical Computer Science. An EATCS Series, Springer, 2004.