Complexity Results on Register Context-Free Grammars and Register Tree Automata

Ryoma Senda[1(✉)], Yoshiaki Takata[2], and Hiroyuki Seki[1]

[1] Graduate School of Information Science, Nagoya University, Furo-cho, Chikusa, Nagoya 464-8601, Japan
ryoma.private@sqlab.jp, seki@i.nagoya-u.ac.jp
[2] Graduate School of Engineering, Kochi University of Technology, Tosayamada, Kami City, Kochi 782-8502, Japan
takata.yoshiaki@kochi-tech.ac.jp

Abstract. Register context-free grammars (RCFG) and register tree automata (RTA) are an extension of context-free grammars and tree automata, respectively, to handle data values in a restricted way. RTA are paid attention as a model of query languages for structured documents such as XML with data values. This paper investigates the computational complexity of the basic decision problems for RCFG and RTA. We show that the membership and emptiness problems for RCFG are EXPTIME-complete and also show how the complexity reduces by introducing subclasses of RCFG. The complexity of these problems for RTA are also shown to be NP-complete and EXPTIME-complete.

1 Introduction

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 the decidability on basic problems and the closure properties, including first-order and monadic second-order logics with data equality, linear temporal logic with freeze quantifier [7] and register automata [14]. Among them, register automata (abbreviated as RA) is 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 expression was extended to regular expression with memory (REM), which have the same expressive power as RA [18].

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 [17,19]. 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

B. Fischer and T. Uustalu (Eds.): ICTAC 2018, LNCS 11187, pp. 415–434, 2018.
https://doi.org/10.1007/978-3-030-02508-3_22

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 [14] and their computational complexities were also analyzed [7,22].

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) were proposed in [6] as an extension of classical context-free grammars (CFG) in a similar way to extending FA to RA. In [6], properties of RCFG were shown including the decidability of the membership and emptiness problems, and the closure properties. However, the computational complexity of these 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 [15,23]. 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, some subclasses of them, and RTA. In the original definition of RCFG [6], 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 RA [14]. In a derivation, a symbol can be loaded to a register only when the value is different from any symbol stored in the other registers, by which the equality checking is indirectly incorporated. In recent studies on RA [18,19], 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 a *register assignment*) 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 registers.

We then show that the membership problems for general RCFG, ε-rule free RCFG, growing RCFG, and RCFG with bounded registers 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 [6]. 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.

Finally, we analyze the computational complexity of these problems for RTA. It is well-known that the class of tree languages accepted by tree automata coincides with the class of derivation trees generated by CFG. 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 is 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 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.

Table 1. Complexity results on RCFG and RTA

	General RCFG	ε-rule free RCFG	Growing RCFG	RCFG w/ bounded regs	RTA
Membership	EXPTIMEc	PSPACEc	NPc	In P	NPc
Emptiness	EXPTIMEc	EXPTIMEc	EXPTIMEc	In P	EXPTIMEc

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.

Register Automata and Register Context-Free Grammars: As already mentioned, register automata (RA) was first introduced in [14] as finite-memory automata where they show that the membership and emptiness problems are decidable, and RA are closed under union, concatenation and Kleene-star. Later, the computational complexity of the former two problems are analyzed in [7,22]. In [6], 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 [4] and pebble

automata (PA) [20]. It is desirable for a query language to have an efficient data complexity for the membership problem. Libkin and Vrgoč [19] 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 [21] 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, $\mathrm{FO}(\sim, <)$ and $\mathrm{EMSO}(\sim, <)$, 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$\downarrow$ with freeze quantifier [7,8]. A data value is bound with 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 [7]. 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 LTL$\downarrow$ of [7]. Also it was shown in [2] 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 [15], 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 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) [12] which correspond to forward XPath were introduced, and the decidability of the emptiness problems for these classes was shown. While RTA work on ranked data trees, ATRA work on unranked data trees. Later, [9,10] extended ATRA so that (1) they 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 shown 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 [11]. 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.

2 Definitions

2.1 Preliminaries

A register context-free grammar was introduced in [6] as a grammar over an infinite alphabet. We define it as a grammar over the product of a finite alphabet and an infinite set of data values, following recent notions [18,19]. 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 $\nu : [k] \to D$ 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 F_k denote the class of assignments to k registers. For $\nu, \nu' \in F_k$, we write $\nu' = \nu[i \leftarrow d]$ if $\nu'(i) = d$ and $\nu'(j) = \nu(j)$ $(j \neq i)$.

Let C_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 $x_i \in \{x_1, \ldots, x_k\}$. 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 $\nu \in F_k$, the satisfaction of $\psi \in C_k$ by (d, ν) is recursively defined as follows. Intuitively, d is a current data value in the input, ν is a current register assignment, $d, \nu \models x_i^{=}$ means that the data value assigned to the i-th register by ν is equal to d and $d, \nu \models x_i^{\neq}$ means they are different.

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

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

For a finite alphabet Σ and a set D of data values disjoint from Σ, a *data word* over $\Sigma \times D$ is a finite sequence of elements of $\Sigma \times D$ and a subset of $(\Sigma \times D)^*$ is called a *data language* over $\Sigma \times D$. For a data word $w = (a_1, d_1)(a_2, d_2) \ldots (a_n, d_n)$, $a_1 a_2 \ldots a_n$ is the label of w and $d_1 d_2 \ldots d_n$ 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 such that $\Sigma \cap D = \emptyset$ 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) where $V \cap (\Sigma \cup D) = \emptyset$,
- R is a finite set of production rules (abbreviated as rules) having either of the following forms:
$$(A, \psi, i) \to \alpha, \quad (A, \psi) \to \alpha$$
where $A \in V$, $\psi \in C_k$, $i \in [k]$ and $\alpha \in (V \cup (\Sigma \times [k]))^*$; we call (A, ψ, 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 $\|G\| = |V| + |R| \max\{(|\alpha| + 1)(\log |V| + \log k) + \|\psi\| \mid (A, \psi, i) \to \alpha \in R \text{ or } (A, \psi) \to \alpha \in R\}$, where $\|\psi\|$ is the description length of ψ. In this definition, we assume that the description length of $\alpha \in (V \cup (\Sigma \times [k]))^*$ 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 $(A, \psi, i) \to \alpha$ is $\log |V| + \|\psi\| + \log k$, we let the description length of this rule be $(|\alpha| + 1)(\log |V| + \log k) + \|\psi\|$. We assume $k \leq \|G\|$ 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 $A \in V$, $\nu \in F_k$ and $X \in ((V \times F_k) \cup (\Sigma \times D))^*$, we say (A, ν) directly derives X, written as $(A, \nu) \Rightarrow_G X$ if there exist $d \in D$ (regarded as an input data value) and $(A, \psi, i) \to c_1 \dots c_n \in R$ (resp. $(A, \psi) \to c_1 \dots c_n \in R$) such that

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

For $X, Y \in ((V \times F_k) \cup (\Sigma \times D))^*$, we also write $X \Rightarrow_G Y$ if there are $X_1, X_2, X_3 \in ((V \times F_k) \cup (\Sigma \times D))^*$ such that $X = X_1(A, \nu)X_2$, $Y = X_1X_3X_2$ and $(A, \nu) \Rightarrow_G X_3$.

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 $\perp$ the register assignment that assigns the initial value $\perp$ to every register. Figure 1 shows an example of a direct derivation from $(S, \perp)$

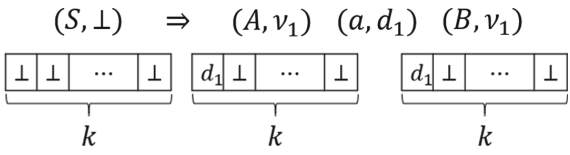

Fig. 1. An example of derivation from $(S, \bot)$ using $(S, \mathrm{tt}, 1) \to A(a, 1)B$

using $(S, \mathrm{tt}, 1) \to A(a, 1)B$. In the figure, d_1 is an arbitrary data value in D and $\nu_1 = \bot[1 \leftarrow d_1]$, because the guard of the rule is tt.

We let

$$L(G) = \{w \mid (S, \nu_0) \overset{+}{\Rightarrow} w \in (\Sigma \times D)^*\}.$$

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

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. Consider the following k-RCFG $G = (V, R, S)$, which satisfies $L(G) = \{(a, \bot)\}$. While $\|G\| = O(k \log k)$, the height of the derivation tree of $(a, \bot)$ is $\Omega(2^k)$.

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

and R consists of the following rules:

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

and for $1 \leq i < k$,

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

The derivation of G from $(S, \bot)$ to $(a, \bot)$ 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 $\bot$ (resp. not $\bot$). The derivation keeps the value of the kth register being $\bot$. For $1 \leq i < k$ and $b \in \{0, 1\}$, derivation $(A_{(i,b)}, \nu) \Rightarrow_G (B_{(i,b)}, \nu)$ can exist if only if the ith bit of the binary counter represented by ν equals b. After this derivation, the ith bit is flipped by the derivation from $(B_{(i,b)}, \nu)$, and

it derives $(A_{(j,0)}, \nu')$ and $(A_{(j,1)}, \nu')$ where ν' is the updated register assignment, and $j = i + 1$ if "the carry to the next bit" exists, and $j = 1$ otherwise. Because $(A_{(k,0)}, \nu)$ for some ν (and $(a, \bot)$) can be derived only when every bit of the binary counter becomes "1", the derivation from $(S, \bot)$ to $(a, \bot)$ must pass through the elements of $\{A_{(1,0)}, A_{(1,1)}\} \times F_k$ 2^{k-1} times.

3 Basic Properties of RCFG

The properties of RCFG in this section were first shown in [6]. We will give sketches of proofs to them to make this paper self-contained. We fix a finite alphabet Σ and a set D of data values.

Proposition 2. *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 a register 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 ν in F_k to be a k-tuple over D, i.e. an element of D^k, and thus $V' \subseteq V \times F_k$.
- $R' = \{(A, \nu) \to X \mid (A, \nu) \in V',\ X \in (V' \cup (\Sigma \times D'))^*,\ \text{and } (A, \nu) \Rightarrow_G X\}$.
- $S' = (S, \bot)$.

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 $(S, \bot) \overset{*}{\Rightarrow}_G X$. This establishes $L(G') = L(G) \cap (\Sigma \times D')^*$.

Proposition 3. *If a k-RCFG G generates a data word of length n, G generates a data word of length n that contains at most $k + 1$ different data values.*

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 $\bot$. Consider a direct derivation of G:

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

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

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

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

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

Proof. Let G be an RCFG. It holds that $w = (a_1, d_1) \ldots (a_n, d_n) \in L(G)$ if and only if $w \in L(G) \cap (\Sigma \times \{d_1, \ldots, d_n\})^*$. By Proposition 2, we can construct a CFG G' from G and w that generates $L(G) \cap (\Sigma \times \{d_1, \ldots, d_n\})$. This implies the decidability of the membership problem of RCFG because the membership problem of CFG is decidable.

Proposition 5. *The emptiness problem for RCFG is decidable.*

Proof. For a given k-RCFG G, let $D_k = \{d_1, \ldots, d_{k+1}\} \subseteq D$ be an arbitrary subset of D consisting of $k+1$ different data values. By Proposition 3, $L(G) = \emptyset$ if and only if $L(G) \cap (\Sigma \times D_k)^* = \emptyset$. By Proposition 2, 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.

RCFG has the following closure properties [6].

Proposition 6. *The class of data languages generated by RCFG are closed under union, concatenation and Kleene-star.*

4 Upper Bounds

Lemma 7. *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 2, 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 2.*

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 2.

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

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

Lemma 8. *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 4. By Lemma 7, $\|G'\| \in O(\|G\| c^{k+1} k)$ 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 9. *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 $S \overset{*}{\Rightarrow}_G \alpha \overset{*}{\Rightarrow}_G w$, $|\alpha| \leq |w|$ holds. and 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 $S \overset{*}{\Rightarrow}_G w$ in polynomial space in $\|G\|$, $|w|$, and k.

Lemma 10. *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 $S \overset{*}{\Rightarrow}_G \alpha \Rightarrow_G \alpha'$, $|\alpha| < |\alpha'|$ holds. 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 $|w|$, and checking step by step whether $S \overset{*}{\Rightarrow}_G w$ in nondeterministical polynomial time in $\|G\|$, $|w|$, and k.

Lemma 11. *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 2. By Lemma 7, $\|G'\| \in O(\|G\| \cdot |D_k|^{k+1} k)$ 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.

5 Lower Bounds

5.1 Alternating Turing Machine

An *alternating Turing machine* [5], [24, *Sect.* 10.3] (abbreviated as ATM) 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

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

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

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

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

We define the accepting condition $g : Q \times \Gamma^* \times N \to \{\mathrm{tt}, \mathrm{ff}\}$ of M as follows.

$$g(q, \alpha, j) = \begin{cases} \mathrm{tt} & q = q_{acc} \\ \mathrm{ff} & q = q_{rej} \\ \bigvee_{(q,\alpha,j) \to (q',\alpha',j')} g(q', \alpha', j') & q \in Q_e \\ \bigwedge_{(q,\alpha,j) \to (q',\alpha',j')} g(q', \alpha', j') & q \in Q_a \end{cases}$$

For an ATM $M = (Q, Q_e, Q_a, \Gamma, \Sigma, \delta, q_0, q_{acc}, q_{rej})$ and $u \in \Sigma^*$, if $g(q_0, u, 1) = \mathrm{tt}$, then M accepts u, and if $g(q_0, u, 1) = \mathrm{ff}$, then M rejects u. Let $L(M) = \{u \in \Sigma^* \mid g(q_0, u, 1) = \mathrm{tt}\}$, 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.

5.2 Membership for General RCFG

Theorem 12. *The membership problem for RCFG is EXPTIME-complete. This holds even for RCFG of which every guard expression refers to at most two registers.*

Proof. By Lemma 8, it is enough to show EXPTIME-hardness, which will be shown 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 a register 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

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

and R is constructed as follows. Without loss of generality, we assume that $\Gamma = \{1, 2, \ldots, |\Gamma|\} \subseteq \mathbb{N}$ 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., $\alpha = \alpha_1 \alpha_2 \ldots \alpha_{|\alpha|}$).

– We construct production rules that load different data values in the first $|\Gamma|$ registers. Note that we keep the initial value $\perp$ in the first register. To the ith register ($i \ge 2$), 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 ($2 \le j < i$) by Rule (2).

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

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

– 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:

$$(T_{(|\Gamma|,|\Gamma|-1)}, \mathrm{tt}) \to W_0, \tag{3}$$

$$(W_{i-1}, x_{u_i}^{=}, |\Gamma| + i) \to W_i \quad \text{for } 1 \le i \le |u|. \tag{4}$$

– 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 $q \in Q$, $1 \le i \le |\Gamma|$, and $1 \le j \le p(n)$. $A_q^{(i,j)}$ represents a part of an ID (q, α, j) of M where $i = \alpha_j$, i.e. the tape symbol at the head position. The remaining information about α of (q, α, j) will be represented by a register assignment of G. More precisely, the content of $(|\Gamma|+j)$th register (i.e. $\nu(|\Gamma|+j)$) equals the data value $\nu(\alpha_j)$ representing the tape symbol α_j for $1 \le j \le |\alpha|$ and $\nu(|\Gamma|+j) = \perp$ for $|\alpha| < j \le p(|u|)$. Let ν_α denote such

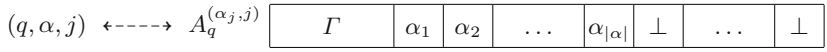

Fig. 2. The correspondence between M's ID and G's nonterminal symbol and registers.

a register assignment that represents the tape contents α. We illustrate the correspondence between an ID of M and a nonterminal symbol and a register 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 $\{(q_1, b_1, m_1), \ldots, (q_t, b_t, m_t)\} = \delta(q, i)$. For each $a \in \Gamma$ and $1 \leq k \leq t$, we construct the following rules. If $q \in Q_e$, then:

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

If $q \in Q_a$, then:

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

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

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 $1 \leq j' \leq p(n)$:

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

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

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

We can show for each ID (q, α, j),

$$g(q, \alpha, j) = \mathrm{tt} \;\text{ iff }\; (A_q^{(\alpha_j, j)}, \nu_\alpha) \overset{*}{\Rightarrow}_G \varepsilon \tag{11}$$

by induction on the application number of the definition of accepting condition for only if part and by induction on the length of the derivation for if part.

We can easily prove that $(S, \perp) \overset{*}{\Rightarrow}_G (A_{q_0}^{(u_1,1)}, \nu_u)$, and moreover, if $(S, \perp) \overset{*}{\Rightarrow}_G \varepsilon$, then this derivation must be $(S, \perp) \overset{*}{\Rightarrow}_G (A_{q_0}^{(u_1,1)}, \nu_u) \overset{*}{\Rightarrow}_G \varepsilon$. By letting $(q, \alpha, k) = (q_0, u, 1)$ in property (11) and by the above-mentioned fact, we obtain $u \in L(M) \Leftrightarrow g(q_0, u, 1) = \mathrm{tt} \Leftrightarrow ((S, \perp) \overset{*}{\Rightarrow} \varepsilon) \Leftrightarrow \varepsilon \in L(G)$. By the definition of G, we can say that this EXPTIME-completeness holds even for RCFG of which every guard expression refers to at most two registers.

5.3 Membership for ε-rule Free RCFG

Theorem 13. *The membership problem for ε-rule free RCFG is PSPACE-complete. This holds even for ε-rule free RCFG with guards referring to at most two registers.*

Proof. By Lemma 9, it is enough to show PSPACE-hardness. We prove it by a polynomial-time reduction from the membership problem for polynomial-space bounded ordinary Turing machines (TM), in a similar way as the proof of Theorem 12. 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 $(A^{(i,j)}_{q_{acc}}, \mathrm{tt}) \to (a, 1)$. The resultant RCFG G is ε-rule free, and we can show $u \in L(M) \Leftrightarrow (a, \bot) \in L(G)$ (because the data value in the first register is always $\bot$).

5.4 Membership for Growing RCFG

Theorem 14. *The membership problem for growing RCFG is NP-complete. This holds even for growing RCFG in which every guard is either tt or $x_1^{=}$.*

Proof. By Lemma 10, it is enough to show NP-hardness. We prove it 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

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

The first register of the constructed RCFG G is used for keeping a data value (possibly) different from $\bot$, and we use that value and $\bot$ 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 $y_1, \overline{y_1}, y_2, \overline{y_2}, \dots, y_n, \overline{y_n}$. 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. It is not difficult to show that ϕ is satisfiable if and only if $(a, d)^{n+m+3} \in L(G)$, where d is an arbitrary data value in $D \setminus \{\bot\}$. Since $(a, d_1)^{n+m+3} \in L(G)$ iff $(a, d_2)^{n+m+3} \in L(G)$ for any $d_1, d_2 \in D \setminus \{\bot\}$, we can choose any $d \in D \setminus \{\bot\}$ to make the input data word for the membership problem. Hence, we have shown the NP-hardness of the problem.

5.5 The Emptiness Problem

Theorem 15. *The emptiness problem for RCFG is EXPTIME-complete, even if RCFG are restricted to be growing and with guards referring to at most two registers.*

Proof. By Lemma 11, it is enough to show EXPTIME-hardness. In the proof of Theorem 12, 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:

$$(A_q^{(i,j)}, \mathrm{tt}) \to (a, 1) \quad \text{if } \delta(q, i) = \emptyset \text{ and } q \in Q_a,$$
$$(A_{q_{acc}}^{(i,j)}, \mathrm{tt}) \to (a, 1).$$

Also, a unit rule, say $A \to B$ $(A, B \in V)$ can be replaced with $A \to (a, 1)B$ $(a \in \Sigma)$. 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 Theorem 12 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 Theorem 14, 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 16. *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 register assignment ν, there must be a data value d that satisfies $d, \nu \models \psi$. That is, regardless of ν, rule $(A, \psi, i) \to \alpha$ or $(A, \psi) \to \alpha$ can be applied to expanding (A, ν).

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 $(A, \psi, i) \to B(a, j)$ with $A \to Ba$). For a string $X \in ((V \times F_k) \cup (\Sigma \times D))^*$, let $r(X) \in (V \cup \Sigma)^*$ be the string obtained from X by removing the register 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.

5.6 RCFG with Bounded Registers

Theorem 17. *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 2 from G and a finite set $D' \subseteq D$. Then $\|G'\| = O(\|G\| \cdot |D''|^{k+1}k)$ holds by Lemma 7. If k is a constant independent of the choice of G, then $\|G'\|$ is a polynomial of $\|G\|$ and $|D''|$. Hence, by Lemmas 8 and 11, 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.

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 $t_j \in T_\Sigma (1 \leq j \leq n)$ 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 $\mathrm{Pos}(t) = \{\varepsilon\} \cup \{jp \mid p \in \mathrm{Pos}(t_j), 1 \leq j \leq n\}$. 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 $t|_{jp} = t_j|_p$ $(1 \leq j \leq n)$. 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 18. *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:*

$$f(q, \psi, i) \to (q_1, \ldots, q_n) \quad or \quad f(q, \psi) \to (q_1, \ldots, q_n)$$

where $f \in \Sigma_n$, $q \in Q$, $\psi \in C_k$, $1 \leq i \leq k$ and q_j $(1 \leq j \leq n)$. When $n = 1$, we omit the parentheses in the right-hand side. When $n = 0$, we write only the left-hand side $f(q, \psi, i)$ or $f(q, \psi)$ to denote the rule. A run *of A on a data tree $\tau = (t, \delta)$ is a mapping $\rho : Pos(t) \to Q \times F_k$ satisfying the following condition: For $p \in Pos(t)$, if $\rho(p) = (q, \nu)$, $lab(t, p) = f$, and $\rho(pj) = (q_j, \mu_j)$ for $1 \leq j \leq n$, then there is $f(q, \psi, i) \to (q_1, \ldots, q_n) \in T$ (resp. $f(q, \psi) \to (q_1, \ldots, q_n) \in T$) such that $\delta(p), \nu \models \psi$ and $\nu_j = \nu[i \leftarrow \delta(p)]$ (resp. $\nu_j = \nu$) $(1 \leq j \leq n)$. 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 19. *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 14.

Theorem 20. *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 15).

- 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:

$$(A, \varphi, i) \to \alpha, \quad (A, \varphi) \to \alpha \quad (\alpha \in V^+)$$
$$(A, \mathrm{tt}) \to (a, j) \quad (a \in \Sigma,\ j \in [k])$$

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

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

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

– $R = \{\, (A_q, \varphi, i) \to A_{q_1} A_{q_2} \dots A_{q_n} \mid$
$f(q, \varphi, i) \to (q_1, q_2, \dots, q_n) \in T,\ f \in \Sigma_n\ (n \geq 1)\}$
$\cup \{\, (A_q, \varphi) \to A_{q_1} A_{q_2} \dots A_{q_n} \mid$
$f(q, \varphi) \to (q_1, q_2, \dots, q_n) \in T,\ f \in \Sigma_n\ (n \geq 1)\}$
$\cup \{(A_q, \varphi, i) \to c \mid c(q, \varphi, i) \in T\} \cup \{(A_q, \varphi) \to c \mid c(q, \varphi) \in T\}.$

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 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) [13] 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(∼), EMSO(∼) and LTL↓ on data trees that corresponds to or subsumes RCFG and RTA is a future study. Also, introducing recursive queries such as datalog in relational databases and fixed point logics as related logical foundations [1, Part D] [16, Chap. 10] would be an interesting topic to be pursued.

Acknowledgements. This work was supported by JSPS KAKENHI Grant Number JP15H02684.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Bojańczyk, M., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data trees and XML reasoning. J. ACM **56**(3), Article no. 13 (2009). https://doi.org/10.1145/1516512.1516515.
3. Bojańczyk, M., David, C., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data words. ACM Trans. Comput. Log. **1**(4), Article no. 27 (2011). https://doi.org/10.1145/1970398.1970403
4. Bouyer, P.: A logical characterization of data languages. Inf. Process. Lett. **84**(2), 75–85 (2002). https://doi.org/10.1016/s0020-0190(02)00229-6
5. Chandra, A.K., Kozen, D., Stockmeyer, L.J.: Alternation. J. ACM **28**(1), 114–133 (1981). https://doi.org/10.1145/322234.322243

6. Cheng, E.Y.C., Kaminski, M.: Context-free languages over infinite alphabets. Acta Inf. **35**(3), 245–267 (1998). https://doi.org/10.1007/s002360050120
7. Demri, S., Lazić, R.: LTL with freeze quantifier and register automata. ACM Trans. Comput. Log. **10**(3), Article no. 16 (2009). https://doi.org/10.1145/1507244.1507246
8. Demri, S., Lazić, R., Nowak, D.: On the freeze quantifier in constraint LTL: decidability and complexity. Inf. Comput. **205**(1), 2–24 (2007). https://doi.org/10.1016/j.ic.2006.08.003
9. Figueira, D.: Forward-XPath and extended register automata on data-trees. In: Proceedings of 13th International Conference on Database Theory, ICDT 2010, Lausanne, March 2010. ACM International Conference Proceedings Series, pp. 231–241. ACM Press, New York (2010). https://doi.org/10.1145/1804669.1804699
10. Figueira, D.: Alternating register automata on finite data words and trees. Log. Methods Comput. Sci. **8**(1), Article no. 22 (2012). https://doi.org/10.2168/lmcs-8(1:22)2012
11. Figueira, D., Segoufin, L.: Bottom-up automata on data trees and vertical XPath. In: Schwentick, T., Dürr, C. (eds.) Proceedings of 28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011, Dortmund, March 2011. Leibniz International Proceedings in Information, vol. 9, pp. 93–104. Dagstuhl Publishing, Saarbrücken/Wadern (2011). https://doi.org/10.4230/lipics.stacs.2011.93
12. Jurdziński, M., Lazić, R.: Alternation-free modal mu-calculus for data trees. In: Proceedings of 22nd IEEE Symposium on Logic in Computer Science, LICS 2007, Wroclaw, July 2007, pp. 131–140. IEEE CS Press, Washington, DC (2007). https://doi.org/10.1109/lics.2007.11
13. Kaji, Y., Nakanishi, R., Seki, H., Kasami, T.: The computational complexity of the universal recognition problem for parallel multiple context-free grammars. Comput. Intell. **10**(4), 440–452 (1994). https://doi.org/10.1111/j.1467-8640.1994.tb00008.x
14. Kaminski, M., Franz, N.: Finite-memory automata. Theor. Comput. Sci. **134**(2), 322–363 (1994). https://doi.org/10.1016/0304-3975(94)90242-9
15. Kaminski, M., Tan, T.: Tree automata over infinite alphabets. In: Avron, A., Dershowitz, N., Rabinovich, A. (eds.) Pillars of Computer Science. LNCS, vol. 4800, pp. 386–423. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78127-1_21
16. Libkin, L.: Elements of Finite Model Theory. TTCS. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-662-07003-1
17. Libkin, L., Martens, W., Vrgoč, D.: Querying graphs with data. J. ACM **63**(2), Article no. 14 (2016). https://doi.org/10.1145/2850413
18. Libkin, T., Tan, T., Vrgoč, D.: Regular expressions for data words. J. Comput. Syst. Sci. **81**(7), 1278–1297 (2015). https://doi.org/10.1016/j.jcss.2015.03.005
19. Libkin, L., Vrgoč, D.: Regular path queries on graphs with data. In: Proceedings of 15th International Conference on Database Theory, ICDT 2012, Berlin, March 2012, pp. 74–85. ACM Press, New York (2012). https://doi.org/10.1145/2274576.2274585
20. Milo, T., Suciu, D., Vianu, V.: Type checking for XML transformers. In: Proceedings of 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2000, Dallas, TX, May 2000, pp. 11–22. ACM Press, New York (2000). https://doi.org/10.1145/335168.335171
21. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Trans. Comput. Log. **5**(3), 403–435 (2004). https://doi.org/10.1145/1013560.1013562

22. Sakamoto, H., Ikeda, D.: Intractability of decision problems for finite-memory automata. Theor. Comput. Sci. **231**, 297–308 (2000). https://doi.org/10.1016/s0304-3975(99)00105-x
23. Segoufin, L.: Automata and logics for words and trees over an infinite alphabet. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 41–57. Springer, Heidelberg (2006). https://doi.org/10.1007/11874683_3
24. Sipser, M.: Introduction to the Theory of Computation, 3rd edn. Cengage Learning, Boston (2013)