A study on Computational Models with Registers and their Application to Software Construction

Ryoma Senda

A Doctoral Dissertation
submitted to Graduate School of Informatics,
Nagoya University
in fulfillment of the requirements for the degree of
Doctor of Philosophy in INFORMATICS

Ryoma Senda

Thesis Committee:

Professor Seki Hiroyuki (Supervisor)
Professor Yuen Shoji
Professor Kaji Yuichi
Associate Professor Yoshiaki Takata (Kochi University of Technology)

Abstract

While most practical programs deal with data values such as ID, timestamp and probability, classical computational models cannot handle data values well. Adding an ability of processing data values to a computational model makes the model Turing equivalent, and many properties become undecidable easily even if only very simple operations to data values are allowed. Register automata (RA) are an extension of finite automata that have a finite number of registers for storing data values from an input word. RA compares the contents of its registers with the current input data value to determine the next transition and is known to have mild powers of processing data values. Register context-free grammars (RCFG), register pushdown automata (RPDA), register tree automata (RTA) and register pushdown systems (RPDS) are extensions of their classical counterparts to handle data values in a restricted way. These models are regarded as models for processing query languages for structured documents such as XML with data values.

The purpose of this dissertation is to build a theoretical foundation for handling programs and databases with data values. To this aim, this dissertation investigates the properties of register models in the following three parts.

- We investigate these models with respect to the complexities of fundamental decision problems such as membership and emptiness. We further generalize RCFG with reference to guard conditions of production rules.
- We apply these models to software verification. We show the forward regularity preservation property for RPDS and provide an algorithm for linear temporal logic (LTL) model checking.
- We investigate realizability problems for the cases that a specification and

an implementation are given by a pair of pushdown automaton (PDA) and pushdown transducer (PDT), and a pair of RPDA and register PDT (RPDT). Because the pushdown stack is useful for representing recursive programs, these problems are important in the field of reactive synthesis of recursive programs.

First, we show the computational complexity of the basic decision problems for RCFG, RPDA, RTA and their subclasses. For example, the membership and emptiness problems for RCFG are EXPTIME-complete, and the membership problem for the ε-rule free RCFG and the growing RCFG is PSPACE-complete and NP-complete, respectively. Furthermore, we define a generalized RCFG (GRCFG) where an arbitrary binary relation is allowed in guards. Although the membership and emptiness problems for GRCFG are undecidable in general, we show that these decidabilities are recovered under two properties of simulation and progress.

Secondly, we show the forward regularity preservation property for RPDS. This property implies the decidability of the forward reachability problem for RPDS. We prove this property by presenting a saturation algorithm that constructs an RA $\mathcal{A}_{post}$ such that $L(\mathcal{A}_{post}) = post^*_{\mathcal{P}}(L(\mathcal{A}))$ from a given RA $\mathcal{A}$ and RPDS $\mathcal{P}$, where $post^*_{\mathcal{P}}$ is the forward image of the mapping induced by $\mathcal{P}$. In addition, we propose algorithms for the LTL model checking problem for RPDS with proper labelings of atomic propositions to configurations. These algorithms are useful for a broader range of software verification such as interprocedural data flow analysis and malware detection.

Finally, we investigate the realizability problems for register pushdown models. The realizability problem for a given specification $\mathcal{S}$ is to decide whether there exists an implementation satisfying $\mathcal{S}$. Although the problem is important, the problem has not been studied yet when pushdown computational models give specification and implementation. We investigate the realizability problem for the cases that a specification and an implementation are given by a pair of a PDA and a pushdown transducer (PDT), and a pair of an RPDA and a register PDT (RPDT). We show the problem is solvable in EXPTIME if specification and implementation are given by deterministic PDA and PDT, solvable in 2-

EXPTIME if they are given by deterministic visibly RPDA and RPDT, and undecidable if the specification is nondeterministic PDA or RPDA.

Through this research, we have developed a computational theory for the application of register models. These results can be applied to various fields of software engineering, such as verification and synthesis because these studies allow us to handle recursive programs with data values in a formal way.

Contents

Acknowledgements

First of all, I would like to send sincere thanks to my supervisor, Professor Seki Hiroyuki, and my collaborator, Associate Professor Takata Yoshiaki, for their dedicated research guidance. Since 2017, when I entered to Seki lab, Professor Seki have taught me the way to research, interest of discussion and a lot of things I needed. After I started researching with Associate Professor Takata in 2018, our discussions and researches become more advanced and interesing. Thanks to my great teachers, I have enjoyed my research and I'm finishing my highest study degree. I am also grateful to Professor Shoji Yuen and Yuichi Kaji for carefully reading a draft of this dissertation and giving valuable comments. I would like to thank the members of Seki, Yuen-Nakazawa and Kaji Laboratories for giving a lot of valuable advice and criticism on this research. All of advices from them were always right on target, and they always gave me the helpful perspective I had not noticed. Again, I would like to thank Seki-sensei and Takata-sensei for supporting me kindly.

Part I

Introduction

Background

An extension of finite automata by adding the ability to process infinite data values, such as two counter machine, easily becomes Turing machine-equivalent. There have been active studies on defining computational models adding mild powers of processing data values to classical models. The decidability of basic problems and the closure properties have been investigated for first-order and monadic second-order logics with data equality, linear temporal logic with freeze quantifier [25] and register automata [41]. Register automata (abbreviated as RA) are a natural extension of finite automata incorporating registers that store data values from input and test the equality of data values between input and registers.

Recently, RA has been investigated actively as a computational model of a query language for structured documents such as XML. A query on a structured document can be specified as the combination of a regular pattern and a condition on data values [46, 48]. For query processing and optimization, the decidability of membership and emptiness (hopefully in polynomial time) is necessary as follows. The membership problem asks whether an element e is in the answer set of a query q where the answer must be returned. The emptiness problem asks whether the answer set of a given query is nonempty, where the query can be considered to be redundant or meaningless if there is no answer. The membership and emptiness problems for RA are decidable [41] and the computational complexities are [25, 59].

While RA are sufficient for expressing regular patterns on *paths* of a tree or a graph, it cannot represent *tree* patterns (or patterns over branching paths). Register context-free grammars (RCFG) and register pushdown automata (RPDA) [20] are extensions of classical context-free grammars (CFG) and pushdown au-

tomata (PDA), respectively, for expressing tree patterns. As operational models, RA are extended to tree automata over an infinite alphabet [42, 61]. We call the latter register tree automata (abbreviated as RTA). The membership and emptiness problems of RCFG and RPDA are decidable [20] as well as the equivalence of RCFG and RPDA in their language expressive powers and the closure properties. The computational complexities of the above decision problems are yet to be shown. RCFG, RPDA and RTA can perform only the equality test of data values between input and registers. This is not sufficient when we apply these register models to software construction.

There have been studies on extending traditional models having recursive control structures by adding registers. A pushdown system (PDS) is a pushdown automaton (PDA) without input, which is useful for the abstraction of recursive programs [13, 72]. For pushdown register systems (PDRS) extended with registers, the backward-reachability of PDRS has been shown to be EXPTIME-complete [50]. However, forward-reachability has not been proved yet. In addition, mathematically plausible properties of RPDS including the decidability of reachability, have not been applied to model checking recursive programs against general classes of temporal properties such as linear temporal logic (LTL).

For software synthesis, register transducer (RT) is defined as an extension of the finite transducer, and they are used for representing an implementation (namely, a target model of the synthesis) of the realizability problems [28, 31, 43, 44]. The realizability problem for a given specification $\mathcal{S}$ is to decide whether there exists an implementation satisfying $\mathcal{S}$. This problem is also important in the field of reactive synthesis of recursive programs. However, the decidability and computational complexities of realizability problems for pushdown and register pushdown models are not studied yet.

Overview

This thesis consists of three parts: (1) The definitions of RCFG, RPDA and RTA, followed by the complexity results on the basic problems for these models and a generalization of RCFG (Part II, Chapters 1 to 4). (2) Software verification of register-equipped models (Part III, Chapters 5 and 6). (3) Software synthesis of PDA and RPDA (Part IV, Chapter 7).

Since many abbreviations of register models appear in this dissertation, we summarize them in Figure 1 for convenience.

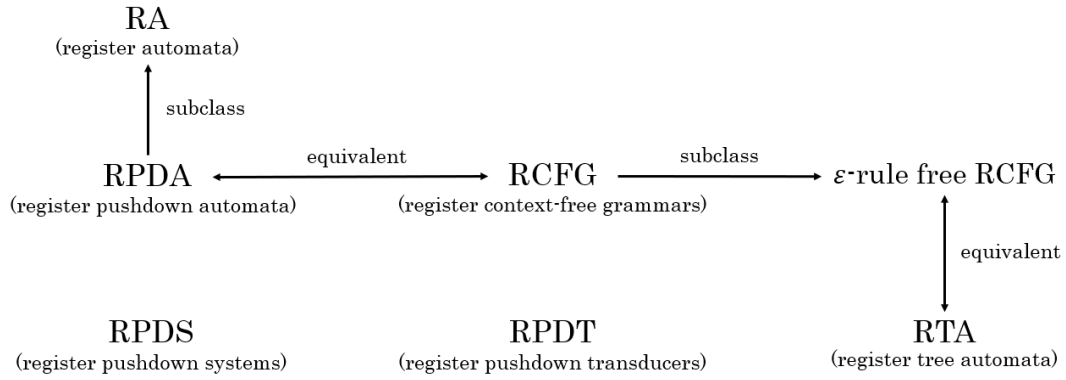

Figure 1: List of register models, abbrebiations and relations. The allow "equivalent" means that two models are conscructable each other in polynomial time.

Part II

Though RCFG, RPDA and RTA are useful computational models for handling stack and tree structure with data values, their complexities of membership and emptiness problems are not known yet. In Part II, we analyze the computational

complexities of these models as Table 1. These results are useful to obtain a computational complexity of more complicated problems we want to solve, such as database queries, model checking and software synthesis with RCFG, RPDA and RTA. Also, for extending range of applications, we expand the definition of guard condition defined as the Boolean combination of the equality check, to the check of an arbitrary relation (such as the total order on numbers). As a result, we prove that the emptiness and membership problems are still decidable if we check other relations, which satisfy *simulation* and *progress* properties.

In Chapter 1, we give an alternative definition of RCFG equivalent to the original definition of RCFG [20] but suitable for modeling a query language as discussed in recent studies on RA [48, 47]. In a derivation of a k-RCFG, k data values are associated with each occurrence of a nonterminal symbol (called an *assignment*) where a production rule is applied only when the guard condition of the rule is satisfied, where a guard condition is a Boolean combination of the equality check between an input data value and the data value in a register. We introduce subclasses of RCFG, including ε-rule free RCFG, growing RCFG, and RCFG with bounded registers. We show that these subclasses have good computational properties in Chapter 2 (see Table 1).

In Chapter 2, we present the *computational complexities of membership and emptiness problems* for RCFG and subclasses of them.

In Chapter 3, we define an RPDA, RTA and subclasses of RPDA and prove their complexities as shown in Table 1.

Table 1: Complexity results on RCFG, RPDA and RTA (Xc means X-complete)

	general RCFG/RPDA	ε-rule free RCFG / non-decreasing RPDA	growing RCFG / growing RPDA	RCFG/RPDA with bounded regs	RTA
Membership	EXPTIMEc	PSPACEc	NPc	in P	NPc
Emptiness	EXPTIMEc	EXPTIMEc	EXPTIMEc	in P	EXPTIMEc

In Chapter 4, we show that ε-removal is possible for RCFG and then we pro-

pose a generalization of RCFG. Beyond theoretical interest, ε-removability is useful because the membership problem for ε-rule free RCFG is PSPACEc while that for general RCFG is EXPTIMEc (see Table 1). In order to remove ε-rules, we introduce a notion called *register type*, which is the quotient of the contents of registers by the equivalence classes induced by the equality relation among data values. In the latter part of the chapter, we reconsider the definition of guard condition, which is defined as the Boolean combination of the equality check in Chapter 1. If we allow the check of an arbitrary relation (such as the total order on numbers), basic problems including membership and emptiness become undecidable in general. By considering this fact, we extend the above mentioned *register type* for an arbitrary relation and then we introduce two properties, namely, simulation and progress, which guarantee that membership and emptiness are decidable. Finally, we prove that the emptiness and membership problems are decidable and ε-removal is possible for *generalized RCFG* (GRCFG) that have both the simulation and progress properties, and these problems become undecidable when either of the properties is missed.

Part III

As a background to explain the contents of Chapter 5, we introduce the regularity preservation property and overview the background of the study. For a mapping T on languages, if we can construct an automaton A_T from a given automaton A such that $L(A_T) = T(L(A))$, then we say that T effectively *preserves regularity*. It is well-known that PDS effectively preserves regularity. More precisely, a PDS $\mathcal{P}$ induces two mappings on the set $ID_{\mathcal{P}}$ of instantaneous descriptions (abbreviated as ID): For $L \subseteq ID_{\mathcal{P}}$, $pre_{\mathcal{P}}(L) = \{c \in ID_{\mathcal{P}} \mid \exists c' \in L : \ c \Rightarrow_{\mathcal{P}} c'\}$ and $post_{\mathcal{P}}(L) = \{c \in ID_{\mathcal{P}} \mid \exists c' \in L : \ c' \Rightarrow_{\mathcal{P}} c\}$. Let $pre^*_{\mathcal{P}}$ and $post^*_{\mathcal{P}}$ be the reflexive transitive closures of $pre_{\mathcal{P}}$ and $post_{\mathcal{P}}$ (as relations), respectively. Both $pre^*_{\mathcal{P}}$ and $post^*_{\mathcal{P}}$ effectively preserve regularity.

In [50], pushdown register systems (PDRS) are introduced as an extension of PDS that can keep data values in the registers and the stack. For PDRS, the saturation algorithm for pre^* was given and the reachability problem was

shown to be EXPTIME-complete in [50]. However, the regularity preservation property for $post^*_{\mathcal{P}}$ is yet to be discussed. Note that the regularity preservation property for $pre^*_{\mathcal{P}}$ does not always imply the property for $post^*_{\mathcal{P}}$ for a given $\mathcal{P}$ in general. Beyond theoretical interest, the property for $post^*_{\mathcal{P}}$ can apply to practical problems such as malware detection. (We will show such an example in Chapter 5.)

In Chapter 5, we first define *register pushdown systems* (abbreviated as RPDS), which are equivalent to PDRS, by using an equivalence relation as the guard condition of a transition rule. As we mentioned before, a guard condition is a partial specification of equality and disequality among the contents of registers and an input data value. In this chapter, the guard conditions of a transition rule of an RPDS and an RA are arbitrary equivalence relations over registers, before and after the transition, and an input data value. This does not change the expressive power because the equivalence relations in the guard conditions correspond to the normal form rules with respect to the register type discussed in Chapter 4. This eases our algorithm of $post^*$ (and pre^*), which constructs a transition rule of RA that simulates consecutive transitions of RPDS and RA by composing the guard conditions of the applied rules. (This algorithm is an extension of the method for proving the regularity preservation property of PDS using *P-automaton* [29].) As a corollary, we show that the *joinability* of regular sets generated by RPDS is decidable. We apply this to malware analysis by using the algorithm.

In Chapter 6, we present model checking register pushdown systems (RPDS) by *Linear Temporal Logic* (LTL). Since RPDS can represent recursive programs with data values using registers and stack, LTL model checking with RPDS can apply to practical verification problems. We show the problem is decidable and investigate its computational complexity. We define Backward-deterministic RA and Büchi RPDS (BRPDS) as related classes of RA and RPDS, respectively. A Backward-deterministic RA is used to define regular valuation for the chapter. The LTL model checking problem for RPDS asks whether every run of $\mathcal{P}$ reachable from a start configuration c_0 satisfies an LTL formula φ under a valuation Λ. We show the decidability of the LTL model checking problem for RPDS with

simple valuations and with regular valuations. In the case of simple valuations, we reduce the model checking problem to the membership problem for RA. The size of the constructed RA is exponential to the input size of the problem and the problem is shown to be EXPTIME-complete. The problem with regular valuation is EXPTIME-complete by reducing the problem to simple valuation.

Part IV

Though stack structure is necessary to represent many practical programs, the theoretical properties of software synthesis, such as decidability of realizability problem, with using pushdown models have not been known well yet. In Chapter 7, we investigate the *realizability problem* that a specification and an implementation are given by a PDA and a *pushdown transducer* (PDT), respectively. Then we further investigate the problem between an RPDA and a *register pushdown transducer* (RPDT). PDT is regarded as a model of a recursive program that emits outputs according to the inputs given from its environment.

The main difficulties to solve the realizability problem here come from the fact that the class of languages recognized by nondeterministic PDA (NPDA) (i.e., context-free languages) does not have the closure properties under the intersection. For these difficulties, this chapter mainly considers deterministic PDA (DPDA). (Note that, PDT is deterministic by definition.) We show that the realizability problem for DPDA is decidable and the problem is undecidable for NPDA. The former is proved by the well-known property that the (two-players zero-sum parity) pushdown game is decidable and a winning strategy of the game can be constructed as a PDT [72]. For the realizability problem for RPDA and register PDT (RPDT), we reduce the problem for DRPDA to the problem for DPDA. In this reduction, we convert a given DRPDA to a DPDA that recognizes the projection onto the finite alphabet of the language recognized by the DRPDA. For this purpose, we assume that a given DPDA is visible with reference to the guard condition inherited from the DRPDA as well as stack operations (see [5] for the visibility of stack operation). Under these assumptions, we show that the realizability problem is decidable for visibly DRPDA and RPDT.

Related work

Early studies on query optimization and static analysis for structured documents used traditional formalizations such as the tree automata, the two-variable logic and the linear temporal logic. Data values associated with documents have been neglected. Later, richer formalizations have been developed so that data value in structured documents can be handled, automata (register automata, pebble automata, data automata) and logics (two-variable logics with data equality, LTL with freeze quantifier). We list these formalizations in view of handling structured documents with data values as follows.

Formalizations for data languages

Register automata: Register automata (RA) were first introduced in [41] as finite-memory automata. The membership and emptiness problems of RA are decidable, and closed under the operation of union, concatenation and Kleene-star but not closed under complementation. The computational complexity of membership and emptiness problems were analyzed in [59, 25]. Both language equivalence and language inclusion problems are undecidable in general [55], whereas for the deterministic RA the problems are decidable in P and PSPACE-complete, respectively [51]. The classical Myhill-Nerode congruence is extended to data languages in [17, 71].

Register context-free grammars and register pushdown automata: Extensions by adding registers are also done for PDA [22, 50, 66] and context-free grammar [20, 63, 64], called register pushdown automaton (RPDA) and register context-free grammar (RCFG), respectively. In [20], the equivalence of the two models as

well as the decidability results and closure properties similar to RA were shown. RPDA is a natural model for recursive programs with data values, while RCFG has an advantage such that explicit representation of pushdown stack is not needed. For other extensions of PDA and verification of them, see [67, 73, 58].

LTL with freeze quantifier: Linear temporal logic (LTL) was extended to LTL$\downarrow$ with freeze quantifier [26, 25]. 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 [25]. Especially, they showed that the emptiness problem for (both nondeterministic and deterministic) RA is 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 $\mathrm{FO}^2(<, +1)$ is decidable and corresponds to Core XPath where $<$ is the ancestor-descendant relation and $+1$ is the parent-child relation. The logic was extended to those with data equality. It was shown in [10] 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 [25]. Also, it was shown in [9] that $\mathrm{FO}^2(\sim, +1)$ and existential $\mathrm{MSO}^2(\sim, +1)$ are decidable on unranked data trees.

Other automata for data words: There are extensions of automata to deal with data in a restricted way other than RA, namely, data automata [15] and pebble automata (PA) [55]. It is desirable for a query language to have an efficient data complexity for the membership problem. Libkin and Vrgoč [48] argue that register automata (RA) are the only model that have this property among the above-mentioned formalisms and adopts RA as the core computational model of their queries on graphs with data. Neven [55] considers variations of RA and PA, whether 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. Other extensions of RA are found in [11, 19, 24, 12]. Nominal automaton [11] is an extension of finite automaton by using nominal sets, which are infinite sets having finite orbits of

group actions and equivariant functions among them.

Applications

In this dissertation, we investigate decision procedures for fundamental properties of the register models along with their computational complexities. These results are useful for analyzing complexities of concrete algorithms when these register models are applied to database queries, software verification and synthesis, language learning, etc. We propose concrete algorithms for model checking and software synthesis of the register models, extending the algorithms for non-register cases. In the following, we introduce some studies of applications relevant to the results of this dissertation.

Tree automata and data XPath: In [42], 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, we introduce top-down tree automata for data trees called alternating tree register automata (ATRA) [39] corresponding to forward XPath. We show the decidability of the emptiness problems for these classes. While RTA works on ranked data trees, ATRA works on unranked data trees. Later, [32, 33] extended ATRA so that (1) they guess a data value to store in a register, and also (2) they universally quantify the data values encountered in a given run; the emptiness problem of the extended ATRA was shown decidable as well-structured transition systems. Also, bottom-up tree automata for unranked data trees, which correspond to vertical XPath were introduced and its emptiness problem was shown decidable in [34]. Since XML documents are modeled as unranked trees, ATRA is regarded as a better model for answering queries by XPath than RTA. However, the complexity of the emptiness problem of ATRA is not elementary and an appropriate subclass would be needed for practical applications.

Regularity preservation of PDS: Regularity preservation property of PDS was shown by Büchi in 1964 (also see [35]). Later, this property of PDS was widely used for PDS model-checking, one of the most successful approaches to infinite-state model-checking [72, 13, 29]. For example, PDS model-checking was applied to security verification of stack inspection as a security enforcement mechanism in Java runtime environment [30, 56]. This property was also studied for term rewrite systems (TRS), where a tree language L is regular if there is a (finite-state) tree automaton that recognizes L. Some subclasses of TRS have been shown to effectively preserve (forward) regularity, such as right-linear monadic (RL-M) [60], linear semi-monadic (L-SM)[23], linear generalized semi-monadic (L-GSM)[36], right-linear semi-monadic (RL-SM)[54] and right-linear finite path overlapping (RL-FPO)[68]. There are proper inclusions among these subclasses: RL-M $\cup$ L-SM $\subset$ RL-SM, L-SM $\subset$ L-GSM, RL-SM $\cup$ L-GSM $\subset$ RL-FPO.

Application to infinite model checking: The infinity of data sets causes difficulty in analyzing programs that deal with data values. In previous studies such as [3, 58, 14], this problem was solved by a computational model with finite elements. In [3], a pushdown automaton with gap-order constraints is defined, which can check whether data values in two registers have a gap larger than a given natural number or not, and the reachability of the model with such gap constraints is shown decidable. [58] introduces an extension of PDS, which can encode global and local variables as well as call and return of a procedure and create unboundedly many objects. They also showed such creation of objects can be simulated by a finite set of natural numbers. The reachability problem of multithreaded PDS dealing with pointer values is addressed in [14] and shown to be decidable on the assumptions of bounded depth heap and finite context switch. Multi-pushdown models with data values have been also studied under certain constraints (e.g., on context switch) [12, 2]. However, none of these studies discusses the computational complexity only the reachability, i.e., runs (or traces) of finite length has been investigated.

Reactive synthesis: Studies on reactive synthesis originated in the 1960s and have been one of central topics in formal methods [21]. Among them, Büchi and Landweber [16] showed EXPTIME-completeness of the problem when a

specification is given by a finite ω-automaton (a finite automaton on infinite words). Pnueli and Rosner [57] showed 2EXPTIME-completeness of the problem when an LTL formula gives a specification. We can find an excellent tutorial and survey of the previous studies on the synthesis problem in [8]. The standard approach to the problem is as follows. Assume that, for example, a specification is given as a deterministic ω-automaton $\mathcal{A}$. We convert $\mathcal{A}$ to a tree automaton (or equivalently, a parity game) $\mathcal{B}$ by separating the input and output streams. Then, we test whether $L(\mathcal{B}) \neq \emptyset$ (or equivalently, there is a winning strategy for the player I in $\mathcal{B}$). The answer to the problem is affirmative if and only if $L(\mathcal{B}) \neq \emptyset$, and any $t \in L(\mathcal{B})$ (or any winning strategy for $\mathcal{B}$) is an implementation of the specification.

Realizability for RA: When considering the realizability for RA, the difficulty is to identify the number of registers needed for implementing the specification, i.e., the number of registers of RT. If the upper bound of the registers of RT is not given as an input, the realizability problem is undecidable for both nondeterministic and universal RA [31]. If the upper bound of the registers is *a priori* known, the problem (called the bounded realizability problem) is shown to be decidable in EXPTIME for universal RA [43]. In [31], it is shown that the bounded realizability problem remains undecidable for nondeterministic RA (NRA) and becomes decidable in 2EXPTIME for a subclass called test-free NRA.

Learning algorithms for data languages: Active automata learning was first studied by Angluin in 1987 (see [7]) and has been extended for various models including register-equipped ones. Howar et al. [37] present a learning algorithm for register automata by adapting the Nerode relation of data languages. A tool for learning register automata is also constructed, called "LearnLib" [38, 49]. Aarts et al. [1] extend this study to register automata with output to generate fresh output values. Vaandrager [70] introduces an application of register automata learning to system-modeling. An et al. [6] present an algorithm for learning deterministic one-clock timed automata under continuous-time semantics. Moerman et al. [53] construct an algorithm to learn nominal automata. Drews et al. [27] present an algorithm to learn symbolic finite automata (s-FA) and investigate a condition of s-FA that the learning algorithm is applicable.

Part II

Computational Results and Generalization

Outline of Part II

- In Chapter 1, we introduce some notions such as *data value*, *register*, *guard expression* and *data word* and define RCFG and its subclasses, ε-rule free RCFG and growing RCFG.
- We show the computational complexity of the basic decision problems for RCFG and their subclasses in Chapter 2, and the computational complexity of problems for RTA, RPDA and their subclasses in Chapter 3. For example, the membership and emptiness problems for RCFG are EXPTIME-complete and the membership problem for ε-rule free RCFG and growing RCFG becomes PSPACE-complete and NP-complete, respectively.
- In Chapter 4, we define generalized RCFG (GRCFG) where an arbitrary binary relation can be specified in the guard condition. Since the basic decision problems are shown to be undecidable in general, we introduce two properties of GRCFG, simulation and progress, which guarantee the decidability of these problems.

Chapter 1 Basic definitions

A *register context-free grammar* (abbreviated as RCFG) was introduced in [20] as a grammar over an infinite alphabet. We will mainly discuss the properties of RCFG in Part I. In this chapter, we define RCFG as a grammar over the product of a finite alphabet and an infinite set of data values, following recent notions [48, 47]. Note that these differences are not essential.

1.1 Preliminaries

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

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

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

where $x_i \in \{x_1, \ldots, x_k\}$. Let $\mathrm{ff}, x_i^{\neq}$ and $\psi_1 \wedge \psi_2$ denote $\neg\mathrm{tt}, \neg x_i^{=}, \neg(\neg\psi_1 \vee \neg\psi_2)$, respectively. The description length of guard expression ψ is defined as

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

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.

- $\theta, d \models \mathrm{tt}$
- $\theta, d \models x_i^{=}$ iff $\theta(i) = d$
- $\theta, d \models \psi_1 \vee \psi_2$ 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 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.

1.2 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 $A \in V$, $\psi \in F_k$, $i \in [k]$ 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 $A \in V$, $\theta \in \Theta_k$ 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 $(A, \psi, i) \to c_1 \dots c_n \in R$ (resp. $(A, \psi) \to c_1 \dots c_n \in R$) such that

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

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

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.

Fig.1.1 shows an example of a direct derivation from $(S, \theta_\perp)$ using $(S, \mathrm{tt}, 1) \to A(a, 1)B$. In the figure, d_1 is an arbitrary data value in D and $\theta_1 = \theta_\perp[1 \leftarrow d_1]$, 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 $(S, \mathrm{tt}, 1) \to (a, 1)S(a, 1)$ and $(S, \mathrm{tt}, 1) \to \varepsilon$, $L(G) =$

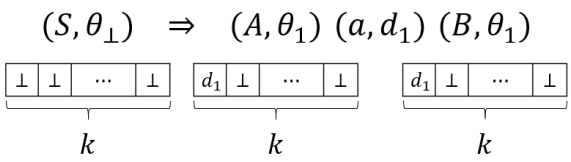

Figure 1.1: An example of derivation from $(S, \theta_\perp)$ using $(S, \mathrm{tt}, 1) \to A(a, 1)B)$

$\{(a, d_1) \dots (a, d_n)(a, d_n) \dots (a, d_1) \mid n \geq 0\}$.

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

Example 1.2.1. *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 $L(G) = \{(a, \perp)\}$. 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:

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

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 $(S, \theta_\perp)$ 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* i*th bit of the binary counter represented by* θ *equals* b. *After this derivation, the* i*th 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.*

Theorem 1.2.1. *For an arbitrary* k*-RCFG* $G = (V, R, S)$, *we can construct an equivalent* $(k+1)$*-RCFG* $G' = (V', R', S')$ *such that* R' *has no rule of the form* $(A, \psi) \to \alpha$.

Proof We construct G' by using the $(k+1)$th register as dummy:

- $V' = V$, $S' = S$,
- $R' = \{(A, \psi, i) \to \alpha \mid (A, \psi, i) \to \alpha \in R \text{ or } ((A, \psi) \to \alpha \in R \wedge i = k+1)\}$.

Chapter 2 The Complexities of RCFG

In this chapter, we will show some results about properties of RCFG, including its complexities of membership and emptiness problems.

2.1 Basic Properties of RCFG

The following properties of RCFG were first shown in [20]. We fix a finite alphabet Σ and a set D of data values.

Proposition 2.1.1. *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 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 θ in Θ_k to be a k-tuple over D, i.e. an element of D^k, and thus $V' \subseteq V \times \Theta_k$.
- $R' = \{(A, \theta) \to X \mid (A, \theta) \in V',\ X \in (V' \cup (\Sigma \times D'))^*,\ \text{and } (A, \theta) \Rightarrow_G X\}$.

- $S' = (S, \bot)$.

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 2.1.2. *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:

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

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 $(A, \psi, i) \to \alpha$ and the content of i-th register of θ is updated as d, i.e., the register assignment appearing in X is $\theta' = \theta[i \leftarrow d]$. Let θ'' be the register 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 $d' \in \{d_1, \ldots, d_{k+1}\}$ such that $\theta(j) \neq d'$ for every j $(1 \leq j \leq k)$. Let $\theta'' = \theta[i \leftarrow d']$.

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 $(S, \bot) \overset{*}{\Rightarrow}_G w$, 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 2.1.3. *The membership problem for RCFG is decidable.*

Proposition 2.1.4. *The emptiness problem for RCFG is decidable.*

Proposition 2.1.5. *The class of data languages generated by RCFG are closed under union, concatenation and Kleene-star.*

2.2 Upper Bounds

Lemma 2.2.1. *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 2.1.1, 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 2.1.1.*

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 2.1.1.

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

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

2.2.1 Membership problems for general, ε-rule free and growing RCFG

Lemma 2.2.2. *The membership problem for RCFG is decidable in EXPTIME.*

Proof Assume we are given a k-RCFG G and a data word $w = (a_1, d_1)\ldots(a_n, d_n)$ as an input. Consider the CFG G' such that $L(G) \cap (\Sigma \times \{d_1, \ldots, d_n\})^*$, which is constructed in the proof of Proposition 2.1.3. By Lemma 2.2.1, $\|G'\| \in O(\|G\|c^{k+1}k)$ where $c = \max\{|w|, k+1\}$. Since the membership problem for CFG is decidable in deterministic polynomial time [62], the problem $w \in L(G)$ for k-RCFG G is decidable in deterministic time exponential to k.

Lemma 2.2.3. *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 $S \overset{*}{\Rightarrow}_G \alpha \overset{*}{\Rightarrow}_G w$, $|\alpha| \leq |w|$ 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 2.2.4. *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 $S \overset{*}{\Rightarrow}_G \alpha \Rightarrow_G \alpha'$, $|\alpha| < |\alpha'|$ 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.

2.2.2 Emptiness problem

Lemma 2.2.5. *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 2.1.1. By Lemma 2.2.1, $\|G'\| \in O(\|G\| \cdot |D_k|^{k+1}k)$ 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.

2.2.3 RCFG with bounded registers

Theorem 2.2.1. *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 2.1.1 from G and a finite set $D' \subseteq D$. Then $\|G'\| = O(\|G\| \cdot |D''|^{k+1}k)$ holds by Lemma 2.2.1. If k is a constant independent of the choice of G, then $\|G'\|$ is a polynomial of $\|G\|$ and $|D''|$. Hence, by Lemma 2.2.2 and Lemma 2.2.5, 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.

2.3 Lower Bounds

2.3.1 Membership problems for general, ε-rule free and growing RCFG

Before showing proofs, we give a definition of *alternating Turing machine*, which will be used to prove Theorem 2.3.1.

Definition 2.3.1. *An* alternating Turing machine *[18, 62] (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

$$\exists a, b \in \Gamma,\ \exists \beta, \gamma \in \Gamma^*,\ |\beta| = j - 1,\ \alpha = \beta a \gamma \text{ such that}$$
$$\alpha' = \begin{cases} \beta b \sqcup & \text{if } |\alpha| = j \text{ (i.e. } \gamma = \varepsilon) \text{ and } j' = j + 1, \\ \beta b \gamma & \text{otherwise,} \end{cases}$$
$$j' = \begin{cases} j - 1 & \delta(q, a) \ni (q', b, L), \text{ or} \\ j + 1 & \delta(q, a) \ni (q', b, R). \end{cases}$$

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.

$$g(q, \alpha, j) = \begin{cases} \mathrm{tt} & q = q_{acc} \\ \mathrm{ff} & q = q_{rej} \\ \bigvee_{(q,\alpha,j) \to (q',\alpha',j')} g(q', \alpha', j') & q \in Q_e \\ \bigwedge_{(q,\alpha,j) \to (q',\alpha',j')} g(q', \alpha', j') & q \in Q_a \end{cases} \tag{2.1}$$

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 [18].

Theorem 2.3.1. *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 2.2.2, 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

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

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 (2.2), and that data value is guaranteed to be different from the value of every jth register ($2 \le j < i$) by Rule (2.3).

$$(T_{(i-1,i-2)}, x_1^{\neq}, i) \to T_{(i,1)} \quad \text{for } 2 \le i \le |\Gamma|, \tag{2.2}$$

$$(T_{(i,j-1)}, x_i^{=} \wedge x_j^{\neq}) \to T_{(i,j)} \quad \text{for } 2 \le j < i \le |\Gamma|. \tag{2.3}$$

- 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:

$$(T_{(|\Gamma|,|\Gamma|-1)}, \mathrm{tt}) \to W_0, \tag{2.4}$$

$$(W_{i-1}, x_{u_i}^{=}, |\Gamma| + i) \to W_i \quad \text{for } 1 \le i \le |u|. \tag{2.5}$$

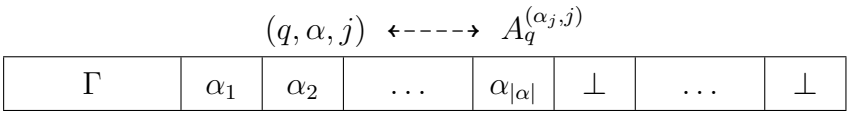

Figure 2.1: The correspondence between M's ID and G's nonterminal symbol and registers.

- 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 register 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 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. 2.1.

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

$$(W_{|u|}, \mathrm{tt}) \to A_{q_0}^{(u_1,1)}. \tag{2.6}$$

  - 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:

$$(A_q^{(i,j)}, x_{b_k}^{=}, |\Gamma| + j) \to B_{q_k}^{(a, j+s(m_k))}. \tag{2.7}$$

    If $q \in Q_a$, then:

$$(A_q^{(i,j)}, \mathrm{tt}) \to C_q^{(i,j,1)} \ldots C_q^{(i,j,t)}, \tag{2.8}$$

$$(C_q^{(i,j,k)}, x_{b_k}^{=}, |\Gamma| + j) \to B_{q_k}^{(a, j+s(m_k))}. \tag{2.9}$$

Note that if $t = 0$, then the right-hand side of Rule (2.8) is ε. We also construct the following rule for each $q' \in Q$, $a \in \Gamma$, and $1 \leq j' \leq p(n)$:

$$(B_{q'}^{(a,j')}, x_a^{=} \wedge x_{|\Gamma|+j'}^{=}) \to A_{q'}^{(a,j')}. \tag{2.10}$$

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

$$(A_{q_{acc}}^{(i,j)}, \mathrm{tt}) \to \varepsilon \tag{2.11}$$

We can show for each ID (q, α, j),

$$g(q, \alpha, j) = \mathrm{tt} \ \text{ iff } \ (A_q^{(\alpha_j, j)}, \theta_\alpha) \stackrel{*}{\Rightarrow}_G \varepsilon \tag{2.12}$$

by induction on the number of applications of the accepting condition (2.1) for only if part and by induction on the length of the derivation for if part.

We can easily prove that $(S, \bot) \stackrel{*}{\Rightarrow}_G (A_{q_0}^{(u_1,1)}, \theta_u)$, and moreover, if $(S, \bot) \stackrel{*}{\Rightarrow}_G \varepsilon$, then this derivation must be $(S, \bot) \stackrel{*}{\Rightarrow}_G (A_{q_0}^{(u_1,1)}, \theta_u) \stackrel{*}{\Rightarrow}_G \varepsilon$. By letting $(q, \alpha, k) = (q_0, u, 1)$ in property (2.12) and by the above-mentioned fact, we obtain $u \in L(M) \Leftrightarrow g(q_0, u, 1) = \mathrm{tt} \Leftrightarrow ((S, \bot) \stackrel{*}{\Rightarrow} \varepsilon) \Leftrightarrow \varepsilon \in L(G)$. 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.

Theorem 2.3.2. *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 2.2.3, 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 2.3.1. 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 (2.8), whose right-hand side is ε if $t = 0$). We modify Rule (2.11), the ε-rule for the accepting state, as $(A_{q_{acc}}^{(i,j)}, \mathrm{tt}) \to (a, 1)$. 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 $\bot$).

Theorem 2.3.3. *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 2.2.4, 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 $(1 \leq i \leq m)$ is a literal y_j or $\overline{y_j}$ for some j $(1 \leq j \leq n)$. For i $(1 \leq i \leq m)$, 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

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

The first register of the constructed RCFG G is used for keeping a data value (possibly) different from $\perp$, 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 $y_1, \overline{y_1}, y_2, \overline{y_2}, \dots, y_n, \overline{y_n}$. 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 $D \setminus \{\perp\}$. 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 $d \in D \setminus \{\perp\}$ to make the input data word for the membership problem. Hence, we have shown the NP-hardness of the problem.

2.3.2 Emptiness problem

Theorem 2.3.4. *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 2.2.5, it is enough to show EXPTIME-hardness. In the proof of Theorem 2.3.1, we construct ε-rules when the state q under consideration is a universal state that has no successor (see Rule (2.8)) or q is the accepting state (see Rule (2.11)). We modify those production rules (2.8) and (2.11) 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 2.3.1 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 2.3.3, 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 2.3.5. *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 $d, \theta \models \psi$. That is, regardless of θ, rule $(A, \psi, i) \to \alpha$ 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 $(A, \psi, i) \to B(a, j)$ 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.

Chapter 3 The Complexities of RTA and RPDA

In this chapter, we will show some results about properties of register tree automata (RTA) and register pushdown automata (RPDA), including its complexities of membership and emptiness problems.

3.1 Register Tree Automata

3.1.1 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 $\mathtt{Lab}(t, p)$ and the subtree $t|_p$ of t at p is defined as $\mathtt{Lab}(t, \varepsilon) = f$, $t|_\varepsilon = t$, $\mathtt{Lab}(t, jp) = \mathtt{Lab}(t_j, p)$ 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 3.1.1. *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 $f \in \Sigma_n$, $q \in Q$, $\psi \in F_k$, $1 \leq i \leq k$ *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* $\rho : Pos(t) \to Q \times \Theta_k$ *satisfying the following condition: For* $p \in Pos(t)$, *if* $\rho(p) = (q, \theta)$, $\mathtt{Lab}(t, p) = f$, *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* $\theta, \delta(p) \models \psi$ *and* $\theta_j = \theta[i \leftarrow \delta(p)]$ *(resp.* $\theta_j = \theta$*)* $(1 \leq j \leq n)$. *A run* ρ *is* accepting *if* $\rho(\varepsilon) = (q_0, \theta_\perp)$. *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* $\tau\}$.

Theorem 3.1.1. *For an arbitrary* k*-RTA* $A = (Q, q_0, T)$, *we can construct an equivalent* $(k+1)$*-RTA* $A' = (Q', q_0', T')$ *such that* T' *has no rule of the form* $f(q, \psi) \to (q_1, \ldots, q_n)$.

Proof Sketch This theorem can be proved in a similar way to Theorem 1.2.1.

3.1.2 Computational complexity

Theorem 3.1.2. *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.

Next, we prove the NP-hardness by providing a polynomial-time reduction from the satisfiability problem for 3-Conjunctive Normal Form (3CNF). Let $\phi = (a_1 \vee b_1 \vee c_1) \ldots (a_m \vee b_m \vee c_m)$ be a 3CNF over Boolean variables $y_1, \ldots, y_n$ where each a_i, b_i, c_i $(1 \leq i \leq m)$ is a literal y_j or $\overline{y_j}$ for some j $(1 \leq j \leq n)$. For i $(1 \leq i \leq m)$, we define guard expressions $\psi_{a_i}, \psi_{a_i}^-$ as $\psi_{a_i} = x_j^=$ if $a_i = y_j$ and $\psi_{a_i} = \mathrm{ff}$ otherwise, and $\psi_{a_i}^- = x_j^=$ if $a_i = \overline{y_j}$ and $\psi_{a_i}^- = \mathrm{ff}$ otherwise. We

also define the same notation $\psi_{b_i}, \psi_{b_i}^{-}, \psi_{c_i}, \psi_{c_i}^{-}$ for b_i and c_i. We construct the $(n+2)$-RTA $A = (Q, q_0, T)$ over $\Sigma = \Sigma_0 \cup \Sigma_1$ with $\Sigma_0 = \{a\}$ and $\Sigma_1 = \{f\}$ where $Q = \{q_0, q_{P_0}, \ldots, q_{P_n}, q_{C_0}, \ldots, q_{C_m}\}$ and

$$\begin{aligned} T = &\{f(q_0, x_1^{\neq}, 2) \to q_{P_0}\} \\ &\cup \{f(q_{P_{i-1}}, x_1^{=} \vee x_2^{=}, i+2) \to q_{P_i} \mid 1 \leq i \leq n\} \\ &\cup \{f(q_{P_n}, \mathrm{tt}) \to q_{C_0}\} \\ &\cup \{f(q_{C_{i-1}}, (x_1^{=} \wedge (\psi_{a_i} \vee \psi_{b_i} \vee \psi_{c_i})) \vee \\ &\qquad\qquad (x_2^{=} \wedge (\psi_{a_i}^{-} \vee \psi_{b_i}^{-} \vee \psi_{c_i}^{-}))) \to q_{C_i} \mid 1 \leq i \leq m\} \\ &\cup \{a(q_{C_m}, \mathrm{tt})\}. \end{aligned}$$

The first and second registers of the constructed RTA A are used for keeping different data values representing tt and ff, respectively. A nondeterministically loads one of those two data values representing tt and ff to the succeeding n registers that encode a truth value assignment to $y_1, \ldots, y_n$. Then A checks whether each clause in ϕ is satisfied by using the conditions in production rules. It is not difficult to show that ϕ is satisfiable if and only if $f^{1+n+1+m}(a) \in L(A)$. Hence, we have shown the NP-hardness of the problem.

Theorem 3.1.3. *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 2.3.4).

- 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. By Theorem 1.2.1, we assume G has no rule whose third component in its left-hand is missing without loss of generality. 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:

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

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

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. By Theorem 3.1.1, we assume G has no rule whose third component in its left-hand is missing without loss of generality. We construct k-register RCFG $G_A = (V, R, A_{q_0})$ where

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

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

3.2 Register Pushdown Automata

3.2.1 Definitions

A register pushdown automaton (abbreviated as RPDA) was originally defined in [20] as a pushdown automaton with registers over an infinite alphabet and the equivalence between RCFG and RPDA was shown in [20]. In this section, we

define RPDA as a pushdown automaton that takes a data word as an input by the same reason for RCFG. We also present equivalance 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 [20] where the translation is divided in three steps by using M-grammar and simple M-grammar as intermediate models.

Definition 3.2.1. *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 start symbol in 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:*

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

where $p, q \in Q$, $a \in (\Sigma \cup \{\varepsilon\})$, $i, j_1, j_2 \in [k]$, and $\psi \in F_k^{top}$ where F_k^{top} is the set of guard expressions defined in the same way as F_k, except adding $top^=$ as an atomic proposition. A rule is called an ε-rule if $a = \varepsilon$. As formally defined below, an instanteneous description of an RPDA satisfies $top^=$ if and only if the input data value equals to the data value at the stack top of the ID.

- $\theta, d, e \models top^=$ *iff* $d = e$

where $\theta \in \Theta_k$, $d, e \in D$ (d and e are supposed to be the input data value and the data value at the stack top, respectively). The other cases of the definition of F_k^{top} are the same as in Section 2.1. 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 $r = (p, \psi, i) \xrightarrow{a} (q, J) \in \delta$ $(0 \leq |J| \leq 2)$ *is* $2\log|Q| + \|\psi\| + 3\log(k+1) + \log(|\Sigma|+1)$ *because there are two states* $p, q \in Q$*, a guard expression* $\psi \in F_k^{top}$*, at most three register indexes* i *and those in* J*, and* $a \in \Sigma \cup \{\varepsilon\}$ *in* r*.*

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 $(p, \psi, i) \xrightarrow{a} (q', J) \in \delta$ (resp. $(p, \psi) \xrightarrow{a} (q', J)$), data values $d, e \in D$ and $\gamma \in D^*$ such that

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

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.

For a k-RPDA $P = (Q, q_0, \delta)$ and $w \in (\Sigma \times D)^*$, if $(q_0, \theta_\perp, w, \perp) \overset{*}{\Rightarrow} (q, \theta, \varepsilon, \varepsilon)$ for some $q \in Q$ and $\theta \in \Theta_k$, then P accepts w, and else P rejects w. In the former case, 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.

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

Theorem 3.2.1. *For an arbitrary* k*-RPDA* $P = (Q, q_0, \delta)$*, we can construct an equivalent* $(k+1)$*-RPDA* $P' = (Q', q_0', \delta')$ *such that* δ' *has no rule of the form* $(p, \psi) \xrightarrow{a} (q, J)$*.*

Proof Sketch This theorem can be proved in a similar way to Theorem 1.2.1.

3.2.2 Computational complexity

Theorem 3.2.2. *The membership problem for RPDA is EXPTIME-complete.*

Proof It suffices to prove the following two properties because the membership problem for RCFG is EXPTIME-complete.

- 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 different data values in the last $|\Sigma| + |V|$ registers:

$$(q_{i-1}, \bigwedge_{j=1}^{i-1} x_j^{\neq}, k + i) \xrightarrow{\varepsilon} (q_i, 1) \quad i \in [|\Sigma| + |V|]. \tag{3.1}$$

  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 abbreviate $x_{\gamma(a)}$ as x_a (and $x_{\gamma(A)}$ as x_A respectively) if there is no ambiguity.

- 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 to 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

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

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

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

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 register assignment associated with S.

$$(q_{(|\Sigma|+|V|)}, tt) \xrightarrow{\varepsilon} (q_{read}, \gamma(S)1 \cdots k). \tag{3.2}$$

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.

– If the stack top is equal to x_A $(A \in V)$ and $r = (A, \psi, i) \to X_1 \cdots X_n \in R$, 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 corresponding data values corresponding to the right-hand side of r:

$$(q_{read}, x_A^{=} \wedge top^{=}) \xrightarrow{\varepsilon} (q_r^0, \varepsilon), \tag{3.3}$$

$$(q_r^{j-1}, top^{=}, j) \xrightarrow{\varepsilon} (q_r^j, \varepsilon) \quad j \in [k-1] \tag{3.4}$$

$$(q_r^{k-1}, top^{=}, k) \xrightarrow{\varepsilon} (q_r^k, 1). \tag{3.5}$$

$$(q_r^k, \bigwedge_{a \in \Sigma} x_a^{\neq} \wedge \bigwedge_{A \in V} x_A^{\neq} \wedge \psi, i) \xrightarrow{\varepsilon} (q_{read}, J_1 \cdots J_n). \tag{3.6}$$

where $J_s = \gamma(a)h$ (if $X_s = (a, h) \in (\Sigma \times [k])$) and $\gamma(X_s)1 \cdots k$ (if $X_s \in V$) for $s \in [n]$.

– If the stack top is equal to x_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:

$$(q_{read}, x_a^{=} \wedge top^{=}) \xrightarrow{\varepsilon} (q_a, \varepsilon), \tag{3.7}$$

$$(q_a, top^{=}) \xrightarrow{a} (q_{read}, \varepsilon). \tag{3.8}$$

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 G.

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 by Theorem 3.2.1. 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

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

and R is constructed as follows.

We construct production rules so that each nonterminal $[p, q]$ $(p, q \in Q)$ together with a $3K$-assignment 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:

$$\begin{aligned} (S, \mathrm{tt}, K+1) &\to S_1, \\ (S_{i-1}, \mathrm{tt}, K+i) &\to S_i \quad (2 \leq i \leq k-1), \\ (S_{k-1}, \mathrm{tt}, K+k) &\to [q_0, q] \quad (q \in Q). \end{aligned}$$

For $\psi \in F_k^{top}$, 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 $r : (p, \psi, j) \xrightarrow{a} (p', j_1 j_2) \in \delta$, 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.

$$\begin{aligned}([p,q], m(\psi), j) &\to (a,j)[p',q]_r^{'} \text{ (if } a \in \Sigma) \text{ or } [p',q]_r^{'} \text{ (if } a = \varepsilon),\\ ([p',q]_r^{'}, x_{j_1}^{=}, K) &\to [p',q]_r^{''},\\ ([p',q]_r^{''}, x_{j_2}^{=}, 3K) &\to [p',q]_{new}^{0}.\end{aligned}$$

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:

$$\begin{aligned}([p',q]_{new}^{i-1}, \mathrm{tt}, 2K+i) &\to [p',q]_{new}^{i} \quad (i \in [k]),\\ ([p',q]_{new}^{k}, \mathrm{tt}) &\to [p',t]_{left}^{0}[t,q]_{right}^{0}.\end{aligned}$$

- 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 K data values in the $(2K+i)$-th registers ($i \in [K]$) to the first K registers associated with nonterminal $[t,q]$ that will simulate the right part:

$$\begin{aligned}([p',t]_{left}^{i-1}, x_{2K+i}^{=}, K+i) &\to [p',t]_{left}^{i} \quad (i \in [k]),\\ ([p',t]_{left}^{k}, x_{3K}^{=}, 2K) &\to [p',t],\\ ([t,q]_{right}^{i-1}, x_{2K+i}^{=}, i) &\to [t,q]_{right}^{i} \quad (i \in [k]),\\ ([t,q]_{right}^{k}, x_{3K}^{=}, K) &\to [t,q].\end{aligned}$$

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

$$\begin{aligned}([p,q], m(\psi), j) &\to (a,j)E_0 \text{ (if } a \in \Sigma) \text{ or } E_0 \text{ (if } a = \varepsilon),\\ (E_{i-1}, x_i^{=} \wedge x_{K+i}^{=}) &\to E_i \quad (i \in [k-1]),\\ (E_{k-1}, x_k^{=} \wedge x_{K+k}^{=}) &\to \varepsilon.\end{aligned}$$

These production rules simulate the above pop rule r by guessing the assignment when the state reaches 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 leghth of an accepting run of P, and $L(G_P) \subseteq L(G_P)$ by induction on the length of a derivation of G_P. $\square$

In the rest of the section, we show that the membership problems for non-decreasing and growing RPDA are PSPACE-complete and NP-complete, respectively. Although the lower-bound proofs are similar to those for ε-rule free and growing RCFG.

Theorem 3.2.3. *The membership problem for non-decreasing RPDA is PSPACE-complete.*

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

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

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

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

and $\delta_{\mathcal{A}}$ is constructed as follows. Without loss of generality, we assume that $\Gamma = \{1, 2, \ldots, |\Gamma|\} \subseteq \mathbb{N}$ 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 transition rules that load different data values in the first $|\Gamma|$ registers. Note that we keep the initial value $\perp$ in the first register. To the ith register ($2 \leq i \leq |\Gamma|$), a data value different from $\perp$ is assigned by Rule (3.9), and that data value is guaranteed to be different from the value of every jth register ($2 \leq j < i$) by Rule (3.10).

$$(T_{(i-1,i-2)}, x_1^{\neq}, i) \xrightarrow{\varepsilon} (T_{(i,1)}, 1) \quad \text{for } 2 \leq i \leq |\Gamma|, \tag{3.9}$$

$$(T_{(i,j-1)}, x_i^{=} \wedge x_j^{\neq}) \xrightarrow{\varepsilon} (T_{(i,j)}, 1) \quad \text{for } 2 \leq j < i \leq |\Gamma|. \tag{3.10}$$

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

$$(T_{(|\Gamma|,|\Gamma|-1)}, \mathrm{tt}) \xrightarrow{\varepsilon} (W_0, 1), \tag{3.11}$$

$$(W_{i-1}, x_{u_i}^{=}, |\Gamma| + i) \xrightarrow{\varepsilon} (W_i, 1) \quad \text{for } i \in [|u|]. \tag{3.12}$$

- Let $s(m) = -1$ if $m = L$ and $s(m) = 1$ if $m = R$. For encoding the state transition and accepting condition of M by $\mathcal{A}$, we introduce a state $A_q^{(i,j)}$ for $q \in Q_M$, $i \in [|\Gamma|]$, and $j \in [p(n)]$. $A_q^{(i,j)}$ represents a part of an ID (q, α, j) of M where $i = \alpha_j$, i.e. the tape symbol at the head position. The remaining information about α of (q, α, j) will be represented by an assignment of $\mathcal{A}$. More precisely, the content of $(|\Gamma| + j)$th register (i.e. $\theta(|\Gamma| + j)$) equals the data value $\theta(\alpha_j)$ representing the tape symbol α_j for $j \in [|\alpha|]$ and $\theta(|\Gamma| + j) = \perp$ for $|\alpha| < j \leq p(|u|)$. Let θ_α denote such an assignment that represents the tape contents α. We illustrate the correspondence between an ID of M and a state and an assignment of $\mathcal{A}$ in Fig. 3.1.

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

$$(W_{|u|}, \mathrm{tt}) \xrightarrow{\varepsilon} (A_{q_0}^{(u_1,1)}, 1). \tag{3.13}$$

$A_q^{(\alpha_j,j)}$	Γ	α_1	α_2	$\ldots$	$\alpha_{\lvert\alpha\rvert}$	$\perp$	$\ldots$	$\perp$

Figure 3.1: $\mathcal{A}$'s state and registers that correspond to M's ID (q, α, j).

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

$$(A_q^{(i,j)}, x_{b'}^{=}, |\Gamma| + j) \xrightarrow{\varepsilon} (B_{q'}^{(a,\max(1,j+s(m')))}, 1). \tag{3.14}$$

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

$$(B_{q'}^{(a,j')}, x_a^{=} \wedge x_{|\Gamma|+j'}^{=}) \xrightarrow{\varepsilon} (A_{q'}^{(a,j')}, 1). \tag{3.15}$$

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

$$(A_{q_f}^{(i,j)}, x_1^{=}) \xrightarrow{a} (E, \varepsilon). \tag{3.16}$$

We can show for each ID (q, α, j),

$$\begin{aligned}&(q,\alpha,j) \xrightarrow{*} (q_f, u', j') \text{ for some } q_f \in F, u' \in \Gamma^* \text{ and } j' \in \mathbb{N}\\ &\text{iff } (A_q^{(\alpha_j,j)}, \theta_\alpha, (a,\perp), \perp) \overset{*}{\Rightarrow}_{\mathcal{A}} (E, \theta, \varepsilon, \varepsilon) \text{ for some } \theta \in \Theta_k\end{aligned} \tag{3.17}$$

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

We can easily prove that $(T_{(1,0)}, \theta_\perp, (a,\perp), \perp) \overset{*}{\Rightarrow}_{\mathcal{A}}$ $(A_{q_0}^{(u_1,1)}, \theta_u, (a,\perp), \perp)$, and moreover, if $(T_{(1,0)}, \theta_\perp, (a,\perp), \perp) \overset{*}{\Rightarrow}_{\mathcal{A}} (E, \theta, \varepsilon, \varepsilon)$ for some $\theta \in \Theta_k$, then this run must be $(T_{(1,0)}, \theta_\perp, (a,\perp), \perp) \overset{*}{\Rightarrow}_{\mathcal{A}} (A_{q_0}^{(u_1,1)}, \theta_u, (a,\perp), \perp) \overset{*}{\Rightarrow}_{\mathcal{A}} (E, \theta, \varepsilon, \varepsilon)$.
By letting $(q, \alpha, j) = (q_0, u, 1)$ in property (3.17) and by the above-mentioned fact, we obtain $u \in L(M) \Leftrightarrow ((T_{(1,0)}, \theta_\perp, (a,\perp), \perp) \overset{*}{\Rightarrow}_{\mathcal{A}} (E, \theta, \varepsilon, \varepsilon)$ for some $\theta \in \Theta_k) \Leftrightarrow (a,\perp) \in L(\mathcal{A})$.

Theorem 3.2.4. *The membership problem for growing RPDA is NP-complete.*

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

We prove NP-hardness by a polynomial-time reduction from the satisfiability problem for 3-Conjunctive Normal Form (3CNF).

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

$$\begin{aligned}
\delta = {} & \{(q_0, \mathrm{tt}, 1) \xrightarrow{a} (P_0, 1)\} \\
& \cup \{(P_{i-1}, x_1^{=}, 2i+j) \xrightarrow{a} (P_i, 1) \mid i \in [n],\ j \in \{0,1\}\} \\
& \cup \{(P_n, x_1^{=}) \xrightarrow{a} (C_0, 1)\} \\
& \cup \{(C_{i-1}, x_r^{=}) \xrightarrow{a} (C_i, 1) \mid i \in [m],\ r \in \{r_{a_i}, r_{b_i}, r_{c_i}\}\} \\
& \cup \{(C_m, x_1^{=}) \xrightarrow{a} (E, \varepsilon)\}.
\end{aligned}$$

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

For both of the RPDA constructed in the proofs of Theorem 3.2.3 and 3.2.4, the height of the stack appearing in any accepting run is at most one. This fact

implies the following property.

Corollary 3.2.1. *The membership problems for RA with and without ε-transition are PSPACE-complete and NP-complete, respectively.*

Note that NP-completeness of the membership for RA without ε-transition was first proved in [59].

Though the complexities of membership problems for non-decreasing and growing RPDA are different with that of general RPDA, the complexities of emptiness problems are the same as shown in the following theorem.

Theorem 3.2.5. *The emptiness problems for non-decreasing RPDA and growing RPDA are both EXPTIME-complete.*

Proof By theorem 3.2.2, it suffices to show that the problem is EXPTIME-hard for growing RPDA. We give a poly-time reduction from the emptiness problem for general RPDA. From an arbitrary RPDA $\mathcal{A} = (Q, q_0, \delta)$, we construct the growing RPDA $\mathcal{A}_g = (Q, q_0, \delta_g)$, where

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

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

Chapter 4 Generalization of RCFG

In this chapter, we define generalizad register context-free grammar (GRCFG) that allows the test of an arbitrary binary relation on the set of data values. However, GRCFG does not have some important properties such as decidabilities of membership and emptiness, which RCFG has. We first define *register type* in Section 4.1 which is used to define two properties of GRCFG, simulation and progress, in Section 4.2. We will prove that simulation and progress properties guarantee the decidability of membership and emptiness problems of GRCFG in Section 4.3.

4.1 Register Type, Normal Forms and ε-rule Removal

4.1.1 Register type

In this subsection, we will define register type, which is useful in expressing equalities among the contents of registers, transforming a given RCFG into a certain normal form and proving some important properties of RCFG. The idea is simple; instead of remembering concrete data values in registers, it suffices to remember the induced equivalence classes of the indices of registers as long as the equalities among data values in the registers are concerned.

Definition 4.1.1. *A decomposition of* $[k]$ *into disjoint non-empty subsets is called a* register type *of* k*-RCFG. Let* Γ_k *denote the collection of all register types of* k*-RCFG. For a register type* $\gamma \in \Gamma_k$*, let* $\gamma[i]$ *(*$i \in [k]$*) denote the subset containing* i*.* □

For example, $\gamma_1 = \{\{1,2\},\{3,5\},\{4\}\}$ is a register type of 5-RCFG and $\gamma_1[1] = \{1,2\}$, $\gamma_1[5] = \{3,5\}$. For an assignment $\theta \in \Theta_k$ and a register type $\gamma \in \Gamma_k$, we define the typing relation as:

$$\theta \models \gamma :\Longleftrightarrow \forall i,j.(\theta(i) = \theta(j) \Longleftrightarrow \gamma[i] = \gamma[j]).$$

For example, $\theta_1 \in \Theta_5$ such that $\theta_1(1) = \theta_1(2) = 8$, $\theta_1(3) = \theta_1(5) = 10$, $\theta_1(4) = 5$ satisfies $\theta_1 \models \gamma_1$. By definition, for each $\theta \in \Theta_k$, there is exactly one $\gamma \in \Gamma_k$ such that $\theta \models \gamma$. In this case, we say that the type of θ is γ.

4.1.2 Normal forms for guard expressions

By using register types, we show that a given RCFG can be transformed into an equivalent RCFG G' such that for any rule $r = (A,\psi,i)/(A,\psi) \to \alpha$, there must exist an input data value d for any (A,θ) that enables r to be applied to (A,θ), that is, the guard ψ never blocks any (A,θ) and only specifies the equality or inequality among an input data value d and the current contents of the registers. This transformation is the key of the ε-rule removal shown in the next subsection.

First, it is easy to transform a given k-RCFG into an equivalent k-RCFG where the guard expression ψ of every rule has the following form:

$$\psi = (x_{i_1}^{=} \wedge \ldots \wedge x_{i_m}^{=}) \wedge (x_{j_1}^{\neq} \wedge \ldots \wedge x_{j_n}^{\neq}). \tag{4.1}$$

The above guard can be obtained by the following equivalence transformations:

1. Transform the guard expression of every rule to an equivalent disjunctive normal form.

2. Replace a rule $(A,\psi_1 \vee \psi_2,i) \to \alpha$ into $(A,\psi_1,i) \to \alpha$ and $(A,\psi_2,i) \to \alpha$.

For a guard expression ψ in (4.1), we let $\psi^{=} = \{i_1,\ldots,i_m\}$ and $\psi^{\neq} = \{j_1,\ldots,j_n\}$. For $\gamma \in \Gamma_k$ and $\psi \in F_k$ in the form of (4.1), we define

$$\gamma \models \psi :\Longleftrightarrow \bigwedge_{i \in \psi^{=}} (\bigwedge_{j \in \psi^{=}} \gamma[i] = \gamma[j] \wedge \bigwedge_{j \in \psi^{\neq}} \gamma[i] \neq \gamma[j]).$$

Note that $\psi^{=} = \emptyset$ implies $\gamma \models \psi$ for any γ. It is easy to see that the following property holds, which means that for an assignment θ that conforms to γ, there is a data value d that satisfies ψ if and only if $\gamma \models \psi$.

$$\theta \models \gamma \;\Rightarrow\; (\gamma \models \psi \Longleftrightarrow \exists d.\ \theta, d \models \psi). \tag{4.2}$$

Lemma 4.1.1. *For an arbitrary k-RCFG G, we can construct a k-RCFG G' such that $L(G') = L(G)$ and the guard expression of every rule in G' is one of the following $k+1$ expressions: $x_1^{=}, x_2^{=}, \ldots, x_k^{=}$, $x_1^{\neq} \wedge \cdots \wedge x_k^{\neq}$.*

Proof Let $\varphi_j = x_j^{=}$ for $j \in [k]$ and $\varphi_* = x_1^{\neq} \wedge \ldots \wedge x_k^{\neq}$. Obviously, the following property holds:

$$\text{For any } \theta,\ d, \text{ and } \psi,\ \ \theta, d \models \psi \text{ iff } \theta, d \models \psi \wedge \varphi_j \text{ for some } j \in [k] \cup \{*\}. \tag{4.3}$$

For an assignment θ whose type is γ, consider an updated assignment $\theta[i \leftarrow d]$ obtained by a single derivation step. Then one of the following possibilities must hold:

(i) $d = \theta(j)$ for some $j \in [k]$ (i.e. $\theta, d \models \varphi_j$).

(ii) $d \neq \theta(j)$ for any $j \in [k]$ (i.e. $\theta, d \models \varphi_*$).

In case (i), the type of $\theta[i \leftarrow d]$ is uniquely determined by γ, i, and j and regardless of θ. Similarly, the type of $\theta[i \leftarrow d]$ in case (ii) is uniquely determined by γ and i. We denote the type of $\theta[i \leftarrow d]$ in cases (i) and (ii) by $\mathit{after}(\gamma, i, j)$ and $\mathit{after}(\gamma, i, *)$, respectively. These register types can be specifically described as follows:

- For $j \in [k]$, $\mathit{after}(\gamma, i, j) = \gamma'$ where $\gamma'[i] = \gamma'[j] = \gamma[j] \cup \{i\}$ and $\gamma'[j'] = \gamma[j'] \setminus \{i\}$ for $\forall j' \in [k] \setminus \gamma'[i]$.
- $\mathit{after}(\gamma, i, *) = \gamma'$ where $\gamma'[i] = \{i\}$ and $\gamma'[j'] = \gamma[j'] \setminus \{i\}$ for $\forall j' \in [k] \setminus \gamma'[i]$.

We construct k-RCFG $G' = (V', S', R')$ from $G = (V, S, R)$ where $V' = V \times \Gamma_k$, $S' = (S, \{[k]\})$, and R' is the smallest set that satisfies the following inference rules, where $\alpha^{\mathrm{aug}(\gamma')}$ is the sequence obtained from α by replacing every occurrence of every nonterminal A in V with (A, γ'); that is, $(X_1 \ldots X_n)^{\mathrm{aug}(\gamma')} = X_1' \ldots X_n'$

where $X'_\ell = (X_\ell, \gamma')$ if $X_\ell \in V$ and $X'_\ell = X_\ell$ otherwise for every $\ell \in [n]$. In the following rules, φ_j is the expression defined in the beginning of the proof of this lemma. We assume that the guard expression of every rule of G has the form of (4.1).

$$\frac{\begin{array}{l}(A, \psi, i) \to \alpha \in R \\ j \in [k] \cup \{*\} \quad \gamma \models \psi \wedge \varphi_j \quad \gamma' = \mathit{after}(\gamma, i, j)\end{array}}{((A, \gamma), \varphi_j, i) \to \alpha^{\mathrm{aug}(\gamma')} \in R'}$$

$$\frac{(A, \psi) \to \alpha \in R \quad \gamma \models \psi}{((A, \gamma), \varphi_1) \to \alpha^{\mathrm{aug}(\gamma)} \in R'}$$

Note that φ_1 in the second inference rule is used as an equivalent of $\mathtt{tt}$, because for any assignment θ, there must be a data value d $(= \theta(1))$ that satisfies φ_1, and d is not stored in a register when that rule is applied.

We can show the following properties, which establish the lemma.

- For every $((A, \gamma), \theta)$ appearing in a derivation in G' starting from $(S', \theta_\perp)$, it holds that $\theta \models \gamma$. This can be proved by induction on the length of the derivation.
- For a derivation of a data word w in G, if we replace every occurrence of $(A, \theta) \in V \times \Theta_k$ with $((A, \gamma), \theta)$ where γ is the type of θ, then we obtain a derivation of w in G'. This can be proved as follows: For each derivation step of G where a rule $(A, \psi, i) \to \alpha$ is applied to (A, θ) with an input data value d (i.e. $\theta, d \models \psi$), there must exist some φ_j $(j \in [k] \cup \{*\})$ such that $\theta, d \models \psi \wedge \varphi_j$ by property (4.3). By property (4.2), $\gamma \models \psi \wedge \varphi_j$ holds, and thus G' has the rule $((A, \gamma), \varphi_j, i) \to \alpha^{\mathrm{aug}(\gamma')}$. This rule allows the simulating derivation step in G' applied to $((A, \gamma), \theta)$ using the same input data value d, which yields the same updated assignment $\theta[i \leftarrow d]$ as the derivation step in G.
- For a derivation of a data word w in G', if we replace each $((A, \gamma), \theta)$ with (A, θ), then we obtain a derivation of w in G. This can be proved as follows: By the property before the previous one, $\theta \models \gamma$ holds. Consider a derivation step of G' where a rule $((A, \gamma), \varphi_j, i) \to \alpha^{\mathrm{aug}(\gamma')}$ is applied to

$((A, \gamma), \theta)$ with an input data value d (i.e. $\theta, d \models \varphi_j$). Then G must have a rule $r = (A, \psi, i) \to \alpha$ and $\gamma \models \psi \wedge \varphi_j$. By property (4.2), there is some d' such that $\theta, d' \models \psi \wedge \varphi_j$. If $j \in [k]$, then d' and d are the same $(= \theta(j))$. If $j = *$, then d must satisfy $\theta, d \models \psi \wedge \varphi_*$ because $\psi \wedge \varphi_*$ must be equivalent to φ_* (if not, $\theta, d' \not\models \psi \wedge \varphi_*$ for any d'). Since $\theta, d \models \psi \wedge \varphi_j$ implies $\theta, d \models \psi$, we can apply r to (A, θ) with the same input data value d.

4.1.3 ε-rule removal

Theorem 4.1.1. *For an arbitrary k-RCFG G, we can construct an ε-rule free k-RCFG G' that satisfies $L(G') = L(G) \setminus \{\varepsilon\}$.*

Proof By Lemma 4.1.1, we can transform any k-RCFG $G = (V, R, S)$ into another k-RCFG $G'' = (V'', R'', S'')$ such that $L(G'') = L(G)$ and the guard expression of every rule in G'' is either $x_1^{=}, \ldots, x_k^{=}$, or $x_1^{\neq} \wedge \ldots \wedge x_k^{\neq}$. Because the guard expressions of G'' never block the application of each rule, we can compute the set Nu of nullable nonterminals (i.e. the set that consists of every nonterminal A such that $(A, \theta) \Rightarrow_{G''}^{*} \varepsilon$ regardless of θ) in the same way as CFG; that is, we can compute Nu as the smallest set that satisfies the following conditions:

- If $(A, \psi, i)/(A, \psi) \to \varepsilon \in R''$, then $A \in Nu$.
- If $(A, \psi, i)/(A, \psi) \to \alpha \in R''$ and α consists of nonterminals in Nu, then $A \in Nu$.

And thus we can remove the ε-rules of G'' also in the same way as CFG; that is, for each $(A, \psi, i)/(A, \psi) \to \alpha \in R''$ such that $\alpha \neq \varepsilon$, we construct $(A, \psi, i)/(A, \psi) \to \alpha'$ where $\alpha' \neq \varepsilon$ and α' is obtained from α by removing zero or more occurrence of nonterminals in Nu. Let $G' = (V'', R', S'')$ be the resultant RCFG. We can show that for every $A \in V''$, $\theta \in \Theta_k$, and a data word w, $(A, \theta) \Rightarrow_{G''}^{*} w$ and $w \neq \varepsilon$ if and only if $(A, \theta) \Rightarrow_{G'}^{*} w$, by induction on the length of the derivation.

4.2 Generalized RCFG

4.2.1 Definitions

We define generalized register context-free grammar by allowing an arbitrary binary relation on the set of data values. Let Σ be a finite alphabet, D be a set of data values such that $\Sigma \cap D = \emptyset$ equipped with a finite set of binary relations $\mathcal{R}$. We call $(D, \mathcal{R})$ a *data structure*. For $k \in \mathbb{N}_0$, a *generalized k-register context-free grammar* (k-GRCFG) is a triple $G = (V, R, S)$ where V, R and S are the same as in k-RCFG except that an atomic formula in a guard expression is $x_i^{\bowtie}$ and $x_i^{\bowtie^{-1}}$ ($i \in [k]$, $\bowtie \in \mathcal{R}$) and its semantics is defined by

$$\theta, d \models x_i^{\bowtie} \text{ iff } \theta(i) \bowtie d \text{ and } \theta, d \models x_i^{\bowtie^{-1}} \text{ iff } d \bowtie \theta(i)$$

for any $\theta \in \Theta_k$ and $d \in D$. We sometimes write k-GRCFG($\mathcal{R}$) to emphasize $\mathcal{R}$ and abbreviate it as k-GRCFG($\bowtie$) when $\mathcal{R} = \{\bowtie\}$. Notions and notations for RCFG such as ε-rule, derivation relation $\Rightarrow$, the data language $L(G)$ generated by G are defined in the same way. We also write k-GRCFG($=$) to denote a (usual) k-RCFG.

Closure and non-closure properties of GRCFG described in the following Theorem 4.1 can be proved in a similar way to the case of the usual RCFG [20, 41]. In that theorem, a homomorphism is a function $h : (\Sigma \times D)^* \to (\Sigma' \times D')^*$ where Σ, Σ' are finite alphabets and D, D' are infinite sets of data values such that $h(w_1 \cdots w_n) = h(w_1) \cdots h(w_n)$ holds for $w_1, \ldots, w_n \in (\Sigma \times D)^*$. Theorem 4.2.2 is an extension of 1.2.1 to GRCFG and can be proved in the same way.

Theorem 4.2.1. *The class of data languages generated by k-GRCFG($\mathcal{R}$) is closed under union, concatenation and Kleene-closure. It is not closed under intersection, complement, homomorphisms or inverse homomorphisms.*

Theorem 4.2.2. *For an arbitrary k-GRCFG $G = (V, R, S)$, we can construct an equivalent $(k+1)$-GRCFG $G' = (V', R', S')$ such that R' has no rule of the form $(A, \psi) \to \alpha$.*

4.2.2 Simulation and progress properties

In Section 4.1, we showed that a given RCFG can be transformed to an equivalent RCFG where the guard expression of a production rule never blocks its application by associating a register type with each nonterminal symbol. We can extend register type to GRCFG in a natural way, but the above transformation cannot guarantee the equivalence because the register type no longer has information enough to represent the applicability of a rule in GRCFG.

Example 4.2.1. *Consider the set of integers with the usual strict total order* $\mathbb{Z} = (Z, \{<_Z, >_Z\})$ *as a data structure. We might extend register type of GRCFG(=) by introducing* $<_Z$ *among the equivalence classes of* $[k]$*. For example, let* $\psi = x_1^{<_Z} \wedge x_2^{>_Z}$ *be a guard expression of 3-GRCFG*$(\{<_Z, >_Z\})$ *and consider assignments* $\theta_1, \theta_2 \in \Theta_3$ *such that* $\theta_1(1) = \theta_1(3) = 4$*,* $\theta_1(2) = 7$ *and* $\theta_2(1) = \theta_2(3) = 5$*,* $\theta_2(2) = 6$*. Also let* γ *be the register type (informally) defined as* $\gamma = \{\{1, 3\} <_Z \{2\}\}$*. Both* $\theta_1 \models \gamma$ *and* $\theta_2 \models \gamma$ *hold. However, there is no* $d \in Z$ *such that* $\theta_2, d \models \psi$ *while* $\theta_1, 5 \models \psi$*.* □

Similarly, the membership and emptiness lose decidability for GRCFG even if a binary relation appearing in a guard expression is a decidable relation.

Theorem 4.2.3. *There exists a data structure* $(D, \mathcal{R})$ *such that the membership and emptiness problems for 1-GRCFG*$(\mathcal{R})$ *are undecidable.*

Proof We show a reduction from the Post's correspondence problem (PCP). Let $\{(u_1, v_1), \ldots, (u_n, v_n)\} \subseteq \Sigma^+ \times \Sigma^+$ be an instance of PCP over an alphabet Σ. Then let $D = \Sigma^* \times \Sigma^*$ and $\mathcal{R} = \{\bowtie_i \mid i \in [n]\} \cup \{EQ\}$ where

$$(x_1, x_2) \bowtie_i (y_1, y_2) \iff x_1 u_i = y_1 \wedge x_2 v_i = y_2,$$
$$(x_1, x_2)\ EQ\ (y_1, y_2) \iff x_1 = x_2.$$

We assume the initial register data value $\bot = (\varepsilon, \varepsilon) \in D$. We construct 1-GRCFG$(\mathcal{R})$ $G = (V, R, S)$ where R consists of

$$(S, x_1^{\bowtie_i}, 1) \to A \text{ and } (A, x_1^{\bowtie_i}, 1) \to A \text{ for } \forall i \in [n] \text{ and } (A, x_1^{EQ}) \to \varepsilon.$$

We can easily show that $\varepsilon \in L(G)$ (iff $L(G) \neq \emptyset$) iff the instance $\{(u_1, v_1), \ldots, (u_n, v_n)\}$ of PCP has a solution. □

In the rest of this paper, we assume $\mathcal{R}$ is a singleton $\mathcal{R} = \{\bowtie\}$ for simplicity. The properties we show below can be extended in a general case that $\mathcal{R}$ has more than one binary relation. We first extend a register type as a binary relation $\gamma : ([k] \times [k]) \backslash \{(i, i) \mid i \in [k]\} \to \{\mathtt{tt}, \mathtt{ff}\}$ (We exclude the diagonal elements $\{(i, i) \mid i \in [k]\}$ from the domain of a register type because the applicability of a rule does not depend on whether $\theta(i) \bowtie \theta(i)$.) We say that the type of an assignment θ is γ (and write $\theta \models \gamma$) iff for all $i, j \in [k]$ $(i \neq j)$,

$$\gamma(i, j) = \mathtt{tt} \ \text{ iff } \ \theta(i) \bowtie \theta(j).$$

We write $\theta \sim_{\bowtie} \theta'$ if the types of assignments θ and θ' are the same. The collection of all register types of k-GRCFG is denoted by Γ_k as before.

In Example 4.2.1, we see that a register type cannot always decide the applicability of a rule because there may exist $\gamma \in \Gamma_k, \psi \in F_k$ such that $\exists d.\ \theta, d \models \psi$ and $\forall d'.\ \theta', d' \not\models \psi$ for some assignments $\theta, \theta' \models \gamma$. We would like to avoid the above case and also to uniquely determine the register-type $\gamma' \in \Gamma_k$ after a rule application. For this purpose, we introduce two predicates (Definition 4.2.1) to describe the desirable property (Definition 4.2.2).

Definition 4.2.1. *Let $pr, npr : \Gamma_k \times F_k \times [k] \times \Gamma_k \to \{\mathtt{tt}, \mathtt{ff}\}$ be the predicates defined as follows: For $\gamma, \gamma' \in \Gamma_k$, $i \in [k]$, and $\psi \in F_k$,*

$$pr(\gamma, \psi, i, \gamma') = \mathtt{tt} \iff \gamma' \in \mathit{after}(\theta, \psi, i) \text{ for } \forall \theta \models \gamma,$$
$$npr(\gamma, \psi, i, \gamma') = \mathtt{tt} \iff \gamma' \notin \mathit{after}(\theta, \psi, i) \text{ for } \forall \theta \models \gamma,$$

where $\mathit{after}(\theta, \psi, i) = \{\gamma' \mid \exists d.\, (\theta, d \models \psi \wedge \theta[i \leftarrow d] \models \gamma')\}$.

Definition 4.2.2 (simulation)**.** *We say that a data structure $(D, \mathcal{R})$ has the simulation property for k registers if the following condition is met:*

> *For all $\gamma, \gamma' \in \Gamma_k$, $i \in [k]$, and $\psi \in F_k$, either $pr(\gamma, \psi, i, \gamma') = \mathtt{tt}$ or $npr(\gamma, \psi, i, \gamma') = \mathtt{tt}$ holds.*

Again, both *pr* and *npr* are undecidable in general because a binary relation appearing in ψ may be undecidable. (It is unknown whether *pr* and *npr* are

decidable when we restrict a binary relation in ψ to be decidable.) Hence, we will introduce the assumption that both *pr* and *npr* are decidable (in EXPTIME).

Definition 4.2.3 (progress)**.** *We say that a data structure $(D, \mathcal{R})$ has the progress property for k registers if the following condition is met:*

> *For all $\gamma, \gamma' \in \Gamma_k$, $i \in [k]$, and $\psi \in F_k$, predicate pr and npr are decidable in EXPTIME.*

For a data structure $\mathbb{D} = (D, \mathcal{R})$, if D is understood, we say that $\mathcal{R}$ (or even $\bowtie$, if $\mathcal{R} = \{\bowtie\}$) has the simulation or progress property.

Finally, we define *data type* as an extension of register type by adding the information on equality between data values in the registers and data values appearing in a given data word w.

Definition 4.2.4 (data type)**.** *Let w be a data word and D_w be the set of data values appearing in w; i.e. $D_w = \{d_i \mid i \in [n],\ w = (a_1, d_1) \dots (a_n, d_n)\}$. Also let $d_{\neq} \notin D$ be a newly introduced symbol. We use a function $e : [k] \to D_w \cup \{d_{\neq}\}$ as a finite representation of an assignment. This function is the same as a given assignment except that every data value that does not appear in w is replaced with $d_{\neq}$. We write $\theta \models e$ iff for all $i \in [k]$,*

$$e(i) = \theta(i) \text{ if } \theta(i) \in D_w \quad \text{and} \quad e(i) = d_{\neq} \text{ otherwise.}$$

The collection of all such functions $e : [k] \to D_w \cup \{d_{\neq}\}$ is denoted by $E_{w,k}$. The data type *of an assignment $\theta \in \Theta_k$ for a data word w is a pair $(\gamma, e) \in \Gamma_k \times E_{w,k}$. We write $\theta \models (\gamma, e)$ iff $\theta \models \gamma$ and $\theta \models e$. We define the simulation and progress properties with data type in the same way as in the case of register types.*

4.3 Properties of GRCFG

4.3.1 ε-rule removal

Theorem 4.3.1. *For an arbitrary GRCFG($\bowtie$) G such that $\bowtie$ has the simulation and progress properties, we can construct an ε-rule free GRCFG($\bowtie$) G'' that satisfies $L(G'') = L(G) \setminus \{\varepsilon\}$.*

Proof The theorem can be proved in a similar way to Theorem 4.1.1 by using the simulation and progress properties. Let $G = (V, R, S)$ be a k-GRCFG($\bowtie$). Without loss of generality, we assume that R has no rule of the form $(A, \psi) \to \alpha$ (by Theorem 4.2.2) and the guard expression of every rule in R is a conjunction of literals (atomic formulas or their negations). We first construct k-GRCFG($\bowtie$) $G' = (V', R', S')$ from G where

- $V' = V \times \Gamma_k$,
- R' is the smallest set of rules satisfying the following conditions. Define the subset of guard expressions F'_k ($\subseteq F_k$) as

$$F'_k = \{ \bigwedge_{i\in[k]} \zeta_i \wedge \bigwedge_{i\in[k]} \eta_i \mid \zeta_i \in \{x_i^{\bowtie}, \neg x_i^{\bowtie}\}, \eta_i \in \{x_i^{\bowtie^{-1}}, \neg x_i^{\bowtie^{-1}}\}\}.$$

  Let $r = (A, \psi, i) \to \alpha \in R$. Also let $\gamma, \gamma' \in \Gamma_k$ and $\varphi \in F'_k$. If $pr(\gamma, \psi \wedge \varphi, i, \gamma') = \mathtt{tt}$, which is decidable by the progress property,

$$((A, \gamma), \psi \wedge \varphi, i) \to \alpha^{\mathrm{aug}(\gamma')} \in R'.$$

  (See the proof of Lemma 4.1.1 for the definition of $\alpha^{\mathrm{aug}(\gamma')}$.) Note that γ' is uniquely determined by γ, $\psi \wedge \varphi$ and i because γ specifies whether $\theta(i) \bowtie \theta(j)$ holds or not for each pair $i, j \in [k]$ $(i \neq j)$ and also φ specifies whether $\theta(i) \bowtie d$ and $d \bowtie \theta(i)$ hold or not for each $i \in [k]$ and an input data value d.
- $S' = (S, \gamma_0)$ where $\theta_\perp \models \gamma_0$.

See the example of the construction in Example 4.3.1. We will show $L(G) = L(G')$ by induction on the length of derivations in G and G', using the simulation property (to show $L(G) \subseteq L(G')$).

First we show that $L(G') \subseteq L(G)$. Assume $w \in L(G')$. Then, there exists a derivation $((S, \gamma_0), \theta_\perp) \stackrel{*}{\Rightarrow}_{G'} w$. We have to show $(S, \theta_\perp) \stackrel{*}{\Rightarrow}_G w$. For this purpose, we will show by induction on the length of the derivation in G' a little more general property that if $((S, \gamma_0), \theta_\perp) \stackrel{*}{\Rightarrow}_{G'} Y'_1 \ldots Y'_m$ $(Y'_j \in ((V \times \Gamma_k) \times \Theta_k) \cup (\Sigma \times D)$ for each $j \in [m])$, then $(S, \theta_\perp) \stackrel{*}{\Rightarrow}_G Y_1 \ldots Y_m$ where $Y_j = (B_j, \theta_j)$ and $\theta_j \models \gamma_j$ if $Y'_j = ((B_j, \gamma_j), \theta_j)$ and $Y_j = Y'_j$ otherwise. We call the former derivation t'.
(Basis) If the length of t' is zero, the claim holds because $\theta_\perp \models \gamma_0$.

(Induction) Assume that the length of t' is greater than zero and the last step in t' is $Y'_1 \dots Y'_{p-1}((A,\gamma),\theta)Y'_{q+1} \dots Y'_m \Rightarrow_{G'} Y'_1 \dots Y'_m$ with $((A,\gamma),\theta) \Rightarrow^d_{G',r'} Y'_p \dots Y'_q$ for some $1 \leq p,q \leq m$. Since $r' \in R'$, by the construction of R', there are $r = (A,\psi,i) \to \alpha \in R$, $\varphi \in F'_k$ and $\gamma' \in \Gamma_k$ such that r' is constructed from r, γ, γ' and φ. We assume that the form of r is $(A,\psi,i) \to \alpha$. By the induction hypothesis, $(S,\theta_\perp) \overset{*}{\Rightarrow}_G Y_1 \dots Y_{p-1}(A,\theta)Y_{q+1} \dots Y_m$ where $Y_j = (B_j,\theta_j)$ and $\theta_j \models \gamma_j$ if $Y'_j = ((B_j,\gamma_j),\theta_j)$ and $Y_j = Y'_j$ otherwise for $1 \leq j < p$ or $q < j \leq n$, and also $\theta \models \gamma$. Since the left-hand side of r' is $((A,\gamma),\psi \wedge \varphi,i)$ and r' is applied to $((A,\gamma),\theta)$ with the input data d, we have $\theta,d \models \psi \wedge \varphi$. Hence $\theta,d \models \psi$ holds obviously and thus r is applicable to (A,θ) also. Let $(A,\theta) \Rightarrow^d_{G,r} \alpha'$, then we obtain a desired derivation $(S,\theta_\perp) \overset{*}{\Rightarrow}_G Y_1 \dots Y_{p-1}(A,\theta)Y_{q+1} \dots Y_m \Rightarrow^d_{G,r} Y_1 \dots Y_{p-1}\alpha' Y_{q+1} \dots Y_m$.

Next we show that $L(G) \subseteq L(G')$. Assume $w \in L(G)$. Then, there exists a derivation $S \overset{*}{\Rightarrow}_G w$. We have to show $((S,\gamma_0),\theta_\perp) \overset{*}{\Rightarrow}_{G'} w$. We will show by induction on the length of the derivation in G that if $(S,\theta_\perp) \overset{*}{\Rightarrow}_G Y_1 \dots Y_m$ ($Y_j \in (V \times \Theta_k) \cup (\Sigma \times D)$ for each $j \in [m]$) then $((S,\gamma_0),\theta_\perp) \overset{*}{\Rightarrow}_{G'} Y'_1 \dots Y'_m$ where $Y'_j = ((B_j,\gamma_j),\theta_j)$ for γ_j such that $\theta_j \models \gamma_j$ if $Y_j = (B_j,\theta_j)$ and $Y'_j = Y_j$ otherwise. We call the former derivation t.

(Basis) If the length of t is zero, the claim holds because $\theta_\perp \models \gamma_0$.

(Induction) Assume that the length of t is greater than zero and the last step in t is $Y_1 \dots Y_{p-1}(A,\theta)Y_{q+1} \dots Y_m \Rightarrow_G Y_1 \dots Y_m$ with $(A,\theta) \Rightarrow^d_{G,r} Y_p \dots Y_q$ for some $1 \leq p,q \leq m$. By the induction hypothesis, $((S,\gamma_0),\theta_\perp) \overset{*}{\Rightarrow}_{G'} Y'_1 \dots Y'_{p-1}((A,\gamma),\theta)$ $Y'_{q+1} \dots Y'_m$ where $Y'_j = ((B_j,\gamma_j),\theta_j)$ for γ_j such that $\theta_j \models \gamma_j$ if $Y_j = (B_j,\theta_j)$ and and $Y'_j = Y_j$ otherwise for $1 \leq j < p$ or $q < j \leq n$, and also $\theta \models \gamma$. We assume that the form of r applied to (A,θ) is $(A,\psi,i) \to \alpha$. This means that $\theta,d \models \psi$, and because of the definition of F'_k, there must be a unique $\varphi \in F'_k$ such that $\theta,d \models \psi \wedge \varphi$. Hence, $\gamma' \in \mathit{after}(\theta,\psi \wedge \varphi,i)$ for the type γ' of $\theta[i \leftarrow d]$. By the simulation property, we have $pr(\gamma,\psi \wedge \varphi,i,\gamma') = \mathtt{tt}$. Therefore, the rule $r' = ((A,\gamma),\psi \wedge \varphi,i) \to \alpha^{\mathrm{aug}(\gamma')}$ has been constructed. The rest of the proof is similar to the converse direction.

We can construct an equivalent ε-rule free k-GRCFG($\bowtie$) G'' from G' in a similar way to Theorem 4.1.1 (because every application of a rule in G' is never blocked by the guard condition and we can compute the set Nu of nullable

nonterminals like Theorem 4.1.1).

Example 4.3.1. *Let $k = 2$ and consider a rule $r = (A, \psi, 1) \to \alpha$ where $\psi = x_1^{\bowtie} \wedge x_2^{\bowtie^{-1}}$. The possible register types are $\gamma_1 = (\delta_{12} \wedge \delta_{21})$, $\gamma_2 = (\delta_{12} \wedge \neg\delta_{21})$, $\gamma_3 = (\neg\delta_{12} \wedge \delta_{21})$ and $\gamma_4 = (\neg\delta_{12} \wedge \neg\delta_{21})$ where $\delta_{12} = (\theta(1) \bowtie \theta(2))$ and $\delta_{21} = (\theta(2) \bowtie \theta(1))$ (for readability, we denote a register type as a Boolean formula on an assignment θ. For example, $\gamma_2(1,2) = \mathtt{tt}$ and $\gamma_2(2,1) = \mathtt{ff}$ if we follow the notation defined in Sec. 4.2.2.) After the elimination of the unsatisfiable ones and Boolean simplification, we can assume that $F'_k = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$ where $\varphi_1 = x_1^{\bowtie^{-1}} \wedge x_2^{\bowtie}$, $\varphi_2 = x_1^{\bowtie^{-1}} \wedge \neg x_2^{\bowtie}$, $\varphi_3 = \neg x_1^{\bowtie^{-1}} \wedge x_2^{\bowtie}$ and $\varphi_4 = \neg x_1^{\bowtie^{-1}} \wedge \neg x_2^{\bowtie}$. If $pr(\gamma, \psi \wedge \varphi_i, 1, \gamma') = \mathtt{tt}$, the possible register type γ' is $\gamma_1, \gamma_2, \gamma_1, \gamma_2$ for $\varphi_1, \varphi_2, \varphi_3, \varphi_4$, respectively. In this example, the type γ' is determined depending only on φ_j and independent of γ because $k = 2$ and an input data value is loaded to the first register when r is applied.*

4.3.2 Emptiness and membership

Theorem 4.3.2. *The emptiness problem for GRCFG($\bowtie$) such that $\bowtie$ has the simulation and progress properties, is EXPTIME-complete.*

Proof Let $G = (V, R, S)$ be an arbitrary k-GRCFG($\bowtie$) that has no rule of the form $(A, \psi) \to \alpha$ and $G' = (V', R', S')$ be the k-GRCFG($\bowtie$) constructed from G in the proof of Theorem 4.3.1. As shown in that proof, $L(G') = L(G)$. We construct CFG $G'' = (V', R'', S')$ from G' where

$$R'' = \{(A, \gamma) \to X_1 \dots X_n \mid ((A, \gamma), \psi, i) \to X'_1 \dots X'_n \in R' \text{ for some } \psi \text{ and } i, \text{ and } X_j = X'_j \text{ if } X'_j \in V' \text{ and } X_j = a \text{ if } X'_j \notin V' \text{ for each } j \in [n]\}.$$

As we will show below, $L(G') = \emptyset \Leftrightarrow L(G'') = \emptyset$ because a rule application is never blocked in G'.

Assume $L(G'') \neq \emptyset$. Then, there exists a derivation $S' \overset{*}{\Rightarrow}_{G''} w$ for some $w \in a^*$. It suffices to show $(S', \theta_\perp) \overset{*}{\Rightarrow}_{G'} w'$ for some $w' \in (\Sigma \times D)^*$. For this purpose, we will show by induction on the length of the derivation in G'' that if $t = S' \overset{*}{\Rightarrow}_{G''} Y_1 \dots Y_m$ ($Y_j \in V'$ or $Y_j = a$ for each $j \in [m]$), then

$(S', \theta_\perp) \stackrel{*}{\Rightarrow}_{G'} Y'_1 \dots Y'_m$ where $Y'_j = (Y_j, \theta_j)$ for some $\theta_j \in \Theta_k$ such that $\theta_j \models \gamma_j$ if $Y_j = (B_j, \gamma_j) \in V'$ and $Y'_j \in (\Sigma \times D)$ if $Y_j = a$.
(Basis) If the length of t is zero, the claim trivially holds.
(Induction) Assume that the length of t is greater than zero and the last step in t is $Y_1 \dots Y_{p-1}(A, \gamma)Y_{q+1} \dots Y_m \Rightarrow_{G'',r} Y_1 \dots Y_m$ with $(A, \gamma) \Rightarrow_{G'',r} Y_p \dots Y_q$ for some $1 \leq p, q \leq m$. Since $r \in R''$, by the construction of R', there are $r' \in R'$ and $\gamma' \in \Gamma_k$ such that $r' = ((A, \gamma), \psi, i) \to \alpha^{\mathrm{aug}(\gamma')} \in R'$, $pr(\gamma, \psi, i, \gamma') = \mathtt{tt}$, and r is constructed from r'. We assume that the form of r' is $((A, \gamma), \psi, i) \to Y'_p \dots Y'_q$. By the induction hypothesis, $(S', \theta_\perp) \stackrel{*}{\Rightarrow}_{G'} Y'_1 \dots Y'_{p-1}((A, \gamma), \theta)Y'_{q+1} \dots Y'_m$ for some $\theta \models \gamma$ and $Y'_j = (Y_j, \theta_j)$ and $\theta_j \models \gamma_j$ if $Y_j = (B_j, \gamma_j) \in V'$ and $Y'_j \in (\Sigma \times D)$ if $Y_j = a$ for $1 \leq j < p$ or $q < j \leq n$. Since $pr(\gamma, \psi, i, \gamma') = \mathtt{tt}$, the rule r' can be applied at $((A, \gamma), \theta)$ in the above derivation in G'. Hence, $L(G') \neq \emptyset$.

The converse direction $L(G') \neq \emptyset \Rightarrow L(G'') \neq \emptyset$ can be proved in a similar way.

$\|G'\|$ is single exponential to $\|G\|$ because $|V'| = |V| \times |\Gamma_k|$ (by the definition of $\Gamma_k : ([k] \times [k]) \backslash \{(i, i) \mid i \in [k]\} \to \{\mathtt{tt}, \mathtt{ff}\}$, $|\Gamma_k| \leq 2^{k^2}$) and $|R'| \leq |R| \times |\Gamma_k|^2 \times |F'_k|$ (F'_k is defined in the proof of Theorem 5.1 and $|F'_k| = 2^{2k}$). In addition, $\|G''\|$ is obviously linear to $\|G'\|$. Therefore, $\|G''\|$ is single exponential to $\|G\|$. Because the emptiness problem for CFG is decidable in linear time, the emptiness problem for GRCFG is decidable in deterministic time exponential to k.

The lower bound can be obtained from EXPTIME-completeness of the emptiness problem for k-GRCFG(=) in Theorem 2.3.4.

Theorem 4.3.3. *The membership problem for GRCFG($\bowtie$) such that $\bowtie$ has the the simulation and progress properties with data type, is EXPTIME-complete.*

Proof sketch This theorem can be proved in a similar way to Theorem 4.3.2 as follows. From a given k-GRCFG($\bowtie$) $G = (V, R, S)$ and a data word w, we construct k-GRCFG($\bowtie$) $G' = (V', R', S')$ in a similar way to Theorem 4.3.1, by using the set of data types $\Gamma_k \times E_{w,k}$ instead of Γ_k (Therefore, $V' = V \times \Gamma_k \times E_{w,k}$). We construct CFG $G'' = (V', R'', S')$ (over a finite alphabet $\Sigma \times (D_w \cup \{d_\neq\})$) from G' where

$$R'' = \{(A, (\gamma, e)) \to X_1 \dots X_n \mid ((A, (\gamma, e)), \psi, i) \to X'_1 \dots X'_n \in R'$$

for some ψ and i, and $X_j = X'_j$ if $X'_j \in V'$ and $X_j = (a, e(h))$ if $X'_j = (a, h) \in (\Sigma \times [k])$ for each $j \in [n]\}$.

Similar to the proof of Theorem 4.3.2, each derivation $S' \overset{*}{\Rightarrow}_{G''} Y_1 \dots Y_m$ in G'' has a corresponding derivation $(S', \theta_\perp) \overset{*}{\Rightarrow}_{G'} Y'_1 \dots Y'_m$ in G'. Moreover, if Y_j is a terminal $(a, d) \in \Sigma \times (D_w \cup \{d_{\neq}\})$, then Y'_j equals (a, d) if $d \in D_w$ and $Y'_j = (a, d')$ for some $d' \notin D_w$ if $d = d_{\neq}$. Therefore, we can prove $w \in L(G') \Leftrightarrow w \in L(G'')$ in a similar way to Theorem 4.3.2. □

If we drop the progress property, the membership and emptiness are not always decidable as shown in the next theorem.

Theorem 4.3.4. *There exists a data structure $(D, \mathcal{R})$ that has the simulation property but does not have the progress property, and the membership and emptiness problems for 2-GRCFG($\mathcal{R}$) are undecidable.*

Proof We use a reduction from the inclusion problem for CFG. Let $D = \Sigma^*$, and consider two CFG A and B over Σ and $\mathcal{R} = \{\bowtie\}$ such that

$$d_1 \bowtie d_2 \iff d_2 \in L(A) \setminus L(B).$$

As defined in Example 4.3.1, there are four possible register types $\gamma_1, \gamma_2, \gamma_3, \gamma_4$ for $k = 2$ and a single binary relation $\bowtie$. For convenience, we give the definition of γ_3 and γ_4 again.

- $\gamma_3 = (\neg(\theta(1) \bowtie \theta(2)) \wedge \theta(2) \bowtie \theta(1))$
- $\gamma_4 = (\neg(\theta(1) \bowtie \theta(2)) \wedge \neg(\theta(2) \bowtie \theta(1)))$

For the binary relation $\bowtie$ defined above, $(\theta(1) \bowtie \theta(2))$ iff $(\theta(2) \in L(A) \setminus L(B))$ and $(\theta(2) \bowtie \theta(1))$ iff $(\theta(1) \in L(A) \setminus L(B))$.

We can prove that $\bowtie$ has the simulation property but does not have the progress property for two registers.

(The simulation property) For given $\psi \in F_2$, $\gamma' \in \Gamma_2$, $i \in [2]$, $\theta \in \Theta_2$, and $d \in D$, the condition $\theta, d \models \psi \wedge \theta[i \leftarrow d] \models \gamma'$ can be transformed into a Boolean

combination of the following atomic propositions:

$$\theta(1) \in L(A) \setminus L(B), \tag{4.4}$$

$$\theta(2) \in L(A) \setminus L(B), \tag{4.5}$$

$$d \in L(A) \setminus L(B). \tag{4.6}$$

(Note that the atomic propositions of F_2 are $x_1^{\bowtie}$, $x_2^{\bowtie}$, $x_1^{\bowtie^{-1}}$, $x_2^{\bowtie^{-1}}$, and for example, $(\theta, d \models x_1^{\bowtie})$ iff $(d \in L(A) \setminus L(B))$ and $(\theta, d \models x_1^{\bowtie^{-1}})$ iff $(\theta(1) \in L(A) \setminus L(B))$.)

Whether the propositions (4.4) and (4.5) hold is determined uniquely by the type γ of θ (because γ uniquely determines whether $\theta(2) \bowtie \theta(1)$ and $\theta(1) \bowtie \theta(2)$ hold). Therefore, whether $\exists d.\, \theta, d \models \psi \wedge \theta[i \leftarrow d] \models \gamma'$ holds or not is determined independent of a choice of θ of type γ, and thus the simulation property holds.

(Absence of the progress property) We can show that $pr(\gamma_4, x_1^{\bowtie}, 1, \gamma_3)$ for the above-mentioned γ_4 and γ_3 is undecidable by a reduction from the inclusion problem $L(A) \subseteq L(B)$ for CFG A and B. We first decide whether $L(B) = \emptyset$. If $L(B) = \emptyset$, then $L(A) \subseteq L(B)$ iff $L(A) = \emptyset$, which is also decidable. Assume that $L(B) \neq \emptyset$. We can prove $L(A) \not\subseteq L(B) \Leftrightarrow pr(\gamma_4, x_1^{\bowtie}, 1, \gamma_3) = \texttt{tt}$ in the following steps:

$$\begin{aligned} L(A) \not\subseteq L(B) &\Leftrightarrow \exists d.\, d \in L(A) \setminus L(B) \\ &\Leftrightarrow \forall \theta \models \gamma_4.\, \exists d.\, \theta, d \models x_1^{\bowtie} \wedge \theta[1 \leftarrow d] \models \gamma_3 \\ &\Leftrightarrow pr(\gamma_4, x_1^{\bowtie}, 1, \gamma_3) = \texttt{tt}. \end{aligned}$$

$L(A) \not\subseteq L(B) \Leftrightarrow \exists d.\, d \in L(A) \setminus L(B)$ is obviously satisfied, and $\forall \theta \models \gamma_4.\, \exists d.\, \theta, d \models x_1^{\bowtie} \wedge \theta[1 \leftarrow d] \models \gamma_3$
$\Leftrightarrow pr(\gamma_4, x_1^{\bowtie}, 1, \gamma_3) = \texttt{tt}$ is obtained by the definition of pr. We prove the middle step $\exists d.\, d \in L(A) \setminus L(B) \Leftrightarrow \forall \theta \models \gamma_4.\, \exists d.\, \theta, d \models x_1^{\bowtie} \wedge \theta[1 \leftarrow d] \models \gamma_3$ as follows.

- $(\Rightarrow)$: Let θ be an arbitrary assignment that satisfies $\theta \models \gamma_4$. Note that $\theta(2) \notin L(A) \setminus L(B)$ by the definition of γ_4. By the assumption, there is a data value $d \in L(A) \setminus L(B)$. This d satisfies $\theta, d \models x_1^{\bowtie}$. Moreover, this d also satisfies $\theta[1 \leftarrow d] \models \gamma_3$ because $\theta[1 \leftarrow d](1) = d \in L(A) \setminus L(B)$ and $\theta[1 \leftarrow d](2) = \theta(2) \notin L(A) \setminus L(B)$.
- $(\Leftarrow)$: Because $L(B) \neq \emptyset$, there exists some $d' \in L(B)$. Let θ be the assignment defined as $\theta(1) = \theta(2) = d'$, which satisfies $\theta \models \gamma_4$. By the

assumption, there exists d such that $\theta, d \models x_1^{\bowtie}$, and this implies that $d \in L(A) \setminus L(B)$.

We construct 2-GRCFG($\bowtie$) $G = (V, R, S)$ such that $R = \{(S, x_1^{\bowtie}) \to \varepsilon\}$. We can easily show that $\varepsilon \in L(G)$ (iff $L(G) \neq \emptyset$) iff $L(A) \not\subseteq L(B)$.

4.3.3 GRCFG with a total order on a dense set

Lemma 4.3.1. *Every GRCFG($<_Q$) has the simulation and progress properties where $<_Q$ is the strict total order on the set Q of all rational numbers. Similarly, it has the simulation and progress properties with data type.*

Proof We abbreviate $<_Q$ as $<$. Let $G = (V, R, S)$ be a k-GRCFG($<$), $\theta \in \Theta_k$, $\gamma \in \Gamma_k$ and $r = (A, \psi, i) \to \alpha \in R$ where ψ is the conjunction of literals (of the form $x_i^{<}$ or $\neg x_j^{<}$). (The case $r = (A, \psi) \to \alpha \in R$ can be treated in a similar way.) Assume that $\theta \models \gamma$. The rule r can be applied to (A, θ) iff there is $d \in Q$ such that $\theta, d \models \psi$. The condition $\theta, d \models \psi$ as well as the assumption $\theta \models \gamma$ can be represented as a set of inequations on $d, \theta(1), \ldots, \theta(k)$. Whether this set of inequations has a contradiction does not depend on the concrete values $\theta(1), \ldots, \theta(k)$, and if it does not have a contradiction, then there must exist $d \in Q$ that satisfies it because Q is dense. Moreover, whether $\theta[i \leftarrow d] \models \gamma'$ holds for a given γ', which can also be represented as the consistency of a set of inequations on $d, \theta(1), \ldots, \theta(k)$, does not depend on θ. Hence, for all $\theta \models \gamma$, whether there exists d such that $\theta, d \models \psi$ and $\theta[i \leftarrow d] \models \gamma'$ can be decided uniquely, and the simulation property holds.

Similarly, for deciding $pr(\gamma, \psi, i, \gamma') = \texttt{tt}$, it suffices to represent the condition

$$\theta \models \gamma \ \wedge \ \theta, d \models \psi \ \wedge \ \theta[i \leftarrow d] \models \gamma'$$

as a set of inequations on $d, \theta(1), \ldots, \theta(k)$ as above and solve it.

We can show the simulation and progress properties with data type in a similar way.

Example 4.3.2. *Consider a 2-GRCFG($<$) (V, R, S) and a rule $(A, \psi, 1) \to B \in R$ where $\psi = x_1^{<} \wedge \neg x_2^{<}$. We see that $\theta, d \models \psi \Leftrightarrow \theta(1) < d \leq \theta(2)$. Because $k = 2$*

and $<$ *is a total order on* Q*, there are three possible register types* $\gamma_1 = (\theta(1) < \theta(2))$*,* $\gamma_2 = (\theta(2) < \theta(1))$ *and* $\gamma_3 = (\theta(1) = \theta(2))$*. As easily known, (i) there is* $d \in Q$ *such that* $\theta, d \models \psi$ *and* $\theta \models \gamma$ *if and only if* $\gamma = \gamma_1$*, and (ii) if* $\gamma = \gamma_1$ *then such* $d \in Q$ *satisfies either (ii-a)* $d < \theta(2)$*,* $\theta[1 \leftarrow d] \models \gamma_1$ *or (ii-b)* $d = \theta(2)$*,* $\theta[1 \leftarrow d] \models \gamma_3$*.*

Corollary 4.3.1. *For a given GRCFG(*$<_Q$*)* G*, we can construct an* ε*-rule free GRCFG(*$<_Q$*)* G' *that satisfies* $L(G') = L(G) \setminus \{\varepsilon\}$*. The emptiness and membership problems are both EXPTIME-complete for GRCFG(*$<_Q$*).*

Proof By Lemma 4.3.1 and Theorems 4.3.1, 4.3.2 and 4.3.3.

Conclusion of Part II

- We have clarified the computational complexity of the membership and emptiness problems of RCFG and the related models. These results are important for the application of these register models to software engineering because some practical problems such as query for database and model checking can be reduced to membership and emptiness problems.
- We defined GRCFG and proposed two constraints, simulation and progress, which guarantees the membership and emptiness problems decidable. Even with these two constraints, GRCFG can deal with total order relation on rational numbers, which is useful for representing practical programs.

Part III

Software Verification

Outline of Part III

- In Chapter 5, we show the forward regularity preservation property of register pushdown systems (abbreviated as RPDS). Before giving the definition of RPDS, we define equivalence relations over the set of k-data values D^k, which is used as a guard condition of RPDS and RA in this part. This does not change the expressive power, comparing with the register models defined in Part I, because using equivalence relations as guard conditions corresponds to using normal form rules with respect to register type as discussed in [64, 65]. We also show the joinability of regular sets by RPDS is decidable as a corollary and give an example of malware analysis by using the algorithm.
- In Chapter 6, we propose algorithms for LTL model checking problems for RPDS with simple and regular valuations. The LTL model checking problem for RPDS takes an RPDS $\mathcal{P}$ with a start configuration c_0, an LTL formula φ and a valuation Λ, and asks whether every run of $\mathcal{P}$ reachable from c_0 satisfies φ under Λ. To prove the EXPTIME-completeness of the problem with simple valuation, we reduce the model checking problem to the membership problem for RA. In the case of the problem with regular valuation, we show the EXPTIME-completeness by reducing the problem to the one with simple valuation. As practical examples, we show solutions of a malware detection and an XML schema checking in the proposed framework.

Chapter 5 Regularity Preservation

5.1 Definitions

5.1.1 Equivalence relation and quotient set

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

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

Note that Φ_k is isomorphic to the set of equivalence relations over $[k]$. That is, for each $\phi \in \Phi_k$, we can define a unique equivalence relation $\sim_\phi$ over $[k]$ as $i \sim_\phi j$ iff every data word $d_1 \cdots d_k \in D^k$ such that $\langle d_1 \cdots d_k\rangle = \phi$ satisfies $d_i = d_j$. For example, for $\phi = \langle 35523\rangle$, $i \sim_\phi i$ $(i \in [5])$, $1 \sim_\phi 5$, $2 \sim_\phi 3$ and $i \not\sim_\phi j$ for every

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

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

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

The description length of an equivalence relation ϕ, denoted as $\|\phi\|$, is defined as $\|\phi\| = k(2k+1)$, because a reflexive symmetric binary relation over $2k+1$ elements can be described with $2k(2k+1)/2 = k(2k+1)$ bits.

We introduce the following lemma, which is given by the definition of equivalence class.

Lemma 5.1.1. *Let $u_1, u_2, u_3, v_1, v_2, v_3 \in D^*$ be any words over D and $d, d' \in D$ be any data values.*

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

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

(iii) *If $u_1, d, u_2, u_3 = v_1, d', v_2, v_3$, $|u_1| = |v_1|$ and $|u_2| = |v_2|$, then $u_1, d, u_2, d, u_3 =$*

v_1, d', v_2, d', v_3 .

5.1.2 Register pushdown systems

We define *register pushdown systems* (abbreviated as RPDS), which are equivalent but more concise than pushdown register systems (PDRS) [50], by using an equivalence relation as the guard condition of a transition rule.

Definition 5.1.1. *A k-register pushdown system (k-RPDS) over an infinite set D of data values is a pair $\mathcal{P} = (P, \Delta)$ where P is a finite set of states and Δ is a finite set of transition rules having one of the following forms:*

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

where $p, q \in P$, $j_1, j_2 \in [k]$, and $\phi \in \Phi_k$.

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

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

Let $\overset{*}{\Rightarrow}_{\mathcal{P}}$ be the reflexive transitive closure of $\Rightarrow_{\mathcal{P}}$. We abbreviate $\Rightarrow_{\mathcal{P}}$ and $\overset{*}{\Rightarrow}_{\mathcal{P}}$ as $\Rightarrow$ and $\overset{*}{\Rightarrow}$ if $\mathcal{P}$ is clear from the context. If we emphasize the rule r and the data value d, we write $\Rightarrow_d^r$. By definition, any ID $(p, \theta, \varepsilon) \in ID_{\mathcal{P}}$ has no successor. That is, there is no transition from an ID with empty stack. A *run* of k-RPDS

Table 5.1: A list of fixed equivalence relations appearing in examples later.

ϕ_i	X_2/ϕ_i
ϕ_0	$\{\{\mathtt{x}_1, \mathtt{x}'_1\}, \{\mathtt{x}_2\}, \{\mathtt{x}'_2, \mathtt{top}\}\}$
ϕ_1	$\{\{\mathtt{x}_1\}, \{\mathtt{x}_2, \mathtt{x}'_2\}, \{\mathtt{x}'_1, \mathtt{top}\}\}$
ϕ_2	$\{\{\mathtt{x}_1\}, \{\mathtt{x}_2\}, \{\mathtt{x}'_1, \mathtt{x}'_2, \mathtt{top}\}\}$
ϕ_3	$\{\{\mathtt{x}_1, \mathtt{x}_2, \mathtt{top}\}, \{\mathtt{x}'_1, \mathtt{x}'_2\}\}$
ϕ_4	$\{\{\mathtt{x}_1, \mathtt{x}'_1\}, \{\mathtt{x}_2, \mathtt{x}'_2\}, \{\mathtt{top}\}\}$
ϕ_5	$\{\{\mathtt{x}_1, \mathtt{x}'_1, \mathtt{top}\}, \{\mathtt{x}_2, \mathtt{x}'_2\}\}$
ϕ_6	$\{\{\mathtt{x}_1, \mathtt{x}'_1\}, \{\mathtt{x}_2, \mathtt{x}'_2, \mathtt{top}\}\}$
ϕ_7	$\{\{\mathtt{x}_1, \mathtt{x}'_1, \mathtt{x}'_2\}, \{\mathtt{x}_2, \mathtt{top}\}\}$
ϕ_8	$\{\{\mathtt{x}_1, \mathtt{x}'_1, \mathtt{x}'_2, \mathtt{top}\}, \{\mathtt{x}_2\}\}$
ϕ_9	$\{\{\mathtt{x}_1, \mathtt{x}'_1, \mathtt{x}'_2\}, \{\mathtt{x}_2\}, \{\mathtt{top}\}\}$

$\mathcal{P}$ is a finite or infinite sequence of IDs $\rho \in (ID_{\mathcal{P}})^{\infty}$ that satisfy $\rho(i) \Rightarrow_{\mathcal{P}} \rho(i+1)$ for $i \geq 0$.

In what follows, we often specify an equivalence relation ϕ over X_k by providing the quotient set (the set of equivalence classes) X_k/ϕ. Also, we fix equivalence relations $\phi_0, \ldots, \phi_9$ in Table 5.1 and will use them in Examples 5.1.1, 5.1.2, 5.2.1 and 5.3.1.

Example 5.1.1. *Let us consider* 2*-RPDS* $\mathcal{P} = (\{q_0, p_1, p_2\}, \{(q_0, \phi_0) \to (p_1, \varepsilon), (p_1, \phi_1) \to (p_2, 12)\})$ *(ϕ_0, ϕ_1 are defined in Table 5.1). In the example, we write* $\theta \in \Theta_2$ *as* $[d, d']$ *where* $\theta(1) = d$ *and* $\theta(2) = d'$*. We show transitions of* $\mathcal{P}$ *from* $(q_0, [d_0, d_1], d_2 d_3 d_3) \in ID_{\mathcal{P}}$ *where* $d_i \neq d_j$ *for all* $d_i, d_j \in D$ *if* $i \neq j$ *(Fig. 5.1). First, assume we apply* $r_0 = (q_0, \phi_0) \to (p_1, \varepsilon)$*. We obtain* $(q_0, [d_0, d_1], d_2 d_3 d_3) \Rightarrow^{r_0}_{d_2} (p_1, [d_0, d_2], d_3 d_3)$ *because* $\mathtt{top} \sim_{\phi_0} \mathtt{x}'_2$ *requires that the data value* d_2 *at the stack top before the transition should be equal to the data value in the second register after the transition, and* $\mathtt{x}_1 \sim_{\phi_0} \mathtt{x}'_1$ *requires that the data value in the first register does not change in the transition. If we apply* $r_1 = (p_1, \phi_1) \to (p_2, 12)$ *next, then we obtain* $(p_1, [d_0, d_2], d_3 d_3) \Rightarrow^{r_1}_{d_3} (p_2, [d_3, d_2], d_3 d_2 d_3)$*.*

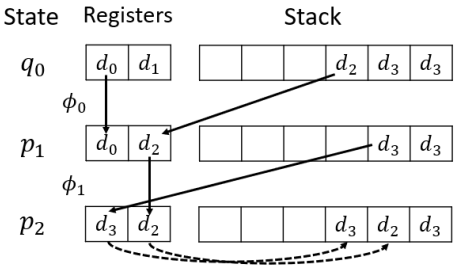

Figure 5.1: The transitions of $\mathcal{P}$ from an ID $(q_0, [d_0, d_1], d_2d_3d_3)$. (The register updates given by ϕ_0, ϕ_1 are shown by solid lines, and the push of data values by r_1 is shown by dotted lines.)

5.1.3 Register automata

We define a register automaton (RA) as an RPDS equipped with initial states and accepting conditions and having only pop rules. Note that this definition of RA is essentially the same as that of other studies such as [41, 25]. While an instantaneous description of RA has the same form (p, θ, w) as an ID of RPDS, the third component w is regarded as an input string to the RA, not a pushdown stack.

Definition 5.1.2. *A k-register automaton (k-RA) over an infinite set D of data values is $\mathcal{A} = (\mathcal{P}, I, \xi)$, where*

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

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

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

In the rest of the paper, we write $\mathcal{A} = (\mathcal{P}, I, \xi)$ with $\mathcal{P} = (Q, \delta)$ as (Q, I, ξ, δ) and $(p, \phi) \to (q, \varepsilon) \in \delta$ as $(p, \phi) \to q$. The language $L(\mathcal{A})$ recognized by a k-RA

$\mathcal{A} = (Q, I, \xi, \delta)$ is defined as $L(\mathcal{A}) = \{(p, \theta, w) \in ID_{\mathcal{A}} \mid p \in I, (p, \theta, w) \vdash^* (q, \theta', \varepsilon)$ for some $(q, \theta', \varepsilon) \in Acc_{\mathcal{A}}\}$.

Example 5.1.2. *Let 2-RA* $\mathcal{A} = (\{\{q_0, q_1, q_2\}, \{q_0\}, \{(q_2, \phi_f)\}, \{(q_0, \phi_0) \to q_1, (q_1, \phi_2) \to q_1, (q_1, \phi_3) \to q_2\})$ *where* $X_2/\phi_f = \{\{\mathtt{x}_1, \mathtt{x}_2\}\}$. *We show transitions of* $\mathcal{A}$ *from* $(q_0, [d_0, d_1], d_2d_3d_3) \in ID_{\mathcal{A}}$ *where* $d_i \neq d_j$ *for all* $d_i, d_j \in D$ *if* $i \neq j$. *First, we apply* $r_2 = (q_0, \phi_0) \to q_1$, *and obtain* $(q_0, [d_0, d_1], d_2d_3d_3) \vdash^{r_2}_{d_2} (q_1, [d_0, d_2], d_3d_3)$. *By applying* $r_3 = (q_1, \phi_2) \to q_1$ *in the next step, we obtain* $(q_1, [d_0, d_2], d_3d_3) \vdash^{r_3}_{d_3}$ $(q_1, [d_3, d_3], d_3)$. *Lastly, we apply* $r_4 = (q_1, \phi_3) \to q_2$ *and obtain* $(q_1, [d_3, d_3], d_3) \vdash^{r_4}_{d_3}$ $(q_2, [d_4, d_4], \varepsilon)$ *where* $d_4 \in D$ *is an arbitrary data value other than* d_3. *Since* $\langle [d_4, d_4] \rangle = \phi_f$, $(q_2, [d_4, d_4], \varepsilon)$ *is an accepting ID of* $\mathcal{A}$ *and thus* $(q_0, [d_0, d_1], d_2d_3d_3) \in L(\mathcal{A})$.

5.2 Regularity Preservation

Let $C \subseteq P \times \Theta_k \times D^*$ be a subset of IDs of a k-RPDS $\mathcal{P} = (P, \Delta)$. We define:

$$pre^*_{\mathcal{P}}(C) := \{c \mid c \overset{*}{\Rightarrow} c_0 \text{ for some } c_0 \in C\},$$
$$post^*_{\mathcal{P}}(C) := \{c \mid c_0 \overset{*}{\Rightarrow} c \text{ for some } c_0 \in C\}.$$

C is *regular* if and only if there exists a k-RA $\mathcal{A} = (Q, P, \xi, \delta)$ over D such that $C = L(\mathcal{A})$. $\mathcal{A}$ is called a k-RA for C. Note that the set of initial states of $\mathcal{A}$ must be P. We omit the subscript $\mathcal{P}$ and write $post^*(C)$ and $pre^*(C)$ if $\mathcal{P}$ is clear from the context.

The following is the main theorem of this paper.

Theorem 5.2.1. *Let* $\mathcal{P} = (P, \Delta)$ *be a k-RPDS and* C *be a regular set of IDs of* $\mathcal{P}$. *Then* $post^*(C)$ *is regular. Moreover, a k-RA for* $post^*(C)$ *can be effectively constructed from* $\mathcal{P}$ *and a k-RA for* C. □

Before describing the proposed construction algorithm for $post^*(C)$ and the lemmas for proving the main theorem, we define some notions for Φ_k.

Definition 5.2.1. *The* inverse *of an equivalence relation $\phi \in \Phi_k$ is $\phi^{-1} \in \Phi_k$ that satisfies the following conditions for all $i, j \in [k]$:*

$$\mathtt{x}_i \sim_{\phi^{-1}} \mathtt{x}_j \Leftrightarrow \mathtt{x}'_i \sim_{\phi} \mathtt{x}'_j, \qquad \mathtt{x}_i \sim_{\phi^{-1}} \mathtt{top} \Leftrightarrow \mathtt{x}'_i \sim_{\phi} \mathtt{top},$$
$$\mathtt{x}_i \sim_{\phi^{-1}} \mathtt{x}'_j \Leftrightarrow \mathtt{x}'_i \sim_{\phi} \mathtt{x}_j, \qquad \mathtt{x}'_i \sim_{\phi^{-1}} \mathtt{top} \Leftrightarrow \mathtt{x}_i \sim_{\phi} \mathtt{top},$$
$$\mathtt{x}'_i \sim_{\phi^{-1}} \mathtt{x}'_j \Leftrightarrow \mathtt{x}_i \sim_{\phi} \mathtt{x}_j.$$

The following lemma states a basic property of ϕ^{-1}, which can be directly proved by its definition.

Lemma 5.2.1. *Let $\theta_1, \theta_2 \in \Theta_k$, $d \in D$ and $\phi \in \Phi_k$. Then, $\langle \theta_1, \theta_2, d \rangle = \phi$ if and only if $\langle \theta_2, \theta_1, d \rangle = \phi^{-1}$.*

Definition 5.2.2. *The* middle composition *of $\phi_1, \phi_2 \in \Phi_k$ is $\phi_1 \circ_{\mathtt{M}} \phi_2 = \{\phi_{12} \in \Phi_k \mid \phi_1$, ϕ_2 and ϕ_{12} satisfy the following (5.1), (5.2), (5.3), (5.4), (5.6) and (5.7)$\}$. The* right composition *of $\phi_1, \phi_2 \in \Phi_k$ is $\phi_1 \circ_{\mathtt{R}} \phi_2 = \{\phi_{12} \in \Phi_k \mid \phi_1$, ϕ_2 and ϕ_{12} satisfy the following (5.1), (5.2), (5.4), (5.5), (5.6) and (5.8)$\}$. In (5.6)–(5.8), $a \not\Leftrightarrow b$ means that exactly one of a and b is true and the other is false.*

(Left-consistency) $$\mathtt{x}_i \sim_{\phi_1} \mathtt{x}_j \Leftrightarrow \mathtt{x}_i \sim_{\phi_{12}} \mathtt{x}_j \quad \text{for } i, j \in [k] \tag{5.1}$$

(Middle-consistency) $$\mathtt{x}'_i \sim_{\phi_1} \mathtt{x}'_j \Leftrightarrow \mathtt{x}_i \sim_{\phi_2} \mathtt{x}_j \quad \text{for } i, j \in [k] \tag{5.2}$$

$$\mathtt{x}'_i \sim_{\phi_1} \mathtt{top} \Leftrightarrow \mathtt{x}_i \sim_{\phi_2} \mathtt{top} \quad \text{for } i \in [k] \tag{5.3}$$

(Right-consistency) $$\mathtt{x}'_i \sim_{\phi_2} \mathtt{x}'_j \Leftrightarrow \mathtt{x}'_i \sim_{\phi_{12}} \mathtt{x}'_j \quad \text{for } i, j \in [k] \tag{5.4}$$

$$\mathtt{x}'_i \sim_{\phi_2} \mathtt{top} \Leftrightarrow \mathtt{x}'_i \sim_{\phi_{12}} \mathtt{top} \quad \text{for } i \in [k] \tag{5.5}$$

(Transitivity) $$(\mathtt{x}_i \sim_{\phi_1} \mathtt{x}'_l \wedge \mathtt{x}_l \sim_{\phi_2} \mathtt{x}'_j) \Rightarrow \mathtt{x}_i \sim_{\phi_{12}} \mathtt{x}'_j \text{ and}$$
$$(\mathtt{x}_i \sim_{\phi_1} \mathtt{x}'_l \not\Leftrightarrow \mathtt{x}_l \sim_{\phi_2} \mathtt{x}'_j) \Rightarrow \mathtt{x}_i \not\sim_{\phi_{12}} \mathtt{x}'_j \text{ for } i, j, l \in [k] \tag{5.6}$$
$$(\mathtt{x}_i \sim_{\phi_1} \mathtt{top} \wedge \mathtt{top} \sim_{\phi_2} \mathtt{x}'_j) \Rightarrow \mathtt{x}_i \sim_{\phi_{12}} \mathtt{x}'_j \text{ and}$$
$$(\mathtt{x}_i \sim_{\phi_1} \mathtt{top} \not\Leftrightarrow \mathtt{top} \sim_{\phi_2} \mathtt{x}'_j) \Rightarrow \mathtt{x}_i \not\sim_{\phi_{12}} \mathtt{x}'_j \text{ for } i, j \in [k] \tag{5.7}$$
$$(\mathtt{x}_i \sim_{\phi_1} \mathtt{x}'_l \wedge \mathtt{x}_l \sim_{\phi_2} \mathtt{top}) \Rightarrow \mathtt{x}_i \sim_{\phi_{12}} \mathtt{top} \text{ and}$$
$$(\mathtt{x}_i \sim_{\phi_1} \mathtt{x}'_l \not\Leftrightarrow \mathtt{x}_l \sim_{\phi_2} \mathtt{top}) \Rightarrow \mathtt{x}_i \not\sim_{\phi_{12}} \mathtt{top} \text{ for } i, l \in [k] \tag{5.8}$$

We extend $\circ_{\mathtt{R}}$ to the subsets of Φ_k in the usual way: For $S \subseteq \Phi_k$ and $\phi_2 \in \Phi_k$, let $S \circ_{\mathtt{R}} \phi_2 = \{\phi_{12} \in \Phi_k \mid \phi_{12} \in \phi_1 \circ_{\mathtt{R}} \phi_2$ for some $\phi_1 \in S\}$.

Note that if ϕ_1 and ϕ_2 do not satisfy (5.2) and (5.3), then $\phi_1 \circ_{\mathtt{M}} \phi_2 = \emptyset$. Let $\phi_{12} \in \phi_1 \circ_{\mathtt{M}} \phi_2$ for ϕ_1 and ϕ_2 that satisfy (5.2) and (5.3). The relation among $\mathtt{x}_1, \ldots, \mathtt{x}_k$ and the relation among $\mathtt{x}'_1, \ldots, \mathtt{x}'_k$ represented by ϕ_{12} are uniquely determined by (5.1) and (5.4), respectively. The relation between an element of $\{\mathtt{x}_1, \ldots, \mathtt{x}_k\}$ and an element of $\{\mathtt{x}'_1, \ldots, \mathtt{x}'_k\}$ represented by ϕ_{12} is (non-uniquely) determined by (5.6) and (5.7). Note that (5.6) and (5.7) are well-defined for ϕ_1 and ϕ_2 that satisfy (5.2) and (5.3): For example, if $\mathtt{x}_i \sim_{\phi_1} \mathtt{x}'_{l_1} \wedge \mathtt{x}_{l_1} \sim_{\phi_2} \mathtt{x}'_j$ and $\mathtt{x}_i \sim_{\phi_1} \mathtt{x}'_{l_2}$, then $\mathtt{x}_{l_2} \sim_{\phi_2} \mathtt{x}'_j$ because $\mathtt{x}_i \sim_{\phi_1} \mathtt{x}'_{l_1} \sim_{\phi_1} \mathtt{x}'_{l_2}$ implies $\mathtt{x}_{l_1} \sim_{\phi_2} \mathtt{x}_{l_2}$ by (5.2). Since no constraint on $\mathtt{top}$ is imposed on ϕ_{12}, both $\mathtt{x}_i \sim_{\phi_{12}} \mathtt{top}$ and $\mathtt{x}_i \not\sim_{\phi_{12}} \mathtt{top}$ (and both $\mathtt{top} \sim_{\phi_{12}} \mathtt{x}'_i$ and $\mathtt{top} \not\sim_{\phi_{12}} \mathtt{x}'_i$) are allowed for any $i \in [k]$, and thus ϕ_{12} is not unique. Moreover, if there are i and j such that $\mathtt{x}_i \not\sim_{\phi_1} \mathtt{x}'_l$ and $\mathtt{x}_l \not\sim_{\phi_2} \mathtt{x}'_j$ for every $l \in [k]$ and $\mathtt{x}_i \not\sim_{\phi_1} \mathtt{top}$ and $\mathtt{top} \not\sim_{\phi_2} \mathtt{x}'_j$, then both $\mathtt{x}_i \sim_{\phi 12} \mathtt{x}'_j$ and $\mathtt{x}_i \not\sim_{\phi 12} \mathtt{x}'_j$ are also allowed. An element of $\phi_1 \circ_{\mathtt{R}} \phi_2$ for ϕ_1 and ϕ_2 that satisfy (5.2) is determined in the same manner.

Example 5.2.1. *For ϕ_0 in Table 5.1, the inverse ${\phi_0}^{-1}$ satisfies $X_2/{\phi_0}^{-1} = \{\{\mathtt{top}, \mathtt{x}_2\}, \{\mathtt{x}_1, \mathtt{x}'_1\}, \{\mathtt{x}'_2\}\}$. Let us compute the middle composition ${\phi_0}^{-1} \circ_{\mathtt{M}} \phi_0$. Since ${\phi_0}^{-1}$ and ϕ_0 satisfy (5.2) and (5.3), ${\phi_0}^{-1} \circ_{\mathtt{M}} \phi_0 \neq \emptyset$ holds. For each $\phi_{12} \in {\phi_0}^{-1} \circ_{\mathtt{M}} \phi_0$, we obtain $\mathtt{x}_1 \not\sim_{\phi_{12}} \mathtt{x}_2$ by (5.1), $\mathtt{x}'_1 \not\sim_{\phi_{12}} \mathtt{x}'_2$ by (5.4), $\mathtt{x}_1 \sim_{\phi_{12}} \mathtt{x}'_1$ by (5.6) and $\mathtt{x}_2 \sim_{\phi_{12}} \mathtt{x}'_2$ by (5.7). Since no constraint on $\mathtt{top}$ is imposed on ϕ_{12}, ${\phi_0}^{-1} \circ_{\mathtt{M}} \phi_0$ consists of three equivalence relations ϕ_4, ϕ_5 and ϕ_6 in Table 5.1. Next, we compute the right composition $\phi_4 \circ_{\mathtt{R}} \phi_2$. We obtain $\phi_4 \circ_{\mathtt{R}} \phi_2 \neq \emptyset$ because ϕ_4 and ϕ_2 satisfy (5.2). Also, for each $\phi_{12} \in \phi_4 \circ_{\mathtt{R}} \phi_2$, we obtain $\mathtt{x}_1 \not\sim_{\phi_{12}} \mathtt{x}_2$ by (5.1), $\mathtt{x}'_1 \sim_{\phi_{12}} \mathtt{x}'_2 \sim_{\phi_{12}} \mathtt{top}$ by (5.4) and (5.5), and $\mathtt{x}_1 \not\sim_{\phi_{12}} \mathtt{x}'_1$ and $\mathtt{x}_2 \not\sim_{\phi_{12}} \mathtt{x}'_2$ by (5.6) (and $\mathtt{x}_1 \not\sim_{\phi_{12}} \mathtt{top}$ and $\mathtt{x}_2 \not\sim_{\phi_{12}} \mathtt{top}$ by (5.8), which are already obtained from other relations). Hence ϕ_{12} must equal ϕ_2 and thus $\phi_4 \circ_{\mathtt{R}} \phi_2 = \{\phi_2\}$. Likewise, $\phi_5 \circ_{\mathtt{R}} \phi_2 = \phi_6 \circ_{\mathtt{R}} \phi_2 = \{\phi_2\}$ holds. Therefore, we obtain $({\phi_0}^{-1} \circ_{\mathtt{M}} \phi_0) \circ_{\mathtt{R}} \phi_2 = \{\phi_4, \phi_5, \phi_6\} \circ_{\mathtt{R}} \phi_2 = \{\phi_2\}$. On the other hand, $({\phi_0}^{-1} \circ_{\mathtt{M}} \phi_0) \circ_{\mathtt{R}} \phi_3 = \emptyset$ because ϕ' and ϕ_3 do not satisfy (5.2) for any $\phi' \in \{\phi_4, \phi_5, \phi_6\}$.*

The following lemmas state basic properties of $\circ_{\mathtt{M}}$ and $\circ_{\mathtt{R}}$.

Lemma 5.2.2. *Let $\theta_1, \theta_3 \in \Theta_k$ and $\phi_1, \phi_2 \in \Phi_k$.*

(i) If there exist $\theta_2 \in \Theta_k$ and $d \in D$ such that $\langle \theta_1, \theta_2, d \rangle = \phi_1$ and $\langle \theta_2, \theta_3, d \rangle = \phi_2$,

then $\phi_{12} \in \phi_1 \circ_{\mathtt{M}} \phi_2$ *holds for every* $\phi_{12} \in \Phi_k$ *satisfying* $\langle \theta_1, \theta_3, d' \rangle = \phi_{12}$ *for some* $d' \in D$.

(ii) If there exists $\phi_{12} \in \phi_1 \circ_{\mathtt{M}} \phi_2$ *satisfying* $\langle \theta_1, \theta_3, d' \rangle = \phi_{12}$ *for some* $d' \in D$, *then there exist* $\theta_2 \in \Theta_k$ *and* $d \in D$ *such that* $\langle \theta_1, \theta_2, d \rangle = \phi_1$ *and* $\langle \theta_2, \theta_3, d \rangle = \phi_2$.

Proof (i) Assume that $\langle \theta_1, \theta_2, d \rangle = \phi_1$ and $\langle \theta_2, \theta_3, d \rangle = \phi_2$ and $\langle \theta_1, \theta_3, d' \rangle = \phi_{12}$. Then ϕ_{12} is uniquely defined by the following formulas: for all $i, j \in [k]$,

$$\begin{aligned}
&\mathtt{x}_i \sim_{\phi_{12}} \mathtt{x}_j \Leftrightarrow \theta_1(i) = \theta_1(j), && \mathtt{x}_i \sim_{\phi_{12}} \mathtt{top} \Leftrightarrow \theta_1(i) = d',\\
&\mathtt{x}_i \sim_{\phi_{12}} \mathtt{x}'_j \Leftrightarrow \theta_1(i) = \theta_3(j), && \mathtt{x}'_j \sim_{\phi_{12}} \mathtt{top} \Leftrightarrow \theta_3(j) = d',\\
&\mathtt{x}'_i \sim_{\phi_{12}} \mathtt{x}'_j \Leftrightarrow \theta_3(i) = \theta_3(j).
\end{aligned}$$

We can show ϕ_{12} satisfies (5.1), (5.4), (5.6) and (5.7) as below.

$$\begin{aligned}
(5.1)\colon\ & \mathtt{x}_i \sim_{\phi_1} \mathtt{x}_j \Leftrightarrow \theta_1(i) = \theta_1(j) \Leftrightarrow \mathtt{x}_i \sim_{\phi_{12}} \mathtt{x}_j.\\
(5.4)\colon\ & \mathtt{x}'_i \sim_{\phi_2} \mathtt{x}'_j \Leftrightarrow \theta_3(i) = \theta_3(j) \Leftrightarrow \mathtt{x}'_i \sim_{\phi_{12}} \mathtt{x}'_j.\\
(5.6)\colon\ & (\mathtt{x}_i \sim_{\phi_1} \mathtt{x}'_l \wedge \mathtt{x}_l \sim_{\phi_2} \mathtt{x}'_j) \Leftrightarrow \theta_1(i) = \theta_2(l) = \theta_3(j)\\
& \quad \Rightarrow \theta_1(i) = \theta_3(j) \Leftrightarrow \mathtt{x}_i \sim_{\phi_{12}} \mathtt{x}'_j, \text{ and}\\
& \quad (\mathtt{x}_i \sim_{\phi_1} \mathtt{x}'_l \not\Leftrightarrow \mathtt{x}_l \sim_{\phi_2} \mathtt{x}'_j)\\
& \quad \Leftrightarrow (\theta_1(i) \neq \theta_2(l) = \theta_3(j) \vee \theta_1(i) = \theta_2(l) \neq \theta_3(j))\\
& \quad \Rightarrow \theta_1(i) \neq \theta_3(j) \Leftrightarrow \mathtt{x}_i \not\sim_{\phi_{12}} \mathtt{x}'_j.\\
(5.7)\colon\ & (\mathtt{x}_i \sim_{\phi_1} \mathtt{top} \wedge \mathtt{top} \sim_{\phi_2} \mathtt{x}'_j) \Leftrightarrow \theta_1(i) = d = \theta_3(j)\\
& \quad \Rightarrow \theta_1(i) = \theta_3(j) \Leftrightarrow \mathtt{x}_i \sim_{\phi_{12}} \mathtt{x}'_j, \text{ and}\\
& \quad (\mathtt{x}_i \sim_{\phi_1} \mathtt{top} \not\Leftrightarrow \mathtt{top} \sim_{\phi_2} \mathtt{x}'_j)\\
& \quad \Leftrightarrow (\theta_1(i) \neq d = \theta_3(j) \vee \theta_1(i) = d \neq \theta_3(j))\\
& \quad \Rightarrow \theta_1(i) \neq \theta_3(j) \Leftrightarrow \mathtt{x}_i \not\sim_{\phi_{12}} \mathtt{x}'_j.
\end{aligned}$$

Also, we can show ϕ_1 and ϕ_2 satisfy (5.2) and (5.3) by $\langle \theta_1, \theta_2, d \rangle = \phi_1$ and $\langle \theta_2, \theta_3, d \rangle = \phi_2$. Hence, $\phi_{12} \in \phi_1 \circ_{\mathtt{M}} \phi_2$ holds.

(ii) Assume that $\phi_{12} \in \phi_1 \circ_{\mathtt{M}} \phi_2$ and $d' \in D$ and $\langle \theta_1, \theta_3, d' \rangle = \phi_{12}$. Since $\phi_1 \circ_{\mathtt{M}} \phi_2 \neq \emptyset$, ϕ_1 and ϕ_2 must satisfy (5.2) and (5.3).

Define θ_2 by the following formulas: for all $i, j \in [k]$,

$$\theta_2(i) = \theta_1(j) \Leftrightarrow \mathtt{x}'_i \sim_{\phi_1} \mathtt{x}_j, \tag{5.9}$$

$$\theta_2(i) = \theta_3(j) \Leftrightarrow \mathtt{x}_i \sim_{\phi_2} \mathtt{x}'_j, \tag{5.10}$$

$$\theta_2(i) = \theta_2(j) \Leftrightarrow \mathtt{x}'_i \sim_{\phi_1} \mathtt{x}'_j \ (\Leftrightarrow \mathtt{x}_i \sim_{\phi_2} \mathtt{x}_j). \tag{5.11}$$

We can show that θ_2 is well-defined. Assume that θ_2 is not well-defined. Then some pair of these conditions (5.9), (5.10) and (5.11) is not consistent for $\theta_2(l)$ for some $l \in [k]$. If the pair (5.9, 5.10) is not consistent, then one of the following conditions holds for some $i, j, l \in [k]$:

- $\mathtt{x}'_l \sim_{\phi_1} \mathtt{x}_i$ and $\mathtt{x}_l \sim_{\phi_2} \mathtt{x}'_j$ and $\theta_1(i) \neq \theta_3(j)$,
- $\mathtt{x}'_l \not\sim_{\phi_1} \mathtt{x}_i$ and $\mathtt{x}_l \sim_{\phi_2} \mathtt{x}'_j$ and $\theta_1(i) = \theta_3(j)$,
- $\mathtt{x}'_l \sim_{\phi_1} \mathtt{x}_i$ and $\mathtt{x}_l \not\sim_{\phi_2} \mathtt{x}'_j$ and $\theta_1(i) = \theta_3(j)$.

However, the first condition never holds because $\mathtt{x}'_l \sim_{\phi_1} \mathtt{x}_i$ and $\mathtt{x}_l \sim_{\phi_2} \mathtt{x}'_j$ implies $\mathtt{x}_i \sim_{\phi_{12}} \mathtt{x}'_j$ by (5.6), which implies $\theta_1(i) = \theta_3(j)$ by $\langle \theta_1, \theta_3, d' \rangle = \phi_{12}$. The second and the third conditions never hold similarly by (5.6) and $\langle \theta_1, \theta_3, d' \rangle = \phi_{12}$. Therefore the pair (5.9, 5.10) must be consistent. Moreover, the pairs of the conditions other than (5.9, 5.10) are always consistent because these pairs of conditions are for the same equivalence relations. Therefore, θ_2 is well-defined.

Next, define d by the following formulas: for all $i \in [k]$,

$$d = \theta_1(i) \Leftrightarrow \mathtt{top} \sim_{\phi_1} \mathtt{x}_i, \tag{5.12}$$

$$d = \theta_3(i) \Leftrightarrow \mathtt{top} \sim_{\phi_2} \mathtt{x}'_i, \tag{5.13}$$

$$d = \theta_2(i) \Leftrightarrow \mathtt{top} \sim_{\phi_1} \mathtt{x}'_i \ (\Leftrightarrow \mathtt{top} \sim_{\phi_2} \mathtt{x}_i). \tag{5.14}$$

We can show that d is well-defined in a similar way to the well-definedness of θ_2. The above θ_2 and d satisfy $\langle \theta_1, \theta_2, d \rangle = \phi_1$ and $\langle \theta_2, \theta_3, d \rangle = \phi_2$ by their definition. □

Lemma 5.2.3. *Let $\theta_1, \theta_3 \in \Theta_k$, $d \in D$ and $\phi_1, \phi_2 \in \Phi_k$.*

(i) If there exist $\theta_2 \in \Theta_k$ and $d' \in D$ such that $\langle \theta_1, \theta_2, d' \rangle = \phi_1$ and $\langle \theta_2, \theta_3, d \rangle = \phi_2$, then $\phi_{12} \in \phi_1 \circ_{\mathtt{R}} \phi_2$ holds for the unique $\phi_{12} \in \Phi_k$ satisfying $\langle \theta_1, \theta_3, d \rangle = \phi_{12}$.

(ii) If there exists $\phi_{12} \in \phi_1 \circ_{\mathtt{R}} \phi_2$ satisfying $\langle \theta_1, \theta_3, d \rangle = \phi_{12}$, then there exist $\theta_2 \in \Theta_k$ and $d' \in D$ such that $\langle \theta_1, \theta_2, d' \rangle = \phi_1$ and $\langle \theta_2, \theta_3, d \rangle = \phi_2$.

(Since it can be proved in a similar way to Lemma 5.2.2, we omit the proof of Lemma 5.2.3.)

Both the middle composition and the right composition are used in the proposed algorithm in the next section for constructing a *composition* of two transition rules. Intuitively, a composition of two transition rules $r_1 = (q_1, \phi_1) \to (q_2, J_1)$ and $r_2 = (q_2, \phi_2) \to (q_3, J_2)$ in this order is a transition rule $r = (q_1, \phi_{12}) \to (q_3, J)$ that satisfies $c_0 \Rightarrow_d^r c_2$ whenever $c_0 \Rightarrow_{d_1}^{r_1} c_1 \Rightarrow_{d_2}^{r_2} c_2$ for some IDs c_0, c_1, c_2. We use $\phi_1 \circ_{\mathtt{M}} \phi_2$ and $\phi_1 \circ_{\mathtt{R}} \phi_2$ to obtain the equivalence relation ϕ_{12} of the composition r. The conditions (5.1), (5.2), (5.4) and (5.6) in Definition 5.2.2 are shared by both $\circ_{\mathtt{M}}$ and $\circ_{\mathtt{R}}$, and the meanings of them are apparent.

The middle composition is used for constructing the composition of the inverse of a transition rule of RPDS $\mathcal{P}$ and a transition rule of RA $\mathcal{A}$. Let $r_1 = (p_0, \phi_1) \to (p_1, J_1)$ and $r_2 = (p_0, \phi_2) \to p_2$ be transition rules of $\mathcal{P}$ and $\mathcal{A}$, respectively. A rule $r_{12} = (p_1, \phi_{12}) \to p_2$ with $\phi_{12} \in {\phi_1}^{-1} \circ_{\mathtt{M}} \phi_2$ can be regarded as the composition of the inverse of r_1 and r_2 in this order. That is, if $c_1 \Leftarrow_d^{r_1} c_0$ and $c_0 \vdash_d^{r_2} c_2$, then $c_1 \vdash_{d'}^{r_{12}} c_2$ for some $d' \in D$. (Note that if r_1 is a push rule, then r_{12} has to reduce the length of stack by two, and so we have to simulate that composition by two sequential transition rules. We also need a similar treatment if r_1 is a pop rule.) We will add the composition r_{12} to $\mathcal{A}$ because $c_0 \Rightarrow_{\mathcal{P}} c_1$ and we want to let $c_1 \in L(\mathcal{A})$ if $c_0 \in L(\mathcal{A})$. The conditions (5.3) and (5.7) in Definition 5.2.2 mean that the transition by r_1 (from c_0 to c_1) and the transition by r_2 (from c_0 to c_2) consume the same data value d at the stack top.

The right composition is used for constructing the composition r_{12} of a transition rule r_1 that does not alter the stack and another rule r_2. We construct r_{12} so that if $c_0 \Rightarrow^{r_1} c_1$ and $c_1 \vdash_d^{r_2} c_2$ and the stack of c_0 and c_1 are the same, then $c_0 \vdash_d^{r_{12}} c_2$. By the conditions (5.5) and (5.8), $\phi_{12} \in \phi_1 \circ_{\mathtt{R}} \phi_2$ inherits the constraints on `top` from ϕ_2, and thus the transitions by r_{12} and r_2 consume the same data value d.

5.3 Saturation Algorithm for $post^*(C)$

Before providing the proposed algorithm that constructs an RA recognizing $post^*(C)$ from a given RPDS and an RA for C, we review the algorithm that constructs a nondeterministic finite automaton (NFA) that recognizes $post^*_{\mathcal{P}}(C)$ for a given PDS $\mathcal{P}$ and an NFA $\mathcal{A}$ recognizing C (Algorithm 5.1) [29]. Let us fix a finite alphabet Γ. A PDS over Γ is $\mathcal{P} = (P, \Delta)$ where P is a finite set of states and Δ is a set of transition rules. An NFA over Γ is $\mathcal{A} = (Q, I, F, \delta)$ where Q is a finite set of states, I and F are the sets of initial and final states, respectively, and δ is a set of transition rules. As in the case of RA, let $\vdash_{\mathcal{A}}$ and $L(\mathcal{A})$ denote the one-step transition relation of $\mathcal{A}$ and the language recognized by $\mathcal{A}$, respectively. The aim of the algorithm is to construct an NFA $\widetilde{\mathcal{A}}$ that satisfies

$$\Big(\exists (p, w) \in L(\mathcal{A}) \text{ s.t. } (p, w) \stackrel{*}{\Rightarrow}_{\mathcal{P}} (p', w')\Big) \text{ iff } (p', w') \in L(\widetilde{\mathcal{A}})$$

from a PDS $\mathcal{P}$ and an NFA $\mathcal{A}$. For this purpose, starting with $\mathcal{A}_0 = (Q_0, P, F, \delta_0)$ where Q_0 and δ_0 are defined in Line 2 of Algorithm 5.1, we gradually extend $\mathcal{A}_0$ to $\mathcal{A}_1, \mathcal{A}_2, \ldots$ by adding transition rules to $\mathcal{A}_h = (Q_0, P, F, \delta_h)$ as follows. For a push rule $r = (p, a) \to (q, bc) \in \Delta$ and a state q' such that $(p, a) \vdash^*_{\mathcal{A}_h} (q', \varepsilon)$, we add to $\mathcal{A}_h$ two rules $(q, b) \to \langle r \rangle$ and $(\langle r \rangle, c) \to q'$ where $\langle r \rangle$ is a state added for r in Line 2 of the algorithm, so that $(q, bcu) \vdash^*_{\mathcal{A}_{h+1}} (q', u)$ holds. We add the former rule only once to guarantee the termination of the algorithm. For a replace rule, a rule is added in a similar (but simpler) way. For a pop rule $(p, a) \to (q, \varepsilon) \in \Delta$ and a state q' such that $(p, a) \vdash^*_{\mathcal{A}_h} (q', \varepsilon)$, we add $(q, \varepsilon) \to q'$ to $\mathcal{A}_h$. Note that when the algorithm choose a state q' in Line 2, the transition sequence $(p, a) \vdash^*_{\mathcal{A}_h} (q', \varepsilon)$ can include zero or more ε-transitions. In the proposed algorithm described below, we use a slightly different construction for a pop rule by one-step lookahead to avoid the construction of an ε-rule.

We show an algorithm that constructs $post^*(C)$ from a given k-RPDS $\mathcal{P}$ and a k-RA for C in Algorithm 5.2.

Fig. 5.2 (a) illustrates how a new rule is added for a push rule in Line 7. Assume there exist a transition $(p, \theta_0, du) \Rightarrow^r_d (q, \theta_1, \theta_1(j_1)\theta_1(j_2)u)$ by a push rule $r = (p, \phi_0) \to (q, j_1 j_2)$, and a transition $(p, \theta_0, du) \vdash^{r_1}_d (q', \theta_2, u)$ by a rule

Alg. 5.1: Saturation algorithm for a given PDA and an NFA

1: Input: PDS $\mathcal{P} = (P, \Delta)$ and NFA $\mathcal{A} = (Q, P, F, \delta)$ for C
2: Let NFA $\mathcal{A}_0 = (Q_0, P, F, \delta_0)$ where
$Q_0 = Q \cup \{\langle r \rangle \mid r = (p, a) \to (q, bc) \in \Delta\}$,
$\delta_0 = \delta \cup \{(q, b) \to \langle r \rangle \mid r = (p, a) \to (q, bc) \in \Delta\}$.
3: $h := 0$
4: **repeat**
5: Choose a rule $r = (p, a) \to (q, u) \in \Delta$ of $\mathcal{P}$
and a state $q' \in Q_0$ such that $(p, a) \vdash^*_{\mathcal{A}_h} (q', \varepsilon)$.
6: **if** $u = bc$ for some $b, c \in \Gamma$ **then**
7: Let $\delta_{new} = \{(\langle r \rangle, c) \to q'\}$.
8: **else if** $u = b$ for some $b \in \Gamma$ **then**
9: Let $\delta_{new} = \{(q, b) \to q'\}$.
10: **else if** $u = \varepsilon$ **then**
11: Let $\delta_{new} = \{(q, \varepsilon) \to q'\}$.
12: **end if**
13: Let k-RA $\mathcal{A}_{h+1} = (Q_0, P, F, \delta_{h+1})$, where $\delta_{h+1} = \delta_h \cup \delta_{new}$.
14: $h := h + 1$.
15: **until** no more transitions can be added.
16: Output: $\widetilde{\mathcal{A}} := \mathcal{A}_h$

$r_1 = (p, \phi_1) \to q'$. Then, we have to add new rules and states that enable the transitions $(q, \theta_1, \theta_1(j_1)\theta_1(j_2)u) \vdash^* (q', \theta_2, u)$. Because r is a push rule, we need to construct two RA rules to reach (q', θ_2, u). For the same reason as Algorithm 5.1, for each push rule r, we add the first of the two RA rules and the intermediate state $\langle r \rangle$ only once in Line 2. This rule does not update registers (enforced by $\forall s.\, \mathtt{x}_s \sim_\phi \mathtt{x}'_s$ in Line 2) but consumes the contents of j_1-th register pushed by the rule r (enforced by $\mathtt{top} \sim_\phi \mathtt{x}_{j_1}$). Thus, this rule r_2 satisfies $(q, \theta_1, \theta_1(j_1)\theta_1(j_2)u) \vdash^{r_2}_{\theta_1(j_1)} (\langle r \rangle, \theta_1, \theta_1(j_2)u)$. In Line 7, we add the second RA rule $r_{new} = (\langle r \rangle, \phi) \to q'$ for all $\phi \in {\phi_0}^{-1} \circ_{\mathtt{M}} \phi_1$, by which the transition $(\langle r \rangle, \theta_1, \theta_1(j_2)u) \vdash^{r_{new}}_{\theta_1(j_2)} (q', \theta_2, u)$ is enabled (thus, $\langle \theta_1, \theta_2, \theta_1(j_2) \rangle = \phi$ is required).

Fig. 5.2 (b) illustrates how a new rule is added for a pop rule $r = (p, \phi_0) \to$

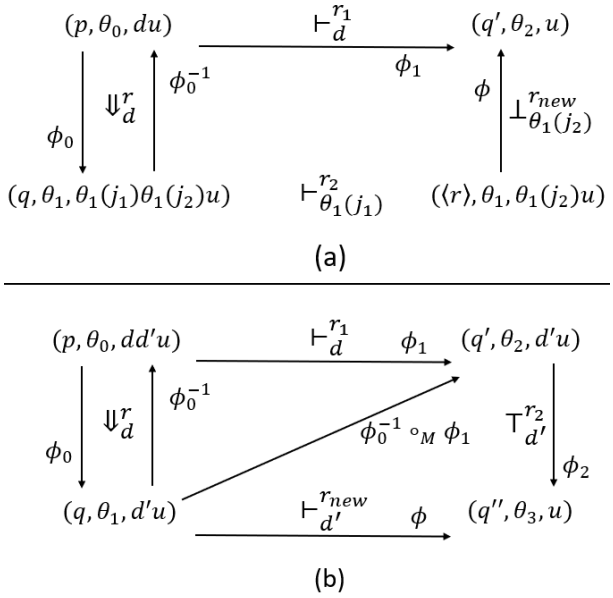

Figure 5.2: Construction of a rule r_{new} for a push rule (a) and a pop rule (b).

(q, ε) in Line 11. The idea is similar to the case of a push rule, but we need to choose two RA rules $r_1 = (p, \phi_1) \to q'$ and $r_2 = (q', \phi_2) \to q''$ and construct the composition of three equivalence relations ${\phi_0}^{-1}, \phi_1, \phi_2$. Thus, we add the transition rule $r_{new} = (p, \phi) \to q''$ for all $\phi \in ({\phi_0}^{-1} \circ_{\mathtt{M}} \phi_1) \circ_{\mathtt{R}} \phi_2$.

Example 5.3.1. *We show an example run of Algorithm 5.2 for 2-RPDS* $\mathcal{P} = (P, \Delta)$ *defined in Example 5.1.1 and 2-RA* $\mathcal{A} = (Q, P, \xi, \delta)$ *which is given by adding the initial states* p_1, p_2 *to the RA in Example 5.1.2. First, assume we choose* $r_0 = (q_0, \phi_0) \to (p_1, \varepsilon) \in \Delta$, $r_2 = (q_0, \phi_0) \to q_1$ *in Line 5. Then,* $\delta_{new} = \delta'_{new} \cup \delta''_{new}$ *where* $\delta'_{new} = \{(p_1, \phi) \to q_1 \mid \phi \in ({\phi_0}^{-1} \circ_{\mathtt{M}} \phi_0) \circ_{\mathtt{R}} \phi_2\}$ *and* $\delta''_{new} = \{(p_1, \phi) \to q_2 \mid \phi \in ({\phi_0}^{-1} \circ_{\mathtt{M}} \phi_0) \circ_{\mathtt{R}} \phi_3\}$. *As shown in Example 5.2.1,* $({\phi_0}^{-1} \circ_{\mathtt{M}} \phi_0) \circ_{\mathtt{R}} \phi_2 = \{\phi_4, \phi_5, \phi_6\} \circ_{\mathtt{R}} \phi_2 = \{\phi_2\}$ *and* $({\phi_0}^{-1} \circ_{\mathtt{M}} \phi_0) \circ_{\mathtt{R}} \phi_3 = \emptyset$. *Thus, we add a rule* $r_5 = (p_1, \phi_2) \to q_1$ *and obtain* $\delta_1 = \delta \cup \{r_5\}$. *In the next iteration, assume we choose* $r_1 = (p_1, \phi_1) \to (p_2, 12) \in \Delta$ *and* $r_5 \in \delta_1$ *in Line 5. Then, we obtain* ${\phi_1}^{-1} \circ_{\mathtt{M}} \phi_2 = \{\phi_7, \phi_8, \phi_9\}$. *Among* ϕ_7, ϕ_8, ϕ_9, *only* ϕ_7 *satisfies* $\mathtt{x}_2 \sim_{\phi_7} \mathtt{top}$. *Thus, we add a rule* $r_6 = (\langle r_1 \rangle, \phi_7) \to q_1$ *and obtain* $\delta_2 = \delta_1 \cup \{r_6\}$. *(Note that the rule* $r_7 = (p_2, \phi_5) \to \langle r_1 \rangle$ *is already added in Line 2.) After this iteration, there is no choice of rules in* Δ *and* δ *that can add transition rules by the algorithm. Hence, we obtain the output k-RA* $\widetilde{\mathcal{A}} = (Q_0, P, \xi_2, \delta_2)$ *(Fig. 5.3). For example,*

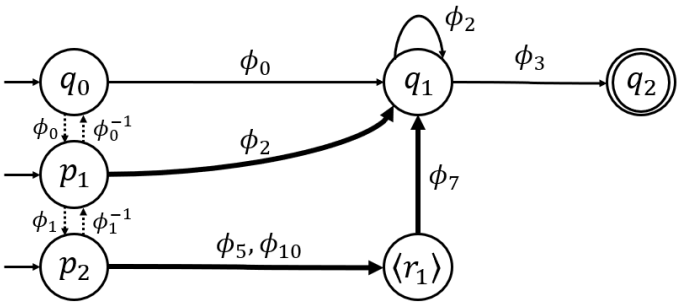

Figure 5.3: Transition diagram of $\widetilde{\mathcal{A}}$ where ϕ_{10} is the equivalence relation that is added in Line 2 and satisfies $X_2/\phi_{10} = \{\{\mathtt{x}_1, \mathtt{x}_2, \mathtt{x}'_1, \mathtt{x}'_2, \mathtt{top}\}\}$. (The transitions of RPDS $\mathcal{P}$ are shown by dotted arrows, and the new transitions added by the algorithm are shown by thick arrows.)

consider an ID $(p_2, [d_3, d_2], d_3d_2d_3)$*, which belongs to* $post^*(L(\mathcal{A}))$ *because* $L(\mathcal{A}) \ni (q_0, [d_0, d_1], d_2d_3d_3) \stackrel{*}{\Rightarrow} (p_2, [d_3, d_2], d_3d_2d_3)$ *by Examples 5.1.1 and 5.1.2. We can see this ID is in* $L(\widetilde{\mathcal{A}})$ *because* $(p_2, [d_3, d_2], d_3d_2d_3) \vdash^{r_7} (\langle r_1 \rangle, [d_3, d_2], d_2d_3) \vdash^{r_6} (q_1, [d_3, d_3], d_3) \vdash^{r_4} (q_2, [d_4, d_4], \varepsilon)$.

Lemma 5.3.1. *For an input* k*-RPDS* $\mathcal{P} = (P, \Delta)$ *and* k*-RA* $\mathcal{A} = (Q, P, \xi, \delta)$ *for* C*, Algorithm 5.2 runs in time polynomial of* $|Q| + |\Delta|$ *and exponential to* k*.*

Proof For every output k-RA $\widetilde{\mathcal{A}} = (Q_0, P, \tilde{\xi}, \tilde{\delta})$, $|\tilde{\delta}| \leq |Q_0| \cdot |\Phi_k| \cdot |Q_0| \leq (|Q| + |\Delta|)^2 \cdot |\Phi_k|$ hold. Thus, the algorithm always halts and every run of the algorithm takes at most $(|Q| + |\Delta|)^2 \cdot |\Phi_k|$ iterations and finishes in polynomial time of $(|Q| + |\Delta|)^2 \cdot |\Phi_k|$. In addition, $|\Phi_k| \leq 2^{(2k+1)k}$ holds because the number of reflexive symmetric binary relations over $2k + 1$ elements is $2^{(2k+1)k}$. (More precisely, the size of Φ_k is known as *Bell number* B_n for $n = 2k + 1$. However, B_{2k+1} is also exponential to k.) Hence, every run finishes in exponential time to k. □

Corollary 5.3.1 (Joinability)**.** *Let* $\mathcal{P}_1 = (P_1, \Delta_1)$, $\mathcal{P}_2 = (P_2, \Delta_2)$, $\mathcal{A}_1 = (Q_1, P_1, \xi_1, \delta_1)$, $\mathcal{A}_2 = (Q_2, P_2, \xi_2, \delta_2)$ *be two* k*-RPDS and two* k*-RAs, respectively. The joinability problem* $post^*_{\mathcal{P}_1}(L(\mathcal{A}_1)) \cap post^*_{\mathcal{P}_2}(L(\mathcal{A}_2)) \neq \emptyset$ *is decidable.*

Proof By Theorem 5.2.1, we can construct two k-RAs that recognize $post^*_{\mathcal{P}_1}(L(\mathcal{A}_1))$ and $post^*_{\mathcal{P}_2}(L(\mathcal{A}_2))$ from $\mathcal{P}_1$, $\mathcal{A}_1$ and $\mathcal{P}_2$, $\mathcal{A}_2$, respectively. The class of languages

recognized by RA is closed under intersection and the emptiness problem for RA is decidable [41]. Hence, the joinability problem is decidable.

5.4 Correctness of the saturation algorithm for $post^*(C)$

In this section, we fix a k-RPDS $\mathcal{P} = (P, \Delta)$ and a k-RA $\mathcal{A} = (Q, P, \xi, \delta)$ given as inputs to Algorithm 5.2.

The following Lemma 5.4.1 establishes one-step soundness for push, replace and pop rules and will be used for proving the multi-step soundness (Lemma 5.4.3).

Lemma 5.4.1. *Let $r = (p, \phi_0) \to (q, J) \in \Delta$ and $r_1 = (p, \phi_1) \to q' \in \delta_h$ be two rules chosen in Line 5 of Algorithm 5.2, and let δ_{new} be the set of rules constructed for r and r_1 in Lines 6–12 of the algorithm. Let $r_{new} \in \delta_{new}$.*

(i) *If $J = j_1 j_2$ and there exist IDs $(q, \theta_1, w_1) \in ID_{\mathcal{P}}$ and $(\langle r \rangle, \theta_1, w_1'), (q', \theta_2, w_2) \in ID_{\mathcal{A}_0}$ that satisfy $(q, \theta_1, w_1) \vdash_{\theta_1(j_1)} (\langle r \rangle, \theta_1, w_1') \vdash^{r_{new}}_{\theta_1(j_2)} (q', \theta_2, w_2)$ in $\mathcal{A}_{h+1}$, then there exists $(p, \theta_0, w_0) \in ID_{\mathcal{P}}$ that satisfies $(q, \theta_1, w_1) \Leftarrow^r_d (p, \theta_0, w_0)$ in $\mathcal{P}$ and $(p, \theta_0, w_0) \vdash^{r_1}_d (q', \theta_2, w_2)$ in $\mathcal{A}_h$ for some $d \in D$.*

(ii) *If $J = j_1$ and there exist IDs $(q, \theta_1, w_1) \in ID_{\mathcal{P}}$ and $(q', \theta_2, w_2) \in ID_{\mathcal{A}_0}$ that satisfy $(q, \theta_1, w_1) \vdash^{r_{new}}_{\theta_1(j_1)} (q', \theta_2, w_2)$ in $\mathcal{A}_{h+1}$, then there exists $(p, \theta_0, w_0) \in ID_{\mathcal{P}}$ that satisfies $(q, \theta_1, w_1) \Leftarrow^r_d (p, \theta_0, w_0)$ in $\mathcal{P}$ and $(p, \theta_0, w_0) \vdash^{r_1}_d (q', \theta_2, w_2)$ in $\mathcal{A}_h$ for some $d \in D$.*

(iii) *If $J = \varepsilon$ and there exist IDs $(q, \theta_1, w_1) \in ID_{\mathcal{P}}$ and $(q'', \theta_3, w_2) \in ID_{\mathcal{A}_0}$ and $d' \in D$ that satisfy $(q, \theta_1, w_1) \vdash^{r_{new}}_{d'} (q'', \theta_3, w_2)$ in $\mathcal{A}_{h+1}$, then there exist (p, θ_0, w_0), $(q', \theta_2, w_1) \in ID_{\mathcal{P}}$ and a rule $r_2 = (q', \phi_2) \to q'' \in \delta_h$ that satisfy $(q, \theta_1, w_1) \Leftarrow^r_d (p, \theta_0, w_0)$ in $\mathcal{P}$ and $(p, \theta_0, w_0) \vdash^{r_1}_d (q', \theta_2, w_1) \vdash^{r_2}_{d'} (q'', \theta_3, w_2)$ in $\mathcal{A}_h$ for some $d \in D$.*

Proof (i) Assume that $r_{new} = (\langle r \rangle, \phi) \to q'$. By the construction of δ_{new}, $\phi \in {\phi_0}^{-1} \circ_{\mathtt{M}} \phi_1$. Since $(\langle r \rangle, \theta_1, w_1') \vdash^{r_{new}}_{\theta_1(j_2)} (q', \theta_2, w_2)$, $\langle \theta_1, \theta_2, \theta_1(j_2) \rangle = \phi$. Thus by Lemmas 5.2.1 and 5.2.2 (ii), there exist $\theta_0 \in \Theta_k$ and $d \in D$ that satisfy $\langle \theta_0, \theta_1, d \rangle = \phi_0$ and $\langle \theta_0, \theta_2, d \rangle = \phi_1$. Hence, we obtain $(q, \theta_1, w_1) \Leftarrow^r_d (p, \theta_0, w_0) \vdash^{r_1}_d (q', \theta_2, w_2)$ for $w_0 = dw_2$.

(ii) Assume that $r_{new} = (q, \phi) \to q'$. By the construction of δ_{new}, $\phi \in {\phi_0}^{-1} \circ_{\mathtt{M}} \phi_1$. Since $(q, \theta_1, w_1) \vdash^{r_{new}}_{\theta_1(j_1)} (q', \theta_2, w_2)$, $\langle \theta_1, \theta_2, \theta_1(j_1) \rangle = \phi$. In the same way as (i), there exist $\theta_0 \in \Theta_k$ and $d \in D$ and we obtain $(q, \theta_1, w_1) \Leftarrow^r_d (p, \theta_0, w_0) \vdash^{r_1}_d (q', \theta_2, w_2)$ for $w_0 = dw_2$.

(iii) Assume that $r_{new} = (q, \phi) \to q''$. By the construction of δ_{new}, $r_2 = (q', \phi_2) \to q'' \in \delta_h$ exists and $\phi \in \phi' \circ_{\mathtt{R}} \phi_2$ (*1) holds for some $\phi' \in {\phi_0}^{-1} \circ_{\mathtt{M}} \phi_1$ (*2). By (*1) and Lemma 5.2.3 (ii), there exist $\theta_2 \in \Theta_k$ and $d'' \in D$ such that $\langle \theta_1, \theta_2, d'' \rangle = \phi'$ (*3) and $\langle \theta_2, \theta_3, d' \rangle = \phi_2$ (*4). By (*2), (*3) and Lemmas 5.2.1 and 5.2.2 (ii), there exist $\theta_0 \in \Theta_k$ and $d \in D$ such that $\langle \theta_0, \theta_1, d \rangle = \phi_0$ (*5) and $\langle \theta_0, \theta_2, d \rangle = \phi_1$ (*6). By (*4), (*5) and (*6), we obtain $(q, \theta_1, w_1) \Leftarrow^r_d (p, \theta_0, w_0) \vdash^{r_1}_d (q', \theta_2, w_1) \vdash^{r_2}_{d'} (q'', \theta_3, w_2)$ for $w_0 = dw_1 = dd'w_2$. □

Before proving Lemma 5.4.3, we prove the following lemma about transitions reaching an ID with a state $\langle r \rangle$ added in Line 2 of Algorithm 5.2. We will also use this lemma for proving Lemma 5.4.3.

Lemma 5.4.2. *For every pair of an ID $(p, \theta_0, w_0) \in ID_{\mathcal{A}}$ and $r = (p_0, \phi_0) \to (q, j_1 j_2) \in \Delta$ that satisfy $(p, \theta_0, w_0) \vdash^*_{\mathcal{A}_h} (\langle r \rangle, \theta_1, w_1)$, $(p, \theta_0, w_0) \overset{*}{\Leftarrow}_{\mathcal{P}} (q, \theta_2, w_2) \vdash_{\mathcal{A}_h} (\langle r \rangle, \theta_1, w_1)$ holds for some $\theta_2 \in \Theta_k$ and $w_2 \in D^*$.*

Proof We show this lemma by the induction on h:

(Basis) If $h = 0$, then every transition rule whose right-hand side is $\langle r \rangle$ must be $(q, \phi) \to \langle r \rangle$ (for some ϕ) constructed in Line 2 of the algorithm. Therefore, the assumption $(p, \theta_0, w_0) \vdash^*_{\mathcal{A}_0} (\langle r \rangle, \theta_1, w_1)$ implies $p = q$, and thus $(q, \theta_0, w_0) \overset{*}{\Leftarrow}_{\mathcal{P}} (q, \theta_0, w_0) \vdash_{\mathcal{A}_0} (\langle r \rangle, \theta_1, w_1)$ holds.

(Induction) Assume $h > 0$ and for each $(p, \theta_0, w_0), (\langle r \rangle, \theta_1, w_1) \in ID_{\mathcal{A}}$ that satisfy $(p, \theta_0, w_0) \vdash^*_{\mathcal{A}_h} (\langle r \rangle, \theta_1, w_1)$, let δ_{new} be the set of rules constructed in the h-th iteration of the algorithm. We assume transition rules in δ_{new} are used m times in total in the transition $(p, \theta_0, w_0) \vdash^*_{\mathcal{A}_h} (\langle r \rangle, \theta_1, w_1)$. We prove the induction step for h by the induction on m. If $m = 0$, $(p, \theta_0, w_0) \vdash^*_{\mathcal{A}_{h-1}} (\langle r \rangle, \theta_1, w_1)$ holds. By the induction hypothesis on h, the induction step for h holds.

If $m > 0$, there exist IDs (s, θ_s, w_s), (t, θ_t, w_t) and $r_{new} \in \delta_{new}$ that satisfy $(p, \theta_0, w_0) \vdash^*_{\mathcal{A}_{h-1}} (s, \theta_s, w_s) \vdash^{r_{new}} (t, \theta_t, w_t) \vdash^*_{\mathcal{A}_h} (\langle r \rangle, \theta_1, w_1)$ and the transitions $(t, \theta_t, w_t) \vdash^*_{\mathcal{A}_h} (\langle r \rangle, \theta_1, w_1)$ use the rules of δ_{new} at most $m - 1$ times.

Assume δ_{new} is constructed from a push rule $r' \in \Delta$ in the h-th iteration of the algorithm. By the construction of δ_{new} in Line 7 of Algorithm 5.2, $s = \langle r' \rangle$ holds. By $(p, \theta_0, w_0) \vdash^*_{\mathcal{A}_{h-1}} (s, \theta_s, w_s)$ and the induction hypothesis on h, $(p, \theta_0, w_0) \overset{*}{\Leftarrow}_{\mathcal{P}} (q', \theta'_0, w'_0) \vdash_{\mathcal{A}_{h-1}} (s, \theta_s, w_s)$ holds (*1) for some $\theta'_0 \in \Theta_k$ and $w'_0 \in D^*$ where q' is the state appearing in the right-hand side of r'. By $(q', \theta'_0, w'_0) \vdash_{\mathcal{A}_{h-1}} (s, \theta_s, w_s) \vdash^{r_{new}} (t, \theta_t, w_t)$ and Lemma 5.4.1 (i), there exists an ID (q'', θ'_1, w'_1) that satisfies $(q', \theta'_0, w'_0) \Leftarrow_{\mathcal{P}} (q'', \theta'_1, w'_1) \vdash_{\mathcal{A}_{h-1}} (t, \theta_t, w_t)$ (*2). By $(q'', \theta'_1, w'_1) \vdash_{\mathcal{A}_{h-1}} (t, \theta_t, w_t) \vdash^*_{\mathcal{A}_h} (\langle r \rangle, \theta_1, w_1)$ and the induction hypothesis on m, $(q'', \theta'_1, w'_1) \overset{*}{\Leftarrow}_{\mathcal{P}} (q, \theta'_2, w'_2) \vdash_{\mathcal{A}_h} (\langle r \rangle, \theta_1, w_1)$ holds for some $\theta'_2 \in \Theta_k$ and $w'_2 \in D^*$ (*3). By (*1), (*2) and (*3), we obtain $(p, \theta_0, w_0) \overset{*}{\Leftarrow}_{\mathcal{P}} (q', \theta'_0, w'_0) \Leftarrow_{\mathcal{P}} (q'', \theta'_1, w'_1) \overset{*}{\Leftarrow}_{\mathcal{P}} (q, \theta'_2, w'_2) \vdash_{\mathcal{A}_h} (\langle r \rangle, \theta_1, w_1)$.

Assume δ_{new} is constructed from a replace rule in the h-th iteration of the algorithm. By the construction of δ_{new}, the state s in the left-hand side of r_{new} must be a state in P. Thus, $(p, \theta_0, w_0) = (s, \theta_s, w_s)$ holds (*4) because an RA has no transitions to an initial state. By $(s, \theta_s, w_s) \vdash^{r_{new}} (t, \theta_t, w_t)$ and Lemma 5.4.1 (ii), there exists an ID (q', θ'_0, w'_0) that satisfies $(s, \theta_s, w_s) \Leftarrow_{\mathcal{P}} (q', \theta'_0, w'_0) \vdash_{\mathcal{A}_{h-1}} (t, \theta_t, w_t)$ (*5). By $(q', \theta'_0, w'_0) \vdash_{\mathcal{A}_{h-1}} (t, \theta_t, w_t) \vdash^*_{\mathcal{A}_h} (\langle r \rangle, \theta_1, w_1)$ and the induction hypothesis on m, $(q', \theta'_0, w'_0) \overset{*}{\Leftarrow}_{\mathcal{P}} (q, \theta'_1, w'_1) \vdash_{\mathcal{A}_h} (\langle r \rangle, \theta_1, w_1)$ holds for some $\theta'_1 \in \Theta_k$ and $w'_1 \in D^*$ (*6). By (*4), (*5) and (*6), we obtain $(p, \theta_0, w_0) = (s, \theta_s, w_s) \Leftarrow_{\mathcal{P}} (q', \theta'_0, w'_0) \overset{*}{\Leftarrow}_{\mathcal{P}} (q, \theta'_1, w'_1) \vdash_{\mathcal{A}_h} (\langle r \rangle, \theta_1, w_1)$.

Assume δ_{new} is constructed from a pop rule in the h-th iteration of the algorithm. As the case of a replace rule, $(p, \theta_0, w_0) = (s, \theta_s, w_s)$ holds. By $(s, \theta_s, w_s) \vdash^{r_{new}} (t, \theta_t, w_t)$ and Lemma 5.4.1 (iii), there exist IDs (q', θ'_0, w'_0) and (q'', θ'_1, w'_1) that satisfies $(s, \theta_s, w_s) \Leftarrow_{\mathcal{P}} (q', \theta'_0, w'_0) \vdash_{\mathcal{A}_{h-1}} (q'', \theta'_1, w'_1) \vdash_{\mathcal{A}_{h-1}} (t, \theta_t, w_t)$. The rest of the proof is similar to that of a replace rule.

By the induction on h, we obtain the lemma. □

Lemma 5.4.3. *For every pair of IDs $(p, \theta_0, w_0), (q, \theta_1, w_1) \in ID_{\mathcal{A}}$ that satisfy $(p, \theta_0, w_0) \vdash^*_{\mathcal{A}_h} (q, \theta_1, w_1)$, there exists an ID (p', θ_2, w_2) such that $(p, \theta_0, w_0) \overset{*}{\Leftarrow}_{\mathcal{P}} (p', \theta_2, w_2) \vdash^*_{\mathcal{A}} (q, \theta_1, w_1)$ holds.*

Proof We show this lemma by the induction on h:
(Basis) If $h = 0$, then for every pair of IDs $(p, \theta_0, w_0), (q, \theta_1, w_1) \in ID_{\mathcal{A}}$ that satisfies $(p, \theta_0, w_0) \vdash^*_{\mathcal{A}_h} (q, \theta_1, w_1)$, we have $(p, \theta_0, w_0) \overset{*}{\Leftarrow}_{\mathcal{P}} (p, \theta_0, w_0) \vdash^*_{\mathcal{A}} (q, \theta_1, w_1)$

because the transition sequence from (p, θ_0, w_0) to (q, θ_1, w_1) cannot use the transition rules added in Line 2 of Algorithm 5.2.
(Induction) We can prove the induction step by induction on the number m of application of rules in δ_{new} constructed in the h-th iteration of the algorithm in a similar way to Lemma 5.4.2. The case that $m = 0$ can be easily proved.

If $m > 0$, there exist IDs (s, θ_s, w_s), (t, θ_t, w_t) and $r_{new} \in \delta_{new}$ that satisfy $(p, \theta_0, w_0) \vdash^*_{\mathcal{A}_{h-1}} (s, \theta_s, w_s) \vdash^{r_{new}} (t, \theta_t, w_t) \vdash^*_{\mathcal{A}_h} (q, \theta_1, w_1)$ and the transitions $(t, \theta_t, w_t) \vdash^*_{\mathcal{A}_h} (q, \theta_1, w_1)$ use the rules of δ_{new} at most $m - 1$ times.

Assume δ_{new} is constructed from a push rule $r' \in \Delta$ in the h-th iteration of the algorithm. By the construction of δ_{new} in Line 7 of Algorithm 5.2, $s = \langle r \rangle$ holds. By $(p, \theta_0, w_0) \vdash^*_{\mathcal{A}_{h-1}} (s, \theta_s, w_s)$ and Lemma 5.4.2, there exists an ID (q', θ'_0, w'_0) such that $(p, \theta_0, w_0) \overset{*}{\Leftarrow}_{\mathcal{P}} (q', \theta'_0, w'_0) \vdash_{\mathcal{A}_{h-1}} (s, \theta_s, w_s)$ holds (*1). By $(q', \theta'_0, w'_0) \vdash_{\mathcal{A}_{h-1}} (s, \theta_s, w_s) \vdash^{r_{new}} (t, \theta_t, w_t)$ and Lemma 5.4.1 (i), there exists an ID (q'', θ'_1, w'_1) that satisfies $(q', \theta'_0, w'_0) \Leftarrow_{\mathcal{P}} (q'', \theta'_1, w'_1) \vdash_{\mathcal{A}_{h-1}} (t, \theta_t, w_t)$ (*2). By $(q'', \theta'_1, w'_1) \vdash_{\mathcal{A}_{h-1}} (t, \theta_t, w_t) \vdash^*_{\mathcal{A}_h} (q, \theta_1, w_1)$ and the induction hypothesis on m, $(q'', \theta'_1, w'_1) \overset{*}{\Leftarrow}_{\mathcal{P}} (p', \theta'_2, w'_2) \vdash^*_{\mathcal{A}_h} (q, \theta_1, w_1)$ holds for some $\theta'_2 \in \Theta_k$ and $w'_2 \in D^*$ (*3). By (*1), (*2) and (*3), we obtain $(p, \theta_0, w_0) \overset{*}{\Leftarrow}_{\mathcal{P}} (q', \theta'_0, w'_0) \Leftarrow_{\mathcal{P}} (q'', \theta'_1, w'_1) \overset{*}{\Leftarrow}_{\mathcal{P}} (p', \theta'_2, w'_2) \vdash^*_{\mathcal{A}_h} (q, \theta_1, w_1)$. The case of δ_{new} is constructed from a replace rule or a pop rule can be proved in a similar way as that of Lemma 5.4.2. By the induction on h, we obtain the lemma. □

For proving the soundness of Algorithm 5.2, we need the following property of accepting conditions added in Line 11 of the algorithm:

Lemma 5.4.4. *If (p, θ, ε) is an accepting ID of $\mathcal{A}_h$ and not an accepting ID of $\mathcal{A}$, then there exist $(p', \theta', d) \in ID_{\mathcal{P}}$ and an accepting ID $(q', \theta_f, \varepsilon)$ of $\mathcal{A}$ that satisfy $(p, \theta, \varepsilon) \Leftarrow_{\mathcal{P}} (p', \theta', d) \vdash_{\mathcal{A}_h} (q', \theta_f, \varepsilon)$.*

Proof By the construction of ξ_h in Line 11 of Algorithm 5.2, there exist two rules $r = (p', \phi_0) \to (p, \varepsilon) \in \Delta$ and $r_1 = (p', \phi_1) \to q' \in \delta_{h-1}$ and $(q', F_{latter}(\phi_1)) \in \xi_{h-1}$ and ${\phi_0}^{-1} \circ_{\mathtt{M}} \phi_1 \neq \emptyset$ hold. Because $\mathcal{A}_{h-1}$ never has a transition rule whose right-hand side is a state in P, $q' \notin P$ holds. Since Algorithm 5.2 adds only an accepting condition that contains a state in P, $(q', F_{latter}(\phi_1))$ is not an accepting condition added by the algorithm and thus $(q', F_{latter}(\phi_1)) \in \xi$. Let $\phi' \in {\phi_0}^{-1} \circ_{\mathtt{M}} \phi_1$

and define $\theta_f \in \Theta_k$ and $d' \in D$ so that $\langle \theta, \theta_f, d' \rangle = \phi'$ holds. By Lemma 5.2.1 and Lemma 5.2.2 (ii), there exist θ' and d that satisfy $\langle \theta', \theta, d \rangle = \phi_0$ and $\langle \theta', \theta_f, d \rangle = \phi_1$. Hence, $(p', \theta', d) \Rightarrow_d^r (p, \theta, \varepsilon)$ in $\mathcal{P}$ and $(p', \theta', d) \vdash_d^{r_1} (q', \theta_f, \varepsilon)$ in $\mathcal{A}_h$. Since $\langle \theta', \theta_f, d \rangle = \phi_1$, $\langle \theta_f \rangle = F_{latter}(\phi_1)$ by definition, and thus $(q', \theta_f, \varepsilon)$ is an accepting ID of $\mathcal{A}$. □

Theorem 5.4.1 (soundness)**.** *For each ID $(p, \theta, w) \in L(\widetilde{\mathcal{A}})$ such that $p \in P$, there exists an ID $(p', \theta', w') \in L(\mathcal{A})$ such that $(p', \theta', w') \Rightarrow_{\mathcal{P}}^* (p, \theta, w)$ holds.*

Proof If $(p, \theta, w) \in L(\widetilde{\mathcal{A}})$, then $(p, \theta, w) \in L(\mathcal{A}_h)$ for some h, and thus $(p, \theta, w) \vdash_{\mathcal{A}_h}^* (q, \theta_1, \varepsilon)$ for some accepting ID $(q, \theta_1, \varepsilon)$ of $\mathcal{A}_h$. There are two cases: $(q, \theta_1, \varepsilon)$ is an accepting ID of $\mathcal{A}$ or not. By Lemma 5.4.3, $(p, \theta, w) \vdash_{\mathcal{A}_h}^* (q, \theta_1, \varepsilon)$ implies that there exists an ID (p', θ', w') such that $(p, \theta, w) \overset{*}{\Leftarrow}_{\mathcal{P}} (p', \theta', w') \vdash_{\mathcal{A}}^* (q, \theta_1, \varepsilon)$. So in the former case, $(p', \theta', w') \in L(\mathcal{A})$, and thus the claim of the theorem holds. In the latter case, we obtain $(p, \theta, w) = (q, \theta_1, \varepsilon)$, because the algorithm adds an accepting condition (q, ϕ) only when $q \in P$ and $\mathcal{A}_h$ never has a transition rule whose right-hand side is a state in P. Applying Lemma 5.4.4 to (p, θ, w), we obtain $(p_2, \theta_2, d) \in ID_{\mathcal{P}}$ and an accepting ID $(q', \theta_f, \varepsilon)$ of $\mathcal{A}$ such that $(p, \theta, w) \Leftarrow_{\mathcal{P}} (p_2, \theta_2, d) \vdash_{\mathcal{A}_h} (q', \theta_f, \varepsilon)$. As in the same way as the former case, by Lemma 5.4.3, there exists (p', θ', w') such that $(p_2, \theta_2, d) \overset{*}{\Leftarrow}_{\mathcal{P}} (p', \theta', w') \vdash_{\mathcal{A}}^* (q', \theta_f, \varepsilon)$. Since $(p_2, \theta_2, d) \Rightarrow_{\mathcal{P}} (p, \theta, w)$, the theorem holds. □

The following lemma establishes one-step completeness for push, replace and pop rules and will be used for proving the completeness of Algorithm 5.2 (Theorem 5.4.2).

Lemma 5.4.5. *Let $r = (p, \phi_0) \to (q, J) \in \Delta$ and $r_1 = (p, \phi_1) \to q' \in \delta_h$ be two rules chosen in Line 5 of Algorithm 5.2, and let δ_{new} be the set of rules constructed for r and r_1 in Lines 6–12 of the algorithm.*

(i) *If r is a push or replace rule and there exist $(p, \theta_0, w_0), (q, \theta_1, w_1) \in ID_{\mathcal{P}}$ and $(q', \theta_2, w_2) \in ID_{\mathcal{A}_0}$ that satisfy $(q, \theta_1, w_1) \Leftarrow_d^r (p, \theta_0, w_0) \vdash_d^{r_1} (q', \theta_2, w_2)$ for some $d \in D$, then $(q, \theta_1, w_1) \vdash^* (q', \theta_2, w_2)$ holds in $\mathcal{A}_{h+1}$.*

(ii) *If r is a pop rule and there exist $(p, \theta_0, w_0), (q, \theta_1, w_1) \in ID_{\mathcal{P}}$ and (q', θ_2, w_1), $(q'', \theta_3, w_2) \in ID_{\mathcal{A}_0}$ that satisfy $(q, \theta_1, w_1) \Leftarrow_d^r (p, \theta_0, w_0) \vdash_d^{r_1} (q', \theta_2, w_1) \vdash_{d'}^{r_2} (q'', \theta_3, w_2)$*

for some $d, d' \in D$ *and some* $r_2 = (q', \phi_2) \to q'' \in \delta_h$, *then* $(q, \theta_1, w_1) \vdash_{d'} (q'', \theta_3, w_2)$ *holds in* $\mathcal{A}_{h+1}$.

Proof (i) If r is a push rule and $J = j_1 j_2$, then w_0, w_1 and w_2 should satisfy $w_0 = dw_2$ and $w_1 = \theta_1(j_1)\theta_1(j_2)w_2$. By the rule $r' = (q, \phi') \to \langle r \rangle$ added in Line 2 of the algorithm, $(q, \theta_1, w_1) = (q, \theta_1, \theta_1(j_1)\theta_1(j_2)w_2) \vdash^{r'}_{\theta_1(j_1)} (\langle r \rangle, \theta_1, \theta_1(j_2)w_2)$ holds (*1). Let ϕ be an equivalence relation that satisfies $\langle \theta_1, \theta_2, \theta_1(j_2) \rangle = \phi$. Then $\phi \in {\phi_0}^{-1} \circ_{\mathtt{M}} \phi_1$ holds by Lemmas 5.2.1 and 5.2.2 (i). Hence, δ_{new} should contain a rule $r_{new} = (\langle r \rangle, \phi) \to q'$ and thus $(\langle r \rangle, \theta_1, \theta_1(j_2)w_2) \vdash^{r_{new}}_{\theta_1(j_2)} (q', \theta_2, w_2)$ holds (*2). By (*1) and (*2), $(q, \theta_1, w_1) \vdash^* (q', \theta_2, w_2)$ holds in $\mathcal{A}_{h+1}$.

If r is a replace rule and $J = j_1$, then w_0, w_1 and w_2 should satisfy $w_0 = dw_2$ and $w_1 = \theta_1(j_1)w_2$. Let ϕ be an equivalence relation that satisfies $\langle \theta_1, \theta_2, \theta_1(j_1) \rangle = \phi$. Then $\phi \in {\phi_0}^{-1} \circ_{\mathtt{M}} \phi_1$ holds by Lemmas 5.2.1 and 5.2.2 (i). Hence, δ_{new} should contain a rule $r_{new} = (q, \phi) \to q'$ and thus $(q, \theta_1, w_1) = (q, \theta_1, \theta_1(j_1)w_2) \vdash^{r_{new}}_{\theta_1(j_1)} (q', \theta_2, w_2)$.

(ii) If r is a pop rule, let ϕ and ϕ' be equivalence relations that satisfy $\langle \theta_1, \theta_2, d \rangle = \phi$ and $\langle \theta_1, \theta_3, d' \rangle = \phi'$. Because $\langle \theta_0, \theta_1, d \rangle = \phi_0$ and $\langle \theta_0, \theta_2, d \rangle = \phi_1$, $\phi \in {\phi_0}^{-1} \circ_{\mathtt{M}} \phi_1$ holds (*3) by Lemmas 5.2.1 and 5.2.2 (i). In addition, because $\langle \theta_1, \theta_2, d \rangle = \phi$ and $\langle \theta_2, \theta_3, d' \rangle = \phi_2$, $\phi' \in \phi \circ_{\mathtt{R}} \phi_2$ holds (*4) by Lemma 5.2.3 (i). By (*3) and (*4), $\phi' \in ({\phi_0}^{-1} \circ_{\mathtt{M}} \phi_1) \circ_{\mathtt{R}} \phi_2$ holds. Hence, δ_{new} should contain a rule $r_{new} = (q, \phi') \to q''$ and thus $(q, \theta_1, w_1) = (q, \theta_1, d'w_2) \vdash^{r_{new}}_{d'} (q'', \theta_3, w_2)$. □

Theorem 5.4.2 (completeness)**.** *For* (p, θ, w), $(p', \theta', w') \in ID_{\mathcal{P}}$, *if* $(p, \theta, w) \in L(\mathcal{A})$ *and* $(p, \theta, w) \Rightarrow^*_{\mathcal{P}} (p', \theta', w')$, *then* $(p', \theta', w') \in L(\widetilde{\mathcal{A}})$.

Proof Assume that $(p, \theta, w) \in L(\mathcal{A})$ and $(p, \theta, w) \Rightarrow^*_{\mathcal{P}} (p', \theta', w')$ and let n be the smallest number of transitions between (p, θ, w) and (p', θ', w') in $\mathcal{P}$. We prove the theorem by the induction on n.

(Basis) If $n = 0$, then $(p, \theta, w) = (p', \theta', w')$. By the definition of $\widetilde{\mathcal{A}}$, it is clear that $L(\mathcal{A}) \subseteq L(\widetilde{\mathcal{A}})$. Therefore, $(p', \theta', w') \in L(\widetilde{\mathcal{A}})$.

(Induction) If $n > 0$, then there exist $(q, \theta_1, w_1) \in ID_{\mathcal{P}}$, a rule $r \in \Delta$ and a data value $d \in D$ that satisfy $(p, \theta, w) \Rightarrow^* (q, \theta_1, w_1) \Rightarrow^r_d (p', \theta', w')$ (*1) and the smallest number of transitions between (p, θ, w) and (q, θ_1, w_1) is $n - 1$. We can assume that $w_1 = du$ for some $u \in D^*$. By the induction hypothesis,

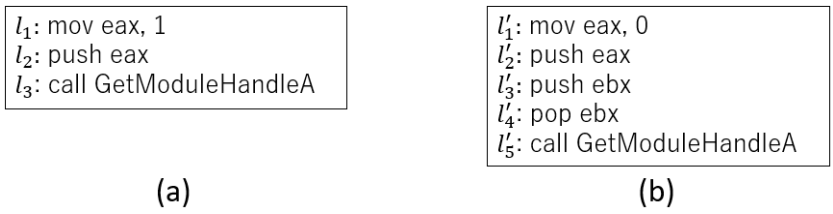

Figure 5.4: **(a)** benign program and **(b)** malware program

$(q, \theta_1, w_1) \in L(\widetilde{\mathcal{A}})$. Since $w_1 \neq \varepsilon$, (q, θ_1, w_1) is not an accepting ID of $\widetilde{\mathcal{A}}$, and thus there exist an ID (q', θ_2, u) and a rule $r_1 = (q, \phi_1) \to q'$ that satisfy $(q, \theta_1, w_1) = (q, \theta_1, du) \vdash_d^{r_1} (q', \theta_2, u) \in L(\widetilde{\mathcal{A}})$ (*2).

If r is a push or replace rule, then by (*1), (*2) and Lemma 5.4.5 (i), there exists a transition sequence $(p', \theta', w') \vdash^* (q', \theta_2, u)$ and thus $(p', \theta', w') \in L(\widetilde{\mathcal{A}})$.

Assume that r is a pop rule ($r = (q, \phi_0) \to (p', \varepsilon)$). In this case, $w' = u$. If $u = \varepsilon$, then (q', θ_2, u) should be an accepting ID of $\widetilde{\mathcal{A}}$, and thus $(q', F_{latter}(\phi_1))$ is an accepting condition of $\widetilde{\mathcal{A}}$. By (*1) and (*2), $\langle \theta_1, \theta', d \rangle = \phi_0$ and $\langle \theta_1, \theta_2, d \rangle = \phi_1$ hold, and thus ${\phi_0}^{-1} \circ_{\mathtt{M}} \phi_1 \neq \emptyset$. By the construction of ξ_{h+1} in Line 11 of Algorithm 5.2, $(p', F_{latter}(\phi_0))$ is also an accepting condition of $\widetilde{\mathcal{A}}$. Since $\langle \theta' \rangle = F_{latter}(\phi_0)$, $(p', \theta', w') = (p', \theta', \varepsilon) \in L(\widetilde{\mathcal{A}})$ holds. If $u \neq \varepsilon$, then (q', θ_2, u) is not an accepting ID of $\widetilde{\mathcal{A}}$, and thus there exist an ID (q'', θ_3, u'), a rule r_2 and a data value $d' \in D$ that satisfy $(q', \theta_2, u) \vdash_{d'}^{r_2} (q'', \theta_3, u') \in L(\widetilde{\mathcal{A}})$ (*3). By (*1), (*2), (*3) and Lemma 5.4.5 (ii), there exists a transition sequence $(p', \theta', w') \vdash^* (q'', \theta_3, u')$, and thus $(p', \theta', w') \in L(\widetilde{\mathcal{A}})$.

By the induction, we obtain $(p', \theta', w') \in L(\widetilde{\mathcal{A}})$ for all $n \geq 0$.

5.5 Application

In this section, we give an example that shows how our algorithm can be applied to practical problems (malware analysis), taken from [67] and simplified.

Let $\mathcal{P}_1$, $\mathcal{P}_2$ be programs as shown in Fig. 5.4 (a), (b), respectively. $\mathcal{P}_1$ and $\mathcal{P}_2$ push a data value assigned to *eax* in l_1 and l'_1, respectively, and call the function *GetModuleHandleA* with reference to the stack top data value as an

argument. Assume calling *GetModuleHandleA* with 0 as its argument allows a calling program to get the information for infecting other files. Then, $\mathcal{P}_1$ is a benign program, but $\mathcal{P}_2$ is considered as a malicious program because $\mathcal{P}_2$ tries to give the unexpected data value 0 to *GetModuleHandleA* as an argument.

Whichever we use traditional signature analysis or pushdown model checking, it is difficult to distinguish the malicious behaviors from the benign ones. Actually, signature analysis does not work if some meaningless instructions (like l'_3 and l'_4 in Fig. 5.4 (b)) are inserted into the code, and pushdown model checking fails to distinguish $\mathcal{P}_1$ and $\mathcal{P}_2$, as discussed in [67], because such analysis assumes data values 0 and 1 are the same. (The example given in [67] assumes the stack contents to be the names of registers, instead of data values in the registers. Therefore, we cannot distinguish benign and malicious programs if the name of register used as an argument of *GetModuleHandleA* is the same as in the case of *eax* appearing in $\mathcal{P}_1$ and $\mathcal{P}_2$.) In [67], a pushdown model checking was extended so that it can deal with a finite number of data values. In our method, we can express the malicious behaviors as $post^*_{\mathcal{P}}(I) \cap Bad \neq \emptyset$ where $\mathcal{P}$ is a program, I is a set of expected initial IDs of $\mathcal{P}$ and Bad is a set of IDs (g_0, θ, du) where g_0 is the entry point of the function *GetModuleHandleA* and d stands for 0 in the original program.

We construct $\mathcal{P}_1$ and $\mathcal{P}_2$ as 3-RPDS whose 1st and 2nd registers are used for keeping the contents of *eax* and *ebx*, respectively, and the data value in 3rd register stands for 0 in the original program. Let $\mathcal{P}_1 = (P_1, \Delta_1)$ be 3-RPDS where $P_1 = \{l_1, l_2, l_3, l_{end}\}$ and $\Delta_1 = \{(l_1, \phi_1) \to (l_2, 2),\ (l_2, \phi_2) \to (l_3, 12),\ (l_3, \phi_3) \to (l_{end}, \varepsilon)\}$ for $X_3/\phi_1 = \{\{\mathtt{x}_1\}, \{\mathtt{x}'_1\}, \{\mathtt{x}_2, \mathtt{x}'_2\}, \{\mathtt{x}_3, \mathtt{x}'_3\}, \{\mathtt{top}\}\}$, $X_3/\phi_2 = \{\{\mathtt{x}_1, \mathtt{x}'_1\}, \{\mathtt{x}_2, \mathtt{x}'_2, \mathtt{top}\}, \{\mathtt{x}_3, \mathtt{x}'_3\}\}$ and $X_3/\phi_3 = \{\{\mathtt{x}_1, \mathtt{x}'_1, \mathtt{top}\}, \{\mathtt{x}_2, \mathtt{x}'_2\}, \{\mathtt{x}_3, \mathtt{x}'_3\}\}$. As defined above, $\mathcal{P}_1$ behaves as Fig. 5.4 (a) by using its registers and stack to restore adequate data value. (Note that $\mathcal{P}_1$ never pushes the data value $\theta(3)$ to the stack.) We can define 3-RPDS $\mathcal{P}_2 = (P_2, \Delta_2)$ in a similar way. The transition $(l'_1, \phi'_1) \to (l'_2, 2) \in \Delta_2$ should save the data value $\theta(3)$ to 1st register, and thus we let $X_3/\phi'_1 = \{\{\mathtt{x}_1\}, \{\mathtt{x}'_1, \mathtt{x}_3, \mathtt{x}'_3\}, \{\mathtt{x}_2, \mathtt{x}'_2\}, \{\mathtt{top}\}\}$. Assume $I_1 = \{(l_1, \theta, u) \mid \theta \in \Theta_3, u \in D^*\}$, $I_2 = \{(l'_1, \theta, u) \mid \theta \in \Theta_3,\ u \in D^*\}$, $Bad_1 = \{(l_3, \theta, du) \mid \theta \in \Theta_3,\ \theta(3) = d \in D,\ u \in D^*\}$ and $Bad_2 = \{(l'_5, \theta, du) \mid \theta \in \Theta_3,\ \theta(3) = d \in D,\ u \in D^*\}$.

Then, $post^*_{\mathcal{P}_1}(I_1) \cap Bad_1 = \emptyset$ because $\mathcal{P}_1$ cannot push the data value $\theta(3)$ to the stack in l_2. On the other hand, $post^*_{\mathcal{P}_2}(I_2) \cap Bad_2 \neq \emptyset$ because $\mathcal{P}_2$ would push the data value $\theta(3)$ to the stack.

While the above example needs only a finite number of data values, RPDS could serve as a formal model in other problems such as the verification of a security protocol that uses nonces from an infinite set and their equality checks.

Alg. 5.2: Saturation algorithm for a given k-RPDS and a k-RA

Input: k-RPDS $\mathcal{P} = (P, \Delta)$ and k-RA $\mathcal{A} = (Q, P, \xi, \delta)$ for C

Let k-RA $\mathcal{A}_0 = (Q_0, P, \xi_0, \delta_0)$ where

$Q_0 = Q \cup \{\langle r \rangle \mid r = (p, \phi_0) \to (q, j_1 j_2) \in \Delta\}$, $\xi_0 = \xi$,

$\delta_0 = \delta \cup \{(q, \phi) \to \langle r \rangle \mid r = (p, \phi_0) \to (q, j_1 j_2) \in \Delta,$
$\forall s.\, \mathtt{x}_s \sim_\phi \mathtt{x}'_s,\ \mathtt{top} \sim_\phi \mathtt{x}_{j_1}\}$.

$h := 0$

repeat

  Choose a rule $r = (p, \phi_0) \to (q, J) \in \Delta$ of $\mathcal{P}$
  and a rule $(p, \phi_1) \to q' \in \delta_h$ of $\mathcal{A}_h$.

  if $J = j_1 j_2$ for some $j_1, j_2 \in [k]$ **then**

    Let $\xi_{h+1} = \xi_h$ and
    $\delta_{new} = \{(\langle r \rangle, \phi) \to q' \mid \phi \in {\phi_0}^{-1} \circ_{\mathtt{M}} \phi_1,\ \mathtt{x}_{j_2} \sim_\phi \mathtt{top}\}$.

  else if $J = j_1$ for some $j_1 \in [k]$ **then**

    Let $\xi_{h+1} = \xi_h$ and
    $\delta_{new} = \{(q, \phi) \to q' \mid \phi \in {\phi_0}^{-1} \circ_{\mathtt{M}} \phi_1,\ \mathtt{x}_{j_1} \sim_\phi \mathtt{top}\}$.

  else if $J = \varepsilon$ **then**

    Let $\xi_{h+1} = \xi_h \cup \{(q, F_{latter}(\phi_0))\}$ if $(q', F_{latter}(\phi_1)) \in \xi_h$ and
    ${\phi_0}^{-1} \circ_{\mathtt{M}} \phi_1 \neq \emptyset$, and let $\xi_{h+1} = \xi_h$ otherwise.
    Also, let $\delta_{new} = \{(q, \phi) \to q'' \mid (q', \phi_2) \to q'' \in \delta_h$ and
    $\phi \in ({\phi_0}^{-1} \circ_{\mathtt{M}} \phi_1) \circ_{\mathtt{R}} \phi_2$ for some $\phi_2 \in \Phi_k\}$.

  end if

  Let k-RA $\mathcal{A}_{h+1} = (Q_0, P, \xi_{h+1}, \delta_{h+1})$, where $\delta_{h+1} = \delta_h \cup \delta_{new}$.

  $h := h + 1$.

until no more transitions and accepting conditions can be added.

Output: $\widetilde{\mathcal{A}} := \mathcal{A}_h$

Chapter 6 LTL Model Checking for RPDS

6.1 Definitions

6.1.1 Infinite sequence

For a set A, let $\mathscr{P}(A)$ be the power set of A, let A^* and A^ω be the sets of finite and infinite words over A, respectively. We denote $A^+ = A^* \setminus \{\varepsilon\}$ and $A^\infty = A^* \cup A^\omega$. For a word $\alpha \in A^\infty$ over a set A, let $\alpha(i) \in A$ be the i-th element of α ($i \geq 0$), $\alpha(i:j) = \alpha(i)\alpha(i+1)\cdots\alpha(j-1)\alpha(j)$ for $i \leq j$ and $\alpha(i:) = \alpha(i)\cdots$ for $i \geq 0$. We write $\exists^\omega i.P(i)$ if there exist infinitely many i that satisfy the condition $P(i)$.s

6.1.2 Linear temporal logic (LTL)

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

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

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

$$\begin{aligned}
&w \models \mathrm{tt} \\
&w \models A && \iff A \in w(0) \\
&w \models \neg\varphi && \iff w \not\models \varphi \\
&w \models \varphi_1 \wedge \varphi_2 && \iff w \models \varphi_1 \text{ and } w \models \varphi_2
\end{aligned}$$

$$
\begin{aligned}
w \models \mathcal{X}\varphi \quad &\iff w(1)w(2)\cdots \models \varphi \\
w \models \varphi_1 \,\mathcal{U}\, \varphi_2 &\iff \text{there exists } j \in \mathbb{N}_0 \text{ such that} \\
&\quad\; w(j)w(j+1)\cdots \models \varphi_2 \text{ and} \\
&\quad\; w(i)w(i+1)\cdots \models \varphi_1 \text{ for all } 0 \leq i < j
\end{aligned}
$$

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

6.1.3 Backward-deterministic RA

As a related class of RA, we define backward-deterministic RA. We use this RA for giving a definition of regular valuation described later.

Definition 6.1.1. *A* backward-deterministic *k-RA $\mathcal{A} = (Q, I, (q_f, \theta_f), \delta)$ is defined as an RA whose accepting condition is given as a pair $(q_f, \theta_f) \in Q \times \Theta_k$ and every pair of ID of the form $(q, \theta, \varepsilon) \in Q \times \Theta_k \times \{\varepsilon\}$ and $d \in D$ determine a unique predecessor ID $(q', \theta', d) \in Q \times \Theta_k \times D$ such that $(q', \theta', d) \vdash (q, \theta, \varepsilon)$.*

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

6.2 Büchi Register Pushdown Systems (BRPDS) and its Properties

6.2.1 Properties of RPDS and RA

Consider the relation $\sim$ over IDs defined by $(p_1, \theta_1, u_1) \sim (p_2, \theta_2, u_2)$ iff $p_1 = p_2$ and $\theta_1, u_1 \ = \theta_2, u_2$. The following lemma states that $\sim$ is a bisimulation that preserves the number of transition steps, which will be used for proving Proposition 6.2.2. This lemma also applies to RA and Büchi RPDS, which will be described later.

Lemma 6.2.1. *For a k-RPDS $\mathcal{P} = (P, \Delta)$, IDs (p, θ_0, u_0), (p, θ_1, u_1) of $\mathcal{P}$ such that $\theta_0, u_0 = \theta_1, u_1$, and $q \in P$, if there exists an ID $(q, \theta_2, u_2) \in ID_{\mathcal{P}}$ that satisfies*

$(p, \theta_0, u_0) \Rightarrow^n (q, \theta_2, u_2)$ for some $n \in \mathbb{N}_0$, then there exists an ID $(q, \theta_3, u_3) \in ID_{\mathcal{P}}$ that satisfies $(p, \theta_1, u_1) \Rightarrow^n (q, \theta_3, u_3)$ and $\theta_2, u_2 = \theta_3, u_3$.

Proof. We prove the lemma by the induction on n. The case of $n = 0$ is obvious. Assume that $n > 0$. Then the transition sequence must be $(p, \theta_0, u_0) \Rightarrow^{n-1} (q', \theta_4, u_4) \Rightarrow_d^r (q, \theta_2, u_2)$ for some $(q', \theta_4, u_4) \in ID_{\mathcal{P}}$, $r = (q', \phi) \to (q, J) \in \Delta$ and $d \in D$, and $u_4 = du_4'$ for some $u_4' \in D^*$ and $\theta_4, \theta_2, d = \phi$. By the induction hypothesis, there is an ID $(q', \theta_5, u_5) \in ID_{\mathcal{P}}$ such that $(p, \theta_1, u_1) \Rightarrow^{n-1} (q', \theta_5, u_5)$ and $\theta_4, u_4 = \theta_5, u_5$. Since $\theta_4, u_4 = \theta_5, u_5$ implies $|u_4| = |u_5|$ and since $u_4 = du_4'$, it holds that $u_5 = d'u_5'$ for some $d' \in D$ and $u_5' \in D^*$. By Lemma 5.1.1 (i), $\theta_4, du_4' = \theta_5, d'u_5'$ implies that there exists $\theta_3 \in \Theta_k$ such that $\theta_4, \theta_2, du_4' = \theta_5, \theta_3, d'u_5'$. Hence, $\theta_4, \theta_2, d = \theta_5, \theta_3, d' = \phi$ by Lemma 5.1.1 (ii), and thus $(q', \theta_5, u_5) \Rightarrow_{d'}^r (q, \theta_3, u_3)$ where $u_3 = \theta_3(j_1)\theta_3(j_2)u_5'$ if $J = j_1j_2$, $u_3 = \theta_3(j_1)u_5'$ if $J = j_1$, and $u_3 = u_5'$ if $J = \varepsilon$. Note that $\theta_4, \theta_2, du_4' = \theta_5, \theta_3, d'u_5'$ also implies $\theta_2, u_4' = \theta_3, u_5'$ by Lemma 5.1.1 (ii). If $J = j_1j_2$, then $u_2 = \theta_2(j_1)\theta_2(j_2)u_4'$, and thus $\theta_2, u_2 = \theta_3, u_3$ holds because $\theta_2, u_4' = \theta_3, u_5'$ implies $\theta_2, \theta_2(j_1)\theta_2(j_2)u_4' = \theta_3, \theta_3(j_1)\theta_3(j_2)u_5'$ by applying Lemma 5.1.1 (iii) twice. In the same way, we can show $\theta_2, u_2 = \theta_3, u_3$ in the cases of $J = j_1$ and $J = \varepsilon$.

The following lemma is a property of RA similar to, but stronger than the property of RPDS stated in the previous lemma in the sense that any run of a k-RA can be simulated by a run that uses only $2k$ fresh data values that do not appear in the input data word of the original run. This lemma will be used for proving Proposition 6.2.1 described below.

Lemma 6.2.2. *Let $\mathcal{A} = (Q, I, \xi, \delta)$ be a k-RA and $(p_0, \theta_0, w_0) \vdash (p_1, \theta_1, w_1) \vdash \cdots \vdash (p_n, \theta_n, w_n)$ be a run of $\mathcal{A}$, where $w_0 = d_1d_2 \cdots d_m$. Let $D_{w_0} = \{d_1, \ldots, d_m\}$ and let $D' \subseteq D$ be a finite subset of D such that $D_{w_0} \subseteq D'$ and $|D' \setminus D_{w_0}| \geq 2k$. Let $\Theta_k' \subseteq \Theta_k$ be the set of assignments whose range is D', and let $\theta_0' \in \Theta_k'$ be an assignment such that $\theta_0, w_0 = \theta_0', w_0$. Then, there exist $\theta_1', \ldots, \theta_n' \in \Theta_k'$ such that $(p_0, \theta_0', w_0) \vdash (p_1, \theta_1', w_1) \vdash \cdots \vdash (p_n, \theta_n', w_n)$ and $\theta_j, w_j = \theta_j', w_j$ for each $j \in [n]$.*

Proof. We prove the lemma by the induction on n. The case of $n = 0$ is obvious. Assume that $n > 0$. Then, $(p_0, \theta_0, w_0) \vdash_{d_1}^r (p_1, \theta_1, w_1)$ for some $r = (p_0, \phi) \to p_1 \in \delta$ and $\theta_0, \theta_1, d_1 = \phi$. There exists $\theta_1' \in \Theta_k'$ that satisfies $\theta_0, \theta_1, w_0 =$

$\theta'_0, \theta'_1, w_0$, because $\theta_0, w_0 = \theta'_0, w_0$ implies $\theta_0, \theta_1, w_0 = \theta'_0, \theta', w_0$ for some $\theta' \in \Theta_k$ by Lemma 5.1.1 (i), and we can replace every $\theta'(i) \notin D'$ with an element in $D' \setminus (D_{w_0} \cup \{\theta'_0(1), \ldots, \theta'_0(k)\})$ without changing the equivalence class.

By Lemma 5.1.1 (ii), $\theta_0, \theta_1, w_0 = \theta'_0, \theta'_1, w_0$ implies $\theta_0, \theta_1, d_1 = \theta'_0, \theta'_1, d_1 = \phi$ and $\theta_1, w_1 = \theta'_1, w_1$, and the former implies $(p_0, \theta'_0, w_0) \vdash^r_{d_1} (p_1, \theta'_1, w_1)$. By the induction hypothesis, the lemma holds.

The following proposition is essentially the same as [41, Proposition 1], and used for showing decidability of the membership problem of an RA.

Proposition 6.2.1. *For a k-RA $\mathcal{A} = (Q, I, \xi, \delta)$, a finite subset D' of D, $p \in I$, and $\theta \in \Theta_k$ such that $\theta(i) \in D'$ for $i \in [k]$, we can construct a nondeterministic finite automaton (NFA) $\mathcal{A}^p_\theta$ over D' such that for any $w \in (D')^*$, $w \in L(\mathcal{A}^p_\theta)$ iff $(p, \theta, w) \in L(\mathcal{A})$. Moreover, the number of states of $\mathcal{A}^p_\theta$ is at most $|Q| \cdot (2k + |D'|)^k$.*

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

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

By the construction, if $((p, \theta), w) \vdash^*_{\mathcal{A}^p_\theta} ((q, \theta'), w')$, then $(p, \theta, w) \vdash^*_{\mathcal{A}} (q, \theta', w')$, obviously. By Lemma 6.2.2, if $(p, \theta, w) \vdash^*_{\mathcal{A}} (q, \theta', w')$ for $w \in (D')^*$, then $((p, \theta), w) \vdash^*_{\mathcal{A}^p_\theta} ((q, \theta''), w')$ for some $\theta'' \in \Theta'_k$ such that $\theta' = \theta''$. Thus for any $w \in (D')^*$, $w \in L(\mathcal{A}^p_\theta)$ iff $(p, \theta, w) \in L(\mathcal{A})$. The number of states of $\mathcal{A}^p_\theta$ is $|Q| \cdot |D''|^k = |Q| \cdot (2k + |D'|)^k$.

6.2.2 Definition of BRPDS

We define an extended model of RPDS called Büchi register pushdown systems (abbreviated as BRPDS) by adding a notion of Büchi acceptance. In the case of

LTL model checking for PDS described in [30], Büchi PDS is defined and used in the algorithm for model checking. Similarly, BRPDS will be used in the proposed model checking algorithm.

Definition 6.2.1. *A k-BRPDS $\mathcal{BP} = (P, \Delta, G)$ is defined as a k-RPDS accompanied with the acceptance condition given by a set of* repeated states *G ($\subseteq P$). The set $ID_{\mathcal{BP}}$ of all IDs of $\mathcal{BP}$ and the transition relation $\Rightarrow_{\mathcal{BP}}$ of $\mathcal{BP}$ are defined in the same way as those of k-RPDS. An* accepting run *of $\mathcal{BP}$ is a run $\rho \in (ID_{\mathcal{BP}})^\omega$ of $\mathcal{BP}$ that satisfies $\exists^\omega i.\ \rho(i) \in G \times \Theta_k \times D^*$. We say that $c \in ID_{\mathcal{BP}}$ is accepted by $\mathcal{BP}$ if there exists an accepting run ρ such that $\rho(0) = c$.*

The following proposition describes a necessary and sufficient condition for an ID to be accepted by a BRPDS. This proposition is a natural but non-trivial extension of [13, Proposition 3.1] to BRPDS.

Proposition 6.2.2. *For a k-BRPDS $\mathcal{BP} = (P, \Delta, G)$ and an ID (q_0, θ, w) of $\mathcal{BP}$, (q_0, θ, w) is accepted by $\mathcal{BP}$ if and only if there exist IDs (p, θ_0, d_0), (g, θ_0', u_0), $(p, \theta_1, d_1 v_1) \in ID_{\mathcal{BP}}$ where $g \in G$ and $d_0, d_1 \in D$ and $u_0, v_1 \in D^*$ such that the following conditions hold:*

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

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

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

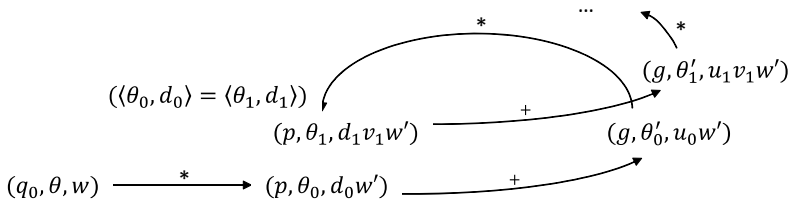

Figure 6.1: An accepting run

Proof. In this proof, for an ID (p, θ, du) of $\mathcal{BP}$ where $d \in D$ and $u \in D^*$, we define the *type* of (p, θ, du) as $(p, \theta, d\ )$.

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

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

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

(If part) Since $\theta_0, d_0 = \theta_1, d_1$ by condition (3), by applying Lemma 6.2.1 to condition (2), there exist $\theta_1', \theta_2 \in \Theta_k$, $u_1, v_2 \in D^*$ and $d_2 \in D$ such that $(p, \theta_1, d_1) \Rightarrow^+ (g, \theta_1', u_1) \Rightarrow^* (p, \theta_2, d_2 v_2)$ and $\theta_0', u_0 = \theta_1', u_1$ and $\theta_1, d_1 v_1 = \theta_2, d_2 v_2$. By Lemma 5.1.1 (ii), $\theta_1, d_1 v_1 = \theta_2, d_2 v_2$ implies $\theta_1, d_1 = \theta_2, d_2$. Repeating this selection of IDs, we obtain $\theta_i', \theta_{i+1} \in \Theta_k$, $u_i, v_{i+1} \in D^*$ and $d_{i+1} \in D$ for $i \geq 1$ such that $(p, \theta_i, d_i) \Rightarrow^+ (g, \theta_i', u_i) \Rightarrow^* (p, \theta_{i+1}, d_{i+1} v_{i+1})$. Concatenating these transition sequences, we obtain a run $(q_0, \theta, w) \Rightarrow^* (p, \theta_0, d_0 w') \Rightarrow^+ (g, \theta_0', u_0 w') \Rightarrow^* (p, \theta_1, d_1 v_1 w') \Rightarrow^+ (g, \theta_1', u_1 v_1 w') \Rightarrow^* (p, \theta_2, d_2 v_2 v_1 w') \Rightarrow^+ \cdots$, where g appears infinitely many times. Therefore this run is an accepting run of

$\mathcal{BP}$, and thus $\mathcal{BP}$ accepts (q_0, θ, w).

6.2.3 Acceptance checking for BRPDS

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

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

Let $pre^*_{\mathcal{P}}$ and $pre^+_{\mathcal{P}}$ be the reflexive transitive closure and the transitive closure of $pre_{\mathcal{P}}$ (as a relation), respectively. It is known that $pre^*_{\mathcal{P}}$ and $pre^+_{\mathcal{P}}$ effectively preserve regularity[50]; that is, if $C \subseteq ID_{\mathcal{P}}$ is regular (i.e. there is an RA $\mathcal{A}$ such that $C = L(\mathcal{A})$), then we can construct an RA $\widetilde{\mathcal{A}}$ such that $pre^*_{\mathcal{P}}(C) = L(\widetilde{\mathcal{A}})$ (or $pre^+_{\mathcal{P}}(C) = L(\widetilde{\mathcal{A}})$).

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

Using pre^*, pre^+, L^p_ϕ and $L^p_{\phi,1}$, Proposition 6.2.2 can be rephrased as follows, which is a BRPDS version of (1') and (2') in [13, Proposition 3.1]: k-BRPDS $\mathcal{BP} = (P, \Delta, G)$ accepts (q_0, θ, w) iff there exist $p \in P$ and $\phi \in \Phi_{k+1}$ (which is θ_0, d_0 for θ_0 and d_0 in Proposition 6.2.2) that satisfy the followings:

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

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

Obviously, (2') is equivalent to $pre^+_{\mathcal{BP}}((G \times \Theta_k \times D^*) \cap pre^*_{\mathcal{BP}}(L^p_\phi)) \cap L^p_{\phi,1} \neq \emptyset$. Note that although Proposition 6.2.2 requires that (p, θ_0, d_0) mentioned in (2') should satisfy $(q_0, \theta, w) \Rightarrow^* (p, \theta_0, d_0 w')$ for some $w' \in D^*$, we do not need to directly test it, because by Lemma 6.2.1, if $(p, \theta_0, d_0) \Rightarrow^+ (g, \theta'_0, u_0) \Rightarrow^* (p, \theta_1, d_1 v_1)$ for some $(g, \theta'_0, u_0) \in G \times \Theta_k \times D^*$ and $(p, \theta_1, d_1 v_1) \in L^p_\phi$, then for *every* $(p, \theta_2, d_2) \in L^p_{\phi,1}$, there exist $(g, \theta'_2, u_2) \in G \times \Theta_k \times D^*$ and $(p, \theta_3, d_3 v_3) \in L^p_\phi$ such that $(p, \theta_2, d_2) \Rightarrow^+ (g, \theta'_2, u_2) \Rightarrow^* (p, \theta_3, d_3 v_3)$.

It is known that the class of languages recognized by RA is closed under intersection, and the membership problem for RA is decidable[41]. Hence, conditions

(1′) and (2′) are decidable. Because both P and Φ_{k+1} are finite, we can decide whether there exist p and ϕ that satisfy (1′) and (2′). We can conclude the above discussion as follows.

Proposition 6.2.3. *For a k-BRPDS $\mathcal{BP} = (P, \Delta, G)$ and an ID (q_0, θ, w) of $\mathcal{BP}$, it is decidable whether (q_0, θ, w) is accepted by $\mathcal{BP}$ in $O((|\Delta| + |w|^{O(k)}) \cdot |P|^{O(1)} \cdot 2^{O(k^2)})$ time.*

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

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

From $\mathcal{A}^p_\phi$, we construct RAs that recognize $pre^*_{\mathcal{BP}}(L^p_\phi)$ and $pre^+_{\mathcal{BP}}((G \times \Theta_k \times D^*) \cap pre^*_{\mathcal{BP}}(L^p_\phi))$, respectively. The algorithm given in [50] constructs an RA $\widetilde{\mathcal{A}}$ such that $L(\widetilde{\mathcal{A}}) = pre^*_{\mathcal{BP}}(L(\mathcal{A}))$ for a given RA $\mathcal{A}$ whose set of initial states equals the set P of states of $\mathcal{BP}$. This construction does not increase the number of states of RA, and the construction takes $O(|\Delta| \cdot |Q|^{O(1)} \cdot |\Phi_{2k+1}|^{O(1)}) = O(|\Delta| \cdot |Q|^{O(1)} \cdot 2^{O(k^2)})$ time, where Q is the set of states of $\mathcal{A}$. An RA recognizing $pre^+_{\mathcal{BP}}(L(\mathcal{A}))$ can be constructed in a similar way. An RA $\mathcal{A}'$ such that $L(\mathcal{A}') = (G \times \Theta_k \times D^*) \cap L(\mathcal{A})$ for a given $\mathcal{A}$ (whose set of initial states equals P) can be obtained simply by removing all transition rules whose left-hand side contains a state in $P \setminus G$. (Note that by the definition of RA, every state in the set P of initial states does not appear in the right-hand side of any transition rule.) Hence, the construction of RAs that recognize $pre^*_{\mathcal{BP}}(L^p_\phi)$ and $pre^+_{\mathcal{BP}}((G \times \Theta_k \times D^*) \cap pre^*_{\mathcal{BP}}(L^p_\phi))$, respectively, can be performed in $O(|\Delta| \cdot |P|^{O(1)} \cdot 2^{O(k^2)})$ time. The set of states of each obtained RA is the same as $\mathcal{A}^p_\phi$.

In the final step, we check the membership $(q_0, \theta, w) \in pre^*_{\mathcal{BP}}(L^p_\phi)$ for given (q_0, θ, w) and $(p, \theta_0, d_0) \in pre^+_{\mathcal{BP}}((G \times \Theta_k \times D^*) \cap pre^*(L^p_\phi))$ for an arbitrary ID $(p, \theta_0, d_0) \in L^p_{\phi,1}$. The latter problem, which is the membership problem of (p, θ_0, d_0) for a k-RA $\mathcal{A} = (P \cup \{q_f\}, P, \xi, \delta)$, can be reduced to a problem to find a transition rule $(p, \phi') \to q \in \delta$ such that $F_{former}(\phi') = \phi$ and $(q, F'_{former}(\phi')) \in \xi$.

This can be performed in $O(|\delta|\cdot|\xi|) = O(|P|^3\cdot|\Phi_{2k+1}|\cdot|\Phi_k|) = O(|P|^3\cdot 2^{O(k^2)})$ time. To check $(q_0,\theta,w) \in pre^*_{\mathcal{BP}}(L^p_\phi)$, we let D' be the finite set that consists of $\theta(i)$ for $i \in [k]$ and every element of w, and then we construct an NFA $\mathcal{A}^{q_0}_\theta$ over D' that satisfies $w \in L(\mathcal{A}^{q_0}_\theta)$ iff $(q_0,\theta,w) \in pre^*_{\mathcal{BP}}(L^p_\phi)$, according to Proposition 6.2.1. Whether $w \in L(\mathcal{A})$ for an NFA $\mathcal{A}$ over Σ can be decided in $O(|w|\cdot|Q|\cdot|\delta|) = O(|w|\cdot|Q|^3\cdot|\Sigma|)$ time where Q and δ are the sets of states and transition rules of $\mathcal{A}$, respectively. In the case of $\mathcal{A}^{q_0}_\theta$, the number of states is at most $(|P|+1)(2k+|D'|)^k$, and thus this step can be performed in $O(|w|\cdot|P|^3\cdot(2k+|D'|)^{O(k)}\cdot|D'|) = O(|w|\cdot|P|^3\cdot(3k+|w|)^{O(k)}) = O(|w|^{O(k)}\cdot|P|^3\cdot 2^{O(k\log k)})$ time.

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

6.3 LTL Model Checking Problem and Valuations

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

A *valuation* for a k-RPDS $\mathcal{P} = (P,\Delta)$ is $\Lambda : ID_\mathcal{P} \to \Sigma$, which labels each ID of $\mathcal{P}$ with atomic propositions. We extend the domain of Λ to $(ID_\mathcal{P})^\infty$ pointwise; that is, $\Lambda(c_1c_2\cdots) = \Lambda(c_1)\Lambda(c_2)\cdots$ for $c_1,c_2,\ldots \in ID_\mathcal{P}$. We define the model checking problem as follows.

Definition 6.3.1.
Input: A k-RPDS $\mathcal{P} = (P,\Delta)$, an LTL formula φ over At, a valuation $\Lambda : ID_\mathcal{P} \to \Sigma$ and an ID $c \in ID_\mathcal{P}$.
Model checking problem: Check whether $\Lambda(\rho) \models \varphi$ for all runs $\rho \in (ID_\mathcal{P})^\omega$ of $\mathcal{P}$ such that $\rho(0) = c$.

For general valuations, LTL model checking problem is undecidable even for PDS [30]. Therefore, we focus on two restricted cases, *simple valuations* (Definition 6.3.2) and *regular valuations* (Definition 6.3.3), which are reasonable extensions of simple and regular valuations for finite-alphabet models [30].

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

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

Definition 6.3.2. *For a k-RPDS $\mathcal{P} = (P, \Delta)$, a subset $C \subseteq ID_{\mathcal{P}}$ is* simple *if $C = L^p_\phi$ for some $p \in P$ and $\phi \in \Phi_{k+1}$. A valuation $\Lambda : ID_{\mathcal{P}} \to \Sigma$ is* simple *if $\Lambda^{-1}(\sigma)$ is a finite union of simple sets of $ID_{\mathcal{P}}$ for every $\sigma \in \Sigma$.*

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

Definition 6.3.3. *For a k-RPDS $\mathcal{P} = (P, \Delta)$, a valuation $\Lambda : ID_{\mathcal{P}} \to \Sigma$ is* backward-deterministic regular *if the set $\{c \in ID_{\mathcal{P}} \mid A \in \Lambda(c)\}$ is a backward-deterministic regular set of $ID_{\mathcal{P}}$ for every $A \in At$.*

In this paper, we call such Λ a *regular valuation*. The definitions of simple and regular valuations are extensions of those of [41] and [30], respectively. We define a regular valuation for RPDS based on backward-deterministic regular sets of $ID_{\mathcal{P}}$, which corresponds to the regular valuation for PDS in [30] based on deterministic finite automata that decide the acceptance of the reverse of stack contents.

6.4 Examples

Example 6.4.1 (malware detection)**.** *This example is taken from [67]. Let $\mathcal{P}_{1a}$ and $\mathcal{P}_{1b}$ be the programs shown in Fig. 6.2 (a) and (b), respectively. $\mathcal{P}_{1a}$ and $\mathcal{P}_{1b}$ push a data value assigned to eax in l_1 and l'_1, respectively, and call the function* GetModuleHandleA *with reference to the stack top data value as an argument. Assume that if a program calls* GetModuleHandleA *with* 0 *as an (unexpected) argument, the latter returns a security sensitive information z to the register eax. If the program calls* UseA *with this z as an argument, the program accomplishes*

its attack. Hence, $\mathcal{P}_{1a}$ is a benign program, but $\mathcal{P}_{1b}$ is considered as a malicious program because $\mathcal{P}_{1b}$ tries to give 0 *to* GetModuleHandleA *as an argument and later $\mathcal{P}_{1b}$ pushes the contents of eax to the stack and calls* UseA.

Whichever we use traditional signature analysis or traditional PDS model checking, it is difficult to distinguish the malicious behaviors from the benign ones. Actually, signature analysis does not work if some meaningless instructions (like l'_3 and l'_4 in Fig. 6.2 (b)) are inserted into the code, and PDS model checking fails to distinguish $\mathcal{P}_{1a}$ and $\mathcal{P}_{1b}$, as discussed in [67], because the analysis assumes data values 0 *and* 1 *are the same. (The example given in [67] assumes the stack contents to be the names of registers, instead of data values in the registers. Therefore, we cannot distinguish benign and malicious programs if the name of register used as an argument of* GetModuleHandleA *is the same as in the case of eax appearing in $\mathcal{P}_{1a}$ and $\mathcal{P}_{1b}$.) In [67], PDS model checking was extended so that it can deal with a finite number of data values.*

We construct $\mathcal{P}_{1a}$ and $\mathcal{P}_{1b}$ as 3-RPDS whose 1st and 2nd registers are used for keeping the contents of eax and ebx, respectively, and the data value in the 3rd register stands for 0 *in the original program. Let $\mathcal{P}_1 = (P_{1a}, \Delta_{1a})$ be 3-RPDS where $P_{1a} = \{l_j \mid j \in [5]\} \cup \{l_{end}\}$ and $\Delta_{1a} = \{(l_1, \phi_1) \to (l_2, 2) \mid \mathtt{x}'_1 \not\sim_{\phi_1} \mathtt{x}_3 \sim_{\phi_1} \mathtt{x}'_3, \mathtt{x}_2 \sim_{\phi_1} \mathtt{x}'_2\} \cup \{(l_2, \phi_{keep}) \to (l_3, 12), (l_4, \phi_{keep}) \to (l_5, 12), (l_5, \phi_{keep}) \to (l_{end}, \varepsilon) \mid \mathtt{x}_1 \sim_{\phi_{keep}} \mathtt{x}'_1, \mathtt{x}_2 \sim_{\phi_{keep}} \mathtt{x}'_2, \mathtt{x}_3 \sim_{\phi_{keep}} \mathtt{x}'_3\} \cup \{(l_3, \phi_{gh}) \to (l_4, \varepsilon) \mid \mathtt{x}_3 \sim_{\phi_{gh}} \mathtt{x}'_3\}$. $\mathcal{P}_{1a}$ behaves as Fig. 6.2 (a) by using its registers and stack to restore adequate data values. (Note that $\mathcal{P}_{1a}$ never pushes the data value $\theta(3)$ to the stack at l_3.) We*

```
                                    l1': move eax, 0
l1: move eax, 1                     l2': push eax
l2: push eax                        l3': push ebx
l3: call GMHA                       l4': pop ebx
l3r: add esp, 4                     l5': call GMHA
l4: push eax                        l5r': add esp, 4
l5: call UseA                       l6': push eax
                                    l7': call UseA
```

Figure 6.2: **(a)** benign program and **(b)** malware program, where GMHA stands for GetModuleHandleA

can define 3*-RPDS* $\mathcal{P}_{1b} = (P_{1b}, \Delta_{1b})$ *in the same way as* $\mathcal{P}_{1a}$ *except that the transition* $(l'_1, \phi'_1) \to (l'_2, 2) \in \Delta_{1b}$ *should transfer the data value* $\theta(3)$ *to the first register, and thus we require* $\mathtt{x}'_1 \sim_{\phi'_1} \mathtt{x}_3 \sim_{\phi'_1} \mathtt{x}'_3$*, and* $\mathtt{x}_2 \sim_{\phi'_1} \mathtt{x}'_2$*. Let*

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

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

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

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

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

The answer to the model checking problem for $\mathcal{P}_1$*,* $\neg\psi_1$*,* Λ_{1a} *and* $(l_1, [d_1, d_2, d_3], d_4)$ *is 'YES', and the answer is 'NO' for* $\mathcal{P}_2$*,* $\neg\psi_1$*,* Λ_{1b} *and* $(l'_1, [d_1, d_2, d_3], d_4)$*. As shown in Fig. 6.3, no run of* $\mathcal{P}_1$ *from* $(l_1, [d_1, d_2, d_3], d_4)$ *satisfies* ψ_1 *because* $\mathtt{top} = \mathtt{x}_3$ *never hold in* l_3 *due to* ϕ_1*. On the other hand, a run of* $\mathcal{P}_2$ *from* $(l'_1, [d_1, d_2, d_3], d_4)$ *may satisfy* ψ_1 *because* ϕ'_1 *forces* $\mathtt{x}'_1 \sim_{\phi'_1} \mathtt{x}_3$*.*

Example 6.4.2 (XML processing)**.** *We consider an XML document as valid if and only if an element name at the top level does not appear inside the contents surrounded by the start and end tags of the top level. For example,*

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

is valid while

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

is not valid because a and /a appear inside the contents surrounded by the top level a and /a . A program $\mathcal{P}_2$ *takes a non-empty XML document, which is a*

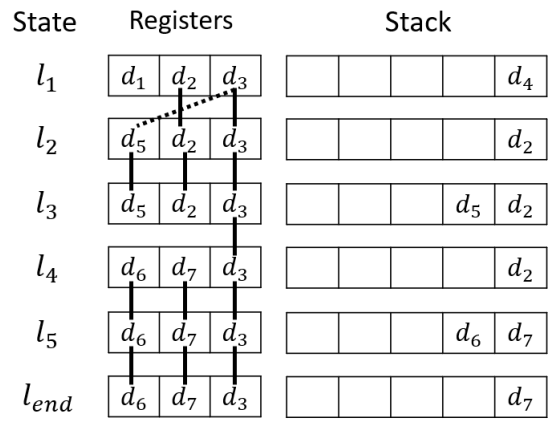

Figure 6.3: A run of $\mathcal{P}_1$ from an ID $(l_1, [d_1, d_2, d_3], d_4)$. (The equality restrictions given by $\sim$ and the disequality restrictions given by $\not\sim$ are shown by solid and dotted lines, respectively.)

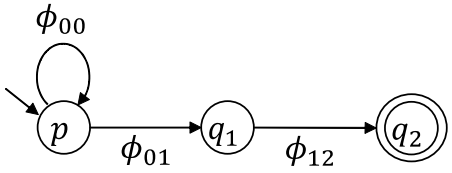

Figure 6.4: XML processing

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

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

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

where ok is true for an ID of $\mathcal{P}_2$ if and only if the stack contents of the ID has the pattern informally described by a regular expression $(any \backslash d_1)^* \cdot d_1 \cdot \perp$ for some data value d_1. Formally, let $\mathcal{P}_2 = (P_2, \Delta_2)$ be an RPDS, $\Lambda_2 : ID_{\mathcal{P}_2} \to \Sigma$ be the regular valuation defined by $\Lambda_2^{-1}(\{ok\}) = L(\mathcal{A}_2)$ where $\mathcal{A}_2 = (P_2 \cup$

$\{q_1, q_2\}, P_2, (q_2, [\bot]), \Delta_2)$ *is the backward-deterministic 1-RA with* $P_2 \cap \{q_1, q_2\} = \emptyset$, $\Delta_2 = \{(p, \phi_{00}) \to p \mid p \in P_2,\ \mathtt{top} \not\sim_{\phi_{00}} \mathtt{x}_1 \sim_{\phi_{00}} \mathtt{x}'_1\} \cup \{(p, \phi_{01}) \to q_1 \mid \mathtt{x}_1 \sim_{\phi_{01}} \mathtt{top} \not\sim_{\phi_{01}} \mathtt{x}'_1\} \cup \{(q_1, \phi_{12}) \to q_2 \mid \mathtt{top} \sim_{\phi_{12}} \mathtt{x}_1 \sim_{\phi_{12}} \mathtt{x}'_1\}$ *(see Fig. 6.4).* $\mathcal{A}_2$ *guesses the element name at the top level in the XML document given to* $\mathcal{P}_2$ *and stores it in the register in the initial ID.* $\mathcal{A}_2$ *verifies whether the guess was correct by* ϕ_{01}*. An example of a run reaching the accepting ID is:* $(p, [d_1], d_2 d_1 \bot) \vdash (p, [d_1], d_1 \bot) \vdash (q_1, [\bot], \bot) \vdash (q_2, [\bot], \varepsilon)$ *where* $d_1 \neq d_2$*.*

6.5 Model Checking LTL with Simple Valuation

We assume that a simple valuation $\Lambda : P \times \Phi_{k+1} \to \Sigma$ is given as a part of an input in Definition 6.3.1 and provide an algorithm for the problem. We first construct a Büchi automaton $\mathcal{B}$ over Σ such that $L(\mathcal{B}) = \{\alpha \in \Sigma^\omega \mid \alpha \models \neg\varphi\}$ in the standard way. After that, we construct the *product* of $\mathcal{P}$ and $\mathcal{B}$ as k-BRPDS $\mathcal{BP}$ defined below.

Definition 6.5.1. *Let* $\mathcal{P} = (P, \Delta)$ *be a* k*-RPDS,* $\mathcal{B} = (Q, q_0, F, \delta)$ *be a Büchi automaton over* Σ*, and* $\Lambda : P \times \Phi_{k+1} \to \Sigma$ *be a simple valuation function. The product of* $\mathcal{P}$ *and* $\mathcal{B}$ *is* k*-BRPDS* $\mathcal{BP} = (P \times Q, \Delta', P \times F)$*, where* $\Delta' = \{((p, q), \phi) \to ((p', q'), J) \mid (p, \phi) \to (p', J) \in \Delta, (q, \sigma) \to q' \in \delta, \sigma = \Lambda(p, F_{former}(\phi))\}$*.*

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

Proposition 6.5.1. *Let* $\mathcal{B} = (Q, q_0, F, \delta)$ *be a Büchi automaton over* Σ *that satisfies* $L(\mathcal{B}) = \{\alpha \in \Sigma^\omega \mid \alpha \models \neg\varphi\}$ *for a given LTL formula* φ *over At. Let* $\Lambda : ID_{\mathcal{P}} \to \Sigma$ *be a simple valuation function. Also, let* $\mathcal{BP} = (P \times Q, \Delta', P \times F)$ *be the product* k*-BRPDS of a* k*-RPDS* $\mathcal{P} = (P, \Delta)$ *and* $\mathcal{B}$ *defined in Definition 6.3.1. Then,* $\mathcal{BP}$ *does not accept an ID* $((p_0, q_0), \theta_0, u_0)$ *if and only if* $\Lambda(\rho) \models \varphi$ *for all run* $\rho \in (ID_{\mathcal{P}})^\omega$ *such that* $\rho(0) = (p_0, \theta_0, u_0)$*.*

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

and only if there is a run ρ of $\mathcal{P}$ such that $\Lambda(\rho) \in L(\mathcal{B})$ and $\rho(0) = (p_0, \theta_0, u_0)$. Below we prove this statement.

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

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

Since the acceptance problem for BRPDS is decidable by Proposition 6.2.3, the model checking problem with simple valuation is decidable.

Theorem 6.5.1. *The LTL model checking problem for RPDS with simple valuation is EXPTIME-complete.*

Proof. Since the LTL model checking problem for PDS with simple valuation is EXPTIME-complete [13], it is enough to show EXPTIME solvability. Let $\mathcal{P} = (P, \Delta)$, φ, and $\Lambda : ID_{\mathcal{P}} \to \Sigma$ be a given k-RPDS, an LTL formula over At, and a simple valuation, respectively. Without loss of generality, we assume every $p \in P$ appears in some rule in Δ, and thus $|P| \leq 2|\Delta|$. We define the description length of $\mathcal{P}$ as $\|\mathcal{P}\| = |\Delta|(\log|P| + k^2)$, because each rule $(p, \phi) \to (q, J) \in \Delta$ where $J \in [k]^2 \cup [k] \cup \{\varepsilon\}$ can be described in $2\log|P| + \log|\Phi_{2k+1}| + \log(k^2 + k + 1) = O(\log|P| + k^2)$ bits. We define the length $|\varphi|$ of φ as the number of operators and atomic propositions appearing in φ. We assume that Λ is given as a list of $\Lambda(p, \phi) \in \Sigma$ for all $p \in P$ and $\phi \in \Phi_{k+1}$, and thus the description length of Λ is $\|\Lambda\| = |P| \cdot |\Phi_{k+1}| \cdot \log|\Sigma| = |P| \cdot 2^{O(k^2)} \cdot |At|$.

In the first step of the model-checing algorithm, we construct the Büchi automaton $\mathcal{B} = (Q, q_0, F, \delta)$ over $\Sigma = 2^{At}$ whose language is the set of infinite words that satisfy the negation of φ. In the standard way of the construction of $\mathcal{B}$, $|Q| \leq 2^{O(|\varphi|)}$ and $|\delta| \leq 2^{O(|\varphi|)}$, and the construction takes at most $2^{O(|\varphi|)}$ time [21].

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

In the final step, we solve the acceptance problem of $((p_0, q_0), \theta_0, u_0)$ for $\mathcal{BP}$. By Proposition 6.2.2, it takes $O((|\Delta'| + |u_0|^{O(k)}) \cdot |P \times Q|^{O(1)} \cdot 2^{O(k^2)}) = O((|\Delta| + |u_0|^{O(k)}) \cdot |P|^{O(1)} \cdot 2^{O(k^2+|\varphi|)})$ time.

6.6 Model Checking LTL with Regular Valuation

Next, we show the decidability of the model checking problem with regular valuation. We prove this by a reduction to the model checking problem with simple valuation as shown in Proposition 6.6.1. The idea behind the proof is based on [30], but we need to extend the idea of [30] to deal with RPDS as follows. To simulate a given RPDS $\mathcal{P}$ with a regular valuation by an RPDS $\mathcal{P}'$ with a simple valuation, the current IDs of backward-deterministic RAs for all atomic propositions are stored in additional registers and the remaining IDs are stored in the stack of $\mathcal{P}'$. The purpose of this transformation is that we can determine which atomic propositions hold under the regular valuation of $\mathcal{P}$ by observing only the finite state, the (extended) register assignments and the stack top of $\mathcal{P}'$. For example, assume $At = \{A\}$ and let $\mathcal{A} = (Q, q_0, F, \delta)$ be a backward-deterministic RA for the proposition A. For stack contents $d_n \cdots d_1 \in$

D^* of $\mathcal{P}$, we associate IDs of $\mathcal{A}$ as $d_n(p_{n-1}, \theta_{n-1}, d_{n-1}) \cdots (p_1, \theta_1, d_1)$ such that $(p_{i+1}, \theta_{i+1}, d_i) \vdash_{\mathcal{A}} (p_i, \theta_i, \varepsilon)$ holds for each $i \in [n-1]$. For a technical reason, (p_n, θ_n) is stored in additonal registers of $\mathcal{P}'$. More accurately, we introduce the encoding table θ_Q that keeps distinct data values $d_{p_1}, \ldots, d_{p_{|Q|}}$ and we store the constant data value d_{p_i} instead of the state p_i of $\mathcal{A}$ to registers and stack of $\mathcal{P}'$ because we cannot store a finite symbol p_i without extending the definition of RPDS.

Proposition 6.6.1. *For a given k-RPDS $\mathcal{P} = (P, \Delta)$, a regular valuation function $\Lambda : ID_{\mathcal{P}} \to \Sigma$ ($\Sigma = 2^{At}$), an ID $c \in P \times \Theta_k \times D$ whose stack is a single data value, and an LTL formula φ over At, we can construct an RPDS $\mathcal{P}' = (P', \Delta')$, a simple valuation function $\Lambda_s : ID_{\mathcal{P}'} \to \Sigma \cup \{\{\sqcup\}\}$, an ID $c' \in ID_{\mathcal{P}'}$ of $\mathcal{P}'$, and an LTL formula φ_s over $At \cup \{\sqcup\}$, where $\sqcup \notin At$, that satisfy the following conditions: for every run $\rho \in (ID_{\mathcal{P}})^\omega$ of $\mathcal{P}$ starting with c and every run $\rho' \in (ID_{\mathcal{P}'})^\omega$ of $\mathcal{P}'$ starting with c', $\Lambda(\rho) \models \varphi \iff \Lambda_s(\rho') \models \varphi_s$.*

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

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

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

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

As the method of [30], we need to

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

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

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

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

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

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

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

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

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

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

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

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

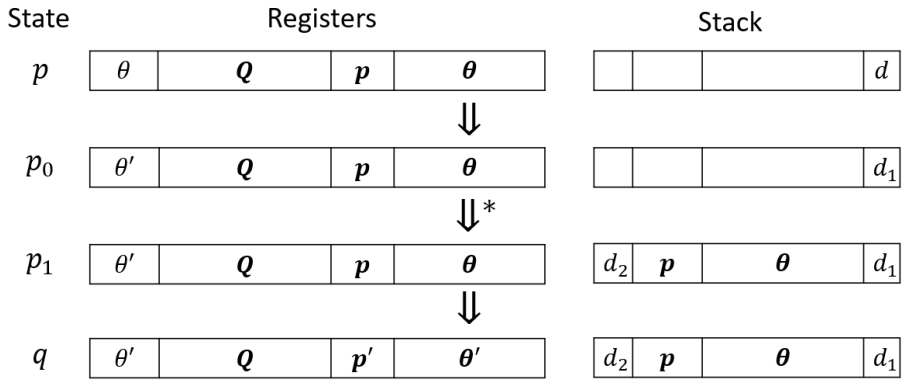

Figure 6.5: An outline of transitions of the push condition

combination of these $|At|$ rules, we add the following replace rule to Δ':

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

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

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

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

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

(end of the proof of the Claim 1)

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

We define a simple valuation Λ_s as satisfying

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

for every $\sigma \in \Sigma$.

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

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

Theorem 6.6.1. *The RPDS model checking problem of LTL with regular valuation is EXPTIME-complete.*

Proof. By Proposition 6.6.1, a model checking problem for a k-RPDS $\mathcal{P} = (P, \Delta)$, an LTL formula φ and a regular valuation $\Lambda : ID_{\mathcal{P}} \to \Sigma$ can be reduced to a model checking problem for a K-RPDS $\mathcal{P}' = (P', \Delta')$ over $\Sigma \cup \{\{\sqcup\}\}$, the LTL formula φ_s over $At \cup \{\sqcup\}$ and the simple valuation $\Lambda_s : ID_{\mathcal{P}'} \to \Sigma \cup \{\{\sqcup\}\}$,

which is also written as $\Lambda_s : P' \times \Phi_K \to \Sigma \cup \{\{\sqcup\}\}$. For K and b appearing in Proposition 6.6.1, $K = k + \sum_{A \in At} |Q_A| + |At| + |At| \cdot k = O(|\Lambda| + k \cdot |\varphi|)$ and $b = (1+k)|At| + 2 = O(k \cdot |\varphi|)$ hold. By the construction of φ_s, $|\varphi_s| \leq |\varphi| \times b = O(k \cdot |\varphi|^2)$ holds.

By Theorem 6.5.1, a model checking for simple valuation with the inputs $\mathcal{P}' = (P', \Delta'), \Lambda_s$ and φ_s can be solved in exponential to K and $|\varphi_s|$, and polynomial to $|P'|$ and $|\Delta'|$. Because both K and $|\varphi_s|$ are polynomial to $k, |\Lambda|$ and $|\varphi|$, it is enough to prove $|P'|$ and $|\Delta'|$ are at most single-exponential to the input size $k + |\Lambda| + |\varphi|$.

For constructing each rules for the push, replace and pop in Proposition 6.6.1, we add $b - 1 = O(k \cdot |\varphi|)$ states to Δ'. Thus, $|P'| = O(k \cdot |\varphi| \cdot |\Delta|)$ holds. By the definition of Δ', $|\Delta'| \leq |P'| \cdot |\Phi_K| \cdot |P'| \cdot 3 = O((k \cdot |\varphi| \cdot |\Delta|)^2 \cdot 2^{O((|\Lambda| + k \cdot |\varphi|)^2)})$. Hence, $|P'|$ and $|\Delta'|$ are exponential to the input size $k + |\Lambda| + |\varphi|$, and thus the EXPTIME solvability holds.

Since the problem for PDS is EXPTIME-complete [13] we obtain the EXPTIME-hardness of the problem.

Conclusion of Part III

- We proved a backward regularity preservation theorem for RPDS. We obtained a non-trivial theoretical result that the regularity conservation property for RPDS holds in both forward and backward. As an application, the Joinability problem can be easily solved by using the backward regularity preservation property.
  Joinability problem of RPDS: Can two regular sets of initial IDs reached to the same ID by runs of RPDS?
- We proposed an algorithm for solving the RPDS-LTL model checking problem. This algorithm can handle the equality check for data values, which PDS-LTL model checking cannot handle. Thus, our method can be applied to the verification of programs handling data values.

Part IV

Software Synthesis

Outline of Part IV

In this part, we investigate the realizability problem for the cases that a specification and an implementation are given by a PDA and a PDT, and an RPDA and an RPDT. We first introduce a transition system (TS), which is a labelled directed graph with initial state and accepting condition and will be used to define accepting lauguages of computational models. After that, we define PDA and PDT over an input alphabet $\Sigma_{\mathbb{i}}$, an output alphabet $\Sigma_{\mathbb{o}}$ and stack alphabet Γ. PDA $\mathcal{A}$ read an infinite alternate sequence $i_1o_1i_2o_2\cdots \in (\Sigma_{\mathbb{i}} \cdot \Sigma_{\mathbb{o}})^\omega$ and accept or reject the input sequence according to a Parity accepting condition, and if accept, we write $i_1o_1i_2o_2 \in L(\mathcal{A})$. A PDT $\mathcal{T}$ is a deterministic PDA with output, and read an infinite input sequence $i_1i_2\cdots \in \Sigma_{\mathbb{i}}^\omega$ and emit an infinite output sequence $o_1o_2\cdots \in \Sigma_{\mathbb{o}}^\omega$, and then we write $i_1o_1i_2o_2 \in L(\mathcal{T})$. The realizability problem for PDA $\mathcal{A}$ asks whether there exists PDT $\mathcal{T}$ such that $L(\mathcal{T}) \subseteq L(\mathcal{A})$. We show that the realizability problem for DPDA is decidable while the problem is undecidable for NPDA. After that, we move to the realizability problem for RPDA and register PDT (RPDT), and also show that the realizability problem for DRPDA is solvable in 2-EXPTIME if we assume the visibility of stack operation.

Chapter 7 Reactive Synthesis for Visibly RPDA

7.1 Definitions

Let $\langle u, w\rangle = u(0)w(0)u(1)w(1)\cdots \in A^\infty$ for words $u, w \in A^\infty$ and $\langle B, C\rangle = \{\langle u, w\rangle \mid u \in B, w \in C\}$ for sets $B, C \subseteq A^\infty$. For a function $f : A \to B$ from a set A to a set B, let $f(w) = f(w(0))f(w(1))\ldots$ for a word $w \in A^\infty$ and let $f(L) = \{f(w) \mid w \in L\}$ for a set $L \subseteq A^\infty$ of words. Let *fst* and *snd* be the functions such that $fst((a, b)) = a$ and $snd((a, b)) = b$ for any pair (a, b). Let *id* be the identity function; i.e., $id(a) = a$ for any a.

7.1.1 Transition systems

Definition 7.1.1. *A transition system (TS) is* $\mathcal{S} = (S, s_0, A, E, \to_{\mathcal{S}}, c)$ *where*

- S *is a (finite or infinite) set of states,*
- $s_0 \in S$ *is the initial state,*
- A, E *are (finite or infinite) alphabets such that* $A \cap E = \emptyset$*,*
- $\to_{\mathcal{S}} \subseteq S \times (A \cup E) \times S$ *is a transition relation, written as* $s \to^a s'$ *if* $(s, a, s') \in \to_{\mathcal{S}}$ *and*
- $c : S \to [n]$ *is a coloring function where* $n \in \mathbb{N}$*.*

An element of A is an observable label and an element of E is an internal label. A *run* of TS $\mathcal{S} = (S, s_0, A, E, \to_{\mathcal{S}}, c)$ is a pair $(\rho, w) \in S^\omega \times (A \cup E)^\omega$ that satisfies $\rho(0) = s_0$ and $\rho(i) \to^{w(i)} \rho(i + 1)$ for $i \geq 0$. Let $\min_{\text{inf}} : S^\omega \to [n]$ be the minimal coloring function such that $\min_{\text{inf}}(\rho) = \min\{m \mid$ there exist an infinite

number of $i \geq 0$ such that $c(\rho(i)) = m\}$. We call $\mathcal{S}$ deterministic if $s \to^a s_1$ and $s \to^a s_2$ implies $s_1 = s_2$ for all $s, s_1, s_2 \in S$ and $a \in A \cup E$.

For $w \in (A \cup E)^\omega$, let $ef(w) = a_0 a_1 \cdots \in A^\infty$ be the sequence obtained from w by removing all symbols belonging to E. Note that $ef(w)$ is not always an infinite sequence even if w is an infinite sequence. We define the *language* of $\mathcal{S}$ as $L(\mathcal{S}) = \{ef(w) \in A^\omega \mid$ there exists a run (ρ, w) such that $\min_{\inf}(\rho)$ is even$\}$. For $m \in \mathbb{N}_0$, we call $\mathcal{S}$ an m-TS if for every run (ρ, w) of $\mathcal{S}$, w contains no contiguous subsequence $w' \in E^*$ such that $|w'| > m$.

Consider two TSs $\mathcal{S}_1 = (S_1, s_{01}, A_1, E_1, \to_{\mathcal{S}_1}, c_1)$ and $\mathcal{S}_2 = (S_2, s_{02}, A_2, E_2, \to_{\mathcal{S}_2}, c_2)$ and a function $\sigma : (A_1 \cup E_1) \to (A_2 \cup E_2)$. We call $R \subseteq S_1 \times S_2$ a *σ-bisimulation relation* from $\mathcal{S}_1$ to $\mathcal{S}_2$ if R satisfies the followings:

(1) $(s_{01}, s_{02}) \in R$.

(2) For any $s_1, s_1' \in S_1$, $s_2 \in S_2$, and $a_1 \in A_1 \cup E_1$, if $s_1 \to_{\mathcal{S}_1}^{a_1} s_1'$ and $(s_1, s_2) \in R$, then $\exists s_2' \in S_2 : s_2 \to_{\mathcal{S}_2}^{\sigma(a_1)} s_2'$ and $(s_1', s_2') \in R$.

(3) For any $s_1 \in S_1$, $s_2, s_2' \in S_2$, and $a_2 \in A_2 \cup E_2$, if $s_2 \to_{\mathcal{S}_2}^{a_2} s_2'$ and $(s_1, s_2) \in R$, then $\exists s_1' \in S_1$, $\exists a_1 \in A_1 \cup E_1 : \sigma(a_1) = a_2$ and $s_1 \to_{\mathcal{S}_1}^{a_1} s_1'$ and $(s_1', s_2') \in R$.

(4) If $(s_1, s_2) \in R$, then $c_1(s_1) = c_2(s_2)$.

We say $\mathcal{S}_1$ is *σ-bisimilar* to $\mathcal{S}_2$ if there exists a σ-bisimulation relation from $\mathcal{S}_1$ to $\mathcal{S}_2$. We call R a bisimulation relation if R is an *id*-bisimulation relation. We say $\mathcal{S}_1$ and $\mathcal{S}_2$ are bisimilar if $\mathcal{S}_1$ is *id*-bisimilar to $\mathcal{S}_2$.

The following lemma can be proved by definition.

Lemma 7.1.1. *If $\mathcal{S}_1 = (S_1, s_{01}, A_1, E_1, \to_{\mathcal{S}_1}, c_1)$ is σ-bisimilar to $\mathcal{S}_2 = (S_2, s_{02}, A_2, E_2, \to_{\mathcal{S}_2}, c_2)$ for a function $\sigma : (A_1 \cup E_1) \to (A_2 \cup E_2)$ that satisfies $a \in A_1 \Leftrightarrow \sigma(a) \in A_2$ for any $a \in A_1 \cup E_1$, then $\sigma(L(\mathcal{S}_1)) = L(\mathcal{S}_2)$.*

7.2 Pushdown Transducers, Automata and Games

In this section, we review definitions of pushdown automaton (PDA), pushdown transducer (PDT) and pushdown game (PDG), together with a well-known

property of PDG. The next section discusses the realizability problem for PDA as specifications and PDT as implementations. As described in the introduction, the approach to solve the realizability problem is as follows. We first convert a given specification (PDA) to a PDG $\mathcal{B}$ by separating the input and output streams. The answer to the realizability problem is affirmative if and only if there is a winning strategy for player I in $\mathcal{B}$. An implementation of the specification is easily obtained as a PDT from any winning strategy for $\mathcal{B}$.

We assume that disjoint sets $\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}$ and Γ are given as a (finite) input alphabet, an output alphabet and a stack alphabet, respectively, and $\Sigma = \Sigma_{\mathbb{i}} \cup \Sigma_{\mathbb{o}}$. Let $\mathtt{Com}(\Gamma) = \{\mathtt{pop}, \mathtt{skip}\} \cup \{\mathtt{push}(z) \mid z \in \Gamma\}$ be the set of stack commands over Γ.

7.2.1 Pushdown transducers

Definition 7.2.1. *A pushdown transducer (PDT) over $\Sigma_{\mathbb{i}}$, $\Sigma_{\mathbb{o}}$ and Γ is $\mathcal{T} = (P, p_0, z_0, \Delta)$ where P is a finite set of states, $p_0 \in P$ is the initial state, $z_0 \in \Gamma$ is the initial stack symbol and $\Delta : P \times \Sigma_{\mathbb{i}} \times \Gamma \to P \times \Sigma_{\mathbb{o}} \times \mathtt{Com}(\Gamma)$ is a finite set of deterministic transition rules having one of the following forms:*

- $(p, a, z) \to (q, b, \mathtt{pop})$ *(pop rule)*
- $(p, a, z) \to (q, b, \mathtt{skip})$ *(skip rule)*
- $(p, a, z) \to (q, b, \mathtt{push}(z))$ *(push rule)*

where $p, q \in P$, $a \in \Sigma_{\mathbb{i}}$, $b \in \Sigma_{\mathbb{o}}$ and $z \in \Gamma$.

For a state $p \in P$ and a finite sequence representing stack contents $u \in \Gamma^*$, (p, u) is called a *configuration* or *instantaneous description (abbreviated as ID)* of PDT $\mathcal{T}$. Let $ID_{\mathcal{T}}$ denote the set of all IDs of $\mathcal{T}$. For $u \in \Gamma^+$ and $\mathtt{com} \in \mathtt{Com}(\Gamma)$, let us define $\mathtt{upds}(u, \mathtt{com})$ as $\mathtt{upds}(u, \mathtt{pop}) = u(1:)$, $\mathtt{upds}(u, \mathtt{skip}) = u$ and $\mathtt{upds}(u, \mathtt{push}(z')) = z'u$.

For two IDs $(p, u), (q, u') \in ID_{\mathcal{T}}$, $a \in \Sigma_{\mathbb{i}}$ and $b \in \Sigma_{\mathbb{o}}$, $((p, u), ab, (q, u')) \in \Rightarrow_{\mathcal{T}}$, written as $(p, u) \Rightarrow_{\mathcal{T}}^{ab} (q, u')$, if there exist a rule $(p, a, z) \to (q, b, \mathtt{com}) \in \Delta$ such that $z = u(0)$ and $u' = \mathtt{upds}(u, \mathtt{com})$. If $\mathcal{T}$ is clear from the context, we abbreviate $\Rightarrow_{\mathcal{T}}^{ab}$ as $\Rightarrow^{ab}$. We will use similar abbreviations for the other models defined later. Note that there is no transition from an ID with empty stack. We define a run

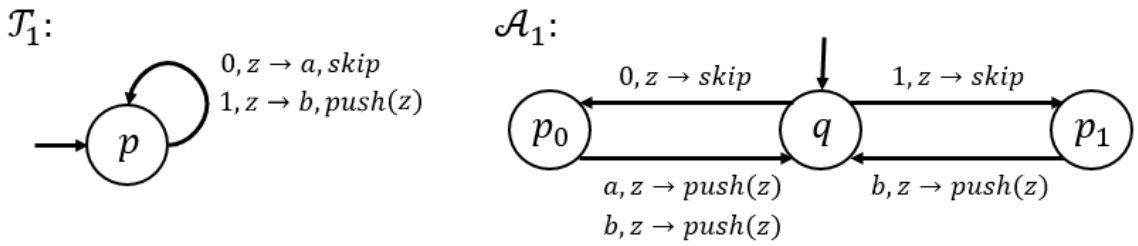

Figure 7.1: States and transitions of $\mathcal{T}_1$ (left) and $\mathcal{A}_1$ (right). Labels $a, b \to c, d$ and $a, b \to c$ from q to q' mean $(q, a, b) \to (q', c, d) \in \Delta$ and $(q, a, b) \to (q', c) \in \delta$, respectively.

and the language $L(\mathcal{T}) \subseteq (\Sigma_{\mathbb{i}} \cdot \Sigma_{\mathbb{o}})^\omega$ of PDT $\mathcal{T}$ as those of deterministic 0-TS $(ID_{\mathcal{T}}, (q_0, z_0), \Sigma_{\mathbb{i}} \cdot \Sigma_{\mathbb{o}}, \emptyset, \Rightarrow_{\mathcal{T}}, c)$ where $c(s) = 2$ for all $s \in ID_{\mathcal{T}}$. In this paper, we assume that no run of PDT reaches an ID whose stack is empty. We can realize this assumption by specifying a unique stack bottom symbol $z_\perp$ and forcing that every rule $(q, a, z_\perp) \to (q', \mathtt{com}) \in \delta$ satisfies $\mathtt{com} \neq \mathtt{pop}$. Let **PDT** be the class consisting of all PDT.

Example 7.2.1. *Let us consider a PDT* $\mathcal{T}_1 = (\{p\}, p, z, \Delta)$ *over* $\{0, 1\}, \{a, b\}$ *and* $\{z\}$ *where* $\Delta = \{(p, 0, z) \to (p, a, \mathtt{skip}), (p, 1, z) \to (p, b, \mathtt{push}(z))\}$. *(See Fig. 7.1, left.) We can see a pair of sequences* (ρ, w) *where* $\rho = (p, z)(p, z)(p, zz)(p, zz)(p, zzz)(p, zzz) \cdots$ *and* $w = (0a1b)^\omega$ *is a run of* $\mathcal{T}_1$. *Also,* $L(\mathcal{T}_1) = (\{0a\} \cup \{1b\})^\omega$.

7.2.2 Pushdown automata

Definition 7.2.2. *A nondeterministic pushdown automata (NPDA) over* $\Sigma_{\mathbb{i}}$, $\Sigma_{\mathbb{o}}$ *and* Γ *is* $\mathcal{A} = (Q, Q_{\mathbb{i}}, Q_{\mathbb{o}}, q_0, z_0, \delta, c)$ *where* Q, $Q_{\mathbb{i}}, Q_{\mathbb{o}}$ *are finite sets of states such that* $Q = Q_{\mathbb{i}} \cup Q_{\mathbb{o}}$ *and* $Q_{\mathbb{i}} \cap Q_{\mathbb{o}} = \emptyset$, $q_0 \in Q_{\mathbb{i}}$ *is the initial state,* $z_0 \in \Gamma$ *is the initial stack symbol,* $c : Q \to [n]$ *is the coloring function where* $n \in \mathbb{N}$ *is the number of priorities and* $\delta : Q \times (\Sigma \cup \{\tau\}) \times \Gamma \to \mathscr{P}(Q \times \mathtt{Com}(\Gamma))$ *is a finite set of transition rules having one of the following forms:*

- $(q_{\mathbb{x}}, a_{\mathbb{x}}, z) \to (q_{\overline{\mathbb{x}}}, \mathtt{com})$ *(input/output rules)*
- $(q_{\mathbb{x}}, \tau, z) \to (q'_{\mathbb{x}}, \mathtt{com})$ *(τ rules, where $\tau \notin \Sigma$)*

where $(\mathbb{x}, \overline{\mathbb{x}}) \in \{(\mathbb{i}, \mathbb{o}), (\mathbb{o}, \mathbb{i})\}$, $q_{\mathbb{x}}, q'_{\mathbb{x}} \in Q_{\mathbb{x}}, q_{\overline{\mathbb{x}}} \in Q_{\overline{\mathbb{x}}}, a_{\mathbb{x}} \in \Sigma_{\mathbb{x}}$, $z \in \Gamma$ *and* $\mathtt{com} \in \mathtt{Com}(\Gamma)$.

We define $ID_{\mathcal{A}} = Q \times \Gamma^*$ and the transition relation $\vdash_{\mathcal{A}} \subseteq ID_{\mathcal{A}} \times (\Sigma \cup \{\tau\}) \times ID_{\mathcal{A}}$ as $((q,u),a,(q',u')) \in \vdash_{\mathcal{A}}$ iff there exist a rule $(q,a,z) \to (q',\mathtt{com}) \in \delta$ and a sequence $u \in \Gamma^*$ such that $z = u(0)$ and $u' = \mathtt{upds}(u,\mathtt{com})$. We write $(q,u) \vdash^a_{\mathcal{A}} (q',u')$ iff $((q,u),a,(q',u')) \in \vdash_{\mathcal{A}}$. We define a run and the language $L(\mathcal{A})$ of $\mathcal{A}$ as those of TS $\mathcal{S}_{\mathcal{A}} = (ID_{\mathcal{A}}, (q_0,z_0), \Sigma, \{\tau\}, \vdash_{\mathcal{A}}, c')$ where $c'((q,u)) = c(q)$ for every $(q,u) \in ID_{\mathcal{A}}$. We call a PDA $\mathcal{A}$ deterministic if $\mathcal{S}_{\mathcal{A}}$ is deterministic. We call $\mathcal{A}$ an m-NPDA (or m-DPDA when $\mathcal{A}$ is deterministic) if $\mathcal{S}_{\mathcal{A}}$ is an m-TS. We abbreviate 0-NPDA (0-DPDA) as NPDA (DPDA). Let **DPDA** and **NPDA** be the classes of DPDA and NPDA, respectively.

Example 7.2.2. *Let us consider a DPDA* $\mathcal{A}_1 = (\{q,p_0,p_1\}, \{q\}, \{p_0,p_1\}, q, z, \delta, c)$ *over* $\{0,1\}, \{a,b\}$ *and* $\{z\}$ *where* $c(q) = c(p_0) = c(p_1) = 2$ *and* $\delta = \{(q,0,z) \to (p_0, \mathtt{skip}), (q,1,z) \to (p_1,\mathtt{skip}), (p_0,a,z) \to (q,\mathtt{push}(z)), (p_0,b,z) \to (q,\mathtt{push}(z)), (p_1, b, z) \to (q,\mathtt{push}(z))\}$. *(See Fig. 7.1, right.) We can see a pair of sequences* (ρ, w) *where* $\rho = (q,z)(p_0,z)(q,zz)(p_1,zz)\cdots$ *and* $w = (0a1b)^\omega$ *is a run of* $\mathcal{A}_1$. *Also,* $L(\mathcal{A}_1) = (\{0a\} \cup \{0b\} \cup \{1b\})^\omega$.

The following lemma states that the class of languages recognized by m-DPDA and 0-DPDA are the same for a fixed m.

Lemma 7.2.1. *For a given* m*-DPDA* $\mathcal{A}$*, we can construct a* 0*-DPDA* $\mathcal{A}'$ *such that* $L(\mathcal{A}) = L(\mathcal{A}')$

Proof. For a given m-DPDA $\mathcal{A}$, we can construct a $2m$-DPDA $\mathcal{A}'$ such that $L(\mathcal{A}) = L(\mathcal{A}')$ and $\mathcal{A}'$ has no skip rule by replacing every skip rule $(q,a,z) \to (q',\mathtt{skip})$ of $\mathcal{A}$ to a pair of push and pop rules $(q,a,z) \to (q'',\mathtt{push}(z')), (q,\tau,z') \to (q',\mathtt{pop})$ of $\mathcal{A}'$ for $a \in \Sigma \cup \{\tau\}$. Thus, we show the lemma for m-DPDA $\mathcal{A}$ that has no skip rule by the induction on m. The case $m = 0$ is obvious. For an arbitrary m, m-DPDA $\mathcal{A} = (Q, Q_{\mathbb{i}}, Q_{\mathbb{o}}, q_0, z_0, \delta, c)$ over $\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}$ and Γ can be converted to an $(m-1)$-DPDA $\mathcal{A}'$ over $\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}$ and Γ^2 such that $L(\mathcal{A}) = L(\mathcal{A}')$. Let $\mathcal{A}' = (Q \cup (Q \times \Gamma), Q_{\mathbb{i}} \cup (Q_{\mathbb{i}} \times \Gamma), Q_{\mathbb{o}} \cup (Q_{\mathbb{o}} \times \Gamma), (q_0,z_0), (z_0,z_0), \delta', c')$ such that $c'(q) = c(q), c'((q,a)) = c(q)$ for all $q \in Q, a \in \Sigma$ and

- $(q, a, z_1) \to (q', \texttt{pop}) \in \delta$ iff $(q, a, (z_1, z_2)) \to ((q', z_2), \texttt{pop})$, $((q, z_1), a, (z_c, z_c')) \to (q', \texttt{skip}) \in \delta'$ for all $z_c, z_c' \in \Gamma$.
- $(q, a, z_1) \to (q', \texttt{skip}) \in \delta$ iff $(q, a, (z_1, z_2)) \to (q', \texttt{skip})$, $((q, z_1), a, (z_c, z_c')) \to ((q', z_1), \texttt{skip}) \in \delta'$ for all $z_c, z_c' \in \Gamma$.
- $(q, a, z_1) \to (q', \texttt{push}(z')) \in \delta$ iff $(q, a, (z_1, z_2)) \to ((q', z'), \texttt{skip})$, $((q, z_1), a, (z_c, z_c')) \to (q', \texttt{push}((z', z_1))) \in \delta'$ for all $z_c, z_c' \in \Gamma$.

for $a \in \Sigma$, and

- $(q, a, z_1) \to (q', \texttt{pop}), (q', b, z_2) \to (q'', \texttt{pop}) \in \delta$ iff $(q, x, (z_1, z_2)) \to (q'', \texttt{pop})$, $((q, z_1), x, (z_2, z_c)) \to ((q'', z_c), \texttt{pop}) \in \delta'$ for all $z_c \in \Gamma$.
- $(q, a, z_1) \to (q', \texttt{push}(z')) \in \delta', (q', b, z') \to (q'', \texttt{pop}) \in \delta$ iff $(q, x, (z_1, z_c)) \to (q'', \texttt{skip})$, $((q, z_1), x, (z_c, z_c')) \to ((q'', z_1), \texttt{skip}) \in \delta'$ for all $z_c, z_c' \in \Gamma$.
- $(q, a, z_1) \to (q', \texttt{push}(z')), (q', b, z') \to (q'', \texttt{push}(z'')) \in \delta$ iff $(q, x, (z_1, z_c)) \to (q'', \texttt{push}((z'', z')))$, $((q, z_1), x, (z_c, z_c')) \to ((q'', z''), \texttt{push}(z', z_1)) \in \delta'$ for all $z_c, z_c' \in \Gamma$.

where $a, b \in \Sigma \cup \{\tau\}$, $x = a$ if $a \in \Sigma$ and $x = b$ otherwise. As the definition of $\mathcal{A}'$, the ID $(q, z_1 z_2 z_3 \cdots z_n)$ of $\mathcal{A}$ corresponds to an ID $(q, (z_1, z_2) \cdots (z_{n-1}, z_n))$ of $\mathcal{A}'$ if n is even and an ID $((q, z_1), (z_2, z_3) \cdots (z_{n-1}, z_n))$ if n is odd. We can check $L(\mathcal{A}) = L(\mathcal{A}')$ by the induction on the length of a sequence $w \in L(\mathcal{A})$.

7.2.3 Pushdown games

Definition 7.2.3. *A pushdown game of DPDA $\mathcal{A} = (Q, Q_{\mathbb{I}}, Q_{\mathbb{O}}, q_0, z_0, \delta, c)$ over $\Sigma_{\mathbb{I}}, \Sigma_{\mathbb{O}}$ and Γ is $\mathcal{G}_{\mathcal{A}} = (V, V_{\mathbb{I}}, V_{\mathbb{O}}, E, C)$ where $V = Q \times \Gamma^*$ is the set of vertices with $V_{\mathbb{I}} = Q_{\mathbb{I}} \times \Gamma^*, V_{\mathbb{O}} = Q_{\mathbb{O}} \times \Gamma^*$, $E \subseteq V \times V$ is the set of edges defined as $E = \{(v, v') \mid v \vdash^a v' \text{for some } a \in \Sigma_{\mathbb{I}} \cup \Sigma_{\mathbb{O}}\}$ and $C : V \to [n]$ is the coloring function such that $C((q, u)) = c(q)$ for all $(q, u) \in V$.*

The game starts with $(q_0, z_0) \in V_{\mathbb{I}}$. When the current vertex is $v \in V_{\mathbb{I}}$, Player II chooses a successor $v' \in V_{\mathbb{O}}$ of v as the next vertex. When the current vertex is $v \in V_{\mathbb{O}}$, Player I chooses a successor $v' \in V_{\mathbb{I}}$ of v. Formally, a finite or infinite sequence $\rho \in V^\infty$ is *valid* if $\rho(0) = (q_0, z_0)$ and $(\rho(i-1), \rho(i)) \in E$ for every

$i \geq 1$. A *play* of $\mathcal{G}_{\mathcal{A}}$ is an infinite and valid sequence $\rho \in V^{\omega}$. Let PL be the set of plays. A play $\rho \in PL$ is *winning* for Player I iff $\min\{m \in [n] \mid$ there exists an infinite number of $i \geq 0$ such that $c(\rho(i)) = m\}$ is even. Note that by definition, a play ρ is winning for Player I iff (ρ, w) is an accepting run of $\mathcal{A}$ for some w.

Since $\mathcal{A}$ is deterministic, the following lemma holds.

Lemma 7.2.2. *Let $f_1 : PL \to (Q \times \mathtt{Com}(\Gamma))^{\omega}$ and $f_2 : (\Sigma_{\mathbb{i}} \cdot \Sigma_{\circledcirc})^{\omega} \to PL$ be the functions defined as follows:*

- *$f_1(\rho) = (q_0, \mathtt{com}_0)(q_1, \mathtt{com}_1) \cdots \in (Q \times \mathtt{Com})^{\omega}$ where $\rho = (q_0, u_0)(q_1, u_1) \cdots \in PL$ and $u_{i+1} = \mathtt{upds}(u_i, \mathtt{com}_i)$ for all $i \geq 0$ and*
- *$f_2(w) = \rho$ where $\rho = (q_0, u_0)(q_1, u_1) \cdots \in PL$ and $\rho(i) \vdash^{w(i)} \rho(i+1)$ for all $i \geq 0$.*

Then, f_1 and f_2 are well-defined, f_1 is an injection and $f_2(L(\mathcal{A}))$ is the set of all the winning plays of Player I.

Theorem 7.2.1. *[72] If player I has a winning strategy of $\mathcal{G}_{\mathcal{A}}$, we can construct a PDT $\mathcal{T}$ over $Q_{\mathbb{i}} \times \mathtt{Com}(\Gamma), Q_{\circledcirc} \times \mathtt{Com}(\Gamma)$ and a stack alphabet Γ' that gives a winning strategy of $\mathcal{G}_{\mathcal{A}}$. That is, $\rho \in PL$ is winning for Player I if $f_1(\rho) \in L(\mathcal{T})$.*

By Lemma 7.2.2, a winning strategy can be also given as a subset of sequences $w \in (\Sigma_{\mathbb{i}} \cdot \Sigma_{\circledcirc})^{\omega}$ such that the play $f_2(w)$ is winning for Player I. Thus, we can obtain the following lemma in a similar way to Theorem 7.2.1.

Corollary 7.2.1. *If player I has a winning strategy of $\mathcal{G}_{\mathcal{A}}$, we can construct a PDT $\mathcal{T}$ over $\Sigma_{\mathbb{i}}, \Sigma_{\circledcirc}$ and a stack alphabet Γ' that gives a winning strategy of $\mathcal{G}_{\mathcal{A}}$. That is, $f_2(w) \in PL$ is winning for Player I if $w \in L(\mathcal{T})$.*

7.3 Realizability Problems for PDA and PDT

For a specification S and an implementation I, we write $I \models S$ if $L(I) \subseteq L(S)$.

Definition 7.3.1. *Realizability problem* Real*$(\mathcal{S}, \mathcal{I})$ for a class of specifications $\mathcal{S}$ and of implementations $\mathcal{I}$: For a specification $S \in \mathcal{S}$, is there an implementation $I \in \mathcal{I}$ such that $I \models S$?*

Example 7.3.1. *By Examples 7.2.1 and 7.2.2,* $L(\mathcal{T}_1) \subseteq L(\mathcal{A}_1)$ *holds for PDT* $\mathcal{T}_1$ *and DPDA* $\mathcal{A}_1$ *defined in the examples. Thus,* $\mathcal{T}_1 \models \mathcal{A}_1$ *holds.*

Theorem 7.3.1. REAL(**DPDA**, **PDT**) *is in EXPTIME.*

Proof. Let $\mathcal{A}$ be a given DPDA. By definitions, $w \in L(\mathcal{A})$ iff $f_2(w)$ is a winning play for Player I of $\mathcal{G}_\mathcal{A}$. By Corollary 7.2.1, if Player I has a winning strategy, we can construct a PDT $\mathcal{T}$ such that $f_2(w)$ is a winning play of $\mathcal{G}_\mathcal{A}$ if $w \in L(\mathcal{T})$. Hence, $\mathcal{T} \models \mathcal{A}$ holds. If Player I does not have a winning strategy, there is no $\mathcal{T}$ such that $\mathcal{T} \models \mathcal{A}$. Because there is an EXPTIME algorithm for constructing $\mathcal{T}$ (if exists) in [72], REAL(**DPDA**, **PDT**) is in EXPTIME.

Theorem 7.3.2. REAL(**NPDA**, **PDT**) *is undecidable.*

Proof. We prove the theorem by a reduction from the universality problem of NPDA, which is undecidable. For a given NPDA $\mathcal{A} = (Q, Q_{\mathbb{i}}, Q_{\mathbb{o}}, q_0, z_0, \delta, c)$ over $\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}$ and Γ, we can construct an NPDA $\mathcal{A}' = (Q \times [2], Q \times \{1\}, Q \times \{2\}, q_0, z_0, \delta', c')$ over $\Sigma'_{\mathbb{i}}, \Sigma'_{\mathbb{o}}$ and Γ where $\Sigma'_{\mathbb{i}} = \Sigma_{\mathbb{i}} \cup \Sigma_{\mathbb{o}}$, $\Sigma'_{\mathbb{o}}$ is an arbitrary (nonempty) alphabet, $c'((q,1)) = c'((q,2)) = c(q)$ for all $q \in Q$ and $((q,1), a, z) \to ((q',2), \texttt{com}) \in \delta'$ iff $(q, a, z) \to (q', \texttt{com}) \in \delta$, and $((q',2), b, z) \to ((q',1), \texttt{skip}) \in \delta'$ for all $b \in \Sigma'_{\mathbb{o}}$ and $z \in \Gamma$.

We show $L(\mathcal{A}) = (\Sigma'_{\mathbb{i}})^\omega$ iff there exists $\mathcal{T}$ such that $\mathcal{T} \models \mathcal{A}$. By the construction of $\mathcal{A}'$, $L(\mathcal{A}') = \langle L(\mathcal{A}), (\Sigma'_{\mathbb{o}})^\omega \rangle$ holds. If $L(\mathcal{A}) = (\Sigma'_{\mathbb{i}})^\omega$, then $L(\mathcal{A}') = \langle (\Sigma'_{\mathbb{i}})^\omega, (\Sigma'_{\mathbb{o}})^\omega \rangle$ and thus $\mathcal{T} \models \mathcal{A}$ holds for every $\mathcal{T}$. Assume that $L(\mathcal{A}) \neq (\Sigma'_{\mathbb{i}})^\omega$. Then, there exists a word $w \in (\Sigma'_{\mathbb{i}})^\omega$ such that $w \notin L(\mathcal{A})$. For any PDT $\mathcal{T}$ and any $u \in (\Sigma'_{\mathbb{i}})^\omega$, there is $v \in (\Sigma'_{\mathbb{o}})^\omega$ such that $\langle u, v \rangle \in L(\mathcal{A}')$. On the other hand, $\langle w, v \rangle \notin L(\mathcal{A}')$ holds for any $v \in (\Sigma'_{\mathbb{o}})^\omega$. Hence, $\mathcal{T} \not\models \mathcal{A}'$ holds for any PDT $\mathcal{T}$. This completes the reduction and the realizability problem for NPDA and PDT is undecidable.

7.4 Register Pushdown Transducers and Automata

7.4.1 Data words and registers

We assume a countable set D of *data values*. For finite alphabets $\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}$, an infinite sequence $(a_1^{\mathbb{i}}, d_1)(a^{\mathbb{o}}, d'_1) \cdots \in ((\Sigma_{\mathbb{i}} \times D) \cdot (\Sigma_{\mathbb{o}} \times D))^\omega$ is called a *data*

word. We let $\mathtt{DW}(\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, D) = ((\Sigma_{\mathbb{i}} \times D) \cdot (\Sigma_{\mathbb{o}} \times D))^{\omega}$. We define the projection $\mathtt{Lab} : \Sigma \times D \to \Sigma$ as $\mathtt{Lab}((a,d)) = a$ for $(a,d) \in \Sigma \times D$. For $k \in \mathbb{N}_0$, a mapping $\theta : [k] \to D$ is called an *assignment* (of data values to k registers). Let Θ_k denote the collection of assignments to k registers. We assume $\perp \in D$ as the initial data value and let $\theta^k_{\perp} \in \Theta_k$ be the initial assignment such that $\theta^k_{\perp}(i) = \perp$ for all $i \in [k]$.

We denote $\mathtt{Tst}_k = \mathscr{P}([k] \cup \{top\})$ and $\mathtt{Asgn}_k = \mathscr{P}([k])$ where $top \notin \mathbb{N}$ is a unique symbol that represents a stack top value. $\mathtt{Tst}_k$ is the set of guard conditions. For $\mathtt{tst} \in \mathtt{Tst}_k$, $\theta \in \Theta_k$ and $d, e \in D$, we denote $\langle \theta, d, e \rangle = \mathtt{tst}$ if $(\theta(i) = d \Leftrightarrow i \in \mathtt{tst})$ and $(e = d \Leftrightarrow top \in \mathtt{tst})$ hold. In the definitions of register pushdown transducer and automaton in the next section, the data values d and e correspond to an input data value and a stack top data value, respectively. $\mathtt{Asgn}_k$ is the set of assignment conditions. For $\mathtt{asgn} \in \mathtt{Asgn}_k$, $\theta \in \Theta_k$ and $d \in D$, let $\theta[\mathtt{asgn} \leftarrow d]$ be the assignment $\theta' \in \Theta_k$ such that $\theta'(i) = d$ for $i \in \mathtt{asgn}$ and $\theta'(i) = \theta(i)$ for $i \notin \mathtt{asgn}$.

7.4.2 Register pushdown transducers

Definition 7.4.1. *A register pushdown transducer with k registers (k-RPDT) over finite alphabets $\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}$ and Γ is $\mathcal{T} = (P, p_0, z_0, \Delta)$ where P is a finite set of states, $p_0 \in P$ is the initial state, $z_0 \in \Gamma$ is the initial stack symbol and $\Delta : P \times \Sigma_{\mathbb{i}} \times \mathtt{Tst}_k \times \Gamma \to P \times \Sigma_{\mathbb{o}} \times \mathtt{Asgn}_k \times [k] \times \mathtt{Com}(\Gamma \times [k])$ is a finite set of deterministic transition rules.*

For $u \in (\Gamma \times D)^+$, $\theta' \in \Theta_k$ and $\mathtt{com} \in \mathtt{Com}(\Gamma \times [k])$, let us define $\mathtt{upds}(u, \theta', \mathtt{com})$ as $\mathtt{upds}(u, \theta', \mathtt{pop}) = u(1:)$, $\mathtt{upds}(u, \theta', \mathtt{skip}) = u$ and $\mathtt{upds}(u, \theta', \mathtt{push}((z, j'))) = (z, \theta'(j'))u$. Let $ID_{\mathcal{T}} = P \times \Theta_k \times (\Gamma \times D)^*$ and $\Rightarrow_{\mathcal{T}} \subseteq ID_{\mathcal{T}} \times ((\Sigma_{\mathbb{i}} \times D) \cdot (\Sigma_{\mathbb{o}} \times D)) \times ID_{\mathcal{T}}$ be the transition relation of $\mathcal{T}$ such that $((p, \theta, u), (a, d^{\mathbb{i}})(b, d^{\mathbb{o}}), (q, \theta', u')) \in \Rightarrow_{\mathcal{T}}$ iff there exists a rule $(p, a, \mathtt{tst}, z) \to (q, b, \mathtt{asgn}, j, \mathtt{com}) \in \Delta$ that satisfies the following conditions: $\langle \theta, d^{\mathbb{i}}, snd(u(0)) \rangle = \mathtt{tst}$, $\theta' = \theta[\mathtt{asgn} \leftarrow d^{\mathbb{i}}]$, $\theta'(j) = d^{\mathbb{o}}$, $z = fst(u(0))$ and $u' = \mathtt{upds}(u, \theta', \mathtt{com})$, and we write $(p, \theta, u) \Rightarrow_{\mathcal{T}}^{(a,d^{\mathbb{i}})(b,d^{\mathbb{o}})} (q, \theta', u')$.

A run and the language $L(\mathcal{T})$ of $\mathcal{T}$ are those of deterministic 0-TS $(ID_{\mathcal{T}}, (q_0, \theta^k_{\perp}, (z_0, \perp)), (\Sigma_{\mathbb{i}} \times D) \cdot (\Sigma_{\mathbb{o}} \times D), \emptyset, \Rightarrow_{\mathcal{T}}, c)$ where $c(s) = 2$ for all $s \in ID_{\mathcal{T}}$. In this

paper, we assume that no run of RPDT reaches an ID whose stack is empty. Let $\mathbf{RPDT}[k]$ be the class of k-RPDT and $\mathbf{RPDT}{=}\bigcup_{k\in\mathbb{N}_0}\mathbf{RPDT}[k]$.

7.4.3 Register pushdown automata

Definition 7.4.2. *A nondeterministic register pushdown automaton with k registers (k-NRPDA) over $\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}$ and Γ is $\mathcal{A} = (Q, Q_{\mathbb{i}}, Q_{\mathbb{o}}, q_0, z_0, \delta, c)$, where Q is a finite set of states, $Q_{\mathbb{i}} \cup Q_{\mathbb{o}} = Q, Q_{\mathbb{i}} \cap Q_{\mathbb{o}} = \emptyset$, $q_0 \in Q$ is the initial state, $z_0 \in \Gamma$ is the initial stack symbol, $c : Q \to [n]$ where $n \in \mathbb{N}$ is the number of priorities and $\delta : Q \times (\Sigma \cup \{\tau\}) \times \mathtt{Tst}_k \times \Gamma \to \mathscr{P}(Q \times \mathtt{Asgn}_k \times \mathtt{Com}(\Gamma \times [k]))$ is a transition function having one of the forms:*

- $(q_{\mathbb{x}}, a_{\mathbb{x}}, \mathtt{tst}, z) \to (q_{\overline{\mathbb{x}}}, \mathtt{asgn}, \mathtt{com})$ *(input/output rule)*
- $(q_{\mathbb{x}}, \tau, \mathtt{tst}, z) \to (q'_{\mathbb{x}}, \mathtt{asgn}, \mathtt{com})$ *(τ rules, where $\tau \notin \Sigma$)*

where $(\mathbb{x}, \overline{\mathbb{x}}) \in \{(\mathbb{i}, \mathbb{o}), (\mathbb{o}, \mathbb{i})\}$, $q_{\mathbb{x}}, q'_{\mathbb{x}} \in Q_{\mathbb{x}}, q_{\overline{\mathbb{x}}} \in Q_{\overline{\mathbb{x}}}, a_{\mathbb{x}} \in \Sigma_{\mathbb{x}}$, $\mathtt{tst} \in \mathtt{Tst}_k$, $z \in \Gamma$, $\mathtt{asgn} \in \mathtt{Asgn}_k$ *and* $\mathtt{com} \in \mathtt{Com}(\Gamma \times [k])$.

Let $ID_{\mathcal{A}} = Q \times \Theta_k \times (\Gamma \times D)^*$. We define the transition relation $\vdash_{\mathcal{A}} \subseteq ID_{\mathcal{A}} \times ((\Sigma \cup \{\tau\}) \times D) \times ID_{\mathcal{A}}$ as $((q, \theta, u), (a, d), (q', \theta', u')) \in \vdash_{\mathcal{A}}$, written as $(q, \theta, u) \vdash_{\mathcal{A}}^{(a,d)} (q', \theta', u')$, iff there exists a rule $(p, a, \mathtt{tst}, z) \to (q, \mathtt{asgn}, \mathtt{com}) \in \delta$ such that $\langle \theta, d, snd(u(0))\rangle = \mathtt{tst}$, $\theta' = \theta[\mathtt{asgn} \leftarrow d]$, $z = fst(u(0))$ and $u' = \mathtt{upds}(u, \theta', \mathtt{com})$. For $s, s' \in ID_{\mathcal{A}}$ and $w \in ((\Sigma_{\mathbb{i}} \times D) \cdot (\Sigma_{\mathbb{o}} \times D))^m$, we write $s \vdash^w s'$ if there exists $\rho \in ID_{\mathcal{A}}^{m+1}$ such that $\rho(0) = s, \rho(m) = s'$, and $\rho(0) \vdash^{w(0)} \cdots \vdash^{w(m-1)} \rho(m)$.

A run and the language $L(\mathcal{A})$ of k-DRPDA $\mathcal{A}$ are those of TS $\mathcal{S}_{\mathcal{A}} = (ID_{\mathcal{A}}, (q_0, \theta_{\perp}^k, (z_0, \perp)), \Sigma \times D, \{\tau\} \times D, \vdash_{\mathcal{A}}, c')$ where $c'((q, \theta, u)) = c(q)$ for all $(q, \theta, u) \in ID_{\mathcal{A}}$. We call $\mathcal{A}$ deterministic, or k-DRPDA, if $\mathcal{S}_{\mathcal{A}}$ is deterministic. We call $\mathcal{A}$ an (m, k)-NRPDA (or an (m, k)-DRPDA when $\mathcal{A}$ is deterministic) if $\mathcal{S}_{\mathcal{A}}$ is an m-TS. We abbreviate $(0, k)$-NRPDA ($(0, k)$-DPDA) as k-NRPDA (k-DRPDA). Let **DRPDA** and **NRPDA** be the unions of k-DRPDA and k-NRPDA for all $k \in \mathbb{N}_0$, respectively.

For simplicity, we assume that the set of stack alphabet Γ is the singleton $\{z\}$. We abbreviate k-RPDT $\mathcal{T} = (P, p_0, z_0, \Delta)$ as $\mathcal{T} = (P, p_0, \Delta)$, the set of all IDs of k-RPDT $P \times \Theta_k \times (\Gamma \times D)^+$ as $P \times \Theta_k \times D^+$, the stack command

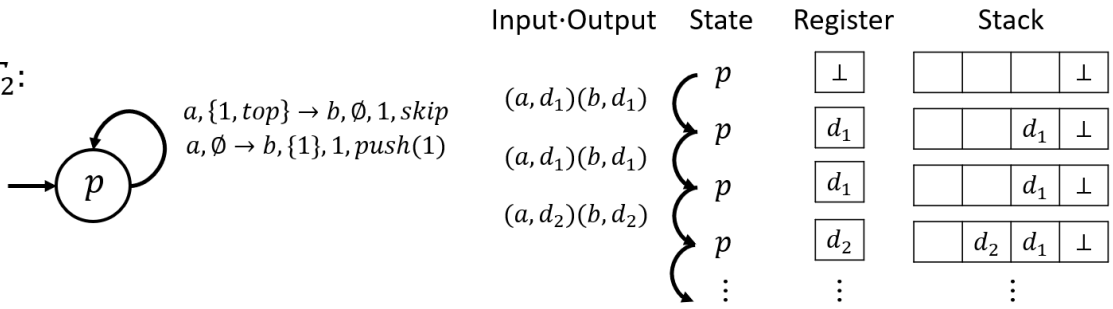

Figure 7.2: A figure of $\mathcal{T}_2$ (left) and the run (ρ, w) of $\mathcal{T}_2$ (right). A label $a, \mathtt{tst} \to b, \mathtt{asgn}, 1, \mathtt{com}$ means $(p, a, \mathtt{tst}) \to (p, b, \mathtt{asgn}, 1, \mathtt{com}) \in \Delta$

$\mathtt{Com}(\Gamma \times [k])$ as $\mathtt{Com}([k])$, every rule $(p, a, \mathtt{tst}, z) \to (q, b, \mathtt{asgn}, j, \mathtt{com})$ of $\mathcal{T}$ as $(p, a, \mathtt{tst}) \to (q, b, \mathtt{asgn}, j, \mathtt{com}')$ where $\mathtt{com} \in \mathtt{Com}(\Gamma \times [k])$ and $\mathtt{com}' \in \mathtt{Com}([k])$ such that $\mathtt{com}' = \mathtt{com}$ if $\mathtt{com} = \mathtt{pop}$ or $\mathtt{skip}$ and $\mathtt{com}' = \mathtt{push}(j')$ if $\mathtt{com}' = \mathtt{push}(z, j')$ for some $j' \in [k]$. except in the proof of Theorem 7.5.1. We apply a similar abbreviation to those of RPDA.

Example 7.4.1. *Let us consider a* 1*-RPDT* $\mathcal{T}_2 = (\{p\}, p, \Delta)$ *over* $\{a\}, \{b\}$ *and* $\{z\}$ *where* $\Delta = \{(p, a, \{1, top\}) \to (p, b, \emptyset, 1, \mathtt{skip}), (p, a, \emptyset) \to (p, b, \{1\}, 1, \mathtt{push}(1))\}$. *(See Fig. 7.2.) Let* $\rho = (p, [\perp], \perp)(p, [d_1], d_1\perp)(p, [d_1], d_1\perp)(p, [d_2], d_2d_1\perp)\cdots$ *where* $[d] \in \Theta_1$ *is the assignment such that* $[d](1) = d$*, and* $w = (a, d_1)(b, d_1)(a, d_1)$ $(b, d_1)(a, d_2)(b, d_2)\cdots$. *Then* (ρ, w) *is a run of* $\mathcal{T}_2$.

7.4.4 Visibly RPDA

Let $\mathtt{Com}_v = \{\mathtt{pop}, \mathtt{skip}, \mathtt{push}\}$ and $v : \mathtt{Com}([k]) \to \mathtt{Com}_v$ be the function such that $v(\mathtt{push}(j)) = \mathtt{push}$ for $j \in [k]$ and $v(\mathtt{com}) = \mathtt{com}$ otherwise. We say that a k-DRPDA $\mathcal{A}$ over $\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}$ and Γ visibly manipulates its stack (or a *stack-visibly* RPDA) if there exists a function $\mathtt{vis} : \Sigma \to \mathtt{Com}_v$ such that every rule $(q, a, \mathtt{tst}) \to (q', \mathtt{asgn}, \mathtt{com})$ of $\mathcal{A}$ satisfies $\mathtt{vis}(a) = v(\mathtt{com})$. We also define a stack-visibly PDA in a similar way. Stack-visibility of RPDA will be used in the proof of Lemma 7.5.2 in order to take the intersection of the two DRPDA.

Also, we say that $\mathcal{A}$ is a *test-visibly* DRPDA if there exists a function $\mathtt{vis}_t : \Sigma \to \mathtt{Tst}_k$ such that every rule $(q, a, \mathtt{tst}) \to (q', \mathtt{asgn}, \mathtt{com})$ of $\mathcal{A}$ satisfies $\mathtt{vis}_t(a) =$

$\mathtt{tst}$. In the next subsection, we prove that the projection of the language recognized by an NRPDA $\mathcal{A}$ onto the finite alphabets can be recognized by an NPDA $\mathcal{A}'$, and if $\mathcal{A}$ is a test-visibly DRPDA, $\mathcal{A}'$ is deterministic.

If $\mathcal{A}$ is a stack-visibly and test-visibly DRPDA, we call $\mathcal{A}$ a visibly DRPDA. Let **DRPDAv** be the union of visibly k-DRPDA for all $k \in \mathbb{N}_0$, respectively.

7.4.5 PDA simulating RPDA

Let $\phi_\bot \in \Phi_k$ be the equivalence relation satisfying $a \sim_{\phi_\bot} b$ for any $a, b \in X_k$.

For $\mathtt{tst} \subseteq [k] \cup \{top\}$ and $\mathtt{asgn} \subseteq [k]$, define a subset $\Phi_k^{\mathtt{tst},\mathtt{asgn}}$ of Φ_k as:

$$\begin{aligned}\Phi_k^{\mathtt{tst},\mathtt{asgn}} = \{\phi \in \Phi_k \mid\ & (\forall i \in \mathtt{tst} :\ \ \forall j \in [k] \cup \{top\} : j \in \mathtt{tst} \Leftrightarrow \mathtt{x}_i \sim_\phi \mathtt{x}_j),\\ & (\forall i \in \mathtt{asgn} : \forall j \in [k] \cup \{top\} : j \in \mathtt{tst} \Leftrightarrow \mathtt{x}_j \sim_\phi \mathtt{x}'_i),\\ & (\forall i, j \in \mathtt{asgn} : \mathtt{x}'_i \sim_\phi \mathtt{x}'_j), (\forall i \in [k] \setminus \mathtt{asgn} : \mathtt{x}_i \sim_\phi \mathtt{x}'_i)\}.\end{aligned}$$

For $j \in [k]$, define $\Phi_k^{=,j} = \{\phi \in \Phi_k \mid \mathtt{top} \sim_\phi \mathtt{x}_j,\ \forall i \in [k] : \mathtt{x}_i \sim_\phi \mathtt{x}'_i\}$. By definition, $\langle \theta, e, \theta' \rangle = \phi$ for $\phi \in \Phi_k^{\mathtt{tst},\mathtt{asgn}}$ iff $\langle \theta, d, e \rangle = \mathtt{tst}$ and $\theta' = \theta[\mathtt{asgn} \leftarrow d]$ for some $d \in D$. Similarly, $\langle \theta, e, \theta' \rangle = \phi$ for $\phi \in \Phi_k^{=,j}$ iff $\theta' = \theta$ and $\theta(j) = e$.

Let $\odot$ and $\odot_{\mathtt{T}}$ be binary predicates over Φ_k defined as:

$$\begin{aligned}\phi_1 \odot \phi_2 \ &:\Leftrightarrow\ \Big(\mathtt{x}'_i \sim_{\phi_1} \mathtt{x}'_j \Leftrightarrow \mathtt{x}_i \sim_{\phi_2} \mathtt{x}_j \text{ for } i, j \in [k]\Big).\\ \phi_1 \odot_{\mathtt{T}} \phi_2 \ &:\Leftrightarrow\ \Big(\phi_1 \odot \phi_2 \text{ and } (\mathtt{x}'_i \sim_{\phi_1} \mathtt{top} \Leftrightarrow \mathtt{x}_i \sim_{\phi_2} \mathtt{top} \text{ for } i \in [k])\Big).\end{aligned}$$

Below we will define the *composition* of two equivalence relations, and $\phi_1 \odot \phi_2$ means that ϕ_1 and ϕ_2 are composable. For $\phi \in \Phi_k$ and $\Phi' \subseteq \Phi_k$, let $\phi \odot \Phi' = \{\phi' \in \Phi' \mid \phi \odot \phi'\}$ and $\phi \odot_{\mathtt{T}} \Phi' = \{\phi' \in \Phi' \mid \phi \odot_{\mathtt{T}} \phi'\}$. By definition, $\phi \odot_{\mathtt{T}} \Phi_k^{\mathtt{tst},\mathtt{asgn}}$ consists of at most one equivalence relation for any $\phi \in \Phi_k$, $\mathtt{tst} \subseteq [k] \cup \{top\}$, and $\mathtt{asgn} \subseteq [k]$. Similarly, $\phi \odot \Phi_k^{=,j}$ consists of exactly one equivalence relation for any $\phi \in \Phi_k$ and $j \in [k]$.

For $\phi_1, \phi_2 \in \Phi_k$ with $\phi_1 \odot \phi_2$, the *composition* $\phi_1 \circ \phi_2$ of them is the equivalence relation in Φ_k that satisfies the followings:

$$\begin{aligned}\mathtt{x}_i \sim_{\phi_1} \mathtt{x}_j \Leftrightarrow \mathtt{x}_i \sim_{\phi_1 \circ \phi_2} \mathtt{x}_j \text{ for } i, j \in [k] \cup \{top\},\\ \mathtt{x}'_i \sim_{\phi_2} \mathtt{x}'_j \Leftrightarrow \mathtt{x}'_i \sim_{\phi_1 \circ \phi_2} \mathtt{x}'_j \text{ for } i, j \in [k],\\ (\exists l \in [k] : \mathtt{x}_i \sim_{\phi_1} \mathtt{x}'_l \wedge \mathtt{x}_l \sim_{\phi_2} \mathtt{x}'_j) \Leftrightarrow \mathtt{x}_i \sim_{\phi_1 \circ \phi_2} \mathtt{x}'_j \text{ for } i \in [k] \cup \{top\},\ j \in [k].\end{aligned}$$

By definition, $\circ$ is associative. We say that $\theta_1, d_1, \theta_2, \theta_3$ satisfy the *freshness* property if for every $i, j \in [k]$, $\Big(\theta_1(i) \neq \theta_2(l)$ for all $l \in [k]$ implies $\theta_1(i) \neq \theta_3(j)\Big)$ and $\Big(d_1 \neq \theta_2(l)$ for all $l \in [k]$ implies $d_1 \neq \theta_3(j)\Big)$. By definition, if $\langle \theta_1, d_1, \theta_2 \rangle = \phi_1$ and $\langle \theta_2, d_2, \theta_3 \rangle = \phi_2$ and $\theta_1, d_1, \theta_2, \theta_3$ satisfy the freshness property, then $\langle \theta_1\, d_1, \theta_3 \rangle = \phi_1 \circ \phi_2$. We extend the freshness property to a sequence $\theta_0, d_0, \theta_1, d_1, \ldots, d_{n-1}, \theta_n$. This sequence satisfies the freshness property if $\theta_i, d_i, \theta_l, \theta_j$ satisfy the property for every i, l, j such that $0 \leq i < l < j \leq n$.

Similarly, we define $\phi_1 \circ_{\mathtt{T}} \phi_2$ for $\phi_1, \phi_2 \in \Phi_k$ with $\phi_1 \odot_{\mathtt{T}} \phi_2$ as follows:

$$\mathtt{x}_i \sim_{\phi_1} \mathtt{x}_j \Leftrightarrow \mathtt{x}_i \sim_{\phi_1 \circ_{\mathtt{T}} \phi_2} \mathtt{x}_j \text{ for } i, j \in [k] \cup \{top\},$$
$$\mathtt{x}'_i \sim_{\phi_2} \mathtt{x}'_j \Leftrightarrow \mathtt{x}'_i \sim_{\phi_1 \circ_{\mathtt{T}} \phi_2} \mathtt{x}'_j \text{ for } i, j \in [k],$$
$$\Big((\exists l \in [k] : \mathtt{x}_i \sim_{\phi_1} \mathtt{x}'_l \wedge \mathtt{x}_l \sim_{\phi_2} \mathtt{x}'_j)$$
$$\vee\, (\mathtt{x}_i \sim_{\phi_1} \mathtt{top} \wedge \mathtt{top} \sim_{\phi_2} \mathtt{x}'_j)\Big) \Leftrightarrow \mathtt{x}_i \sim_{\phi_1 \circ_{\mathtt{T}} \phi_2} \mathtt{x}'_j \text{ for } i \in [k] \cup \{top\},\ j \in [k].$$

2 By definition, $\circ_{\mathtt{T}}$ is associative.

Let $\mathcal{A} = (Q, Q_{\mathbb{i}}, Q_{\mathbb{o}}, q_0, \delta, c)$ be a k-NRPDA over $\Sigma_{\mathbb{i}}$, $\Sigma_{\mathbb{o}}$, and Γ. As mentioned in Section 7.4.3, we assume that Γ is a singleton and use the simplified definition of δ for readability. Note that we can extend the following results to arbitrary Γ by replacing Φ_k used as the stack alphabet of the constructed PDA with $\Gamma \times \Phi_k$. From $\mathcal{A}$, we construct a PDA $\mathcal{A}' = (Q', Q'_{\mathbb{i}}, Q'_{\mathbb{o}}, q'_0, \phi_\perp, \delta', c')$ over $\Sigma_{\mathbb{i}}$, $\Sigma_{\mathbb{o}}$, and Φ_k, where $Q' = Q \times \Phi_k$, $Q'_{\mathbb{i}} = Q_{\mathbb{i}} \times \Phi_k$, $Q'_{\mathbb{o}} = Q_{\mathbb{o}} \times \Phi_k$, $q'_0 = (q_0, \phi_\perp)$, $c'((q, \phi)) = c(q)$ for any $q \in Q$ and $\phi \in \Phi_k$, and for any $(q, \phi_2) \in Q'$, $a \in \Sigma \cup \{\tau\}$, and $\phi_1 \in \Phi_k$, $\delta'((q, \phi_2), a, \phi_1)$ is the smallest set satisfying the following inference rules:

$$\frac{\delta(q, a, \mathtt{tst}) \ni (q', \mathtt{asgn}, \mathtt{skip}),\ \phi_1 \odot \phi_2,\ \phi_3 \in \phi_2 \odot_{\mathtt{T}} \Phi_k^{\mathtt{tst},\mathtt{asgn}}}{\delta'((q, \phi_2), a, \phi_1) \ni ((q', \phi_2 \circ_{\mathtt{T}} \phi_3), \mathtt{skip})} \tag{7.1}$$

$$\frac{\delta(q, a, \mathtt{tst}) \ni (q', \mathtt{asgn}, \mathtt{pop}),\ \phi_1 \odot \phi_2,\ \phi_3 \in \phi_2 \odot_{\mathtt{T}} \Phi_k^{\mathtt{tst},\mathtt{asgn}}}{\delta'((q, \phi_2), a, \phi_1) \ni ((q', \phi_1 \circ (\phi_2 \circ_{\mathtt{T}} \phi_3)), \mathtt{pop})} \tag{7.2}$$

$$\frac{\delta(q, a, \mathtt{tst}) \ni (q', \mathtt{asgn}, \mathtt{push}(j)),\ \phi_1 \odot \phi_2,\ \phi_3 \in \phi_2 \odot_{\mathtt{T}} \Phi_k^{\mathtt{tst},\mathtt{asgn}},\ \phi_4 \in \phi_3 \odot \Phi_k^{=,j}}{\delta'((q, \phi_2), a, \phi_1) \ni ((q', \phi_4), \mathtt{push}(\phi_2 \circ_{\mathtt{T}} \phi_3))} \tag{7.3}$$

Note that if $\mathcal{A}$ is a test-visibly DRPDA, $\mathcal{A}'$ is deterministic. The number of equivalence relations in Φ_k equals the $(2k+1)$th Bell number and is $2^{O(k \log k)}$.

When constructing δ', we choose arbitrary ϕ_1 and ϕ_2 for each transition rule of $\mathcal{A}$ in general, and thus the size of $\mathcal{A}'$ is exponential to the one of $\mathcal{A}$.

The main idea of this construction is as follows: $\mathcal{A}'$ simulates $\mathcal{A}$ without keeping data values in the stack. When `pop` is performed, $\mathcal{A}'$ must know whether or not the data value in the new stack top of $\mathcal{A}$ equals the *current* value of each register. For this purpose, $\mathcal{A}'$ keeps an abstract "history" of the register assignments in the stack, which tells whether each of the data values in the stack of $\mathcal{A}$ equals the current value of each register. The precise meanings of the stack of $\mathcal{A}'$ will become clear by considering the `Lab`-bisimulation relation shown in the proof of Lemma 7.4.1 below.

Let $\mathcal{S}_{\mathcal{A}'} = (ID_{\mathcal{A}'}, (q'_0, \phi_\bot), \Sigma, \{\tau\}, \vdash_{\mathcal{A}'}, c_{\mathcal{A}'})$ be the TS that represents the semantics of $\mathcal{A}'$. We define a TS $\mathcal{S}^{\mathrm{aug}}_{\mathcal{A}} = (ID^{\mathrm{aug}}_{\mathcal{A}}, (q_0, \theta_\bot, (\bot, \theta_\bot)), \Sigma \times D, \{\tau\} \times D, \vdash_{\mathcal{A}^{\mathrm{aug}}}, c_{\mathcal{A}})$ where $ID^{\mathrm{aug}}_{\mathcal{A}} = Q \times \Theta_k \times (D \times \Theta_k)^*$, $c_{\mathcal{A}}((q, \theta, u)) = c(q)$ for any $(q, \theta, u) \in ID^{\mathrm{aug}}_{\mathcal{A}}$, and $\vdash_{\mathcal{A}^{\mathrm{aug}}}$ is defined as follows: $(q, \theta, u) \vdash^{(a,d)}_{\mathcal{A}^{\mathrm{aug}}} (q', \theta', u')$ iff $\delta(q, a, \mathtt{tst}) \ni (q', \mathtt{asgn}, \mathtt{com})$, $\langle \theta, d, \mathit{fst}(u(0)) \rangle = \mathtt{tst}$, $\theta' = \theta[\mathtt{asgn} \leftarrow d]$, and $u' = u(1{:}), u$, or $(\theta'(j'), \theta')u$ if $\mathtt{com} = \mathtt{pop}$, `skip`, or $\mathtt{push}(j')$, respectively. $\mathcal{S}^{\mathrm{aug}}_{\mathcal{A}}$ is essentially the same as the TS $\mathcal{S}_{\mathcal{A}}$ for $\mathcal{A}$, but $\mathcal{S}^{\mathrm{aug}}_{\mathcal{A}}$ "saves" the current register assignment in the stack when performing `push`. The saved assignments do not take part in transitions and thus the behavior of $\mathcal{S}^{\mathrm{aug}}_{\mathcal{A}}$ is the same as $\mathcal{S}_{\mathcal{A}}$. Additionally, we define the freshness property of $\mathcal{S}^{\mathrm{aug}}_{\mathcal{A}}$ as follows: A transition $(q, \theta, u) \vdash^{(a,d)}_{\mathcal{A}^{\mathrm{aug}}} (q', \theta', u')$ of $\mathcal{S}^{\mathrm{aug}}_{\mathcal{A}}$ by a rule $\delta(q, a, \mathtt{tst}) \ni (q', \mathtt{asgn}, \mathtt{com})$ with $\mathtt{tst} = \emptyset$ and $\mathtt{asgn} \neq \emptyset$ is allowed only when d does not appear in any saved assignment in u. Intuitively, this property means that when `tst` designates a data value not in the registers or the stack top, the RPDA chooses a *fresh* data value that has never been used before. (Note that the freshness property for RA was introduced by Tzevelekos [69].)

We assume that $\mathcal{S}^{\mathrm{aug}}_{\mathcal{A}}$ satisfies the freshness property. This assumption guarantees that for every $(q, \theta_n, (d_{n-1}, \theta_{n-1}) \dots (d_1, \theta_1)(d_0, \theta_0)) \in ID^{\mathrm{aug}}_{\mathcal{A}}$ reachable from the initial state of $\mathcal{S}^{\mathrm{aug}}_{\mathcal{A}}$, (i) the sequence $\theta_0, d_0, \theta_1, d_1, \dots, d_{n-1}, \theta_n$ satisfies the freshness property; (ii) for $\theta' = \theta_n[\mathtt{asgn} \leftarrow d]$ where d is a data value chosen by the RPDA to satisfy $\langle \theta_n, d, d_{n-1} \rangle = \mathtt{tst}$, $\langle \theta_{n-1}, d_{n-1}, \theta_n \rangle = \phi_1$ and $\langle \theta_n, d_{n-1}, \theta' \rangle = \phi_2$ imply $\langle \theta_{n-1}, d_{n-1}, \theta' \rangle = \phi_1 \circ_{\mathtt{T}} \phi_2$.

Each equivalence relation in the stack of $\mathcal{A}'$ represents the relation among each data value in the stack and two adjacent saved assignments of $\mathcal{S}^{\mathrm{aug}}_{\mathcal{A}}$, which yields $\mathtt{Lab}$-bisimilarity from $\mathcal{S}^{\mathrm{aug}}_{\mathcal{A}}$ to $\mathcal{S}_{\mathcal{A}'}$, as shown in the following lemma.

Lemma 7.4.1. *If $\mathcal{S}^{\mathrm{aug}}_{\mathcal{A}}$ satisfies the freshness property, then $\mathcal{S}^{\mathrm{aug}}_{\mathcal{A}}$ is $\mathtt{Lab}$-bisimilar to $\mathcal{S}_{\mathcal{A}'}$.*

Proof. Let $R \subseteq ID^{\mathrm{aug}}_{\mathcal{A}} \times ID_{\mathcal{A}'}$ be the relation that satisfies for every $q \in Q$, $u = (d_{n-1}, \theta_{n-1}) \dots (d_1, \theta_1)(d_0, \theta_0) \in (D \times \Theta_k)^*$, and $v = \phi_{n-1} \dots \phi_1 \phi_0 \in \Phi_k^*$, $((q, \theta_n, u), ((q, \phi_n), v)) \in R$ iff $\forall i \in [n] : \theta_{i-1}, d_{i-1}, \theta_i \models \phi_i$ and $\theta_\perp, \perp, \theta_0 \models \phi_0$ and (q, θ_n, u) is reachable from the initial state of $\mathcal{S}^{\mathrm{aug}}_{\mathcal{A}}$. We can show that R is a $\mathtt{Lab}$-bisimulation relation from $\mathcal{S}^{\mathrm{aug}}_{\mathcal{A}}$ to $\mathcal{S}_{\mathcal{A}'}$.

Assume that $((q, \theta_n, u), ((q, \phi_n), v)) \in R$ for $u = (d_{n-1}, \theta_{n-1}) \dots (d_0, \theta_0)$ and $v = \phi_{n-1} \dots \phi_0$. Because $\forall i \in [n] : \theta_{i-1}, d_{i-1}, \theta_i \models \phi_i$ and $\theta_\perp, \perp, \theta_0 \models \phi_0$, $\forall i \in [n] : \phi_{i-1} \odot \phi_i$. If $(q, \theta_n, u) \vdash^{(a,d)}_{\mathcal{A}^{\mathrm{aug}}} (q', \theta', u')$, then there exist $\mathtt{tst}$, $\mathtt{asgn}$, $\mathtt{com}$, and d such that $\delta(q, a, \mathtt{tst}) \ni (q', \mathtt{asgn}, \mathtt{com})$, $(\theta_n, d, d_{n-1}) \models \mathtt{tst}$, $\theta' = \theta_n[\mathtt{asgn} \leftarrow d]$, and $u' = u(1{:}), u$, or $(\theta'(j'), \theta')u$ if $\mathtt{com} = \mathtt{pop}$, $\mathtt{skip}$, or $\mathtt{push}(j')$, respectively. There exists $\phi' \in \phi_n \odot_{\mathtt{T}} \Phi_k^{\mathtt{tst},\mathtt{asgn}}$ because $\theta_{n-1}, d_{n-1}, \theta_n \models \phi_n$ and $(\theta_n, d, d_{n-1}) \models \mathtt{tst}$. Thus, by definition, $\delta'((q, \phi_n), a, \phi_{n-1}) \ni ((q', \phi''), \mathtt{com}')$ where $\mathtt{com}' = \mathtt{pop}$ and $\phi'' = \phi_{n-1} \circ (\phi_n \circ_{\mathtt{T}} \phi')$ if $\mathtt{com} = \mathtt{pop}$, $\mathtt{com}' = \mathtt{skip}$ and $\phi'' = \phi_n \circ_{\mathtt{T}} \phi'$ if $\mathtt{com} = \mathtt{skip}$, and $\mathtt{com}' = \mathtt{push}(\phi_n \circ_{\mathtt{T}} \phi')$ and $\phi'' \in \phi' \odot \Phi_k^{=,j'}$ if $\mathtt{com} = \mathtt{push}(j')$. Therefore, $((q, \phi_n), v) \vdash^{a}_{\mathcal{A}'} ((q', \phi''), v')$ where $v' = \mathtt{upds}(v, \mathtt{com}')$. Because $\theta_{n-1}, d_{n-1}, \theta_n \models \phi_n$ and $\theta_n, d_{n-1}, \theta' \models \phi'$, $\theta_{n-1}, d_{n-1}, \theta' \models \phi_n \circ_{\mathtt{T}} \phi'$. In the case of $\mathtt{com} = \mathtt{pop}$, $\theta_{n-2}, d_{n-2}, \theta' \models \phi_{n-1} \circ (\phi_n \circ_{\mathtt{T}} \phi')$ because $\theta_{n-2}, d_{n-2}, \theta_{n-1} \models \phi_{n-1}$ and $\theta_{n-1}, d_{n-1}, \theta' \models \phi_n \circ_{\mathtt{T}} \phi'$. In the case of $\mathtt{com} = \mathtt{push}(j')$, $\theta', \theta'(j'), \theta' \models \phi''$ because $\phi'' \in \Phi_k^{=,j'}$. Hence, $((q', \theta', u'), ((q', \phi''), v')) \in R$ in any case.

On the other hand, if $((q, \phi_n), v) \vdash^{a}_{\mathcal{A}'} ((q', \phi''), v')$, then $\delta'((q, \phi_n), a, \phi_{n-1}) \ni ((q', \phi''), \mathtt{com}')$ where $v' = \mathtt{upds}(v, \mathtt{com}')$ holds. By definition, there exist $\mathtt{tst}$, $\mathtt{asgn}$, $\mathtt{com}$, and $\phi' \in \phi_n \odot_{\mathtt{T}} \Phi_k^{\mathtt{tst},\mathtt{asgn}}$ such that $\delta(q, a, \mathtt{tst}) \ni (q', \mathtt{asgn}, \mathtt{com})$, and $\mathtt{com} = \mathtt{pop}$ and $\phi'' = \phi_{n-1} \circ (\phi_n \circ_{\mathtt{T}} \phi')$ if $\mathtt{com}' = \mathtt{pop}$, $\mathtt{com} = \mathtt{skip}$ and $\phi'' = \phi_n \circ_{\mathtt{T}} \phi'$ if $\mathtt{com}' = \mathtt{skip}$, or $\mathtt{com} = \mathtt{push}(j')$, $\phi'' \in \phi' \odot \Phi_j^{=,j'}$ and $\phi''' = \phi_n \circ_{\mathtt{T}} \phi'$ if $\mathtt{com}' = \mathtt{push}(\phi''')$. Because $\theta_{n-1}, d_{n-1}, \theta_n \models \phi_n$ and $\phi' \in \phi_n \odot_{\mathtt{T}} \Phi_k^{\mathtt{tst},\mathtt{asgn}}$, there exists $d \in D$ satisfying $(\theta_n, d, d_{n-1}) \models \mathtt{tst}$ and $\theta_n, d_{n-1}, \theta' \models \phi'$ for $\theta' = \theta_n[\mathtt{asgn} \leftarrow d]$. Therefore, $(q, \theta_n, u) \vdash^{(a,d)}_{\mathcal{A}^{\mathrm{aug}}} (q', \theta', u')$ where $u' = u(1{:}), u$, or $(\theta'(j'), \theta')u$ if $\mathtt{com} =$

$\mathtt{pop}$, $\mathtt{skip}$, or $\mathtt{push}(j')$, respectively. We can show that $((q', \theta', u'), ((q', \phi''), v')) \in R$ in the same way as the last paragraph.

By Lemmas 7.1.1 and 7.4.1, we obtain the following theorem.

Theorem 7.4.1. *For a given (m, k)-NRPDA (resp. test-visibly (m, k)-DRPDA) $\mathcal{A}$, we can construct an m-NPDA (resp. m-DPDA) $\mathcal{A}'$ such that $\mathtt{Lab}(L(\mathcal{A})) = L(\mathcal{A}')$, if we assume the freshness property on the semantics of $\mathcal{A}$.*

7.5 Realizability Problems for RPDA and RPDT

7.5.1 Finite actions

In [31], the abstraction of the behavior of k-register transducer (k-RT), called finite actions, was introduced to reduce the realizability problem for register automata (RA) and RT to the problem on finite alphabets. We extend the idea of [31] and define the finite actions of k-RPDT.

For $k \in \mathbb{N}_0$, we define the set of finite input actions as $A_k^{\mathbb{i}} = \Sigma_{\mathbb{i}} \times \mathtt{Tst}_k$ and the set of finite output actions as $A_k^{\mathbb{o}} = \Sigma_{\mathbb{o}} \times \mathtt{Asgn}_k \times [k] \times \mathtt{Com}([k])$. Note that $\mathtt{Com}([k])$ appearing in the definition of $A_k^{\mathbb{o}}$ is not the abbreviation of $\mathtt{Com}(\Gamma \times [k])$. Finite actions have no information on finite stack alphabet Γ even if Γ is not a singleton. A sequence $w = (a_0^{\mathbb{i}}, d_0^{\mathbb{i}})(a_0^{\mathbb{o}}, d_0^{\mathbb{o}}) \cdots \in \mathtt{DW}(\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, D)$ is *compatible* with a sequence $\overline{a} = (a_0^{\mathbb{i}}, \mathtt{tst}_0)(a_0^{\mathbb{o}}, \mathtt{asgn}_0, j_0, \mathtt{com}_0) \cdots \in (A_k^{\mathbb{i}} \cdot A_k^{\mathbb{o}})^\omega$ iff there exists a sequence $(\theta_0, u_0)(\theta_1, u_1) \cdots \in (\Theta_k \times D^*)^\omega$, called a *witness*, such that $\theta_0 = \theta_\perp^k$, $u_0 = \perp$, $\langle \theta_i, d_i^{\mathbb{i}}, u_i(0) \rangle = \mathtt{tst}_i, \theta_{i+1} = \theta_i[\mathtt{asgn}_i \leftarrow d_i^{\mathbb{i}}], \theta_{i+1}(j_i) = d_i^{\mathbb{o}}$ and $u_{i+1} = \mathtt{upds}(u_i, \theta_{i+1}, \mathtt{com}_i)$. Let $\mathtt{Comp}(\overline{a}) = \{w \in \mathtt{DW}(\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, D) \mid w \text{ is compatible with } \overline{a}\}$. For a specification $S \subseteq \mathtt{DW}(\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, D)$, we define $W_{S,k} = \{\overline{a} \mid \mathtt{Comp}(\overline{a}) \subseteq S\}$.

For a data word $w \in \mathtt{DW}(\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, D)$ and a sequence $\overline{a} \in (A_k^{\mathbb{i}} \cdot A_k^{\mathbb{o}})^\omega$ such that for each $i \geq 0$, there exists $a \in \Sigma$ and we can write $w(i) = (a, d)$ and $\overline{a}(i) = (a, \mathtt{tst})$ if i is even and $\overline{a}(i) = (a, \mathtt{asgn}, j, \mathtt{com})$ if i is odd, we define $w \otimes \overline{a} \in \mathtt{DW}(A_k^{\mathbb{i}}, A_k^{\mathbb{o}}, D)$ as $w \otimes \overline{a}(i) = (\overline{a}(i), d)$ where $w(i) = (a, d)$.

7.5.2 Decidability and undecidability of realizability problems

Lemma 7.5.1. *$L_k = \{w \otimes \overline{a} \mid w \in \mathtt{Comp}(\overline{a})\}$ is definable as the language of a $(2, k+2)$-DRPDA.*

Proof. Let $(2,k+2)$-DRPDA $\mathcal{A}_1 = (Q_1, Q_1^{\mathbb{i}}, Q_1^{\mathbb{o}}, q_1^0, \delta_1, c_1)$ over $A_k^{\mathbb{i}}, A_k^{\mathbb{o}}$ and D where $Q_1 = \{p,q\} \cup (\mathtt{Asgn}_k \times [k] \times \mathtt{Com}([k])) \cup [k]$, $Q_1^{\mathbb{i}} = \{p\}$, $Q_1^{\mathbb{o}} = Q_1 \setminus Q_1^{\mathbb{i}}$, $q_1^0 = p$, $c_1(s) = 2$ for every $s \in Q$ and δ_1 consists of rules of the form

$$(p, (a_{\mathbb{i}}, \mathtt{tst}), \mathtt{tst} \cup \mathtt{tst}') \to (q, \{k+1\}, \mathtt{skip}) \tag{7.4}$$

$$(q, (a_{\mathbb{o}}, \mathtt{asgn}, j, \mathtt{com}), \mathtt{tst}'') \to ((\mathtt{asgn}, j, \mathtt{com}), \{k+2\}, \mathtt{skip}) \tag{7.5}$$

$$((\mathtt{asgn}, j, \mathtt{com}), \tau, \{k+1\} \cup \mathtt{tst}'') \to (j, \mathtt{asgn}, \mathtt{com}) \tag{7.6}$$

$$(j, \tau, \{j, k+2\} \cup \mathtt{tst}'') \to (p, \emptyset, \mathtt{skip}) \tag{7.7}$$

for all $(a_{\mathbb{i}}, \mathtt{tst}) \in A_k^{\mathbb{i}}$, $(a_{\mathbb{o}}, \mathtt{asgn}, j, \mathtt{com}) \in A_k^{\mathbb{o}}$, $\mathtt{tst}' \subseteq \{k+1, k+2\}$ and $\mathtt{tst}'' \in \mathtt{Tst}_{k+2}$.

We show $L(\mathcal{A}_k) = L_k$. For this proof, we redefine compatibility for finite sequences $w \in ((\Sigma_{\mathbb{i}} \times D) \cdot (\Sigma_{\mathbb{o}} \times D))^*$ and $\overline{a} \in (A_k^{\mathbb{i}} \cdot A_k^{\mathbb{o}})^*$. We show the following claim.

Claim. Let $n \in \mathbb{N}_0$ and $w \otimes \overline{a} = ((a_0^{\mathbb{i}}, \mathtt{skip}), d_0^{\mathbb{i}})((a_0^{\mathbb{o}}, \mathtt{asgn}_0, j_0, \mathtt{com}_0), d_0^{\mathbb{o}}) \cdots \in ((A_k^{\mathbb{i}} \times D) \cdot (A_k^{\mathbb{o}} \times D))^*$ whose length is $2n$ and $\rho = (\theta_0, u_0)(\theta_1, u_1) \cdots \in (\Theta_k \times D^*)^*$ whose length is $n+1$ and $(\theta_0, u_0) = (\theta_\perp^k, \perp)$. Then, ρ is a witness of the compatibility between w and $\overline{a}$ iff $(p, \theta_0', u_0) \vdash^{w \otimes \overline{a}(0:1)(\tau, d_0^{\mathbb{i}})(\tau, d_0^{\mathbb{o}})} (\theta_1', u_1) \vdash^{w \otimes \overline{a}(2:3)(\tau, d_1^{\mathbb{i}})(\tau, d_1^{\mathbb{o}})} \cdots \vdash^{w \otimes \overline{a}(2n-2:2n-1)(\tau, d_{n-1}^{\mathbb{i}})(\tau, d_{n-1}^{\mathbb{o}})} (p, \theta_n', u_n)$ where $\theta_i' \in \Theta_{k+2}$ $(i \in [n])$ satisfies $\theta_i'(j) = \theta_i(j)$ for $j \in [k]$.

(Proof of the claim) We show the claim by induction on n. The case of $n = 0$ is obvious. We show the claim for arbitrary $n > 0$ with the induction hypothesis.

We first show left to right. By the induction hypothesis, $(p, \theta_0', u_0) \vdash^{w \otimes \overline{a}(0:1)(\tau, d_0^{\mathbb{i}})(\tau, d_0^{\mathbb{o}}) \cdots w \otimes \overline{a}(2n-4:2n-3)(\tau, d_{n-2}^{\mathbb{i}})(\tau, d_{n-2}^{\mathbb{o}})} (p, \theta_{n-1}', u_{n-1})$ holds. By the assumption, because ρ is the witness, (a) $\theta_{n-1}, d_{n-1}^{\mathbb{i}}, u_{n-1}(0) \models \mathtt{tst}_{n-1}$, (b) $\theta_n = \theta_{n-1}[\mathtt{asgn}_{n-1} \leftarrow d_{n-1}^{\mathbb{i}}]$, (c) $\theta_n(j_{n-1}) = d_{n-1}^{\mathbb{o}}$ and (d) $u_n = \mathtt{upds}(u_{n-1}, \theta_n, \mathtt{com}_{n-1})$ hold. By the condition (a), $\mathcal{A}_k$ can do a transition $(p, \theta_{n-1}', u_{n-1}) \vdash^{w \otimes \overline{a}(2n-2)} (q, \theta_{n-1}^1, u_{n-1})$ for

unique $\theta^1_{n-1} \in \Theta_{k+2}$ by the rule $(p, (a^{\mathbb{i}}_{n-1}, \mathtt{tst}_{n-1}), \mathtt{tst}_{n-1} \cup \mathtt{tst}') \to (q, \{k+1\}, \mathtt{skip})$ of the form (7.4). We can also say $(q, \theta^1_{n-1}, u_{n-1}) \vdash^{w \otimes \overline{a}(2n-1)} ((\mathtt{asgn}_{n-1}, j_{n-1}, \mathtt{com}_{n-1}), \theta^2_{n-1}, u_{n-1})$ by the rule of the form (7.5). Note that $\theta^2_{n-1}(j) = \theta_{n-1}(j)$ if $j \in [k]$, $\theta^2_{n-1}(k+1) = d^{\mathbb{i}}_{n-1}$ and $\theta^2_{n-1}(k+2) = d^{\mathbb{o}}_{n-1}$. $((\mathtt{asgn}_{n-1}, j_{n-1}, \mathtt{com}_{n-1}), \theta^2_{n-1}, u_{n-1}) \vdash^{(\tau, d^{\mathbb{i}}_{n-1})} (j_{n-1}, \theta^3_{n-1}, u_n)$ is also valid transition of $\mathcal{A}_k$ of the form (7.6) by the conditions (b) and (d) where $\theta^{n-1}_3(j) = \theta_n(j)$ for $j \in [k]$ and $\theta^3_{n-1}(k+2) = d^{\mathbb{o}}_{n-1}$. By the condition (c), $\theta^{n-1}_3(j_{n-1}) = \theta^{n-1}_3(k+1) = d^{\mathbb{o}}_{n-1}$ holds. Thus, a transition $(j_{n-1}, \theta^3_{n-1}, u_n) \vdash^{(\tau, d^{\mathbb{o}}_{n-1})} (p, \theta'_n, u_n)$ is valid with the rule of the form (7.7). In conclusion, $(p, \theta'_{n-1}, u_{n-1}) \vdash^{w \otimes \overline{a}(2n-2:2n-1)(\tau, d^{\mathbb{i}}_{n-1})(\tau, d^{\mathbb{o}}_{n-1})} (p, \theta'_n, u_n)$ holds, and with the induction hypothesis, we obtain the left to right of the claim.

Next, we prove right to left. By the assumption, $(p, \theta'_{n-1}, u_{n-1}) \vdash^{w \otimes \overline{a}(2n-2:2n-1)(\tau, d^{\mathbb{i}}_{n-1})(\tau, d^{\mathbb{o}}_{n-1})} (p, \theta'_n, u_n)$ holds. By checking four transition rules that realize the above transition relation, we can obtain that $\rho(n-1), \rho(n), w \otimes \overline{a}(2n-2)$ and $w \otimes \overline{a}(2n-1)$ satisfies the conditions (a) to (d) described in the previous paragraph. Thus, by the induction hypothesis, we obtain ρ is a witness of the compatibility between w and $\overline{a}$.

(end of the proof of the claim)

By the claim, $w \otimes \overline{a} \in L_k \Leftrightarrow$ there exists a witness $(\theta_0, u_0)(\theta_1, u_1) \cdots \in (\Theta_k \times D^*)^\omega$ of w and $\overline{a} \Leftrightarrow$ there exists a run $(p, \theta'_0, u_0) \vdash^{w \otimes \overline{a}(0:1)(\tau, d^{\mathbb{i}}_0)(\tau, d^{\mathbb{o}}_0)} (\theta'_1, u_1) \vdash^{w \otimes \overline{a}(2:3)(\tau, d^{\mathbb{i}}_1)(\tau, d^{\mathbb{o}}_1)} \cdots$ of $\mathcal{A} \Leftrightarrow w \otimes \overline{a} \in L(\mathcal{A}_k)$ holds for all $w \otimes \overline{a} \in \mathtt{DW}(A^{\mathbb{i}}_k, A^{\mathbb{o}}_k, D)$.

Lemma 7.5.2. *For a specification $\mathcal{S}$ defined by some visibly k'-DRPDA, $L_{\overline{S},k} = \{w \otimes \overline{a} \mid w \in \mathtt{Comp}(\overline{a}) \cap \overline{S}\}$ is definable as the language of a $(4, k+k'+4)$-DRPDA.*

Proof. Let $L_{\overline{S}} = \{w \otimes \overline{a} \mid w \in \overline{S}, \overline{a} \in (A^{\mathbb{i}}_k \cdot A^{\mathbb{o}}_k)^\omega\}$. We can construct a visibly k'-DRPDA $\mathcal{A}_2 = (Q_2, Q^{\mathbb{i}}_2, Q^{\mathbb{o}}_2, q^0_2, \delta_2, c_2)$ over $A^{\mathbb{i}}_k, A^{\mathbb{o}}_k$ and D such that $L(\mathcal{A}_2) = L_{\overline{S}}$. Let $\mathcal{A}_1$ be the $(2, k+2)$-DRPDA such that $L(\mathcal{A}_1) = L_k$, which is given in Lemma 7.5.1. Because $L_{\overline{S},k} = L_{\overline{S}} \cap L_k$, it is enough to show that we can construct a $(4, k+k'+4)$-DRPDA $\mathcal{A}$ over $A^{\mathbb{i}}_k, A^{\mathbb{o}}_k$ and D such that $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.

To construct $\mathcal{A}$, we use the facts that $c_1(q)$ is even for every $q \in Q_1$ and δ_1

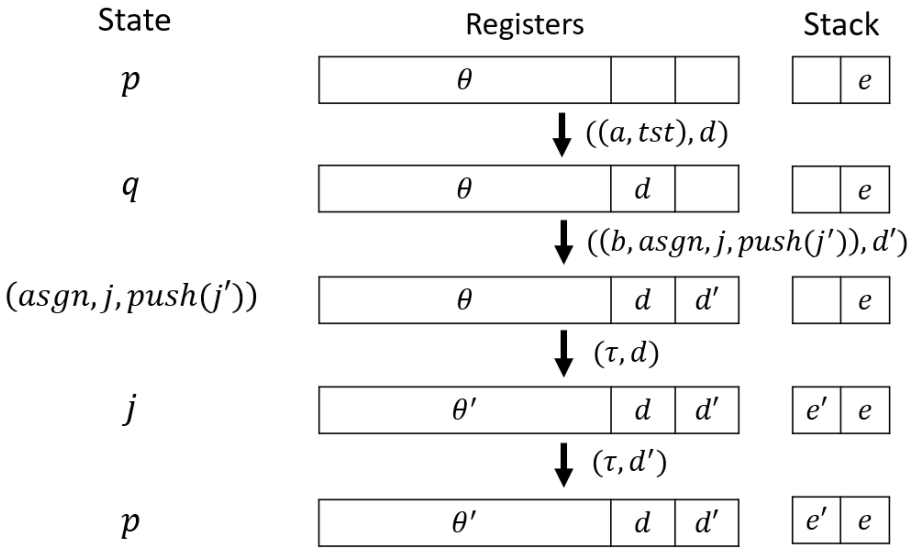

Figure 7.3: An example of transitions of $\mathcal{A}_k$.

consists of several groups of three consecutive rules having the following forms:

$$(q_1, a, \mathtt{tst}_1) \to (q_2, \mathtt{asgn}_1, \mathtt{skip}) \tag{7.5'}$$

$$(q_2, \tau, \mathtt{tst}_2) \to (q_3, \mathtt{asgn}_2, \mathtt{com}_1) \tag{7.6'}$$

$$(q_3, \tau, \mathtt{tst}_3) \to (q_4, \mathtt{asgn}_3, \mathtt{skip}). \tag{7.7'}$$

Note that $\mathtt{vis}(a) = v(\mathtt{com}_1)$ always holds for every consecutive rules. (7.5′), (7.6′) and (7.7′) correspond to (7.5), (7.6) and (7.7), respectively, and (7.4) is also converted to three consecutive rules like (7.5′)-(7.7′) by adding dummy τ rules.

We let $k_1 = k + 2$ and $k_2 = k'$. We construct $(4, k_1 + k_2 + 2)$-DRPDA $\mathcal{A} = (Q_{\mathbb{i}} \cup Q_{\mathbb{o}} \cup \{q_0\}, Q_{\mathbb{i}} \cup \{q_0\}, Q_{\mathbb{o}}, q_0, \delta, c)$ where $Q_{\mathbb{i}} = Q_1^{\mathbb{i}} \times Q_2^{\mathbb{i}} \times [5]$, $Q_{\mathbb{o}} = Q_1^{\mathbb{o}} \times Q_2^{\mathbb{o}} \times [5]$. c is defined as $c(q_0) = 1$ and $c((q_1, q_2, i)) = c_2(q_2)$ for all $(q_1, q_2, i) \in Q$. δ has first τ rule $(q_0, \tau, [k] \cup \{top\}) \to ((q_0^1, q_0^2, 1), \mathtt{push}(1))$. For all rules (7.5'), (7.6'), (7.7') in δ_1 and

$$(q, a, \mathtt{tst}) \to (q', \mathtt{asgn}, \mathtt{com}) \in \delta_2 \tag{7.8}$$

such that $v(\mathtt{com}_1) = v(\mathtt{com})$ $(= \mathtt{vis}(a))$ for $a \in A_k^{\mathbb{i}} \cup A_k^{\mathbb{o}}$, let $\mathtt{tst}^{+k_1} = \{i + k_1 \mid i \in \mathtt{tst}\} \cup \{top \mid top \in \mathtt{tst} \setminus [k_1]\}, \mathtt{asgn}^{+k_1} = \{i + k_1 \mid i \in \mathtt{asgn}\}$ and $\mathtt{com}^{+k_1} = \mathtt{push}(j + k_1)$ if $\mathtt{com} = \mathtt{push}(j)$ and $\mathtt{com}^{+k_1} = \mathtt{com}$ otherwise, then δ

consists of the rules

$$((q_1, q, 1), \tau, T_1^{k_1+k_2+2} \cup \{top\}) \to ((q_1, q, 2), \{k_1 + k_2 + 1\}, \mathtt{pop}) \tag{7.9}$$

$$((q_1, q, 2), \tau, T_1^{k_1+k_2+2} \cup \{top\}) \to ((q_1, q, 3), \{k_1 + k_2 + 2\}, \mathtt{push}(k_1 + k_2 + 1)) \tag{7.10}$$

$$((q_1, q, 3), a, \mathtt{tst}_1 \cup ((\mathtt{tst}^{+k_1} \setminus \{top\}) \cup \mathtt{Top})) \to ((q_2, q', 4), \mathtt{asgn}_1 \cup \mathtt{asgn}^{+k_1}, \mathtt{com}^{+k_1}) \tag{7.11}$$

$$((q_2, q', 4), \tau, ((\mathtt{tst}_2 \setminus \{top\}) \cup \mathtt{Top}') \cup T_{k_1+1}^{k_1+k_2+1}) \to ((q_3, q', 5), \mathtt{asgn}_2, \mathtt{com}_1) \tag{7.12}$$

$$((q_3, q', 5), \tau, \mathtt{tst}_3 \cup T_{k_1+1}^{k_1+k_2+2}) \to ((q_4, q', 1), \mathtt{asgn}_3, \mathtt{skip}) \tag{7.13}$$

for all $T_i^j \in \mathtt{Tst}_j \setminus \mathtt{Tst}_{i-1}$ (we assume $\mathtt{Tst}_0 = \emptyset$), and $\mathtt{Top} = \{k_1 + k_2 + 2\}$ ($\mathtt{Top}' = \{k_1 + k_2 + 1\}$) if $top \in \mathtt{tst}$ ($top \in \mathtt{tst}_2$) and $\mathtt{Top} = \emptyset$ ($\mathtt{Top}' = \emptyset$) otherwise.

Fig. 7.4 illustrates an example of transitions of $\mathcal{A}$ from $(q_1, q, 1)$ to $(q_4, q', 1)$ with updating contents of its registers and stack. The first to k_1-th registers simulate the registers of $\mathcal{A}_1$, $(k_1 + 1)$-th to $(k_1 + k_2)$-th registers simulate the registers of $\mathcal{A}_2$ and $(k_1 + k_2 + 1)$-th and $(k_1 + k_2 + 2)$-th registers are for keeping the first and second stack top contents, respectively. The stack contents of $\mathcal{A}$ simulates those of $\mathcal{A}_1$ and $\mathcal{A}_2$ by restoring the contents of stacks of $\mathcal{A}_1$ and $\mathcal{A}_2$ alternately.

The transition rules (7.9) and (7.10) are for moving the two data values at the stack top to $(k_1 + k_2 + 1)$-th and $(k_1 + k_2 + 2)$-th registers. The transition rule (7.11) is for updating states, registers and stacks by simulating the rules (7.5') and (7.8). Because the first to k_2-th registers of $\mathcal{A}_2$ are simulated by $(k_1 + 1)$-th to $(k_1 + k_2)$-th registers of $\mathcal{A}$, we use $\mathtt{tst}^{+k_1}$, $\mathtt{asgn}^{+k_1}$ and $\mathtt{com}^{+k_1}$ instead of $\mathtt{tst}$, $\mathtt{asgn}$ and $\mathtt{com}$, and replace *top* in $\mathtt{tst}^{+k_1}$ to $k_1 + k_2 + 2$. The transition rules (7.12) and (7.13) simulate the rules (7.6') and (7.7'), respectively.

For two assignments $\theta_1 \in \Theta_{k_1}$ and $\theta_2 \in \Theta_{k_2}$, let $[\theta_1, \theta_2, d, d'] \in \Theta_{k_1+k_2+2}$ be the assignment such that $[\theta_1, \theta_2, d, d'](i) = \theta_1(i)$ if $i \in [k_1]$, $[\theta_1, \theta_2, d, d'](i) = \theta_2(i)$ if $k_1 + 1 \leq i \leq k_2$, $[\theta_1, \theta_2, d, d'](k_1 + k_2 + 1) = d$ and $[\theta_1, \theta_2, d, d'](k_1 + k_2 + 2) = d'$. To prove $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$, we show the following claim.

Claim. For all $n \in \mathbb{N}_0$ and $w \in ((A_k^{\mathrm{i}} \cup A_k^{\mathrm{o}}) \times D)^n$, there exists sequences of transitions $(q_0^1, \theta_0^1, u_0^1) \vdash_{\mathcal{A}_1}^{w(0)(\tau,d_0)(\tau,d_0')} (q_1^1, \theta_1^1, u_1^1) \vdash_{\mathcal{A}_1}^{w(1)(\tau,d_1)(\tau,d_1')} \cdots \vdash_{\mathcal{A}_1}^{w(n-1)(\tau,d_{n-1})(\tau,d_{n-1}')} (q_n^1, \theta_n^1, u_n^1)$ and $(q_0^2, \theta_0^2, u_0^2) \vdash_{\mathcal{A}_2}^{w(0)} (q_1^2, \theta_1^2, u_1^2) \vdash_{\mathcal{A}_2}^{w(1)} \cdots \vdash_{\mathcal{A}_2}^{w(n-1)} (q_n^2, \theta_n^2, u_n^2)$ iff
$(q_0, \theta_\perp^{k_1+k_2+2}, \perp) \vdash_{\mathcal{A}}^{(\tau,\perp)} ((q_0^1, q_0^2, 1), [\theta_0^1, \theta_0^2, d_0^1, d_0^2], u_0^1, u_0^2\ )$
$\vdash_{\mathcal{A}}^{(\tau,u_0^1(0))(\tau,u_0^2(0))w(0)(\tau,d_0)(\tau,d_0')} ((q_1^1, q_1^2, 1), [\theta_1^1, \theta_1^2, d_1^1, d_1^2], u_1^1, u_1^2\ )$
$\vdash_{\mathcal{A}}^{(\tau,u_1^1(0))(\tau,u_1^2(0))w(1)(\tau,d_1)(\tau,d_1')} \cdots \vdash_{\mathcal{A}}^{(\tau,u_{n-1}^1(0))(\tau,u_{n-1}^2(0))w(n-1)(\tau,d_{n-1})(\tau,d_{n-1}')}$
$((q_n^1, q_n^2, 1), [\theta_n^1, \theta_n^2, d_n^1, d_n^2], u_n^1, u_n^2)$ holds where $b \in \{1, 2\}, i \in [n]$, $\theta_0^b = \theta_\perp^{k_b}, u_0^b = \perp$, $q_i^b \in Q_b$, $\theta_i^b \in \Theta_{k_b}$, $u_i^b \in D^*$ and $d_{i-1}, d_{i-1}' \in D$.

(Proof of the claim) We show the claim by the induction on n. The case $n = 0$ is obvious.

We first show left to right. By induction hypothesis, $(q_0, \theta_\perp^{\mathcal{A}}, \perp) \vdash_{\mathcal{A}}^{(\tau,\perp)}$
$((q_0^1, q_0^2, 1), [\theta_0^1, \theta_0^2, d_0^1, d_0^2], u_0^1, u_0^2\ ) \vdash_{\mathcal{A}}^{(\tau,u_0^1(0))(\tau,u_0^2(0))w(0)(\tau,d_0)(\tau,d_0')} \cdots$
$\vdash_{\mathcal{A}}^{(\tau,u_{n-2}^1(0))(\tau,u_{n-2}^2(0))w(n-2)(\tau,d_{n-2})(\tau,d_{n-2}')} ((q_{n-1}^1, q_{n-1}^2, 1), [\theta_{n-1}^1, \theta_{n-1}^2, d_{n-1}^1, d_{n-1}^2],$
$u_{n-1}^1, u_{n-1}^2)$ holds. Also, by the assumption, $(q_{n-1}^1, \theta_{n-1}^1, u_{n-1}^1) \vdash_{\mathcal{A}_1}^{w(n-1)} (q', \theta', u_{n-1}^1)$
$\vdash_{\mathcal{A}_1}^{(\tau,d)} (q'', \theta'', u_n^1) \vdash_{\mathcal{A}_1}^{(\tau,d')} (q_n^1, \theta_n^1, u_n^1)$ and $(q_{n-1}^2, \theta_{n-1}^2, u_{n-1}^2) \vdash_{\mathcal{A}_2}^{w(n-1)} (q_n^2, \theta_n^2, u_n^2)$ for some $q', q'' \in Q_1, \theta', \theta'' \in \Theta_{k_1}$ and $d, d' \in D$, and let the following be the rules used in these transitions.

$$(q_{n-1}, (a, v(\mathtt{com}_1)), \mathtt{tst}_1) \to (q', \mathtt{asgn}_1, \mathtt{skip}) \in \delta_1 \tag{7.5"}$$

$$(q', \tau, \mathtt{tst}_2) \to (q'', \mathtt{asgn}_2, \mathtt{com}_1) \in \delta_1 \tag{7.6"}$$

$$(q'', \tau, \mathtt{tst}_3) \to (q_n, \mathtt{asgn}_3, \mathtt{skip}) \in \delta_1 \tag{7.7"}$$

$$(q_{n-1}, (a, v(\mathtt{com})), \mathtt{tst}) \to (q_n, \mathtt{asgn}, \mathtt{com}) \in \delta_2 \tag{7.8'}$$

We can check the follows.

$$\begin{aligned}
&((q_{n-1}^1, q_{n-1}^2, 1), [\theta_{n-1}^1, \theta_{n-1}^2, d_{n-1}^1, d_{n-1}^2], u_{n-1}^1, u_{n-1}^2\ ) \vdash_{\mathcal{A}}^{(\tau,u_{n-1}^1(0))(\tau,u_{n-1}^2(0))} \\
&((q_{n-1}^1, q_{n-1}^2, 3), [\theta_{n-1}^1, \theta_{n-1}^2, u_{n-1}^1(0), u_{n-1}^1(0)], u_{n-1}^1, u_{n-1}^2\ ) \vdash_{\mathcal{A}}^{w(n-1)} \\
&((q_{n-1}^1, q_{n-1}^2, 4), [\theta_n^1, \theta', u_{n-1}^1(0), u_{n-1}^1(0)], u_{n-1}^1, u_n^2\ ) \vdash_{\mathcal{A}}^{(\tau,d_{n-1})} \\
&((q_{n-1}^1, q_{n-1}^2, 5), [\theta_n^1, \theta'', u_{n-1}^1(0), u_{n-1}^1(0)], u_n^1, u_n^2\ ) \vdash_{\mathcal{A}}^{(\tau,d_{n-1}')} \\
&((q_n^1, q_n^2, 1), [\theta_n^1, \theta_n^2, u_{n-1}^1(0), u_{n-1}^1(0)], u_n^1, u_n^2\ )
\end{aligned}$$

Thus, the right side of the claim holds. In a similar way, we can also show right to left.

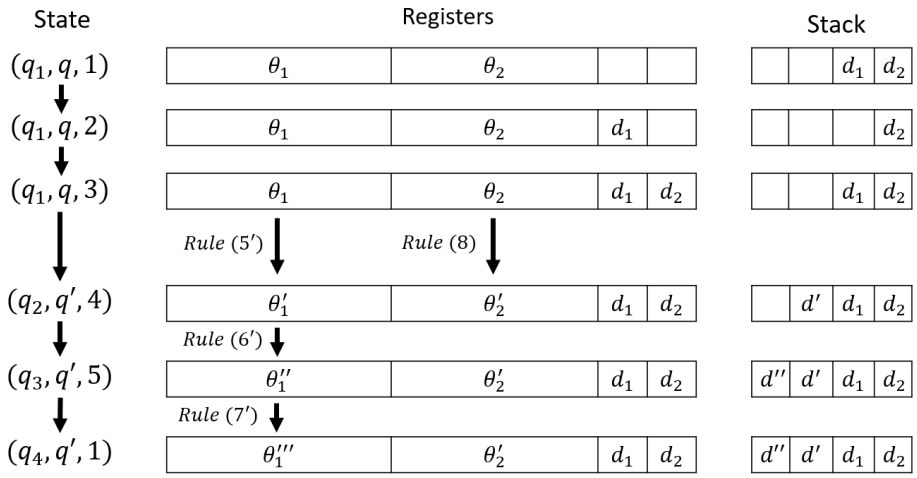

Figure 7.4: An example of transitions of $\mathcal{A}$ with $\mathtt{vis}(a) = \mathtt{push}$.

(end of the proof of the claim)

By the claim, $w \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2) \Leftrightarrow$ there exists runs $(q_0^1, \theta_0^1, u_0^1) \vdash_{\mathcal{A}_1}^{w(0)(\tau,d_0)(\tau,d_0')} (q_1^1, \theta_1^1, u_1^1) \vdash_{\mathcal{A}_1}^{w(1)(\tau,d_1)(\tau,d_1')} \cdots$ and $(q_0^2, \theta_0^2, u_0^2) \vdash_{\mathcal{A}_2}^{w(0)} (q_1^2, \theta_1^2, u_1^2) \vdash_{\mathcal{A}_2}^{w(1)} \cdots$ that satisfies the minimum number appearing in the sequence $q_0^1, q_1^1, \cdots$ infinitely is even. $\Leftrightarrow$ there exists a run $(q_0, \theta_\perp^{\mathcal{A}}, \perp) \vdash_{\mathcal{A}}^{(\tau,\perp)} ((q_0^1, q_0^2, 1), [\theta_0^1, \theta_0^2, d_0^1, d_0^2], u_0^1, u_0^2) \vdash_{\mathcal{A}}^{(\tau,u_0^1(0))} {}^{(\tau,u_0^2(0))w(0)(\tau,d_0)(\tau,d_0')}((q_1^1, q_1^2, 1), [\theta_1^1, \theta_1^2, d_1^1, d_1^2], u_1^1, u_1^2) \vdash_{\mathcal{A}}^{(\tau,u_1^1(0))(\tau,u_1^2(0))w(1)(\tau,d_1)(\tau,d_1')} \cdots$ that satisfies the minimum number appearing in the sequence $(q_0^1, q_0^2, 1), (q_0^1, q_0^2, 2) \cdots, (q_1^1, q_1^2, 1), \cdots$ infinitely is even. $\Leftrightarrow w \in L(\mathcal{A})$ holds.

Lemma 7.5.3. $W_{S,k} = \overline{\mathtt{Lab}(L_{\overline{S},k})}$.

Proof. For every $\overline{a} \in (A_k^{\mathrm{i}} A_k^{\mathrm{o}})^\omega$, $\overline{a} \notin W_{S,k} \Leftrightarrow \mathtt{Comp}(\overline{a}) \not\subseteq S \Leftrightarrow \exists w.w \in \mathtt{Comp}(\overline{a}) \cap \overline{S} \Leftrightarrow \exists w.w \otimes \overline{a} \in L_{\overline{S},k} \Leftrightarrow \overline{a} \in \mathtt{Lab}(L_{\overline{S},k})$. Thus, $W_{S,k} = \overline{\mathtt{Lab}(L_{\overline{S},k})}$ holds.

Theorem 7.5.1. *For all $k \geq 0$,* Real(**DRPDAv**, **RPDT**$[k]$) *is in 2EXPTIME.*

Proof. By Lemma 7.5.2 and Theorem 7.4.1, $W_{S,k}$ is definable by a 4-DPDA $\mathcal{A}_f$. By Lemma 7.2.1, we can construct a 0-DPDA $\mathcal{A}_f'$ where $L(\mathcal{A}_f') = L(\mathcal{A}_f)$. By the construction of $\mathcal{A}$ in Lemma 7.5.2, the stack height of the current ID increases by two during the transitions from $(q_1, q, 1)$ to $(q_4, q', 1)$ in Fig. 7.4 if $\mathtt{vis}(a) = \mathtt{push}$, does not change if $\mathtt{vis}(a) = \mathtt{skip}$ and decreases by two if $\mathtt{vis}(a) = \mathtt{pop}$. Thus, $\mathcal{A}_f'$ is a stack-visibly DPDA, that is, every transition rule

$(p, a, z) \to (q, \mathtt{com})$ of $\mathcal{A}'_f$ satisfies $\mathtt{vis}(a) = v(\mathtt{com})$. We show the following two conditions are equivalent.

- There exists a k-RPDT $\mathcal{T}$ such that $L(\mathcal{T}) \subseteq S$.
- There exists a PDT $\mathcal{T}'$ such that $L(\mathcal{T}') \subseteq W_{S,k}$.

Assume that a k-RPDT $\mathcal{T}$ over $\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}$ and Γ satisfies $L(\mathcal{T}) \subseteq S$. Then, consider the PDT $\mathcal{T}'$ over $A^{\mathbb{i}}_k, A^{\mathbb{o}}_k$ and Γ such that $(q, (a, \mathtt{tst}), z) \to (q', (b, \mathtt{asgn}, j, \mathtt{com}), \mathtt{com}')$ is a rule of $\mathcal{T}'$ iff $(q, a, \mathtt{tst}, z) \to (q', b, \mathtt{asgn}, j, \mathtt{com})$ is a rule of $\mathcal{T}$ where $\mathtt{com}' = \mathtt{com}$ if $v(\mathtt{com}) = \mathtt{pop}$ or $\mathtt{skip}$ and $\mathtt{com}' = \mathtt{push}(z')$ if $\mathtt{com} = \mathtt{push}(z', j')$ for some $j' \in [k]$. For $\overline{a} \in L(\mathcal{T}')$, every $w \in \mathtt{Comp}(\overline{a})$ has a witness (see Section 6.1) and thus $w \in L(\mathcal{T})$ holds. By the assumption $L(\mathcal{T}) \subseteq S$, $\mathtt{Comp}(\overline{a}) \subseteq S$ holds and thus $\overline{a} \in W_{S,k}$. Hence, we obtain $L(\mathcal{T}') \subseteq W_{S,k}$.

Conversely, assume there exists a PDT $\mathcal{T}'$ over $A^{\mathbb{i}}_k, A^{\mathbb{o}}_k$ and Γ that satisfies $L(\mathcal{T}') \subseteq W_{S,k}$. We can in particular construct a PDT $\mathcal{T}''$ such that $L(\mathcal{T}'') \subseteq W_{S,k}$ and every rule $(q, (a, \mathtt{tst}), z) \to (q', (b, \mathtt{asgn}, j, \mathtt{com}), \mathtt{com}')$ satisfies $\mathtt{vis}(b) = v(\mathtt{com}')$ (note that $\mathtt{vis}(a) = \mathtt{skip}$ always holds) by the construction algorithm in [72]. In the rule, $v(\mathtt{com}) = v(\mathtt{com}')$ holds because $\mathcal{A}'_f$ is a stack-visibly PDA and thus $\mathtt{vis}(b) = v(\mathtt{com})$ holds. Consider the k-RPDT $\mathcal{T}$ over $\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}$ and Γ such that $(q, a, \mathtt{tst}, z) \to (q', b, \mathtt{asgn}, j, \mathtt{com}'')$ is a rule of $\mathcal{T}$ iff $(q, (a, \mathtt{tst}), z) \to (q', (b, \mathtt{asgn}, j, \mathtt{com}), \mathtt{com}')$ is a rule of $\mathcal{T}''$ where $\mathtt{com}'' = \mathtt{com}'$ if $\mathtt{com}' = \mathtt{pop}$ or $\mathtt{skip}$ and $\mathtt{com}'' = \mathtt{push}(z', j')$ if $\mathtt{com}' = \mathtt{push}(z')$ and $\mathtt{com} = \mathtt{push}(j')$. Further, assume $w \in L(\mathcal{T})$, and let $\overline{a} \in (A^{\mathbb{i}}_k \cdot A^{\mathbb{o}}_k)^\omega$ be the sequence with which w is compatible. Then, by the definition of $\mathcal{T}$, $\overline{a} \in L(\mathcal{T}'')$. By the assumption $L(\mathcal{T}'') \subseteq W_{S,k}$, every $\overline{a} \in L(\mathcal{T}'')$ satisfies $\mathtt{Comp}(\overline{a}) \subseteq S$, and thus $w \in S$ holds. Hence, we obtain $L(\mathcal{T}) \subseteq S$.

By the equivalence, we can check Real(**DPDA**, **PDT**) for $\mathcal{A}'_f$, which is shown to be EXPTIME in Theorem 7.3.1, instead of checking Real(**DRPDAv**, **RPDT**$[k]$). Because the size of $\mathcal{A}'_f$ is exponential to $k + k'$, Real(**DRPDAv**, **RPDT**$[k]$) is in 2EXPTIME.

Theorem 7.5.2. *For all $k \geq 0$,* Real(**NRPDA**, **RPDT**$[k]$) *is undecidable.*

Proof. We can easily reduce the Real(**NPDA**, **PDT**), whose undecidability is proved in Theorem 7.3.2, to this problem.

Conclusion of Part IV

This chapter discusses the relizability problems of a pair of PDA and PDT, and a pair of RPDA and RPDT. Conparing with previous research, our algorithms can deal with specifications that handle both recursive structure and data values. Our method expands the scope of formal software synthesis from the aspect of the stack.

Part V

Conclusion

The purpose of this dissertation is to build a theoretical foundation for handling programs and databases with data values. To this aim, we investigated the computational complexity of the membership and emptiness problems for RCFG, RPDA and RTA and their subclasses in Part II. 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. These results are useful for analyzing the computational complexity of algorithms using these register models. For example, for a set of trees whose nodes have a data value, the query "find a tree where a data value of every node is different from the data value of the parent node" can be expressed by using RTA.

Next, we have introduced register type to RCFG and shown ε-rule removal property of RCFG. Using the register type technique, the decidability of emptiness and membership for generalized RCFG (GRCFG) under the assumption of simulation and progress and an ε-rule removability of GRCFG have been shown. An example of such a binary relation with simulation and progress is the total order on rational numbers. Our results show that if a query using such binary relation is represented as a GRCFG, the query is decidable.

In Part III, we newly defined RPDS whose guard conditions and register updates are represented by equivalence relations over the set of registers. We have shown the forward regularity preservation property of RPDS by providing a saturation algorithm. And we prove its soundness and completeness of the algorithm. After that, we have defined Büchi RPDS with the fundamental properties of RPDS, RA and Büchi RPDS. We have shown that the LTL model checking problem for RPDS is EXPTIME-complete if either simple valuation or regular valuation is assumed. This model checking can deal with the equality check for data values, which PDS-LTL model checking cannot deal with. Hence, our method can be applied to practical verification problems of a program handling data values.

In Part IV, we have discussed the realizability problem whose specification and implementation are DPDA (NPDA) and PDT (see Table 7.1). By using the result in [72], we show Real(**DPDA**,**PDT**) is in EXPTIME. We also show

the undecidability of Real(**NPDA**,**PDT**) by a reduction from the universality problem of NPDA. Also, we have defined RPDT, RPDA and shown a way to recognize the label of the language of RPDA by PDA. We have introduced the notions of stack-visibly [5] and test-visibly to discuss the decidability of realizability for RPDA and RPDT. We show that the behavior of registers and stack of RPDT can be simulated by the finite alphabets defined as finite actions, and prove that Real(**DRPDAv**,**RPDT**$[k]$) can be reduced to Real(**DPDA**,**PDT**) and is in 2EXPTIME.

Table 7.1: Complexity results of realizability problems for non-register, bounded numbers of registers and general cases.

	DFA	NFA	UFA
	EXPTIMEc [16]		
	DPDA	NPDA	UPDA
	EXPTIMEc	Undecidable	Open
	DRA	NRA	URA
Bounded	EXPTIME [31]	Undecidable [31]	2EXPTIME [31]
General	EXPTIME [31]	Undecidable [31]	Undecidable [31]
	DRPDAv	NRPDA	URPDA
Bounded	2EXPTIME	Undecidable	Open
General	Open	Undecidable	Open

Future work

To describe constraints in register models, logics are important. 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 our future work. Looking into the relation between GRCFG and nominal CFG [11] in-depth leads to further investigation of theoretical properties of GRCFG.

We have implemented a tool for the reachability analysis of RPDS based on the proposed algorithm in Chapter 5. The tool is applied to detection of malware. Implementing a tool for LTL model checking proposed in Chapter 6 is also our

future work.

For software synthesis discussed in part IV, it is still unknown whether the realizability problem is decidable in several cases (see *Open* in Table 7.1). For the problem, a specification is given by a universal PDA, DRPDA without restriction on visibility nor the fixed number of registers of RPDT. These open problems are also necessary to represent important and practical problems, for example, universal PDA is useful for describing the safety constraint "Do all runs of given reactive system satisfy a given condition?". Thus, we would like to investigate the decidability of these open cases.

References

[1] F. Aarts, P. Fiterau-Brostean, H. Kuppens, and F. W. Vaandrager, Learning register automata with fresh value generation, 12th International Colloquium on Theoretical Aspects of Computing (ICTAC 2015), LNCS 9399, 165—183.

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

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

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

[5] R. Alur and P. Madhusudan, Visibly pushdown languages, 36th ACM Symp. Theory of Computing (STOC 2004).

[6] J. An, M. Chen, B. Zhan, N. Zhan, M. Zhang, Learning One-Clock Timed Automata, 26th International Conference of Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2020) LNCS 12078, 444–462.

[7] D. Angluin, Learning regular sets from queries and counterexamples, Information and Computation, 75(2), 87–106, 1987.

[8] R. Bloem, K. Chatterjee and B. Jobstmann, Graph Games and Reactive Synthesis, E. M. Clarke et al. (eds.), Handbook of Model Checking, Chapter 27, 921–962, Springer, 2018.

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

[10] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick and L. Segoufin, Two-variable logic on data words,

[11] M. Bojańczyk, B. Klin, S. Lasota, Automata theory in nominal sets, Logical Methods in Computer Science, 10(3:4), 1–44, 2014.

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

[13] A. Bouajjani, J. Esparza and O. Maler, Reachability analysis of pushdown automata: Application to model-checking, 8th International Conference on Concurrency Theory (CONCUR'97), LNCS 1243, 135–150.

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

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

[16] J. R. Büchi and L. H. Landweber, Solving sequential conditions by finite-state strategies, Trans. American Mathematical Society 138, 295–311, 1969.

[17] S. Cassel, F. Howar, B. Jonsson and B. Steffen, Active learning for extended finite state machines, Formal Aspects of Computing 28: 233–263, 2016.

[18] A. K. Chandra and L. J. Stockmeyer, Alternation, 17th Annual Symposium on Foundations of Computer Science (FOCS 1976), also in: Journal of the ACM, 28, 114-133, 1981.

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

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

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

[22] L. Clemente and S Lasota, Reachability analysis of first-order definable pushdown systems, 24th EACSL Annual Conf. on Computer Science Logic (CSL 2015), 244–259.

[23] J. L. Coquidé, M. Dauchet, R. Gilleron and S. Vágvölgyi, Bottom-up tree pushdown automata: classification and connection with rewrite systems, Theoretical Computer Science, 127, 69–98, 1994.

[24] L. D'Antoni, T. Ferreira, M. Sammartino and A. Silva, Symbolic register automata, 31th International Conference on Computer Aided Verification, LNCS 11561, 2019.

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

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

[27] S. Drews, and L. D ' Antoni, Learning symbolic automata, International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2017), LNCS 10205, 173-–189.

[28] R. Ehlers, S. A. Seshia and H. Kress-Gazit, Synthesis with identifiers, 15th Int. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI 2014), 415–433.

[29] J. Esparza, D. Hansel, P. Rossmanith and S. Schwoon, Efficient algorithms for model checking pushdown systems, 12th International Conference on Computer Aided Verification (CAV 2000), LNCS 1855, 232–247.

[30] J. Esparza, A. Kučera and S. Schwoon, Model-checking LTL with regular valuations for pushdown systems, 4th International Symposium on Theoretical Aspects of Computer Software (TACS 2001), LNCS 2215, 316–339.

[31] L. Exibard, E. Filiot and P.-A. Reynier, Synthesis of data word transducers, 30th Int. Conf. on Concurrency Theory (CONCUR 2019).

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

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

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

[35] S. A. Greibach, A note on pushdown store automata and regular systems, Proc. the American Mathematical Society 18, 263—268, 1967.

[36] P. Gyenizse, and S. Vágvölgyi, Linear generalized semi-monadic rewrite systems effectively preserve recognizability, Theoretical Computer Science, 194, 87–122, 1998.

[37] F. Howar, B. Steffen, B. Jonsson and S. Cassel, Inferring canonical register automata, 13rd International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2012), LNCS 7148, 251–266, 2012.

[38] M. Isberner, F. Howar, and B. Steffen, The open-source LearnLib - A framework for active automata learning, 27th International Conference on Computer Aided Verification (CAV 2015), LNCS 9206, 487-–495.

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

[40] Y. Kaji, R. Nakanishi, H. Seki and T. Kasami, The computational complexity of the universal recognition problem for parallel multiple context-free grammars, Computational Intelligence, 10(4), 440–452, 1994.

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

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

[43] A. Khalimov and O. Kupferman, Register-bounded synthesis, 30th Int. Conf. on Concurrency Theory (CONCUR 2019).

[44] A. Khalimov, B. Maderbacher and R. Bloem, Bounded synthesis of register transducers, 16th Int Symp on Automated Technology for Verification and Analysis (ATVA 2018), 494–510.

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

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

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

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

[49] M. Merten, F. Howar, B. Steffen, S. Cassel, and B. Jonsson, Demonstrating learning of register automata, 18th International Conference of Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2012) LNCS 7214, 466-–471.

[50] A. S. Murawski, S. J. Ramsay and N. Tzevelekos, Reachability in pushdown register automata, Journal of Computer and System Sciences, 87, 58–83, 2017.

[51] A. S. Murawski, S. J. Ramsay, and N. Tzevelekos, Polynomial-time equivalence testing for deterministic fresh-register automata, 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018), LIPIcs 117(72), 72:1–72:14.

[52] T. Milo, D. Suciu and V. Vianu, Type checking for XML transformers, 19th ACM Symposium on Principles of Database Systems (PODS 2000), 11–22.

[53] J. Moerman, M. Sammartino, A. Silva, B. Klin, M. Szynwelski, Learning nominal automata, 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017), 52(1), 613–625.

[54] T. Nagaya and Y. Toyama, Decidability for left-linear growing term rewriting systems, 10th International Conference on Rewriting Techniques and Applications (RTA99), LNCS 1631, 256–270.

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

[56] N. Nitta, Y. Takata, and H. Seki, An efficient security verification method for programs with stack inspection, 8th ACM Conference on Computer and Communications Security (ACM CCS 2001), 68–77.

[57] A. Pnueli and R. Rosner, On the synthesis of a reactive module, 16th ACM Symp. on Principles of Programming Languages (POPL 1989), 179–190.

[58] J. Rot, F. de Boer and M. Bonsangue, Pushdown system representation for unbounded object creation, Tech. Rep. KIT-13, Karlsruhe Institute of Technology, 38–52, 2010.

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

[60] K. Salomaa, Deterministic tree pushdown automata and monadic tree rewriting systems, J. Comput. System Sci., 37, 367–394, 1988.

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

[62] M. Sipser, Introduction to the Theory of Computation, 3rd Edition, Cengage Learning, 2013.

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

[64] R. Senda, Y. Takata and H. Seki, Generalized register context-free grammars, 13th International Conference on Language and Automata Theory and Applications (LATA 2019), LNCS 11417, 259–271.

[65] R. Senda, Y. Takata and H. Seki, Generalized register context-free grammars, IEICE Transactions on Information and Systems, E103-D(3), 540–548, 2020.

[66] R. Senda, Y. Takata and H. Seki, Forward regularity preservation property of register pushdown systems, IEICE Trans. Inf. & Syst., E104-D(3), 370–380, 2021.

[67] F. Song and T. Touili, Pushdown model checking for malware detection, TACAS 2012, extended version, IJSTTT, 16, 147–173, 2014.

[68] T. Takai, Y. Kaji and H. Seki, Right-linear finite path overlapping term rewriting systems effectively preserve recognizability, 11th International Conference on Rewriting Techniques and Applications (RTA 2000), LNCS 1833, 246–260.

[69] N. Tzevelekos, Fresh-register automata, 36th ACM Annual Symp. on Principles of Programming Languages (POPL 2009), 295–306.

[70] F. W. Vaandrager, Model learning, Commun. ACM, 60(2), 86—95, 2017.

[71] F. Vaandrager and A. Midya, A Myhill-Nerode theorem for register automata and symbolic trace languages, 17th International Colloquium on Theoretical Aspects of Computing (ICTAC 2020), LNCS 12545, 43–63.

[72] I. Walukiewicz, Pushdown processes: Games and model checking, 8th International Conference on Computer Aided Verification (CAV'96), LNCS 1102, 62–74.

[73] C. Xiaojuan and M. Ogawa, Well-structured extensions of pushdown systems, 24th Int. Conf. on Concurrency Theory (CONCUR 2013), 121–136.

Publication List

Journal Papers

- Ryoma Senda, Yoshiaki Takata and Hiroyuki Seki, Generalized Register Context-Free Grammars, IEICE Transactions on Information and Systems, Vol. E103-D, No.3, pp.540-548, March 2020.
- Ryoma Senda, Yoshiaki Takata and Hiroyuki Seki, Forward Regularity Preservation Property of Register Pushdown Systems, IEICE Transactions on Information and Systems, Vol.E104-D, No.3, pp.370-380, March 2021.
- Ryoma Senda, Yoshiaki Takata and Hiroyuki Seki, LTL Model Checking for Register Pushdown Systems, IEICE Transactions on Information and Systems, Vol. E104-D, No.12, pp.2131-2144, Dec 2021.
- Ryoma Senda, Yoshiaki Takata and Hiroyuki Seki, Complexity Results on Register Context-Free Grammars and Related Formalisms, Submitted to an international journal (special issue of ICTAC 2018).

Peer-Reviewed International Conference Papers

- Ryoma Senda, Yoshiaki Takata and Hiroyuki Seki, Complexity Results on Register Context-Free Grammars and Register Tree Automata, 15th International Colloquium on Theoretical Aspects of Computing (ICTAC 2018), Lecture Notes in Computer Science, Vol.11187, pp.415-434, 2018.
- Ryoma Senda, Yoshiaki Takata and Hiroyuki Seki, Generalized Register Context-Free Grammars, 13th International Conference on Language and Automata Theory and Applications (LATA 2019), Lecture Notes in Com-

puter Science, Vol.11417, pp.259-271, 2019.

- Ryoma Senda, Yoshiaki Takata and Hiroyuki Seki, Complexity Results on Register Pushdown Automata, Software Foundations for Data Interoperability (SFDI 2019+), https://arxiv.org/abs/1910.10357v1, 2019.
- Ryoma Senda, Yoshiaki Takata and Hiroyuki Seki, Reactive Synthesis from Visibly Register Pushdown Automata, 18th International Colloquium on Theoretical Aspects of Computing (ICTAC 2021), Lecture Notes in Computer Science, Vol.12819, pp.334-353, 2021.

Non-Reviewed Meetings

- Ryoma Senda and Hiroyuki Seki, Computational Complexity of Membership and Emptiness Problems for Register Context-Free Grammars, IEICE Technical Report, SS2017-41, Vol.117, No.381, pp.41-46, 2018, in Japanese.
- Ryoma Senda, Yoshiaki Takata and Hiroyuki Seki, Complexity Results on Register Context-Free Grammars and register tree automata, TRS meeting, 2019.
- Ryoma Senda, Yoshiaki Takata and Hiroyuki Seki, On the Regularity Preservation Property of Register Pushdown Systems, LA symposium, 2019.
- Ryoma Senda, Yoshiaki Takata and Hiroyuki Seki, LTL Model Checking for Register Pushdown Systems, IEICE Technical Report, SS2020-6, Vol.120, No.193, pp.7-12, 2020, online, in Japanese.