


Bisimulation Equivalence of Pushdown Automata Is Ackermann-Complete

Wenbo Zhang 


BASICS, Shanghai Jiao Tong University, Shanghai, China
wbzhang@sjtu.edu.cn

Qiang Yin¹ 

Alibaba Group, Shanghai, China
qiang.yq@alibaba-inc.com

Huan Long 

BASICS, Shanghai Jiao Tong University, Shanghai, China
longhuan@sjtu.edu.cn

Xian Xu 

East China University of Science and Technology, Shanghai, China
xuxian@ecust.edu.cn

Abstract

Deciding bisimulation equivalence of two pushdown automata is one of the most fundamental problems in formal verification. Though Sénizergues established decidability of this problem in 1998, it has taken a long time to understand its complexity: the problem was proven to be non-elementary in 2013, and only recently, Jančar and Schmitz showed that it has an ACKERMANN upper bound. We improve the lower bound to ACKERMANN-hard, and thus close the complexity gap.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Formal languages and automata theory

Keywords and phrases PDA, Bisimulation, Equivalence checking

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.141

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Acknowledgements This work is supported by NSF of China (61772336, 61872142, 61572318). The authors are grateful to Yuxi Fu for insightful discussions on this topic and support. We also thank the anonymous reviewers for their helpful comments and suggestions.

1 Introduction

In the area of formal verification, equivalence checking plays a central role in characterizing when two systems should be considered as the same. A classical equivalence is language equivalence, which asks if two processes recognize the same language. To characterize more refined behavioural relations, Milner proposed a fundamental equivalence called bisimulation equivalence (a.k.a. bisimilarity) [15]. Two processes are bisimilar to each other if every transition from one process can be simulated by the other one, and the resulting two processes keep in the same bisimilarity relation. If internal actions are allowed in a bisimulation step, we will get a more complicated equivalence called weak bisimilarity [15]. A seminal result proven in [1] shows that bisimulation equivalence is decidable for processes generated by context-free grammars, while the language equivalence between context-free grammars is well-known to be undecidable [6]. Extensive works followed up ever since, studying different equivalence relations on various infinite-state systems. See [13] for a survey.

¹ corresponding author



■ **Table 1** Complexity results of bisimulation equivalence problems for (variants of) PDA.

Model	Lower bound	Upper bound
DPDA	P-hard	TOWER [9, 28]
PDA	Ackermann-hard	ACKERMANN [10]
FOG	ACKERMANN-hard [9]	ACKERMANN [10]

Pushdown automata (PDA) extend finite-state automata with a stack memory and can be used to model recursive programs naturally. Since PDA recognize the same language as context-free grammars [6], language equivalence is undecidable. The language equivalence of deterministic PDA (DPDA), raised by Ginsburg and Greibach in [5], was first proved to be decidable for the real-time subclass, i.e., DPDA without internal actions [17, 16]. The decidable result was extended to DPDA in Sénizergues’s remarkable work [23]. Observe that on DPDA, language equivalence coincides with weak bisimulation equivalence, which implies the decidability of weak bisimilarity for DPDA. Sénizergues later generalized the decidability result to weak bisimilarity of PDA with deterministic internal actions [22, 24]. An internal action is deterministic if there is no alternative. There are also quite a few works trying to simplify Sénizergues’s proof [27, 25, 26, 7, 8]. Stirling revisited the decidability proof for DPDA via a tableau system [27]. He also generalized the tableau system for PDA [26]. Another line of work is conducted by Jančar in the framework of first-order grammars (FOG) [7, 8]. FOG has very close relationship with PDA [4]. It can describe PDA with deterministic internal actions by collapsed graphs where all internal actions are absorbed.

Concerning the complexity issue of bisimilarity of PDA, the best known upper bound for the deterministic case (DPDA) is TOWER [28, 9], while only P-hardness is known. For general PDA, EXPTIME-hardness was proven by Kučera and Mayr [14]. The EXPTIME-hardness even holds for a subclass of PDA, named BPA (Basic Process Algebra), of which the set of control states is a singleton [12]. The EXPTIME-hardness was further improved to non-elementary (TOWER-hard actually) [2]. This non-elementary lower bound also holds for the normed subclass, where every PDA process can empty its stack. As for the upper bound for bisimilarity of PDA, little was known until very recently. Jančar and Schmitz gave an ACKERMANN algorithm in [10]. This upper bound is actually proven in the framework of FOG. It also matches the ACKERMANN-hard lower bound for bisimilarity of FOG [9].

Observe that FOG are equivalent to PDA with deterministic internal actions. Without internal actions, the ACKERMANN-hardness of FOG cannot be applied to PDA. Thus the best known lower-bound for bisimilarity of PDA is still TOWER-hard.

Our Contribution. We show that bisimilarity of PDA is actually ACKERMANN-complete by improving the TOWER-hard lower bound to ACKERMANN-hard. This is done by a reduction from the coverability problem of reset Petri net [19]. Moreover, our reduction also gives rise to a parametric complexity result, i.e., \mathbf{F}_{d-1} -hardness if the number of control states $d \geq 4$ is fixed. Our proof extends an early work by Jančar [9], where similar results for first-order grammars are established. We improve the reduction by avoiding ε -rules.

We summarize some mentioned results in Table 1 with our result presented in bold.

Further Comments. According to Table 1, the complexity classes of bisimilarity problems for PDA and first-order grammars happen to be the same. Thus one may wonder whether these two models are actually equal with respect to bisimilarity. The answer was known to be negative [3]. In this paper, we present a new proof which shows that pushdown automata are strictly weaker than first-order grammars as far as bisimilarity is considered. It also demonstrates why the reduction in [9] cannot be applied to real-time PDA directly.

Organization. The rest of the paper is organized as follows. Section 2 introduces the background knowledge and necessary notations. Section 3 establishes the ACKERMANN hardness for bisimilarity of PDA. Section 4 shows that PDA are strictly weaker than FOG considering bisimilarity. Section 5 concludes this paper.

2 Preliminaries

2.1 Pushdown Automata

► **Definition 1** (PDA). A pushdown automaton $\mathcal{A} = (\mathcal{Q}, \Gamma, \Sigma, \mathcal{R})$ consists of

- a finite set of control states \mathcal{Q} ranged over by r, p, q ;
- a finite set of stack symbols Γ ranged over by X, Y, Z ;
- a finite set of actions Σ ranged over by a, b, c, d, f ;
- a finite set of rules $\mathcal{R} \subseteq \mathcal{Q} \times \Gamma \times \Sigma \times \mathcal{Q} \times \Gamma^*$.

The set Σ^* of words will be ranged over by u, v, w , and the set Γ^* of finite strings of stack symbols will be ranged over by α, β . We write $\alpha\beta$ (respectively uv) for the concatenation of α and β (respectively u and v). As usual, $|\alpha|$ and $|u|$ represent the length of α and u respectively. For $n \in \mathbb{N}$, we use a^n to denote n consecutive actions a , and similarly for X^n . We write $pX \xrightarrow{a} q\alpha$ to mean $(p, X, a, q, \alpha) \in \mathcal{R}$.

The syntax of a PDA process is $p\alpha$, where $p \in \mathcal{Q}$ and $\alpha \in \Gamma^*$. The size of process $p\alpha$, denoted by $|p\alpha|$, is defined as its stack height $|\alpha|$. The set of PDA processes \mathcal{P} is ranged over by O, P, Q . The semantics of the PDA processes is defined by the following rule:

$$\frac{pX \xrightarrow{a} q\alpha \in \mathcal{R}}{pX\beta \xrightarrow{a} q\alpha\beta} \quad (1)$$

If $w = a_1a_2 \dots a_n$ with $a_i \in \Sigma$ ($i \in 1, \dots, n$), then $P \xrightarrow{w} Q$ stands for $P \xrightarrow{a_1} P_1 \xrightarrow{a_2} \dots P_{n-1} \xrightarrow{a_n} Q$ for some P_1, P_2, \dots, P_{n-1} . A process P is *normed* if $P \xrightarrow{w} p$ for some $w \in \Sigma^*$ and $p \in \mathcal{Q}$, i.e., P can empty its stack. A PDA \mathcal{A} is *normed* (denoted as nPDA) if every process defined in \mathcal{A} is normed.

► **Definition 2** (Bisimulation). A binary relation $R \subseteq \mathcal{P} \times \mathcal{P}$ is a bisimulation if, for all $a \in \Sigma$, the following statements are valid:

1. whenever $(P, Q) \in R$ and $P \xrightarrow{a} P'$, then $Q \xrightarrow{a} Q'$ and $(P', Q') \in R$ for some Q' ;
2. whenever $(P, Q) \in R$ and $Q \xrightarrow{a} Q'$, then $P \xrightarrow{a} P'$ and $(P', Q') \in R$ for some P' .

The largest bisimulation relation, denoted by \sim , is an equivalence relation called *bisimulation equivalence* or *bisimilarity* [15].

When silent actions are considered, we use a special symbol ε to represent a silent action. Note that in Definition 1 we assume $\varepsilon \notin \Sigma$. PDA without silent actions are called *real-time* PDA. When silent actions are allowed, we will specify action set as $\Sigma_\varepsilon = \Sigma \uplus \{\varepsilon\}$. A rule of the form $pX \xrightarrow{\varepsilon} q\alpha$ is referred to as an ε -rule. For an ε -rule $pX \xrightarrow{\varepsilon} q\alpha$, we say it is popping if $|\alpha| < 1$; it is pushing if $|\alpha| > 1$; it is deterministic if $pX \xrightarrow{a} q'\alpha'$ implies $a = \varepsilon$, $q' = q$, and $\alpha' = \alpha$. We will write $\xRightarrow{\varepsilon}$ for the reflexive and transitive closure of $\xrightarrow{\varepsilon}$; and write \xRightarrow{a} for $\xRightarrow{\varepsilon} \xrightarrow{a} \xRightarrow{\varepsilon}$ if $a \neq \varepsilon$.

► **Definition 3** (Weak Bisimulation). A binary relation $R \subseteq \mathcal{P} \times \mathcal{P}$ is a weak bisimulation if for all $a \in \Sigma_\varepsilon$, the following statements are valid:

1. whenever $(P, Q) \in R$ and $P \xrightarrow{a} P'$, then $Q \xRightarrow{a} Q'$ and $(P', Q') \in R$ for some Q' ;
2. whenever $(P, Q) \in R$ and $Q \xrightarrow{a} Q'$, then $P \xRightarrow{a} P'$ and $(P', Q') \in R$ for some P' .

The largest weak bisimulation, denoted by \approx is called weak bisimilarity [15], and is also an equivalence relation.

Fast-Growing Complexity. We will use an ordinal-indexed hierarchy of “fast-growing” complexity classes defined in [18]. This hierarchy grows as $\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_3, \dots, \mathbf{F}_\omega, \mathbf{F}_{\omega+1} \dots$ and allows the classification of many decision problems with a non-elementary complexity. Here $\mathbf{F}_3 = \text{TOWER}$ is the lowest non-elementary complexity class; $\bigcup_{k \in \mathbb{N}} \mathbf{F}_k$ is the primitive-recursive complexity class; and $\mathbf{F}_\omega = \text{ACKERMANN}$ is the lowest non-primitive-recursive complexity class. A complexity class is closed under reduction functions from “lower” complexity class. For example, all classes starting from \mathbf{F}_3 are closed under elementary reduction.

Bisimilarity Problem for PDA. We are interested in the *bisimilarity problem* for PDA defined as follows. Given a PDA $\mathcal{A} = (\mathcal{Q}, \Gamma, \Sigma, \mathcal{R})$ and two process $p\alpha$ and $q\beta$, where $p, q \in \mathcal{Q}$ and $\alpha, \beta \in \Gamma^*$, the bisimilarity problem asks if $p\alpha \sim q\beta$.

Our main result stated as follows improves the previous TOWER-hard lower bound [2].

► **Theorem 4.** *The bisimilarity problem for PDA is ACKERMANN-hard and is \mathbf{F}_{d-1} -hard if the number of control states $d \geq 4$ is fixed.*

Combining with ACKERMANN upper bound from [10], we have the following result.

► **Corollary 5.** *The bisimilarity problem for PDA is ACKERMANN-complete.*

2.2 Bisimulation Game

Bisimulation equivalence has a nice game characterization called the *bisimulation game*.

Bisimulation Game. Given a pair of processes (P_0, Q_0) , a bisimulation game for (P_0, Q_0) is played between *Attacker* and *Defender*. The game is played in rounds. In round i , Attacker chooses a transition $P_{i-1} \xrightarrow{a_i} P_i$ (resp. $Q_{i-1} \xrightarrow{a_i} Q_i$), then Defender chooses a transition with a same action $Q_{i-1} \xrightarrow{a_i} Q_i$ (resp. $P_{i-1} \xrightarrow{a_i} P_i$). We use $(P_{i-1}, Q_{i-1}) \xrightarrow{a_i} (P_i, Q_i)$ to denote a round. Defender wins if it never gets stuck; otherwise Attacker wins. We say that one player has a winning strategy if it can always win no matter how the opponent plays. The following result is well known.

► **Lemma 6.** *$P \sim Q$ if and only if Defender has a winning strategy in the bisimulation game for (P, Q) .*

Macro Rules. Following [2], we introduce two kinds of macro rules to facilitate the design of bisimulation game and make our presentation concise.

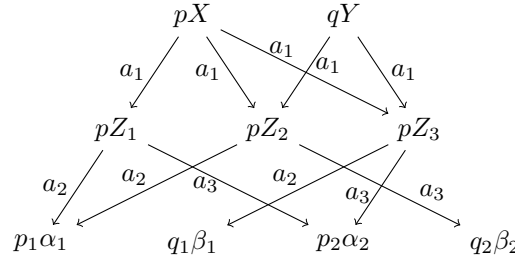
(1). A macro rule $(pX, qY) \xrightarrow{ATT} (p_1\alpha_1, q_1\beta_1)$ denotes a pair of transitions:

$$pX \xrightarrow{a} p_1\alpha_1 \quad qY \xrightarrow{a} q_1\beta_1$$

Here action a is *fresh*. This macro rule favours Attacker. In a bisimulation game for (pX, qY) , if Attacker chooses transition $pX \xrightarrow{a} p_1\alpha_1$ (or $qY \xrightarrow{a} q_1\beta_1$), then Defender is forced to choose transition $qY \xrightarrow{a} q_1\beta_1$ (or $pX \xrightarrow{a} p_1\alpha_1$).

(2). A macro rule $(pX, qY) \xrightarrow{DEF} \{(p_1\alpha_1, q_1\beta_1), (p_2\alpha_2, q_2\beta_2)\}$ denotes a set of transitions:

$$\begin{array}{lll} pX \xrightarrow{a_1} pZ_1 & pX \xrightarrow{a_1} pZ_2 & pX \xrightarrow{a_1} pZ_3 \\ qY \xrightarrow{a_1} pZ_2 & qY \xrightarrow{a_1} pZ_3 & \\ pZ_1 \xrightarrow{a_2} p_1\alpha_1 & pZ_1 \xrightarrow{a_3} p_2\alpha_2 & pZ_2 \xrightarrow{a_2} p_1\alpha_1 \quad pZ_2 \xrightarrow{a_3} q_2\beta_2 \\ pZ_3 \xrightarrow{a_2} q_1\beta_1 & pZ_3 \xrightarrow{a_3} p_2\alpha_2 & \end{array}$$



■ **Figure 1** State transition diagram of macro rules \xrightarrow{DEF} .

Here a_1, a_2, a_3 are fresh actions, and Z_1, Z_2 and Z_3 are fresh stack symbols. This macro rule favours Defender. It is powered by a useful technique called *Defender's forcing* [11]. The state transition diagram of this macro rule is shown in Fig. 1. In a nutshell, with this macro rule, Defender can decide whether the game should continue with $(p_1\alpha_1\alpha, q_1\beta_1\alpha)$ or $(p_2\alpha_2\alpha, q_2\beta_2\alpha)$ in the bisimulation game for $(pX\alpha, qY\alpha)$.

Let us take a look at the development of the game for $(pX\alpha, qY\alpha)$.

1. If the game reaches a configuration with two identical processes, Defender wins immediately. Thus Attacker's optimal choice in the first step is $pX\alpha \xrightarrow{a_1} pZ_1\alpha$.
2. Then Defender can make a choice between $qY\alpha \xrightarrow{a_1} pZ_2\alpha$ and $qY\alpha \xrightarrow{a_1} pZ_3\alpha$. If Defender chooses transition $qY\alpha \xrightarrow{a_1} pZ_2\alpha$, the game continues with $(pZ_1\alpha, pZ_2\alpha)$. If Defender chooses transition $qY\alpha \xrightarrow{a_1} pZ_3\alpha$, the game continues with $(pZ_1\alpha, pZ_3\alpha)$.
3. In the case of $(pZ_1\alpha, pZ_2\alpha)$, Attacker is forced to choose action a_3 , the game comes into $(p_2\alpha_2\alpha, q_2\beta_2\alpha)$. Similarly, in the case of $(pZ_1\alpha, pZ_3\alpha)$, Attacker is forced to choose action a_2 and the game reaches $(p_1\alpha_1\alpha, q_1\beta_1\alpha)$.

3 Lower Bound

We prove our main result by a reduction from the coverability problem of reset Petri net. We recall reset Petri net and its ACKERMANN-complete coverability problem in Section 3.1. We then construct an exponential time reduction in Section 3.2. Although the exponential time reduction suffices for our purpose, we revise it to a polynomial one in Section 3.3.

3.1 Reset Petri Net

Reset Petri Net (RPN). A reset Petri net is a tuple $\mathcal{N} = (\mathcal{S}, \mathcal{C}, \delta)$ consists of

- a finite set of control states \mathcal{S} ranged over by s, t ;
- a finite set of counters $\mathcal{C} = \{c_1, c_2, \dots, c_d\}$;
- a finite set of instructions $\delta \subseteq \mathcal{S} \times \mathcal{O} \times \mathcal{S}$, where the set of operations \mathcal{O} consists of $\text{INC}(c_i)$, $\text{DEC}(c_i)$ and $\text{RESET}(c_i)$ for $i = 1, 2, \dots, d$.

A configuration is a tuple (s, n_1, \dots, n_d) with $s \in \mathcal{S}$ representing the current state, and $n_1, \dots, n_d \in \mathbb{N}$ representing the current contents of the counters. If $(s, op, t) \in \delta$ then we have $(s, n_1, \dots, n_d) \rightarrow (t, n'_1, \dots, n'_d)$ in the following cases:

- $op = \text{INC}(c_i)$, $n'_i = n_i + 1$, and $n'_j = n_j$ for all $j \neq i$; or
- $op = \text{DEC}(c_i)$, $n_i > 0$, $n'_i = n_i - 1$, and $n'_j = n_j$ for all $j \neq i$; or
- $op = \text{RESET}(c_i)$, $n'_i = 0$, and $n'_j = n_j$ for all $j \neq i$.

By \rightarrow^* we denote the reflexive and transitive closure of \rightarrow . We define a partial order \leq on the configurations of \mathcal{N} :

$$(s, n_1, \dots, n_d) \leq (t, m_1, \dots, m_d) \text{ if } s = t \wedge n_1 \leq m_1 \wedge \dots \wedge n_d \leq m_d.$$

We say σ_2 is *coverable* from σ_1 if there is some σ such that $\sigma_1 \rightarrow^* \sigma$ and $\sigma \geq \sigma_2$.

Coverability Problem of RPN. Given a Reset Petri Net $\mathcal{N} = (\mathcal{S}, \mathcal{C}, \delta)$, an initial configuration σ_1 and a final configuration σ_2 , where the counters of σ_1 and σ_2 are given in binary, the *coverability* problem asks if σ_2 is coverable from σ_1 .

We recall the following complexity result for coverability problem of RPN [19, 20, 21].

► **Theorem 7.** *The coverability problem of RPN is ACKERMANN-complete, and is \mathbf{F}_d -complete if the number of counters $d \geq 3$ is fixed.*

3.2 An Exponential Time Reduction

Given a RPN $\mathcal{N} = (\mathcal{S}, \mathcal{C}, \delta)$, an initial configuration σ_1 , and a final configuration σ_2 , we will construct a PDA $\mathcal{A} = (\mathcal{Q}, \Gamma, \Sigma, \mathcal{R})$ and two processes P, Q of \mathcal{A} in exponential time such that

$$\sigma_2 \text{ is coverable from } \sigma_1 \text{ if and only if } P \not\sim Q. \quad (\star)$$

Reduction Overview. Our reduction encodes the run of \mathcal{N} from configuration σ_1 as a bisimulation game for (P, Q) . In the bisimulation game, Attacker aims to show that σ_2 is coverable from σ_1 , while Defender aims to show the opposite.

In order to complete the reduction, we should pay attention to the following aspects.

- A configuration σ' of \mathcal{N} corresponds to a game for (P', Q') in the sense that the state and counter values of σ' are both encoded on the stack of both P' and Q' .
- To track the counters in the run from σ , the bisimulation game pushes every counter operation from the initial configuration on the stacks. Observe that the value of a counter of \mathcal{N} can never become negative. Our reduction will guarantee that Attacker can never cheat by decreasing a counter with value zero. This is fulfilled by Defender's forcing. More specifically, for every $\text{DEC}(c_i)$ operation, Defender has the power to verify its validity by initiating what we shall call a *zero check* for c_i .
- If a configuration that covers σ_2 is reached, Attacker wins the game. In the reduction, we will introduce a special witness action f . When a configuration $\sigma \geq \sigma_2$ is reached, the game will finally come into some (P', Q') where P' can do action f while Q' cannot.

As mentioned earlier, the basic idea of our reduction follows from [9], where ACKERMANN-hardness is proven for the bisimilarity problem of first-order grammars. However, as what will become clear in Section 4, first-order grammars are strictly more powerful than PDA w.r.t. bisimilarity. Here we highlight the main differences between our reduction and the one in [9].

- The reduction in [9] records the increasing and decreasing operations of d counters into $2d$ sub-processes (i.e. sub-terms in the terminology of first-order grammars). These sub-processes can work “in parallel” and be accessed without being interfered by each other. Zero check of a specific counter is achieved by skipping irrelevant sub-processes and comparing the relevant ones directly. Due to the sequential nature of stacks, this is beyond the reach of (real-time) PDA (see also Section 4). Instead, the increasing and

decreasing operations recorded on the stack are interfered by each other unavoidably. To check zero for a counter, we introduce a novel mechanism by comparing the stack as a whole (Lemma 11).

- Our reduction uses $d+1$ PDA control states to encode a RPN with d counters. This yields a \mathbf{F}_{d-1} -hardness result, a first parametric lower-bound for the bisimilarity problem for PDA. In contrast, parametric complexity is not studied in [9]. Moreover, if we transform the reduction for first-order grammars [9] directly to a new reduction for PDA, the resulting PDA would require (i) at least $2d$ control states and (ii) popping ε -rules.

► **Remark 8.** In [9], the ACKERMANN-hardness for bisimilarity of FOG is established by reduction from the *control state reachability problem* of RPN. When the counters in the configurations of RPN are given in binary, the coverability problem of RPN is equivalent to the control state reachability problem under exponential time reduction. Incorporating the counter encoding and zero check trick into the reduction of [9], we can get the ACKERMANN-hard lower bound for bisimilarity of PDA as well. The main reason that we choose the coverability problem of RPN instead is to build a parametric lower bound under polynomial time reduction.

The Reduction. We fix a RPN $\mathcal{N} = (\mathcal{S}, \mathcal{C}, \delta)$, an initial configuration $\sigma_1 = (t_s, n_1, \dots, n_d)$, a final configuration $\sigma_2 = (t_f, m_1, \dots, m_d)$ in this subsection. The corresponding PDA $\mathcal{A} = (\mathcal{Q}, \Gamma, \Sigma, \mathcal{R})$ and two processes P, Q are defined as follows.

1. We introduce $d+1$ control states:

$$\mathcal{Q} \stackrel{\text{def}}{=} \{p, q_1, q_2, \dots, q_d\}$$

Here q_1, q_2, \dots, q_d correspond to c_1, c_2, \dots, c_d of \mathcal{C} , respectively. The usage of state p will become clear later.

2. To record the operations on counters in \mathcal{C} , we introduce the following stack symbols:

$$\Gamma_{\mathcal{C}} \stackrel{\text{def}}{=} \{X_i^+, X_i^-, X_i^0 : i = 1, 2, \dots, d\} \subseteq \Gamma$$

Here X_i^+, X_i^- and X_i^0 represent operations $\text{INC}(c_i)$, $\text{DEC}(c_i)$ and $\text{RESET}(c_i)$, respectively.

3. To record the states of \mathcal{N} , for every state $s \in \mathcal{S}$, we introduce a pair of stack symbols:

$$\{Y_s, Y'_s\} \subseteq \Gamma$$

4. The two processes P and Q of the initial bisimulation game configuration are defined by

$$P \stackrel{\text{def}}{=} pY_s(X_1^+)^{n_1} \dots (X_d^+)^{n_d} \quad Q \stackrel{\text{def}}{=} pY'_s(X_1^+)^{n_1} \dots (X_d^+)^{n_d}$$

In the rest of this subsection, we will complete the definition of \mathcal{A} to validate property (\star) . We start with encoding counters of \mathcal{N} on the stacks of PDA processes. We then introduce rules for \mathcal{A} to manipulate counters and perform zero checks. Last we show how to verify the coverability property in bisimulation games.

Counter Encoding. For each $i = 1, 2, \dots, d$, we introduce a function $\text{count}_i : \Gamma_{\mathcal{C}}^* \rightarrow \mathbb{Z}$. Let $\alpha = X_n X_{n-1} \dots X_1 \in \Gamma_{\mathcal{C}}^*$. Intuitively, $\text{count}_i(\alpha)$ computes the value of c_i after a sequence of operations X_1, X_2, \dots, X_n . More specifically, let K_i be the largest index such that $X_{K_i} = X_i^0$, i.e., the leftmost occurrence of a reset operation of counter c_i . If X_i^0 does not appear in α , then $K_i = 0$. Let I_i and D_i be the respective numbers of occurrences of X_i^+ and X_i^- in the subsequence $X_n X_{n-1} \dots X_{K_i+1}$ of α . The function count_i is defined by $\text{count}_i(\alpha) = I_i - D_i$.

► **Remark 9.** Since the counters in reset Petri net cannot be negative, some strings, e.g. $X_1^- X_1^- X_1^+$, are invalid. For simplicity, the functions count_i are defined on all strings in Γ_C^* .

► **Example 10.** $\text{count}_1(X_2^+ X_1^0 X_2^+ X_1^+) = 0$ and $\text{count}_2(X_2^+ X_1^0 X_2^+ X_1^+) = 2$.

We next introduce rules for counter manipulations. The PDA processes record every operation to the counters on their stacks. The trick here is to avoid invalid operations on counters. For example, at configuration $(pY_s X_1^- X_1^+, pY'_s X_1^- X_1^+)$, the counter c_1 is zero and it is not supposed to push another X_1^- . Here we use the Defender's forcing technique to fulfill this goal. Defender has a chance to force Attacker to check whether a decrease operation is valid. If it is not valid, Defender can win the game.

Counter Manipulation. For each instruction $I = (s, op, t)$ of \mathcal{N} , we introduce a set of PDA rules to \mathcal{R} for counter manipulations as follows.

- If $op = \text{INC}(c_i)$, we introduce $(pY_s, pY'_s) \xrightarrow{ATT} (pY_t X_i^+, pY'_t X_i^+)$.
- If $op = \text{RESET}(c_i)$, we introduce $(pY_s, pY'_s) \xrightarrow{ATT} (pY_t X_i^0, pY'_t X_i^0)$.
- If $op = \text{DEC}(c_i)$, we introduce $(pY_s, pY'_s) \xrightarrow{DEF} \{(pY_t X_i^-, pY'_t X_i^-), (p, q_i)\}$.

We will make more comments on the decrease operation. The control states q_1, q_2, \dots, q_d are mainly used to zero check the counters c_1, c_2, \dots, c_d , respectively. In the sequel, we will introduce some rules to ensure the correctness of zero checks as stated in the following lemma.

► **Lemma 11.** $q_i \alpha \sim p \alpha$ if and only if $\text{count}_i(\alpha) = 0$, where $\alpha \in \Gamma_C^*$.

By Lemma 11, if Attacker intends to do an invalid decrease operation on counter c_i , Defender can force the game to the branch of (p, q_i) and win. If Attacker intends to do a valid decrease operation on counter c_i , Defender would lose in the branch of (p, q_i) . In that case, Defender will force the game to the branch of $(pY_t X_i^-, pY'_t X_i^-)$ and the game continues.

Zero Check. We introduce the following rules to ensure the correctness of zero check. For the sake of concision, we introduce another macro rule. Given $w = a_1 a_2 \dots a_n$, we use $pX \xrightarrow{w} q\alpha$ to stand for the sequence of transitions $pX \xrightarrow{a_1} pX_1 \xrightarrow{a_2} \dots \xrightarrow{a_{n-1}} pX_{n-1} \xrightarrow{a_n} q\alpha$. Note that these macro rules do not introduce any new state. All the stack symbols introduced in these macro rules are fresh.

$$\begin{array}{ll}
pX \xrightarrow{b_{pop} b_{pop}} p & \text{For } X \in \Gamma_C \\
q_i X_i^- \xrightarrow{b_{pop}} q_i & \text{For } i = 1, 2, \dots, d \\
q_i X_i^+ \xrightarrow{b_{pop} b_{pop} b_{pop}} q_i & \text{For } i = 1, 2, \dots, d \\
q_i X_i^0 \xrightarrow{b_{pop} b_{pop}} p & \text{For } i = 1, 2, \dots, d \\
q_i X \xrightarrow{b_{pop} b_{pop}} q_i & \text{For } i = 1, 2, \dots, d, \quad X \in \Gamma_C \setminus \{X_i^-, X_i^+, X_i^0\}
\end{array}$$

Proof of Lemma 11. Observe that the bisimilarity between $q_i \alpha$ and $p \alpha$ only depends on the total numbers of consecutive actions b_{pop} from $q_i \alpha$ and $p \alpha$. Process $p \alpha$ can do action b_{pop} twice for every stack symbol. Process $q_i \alpha$ can do three b_{pop} actions for each X_i^+ and one b_{pop} action for each X_i^- before it meets the first X_i^0 . Let I_i and D_i be the numbers of X_i^+ and X_i^- before it meets the first X_i^0 . It follows that $q_i \alpha \sim p \alpha$ if and only if $3I_i + D_i + 2(|\alpha| - I_i - D_i) = 2|\alpha|$. Note that $\text{count}_i(\alpha) = I_i - D_i$. Thus $q_i \alpha \sim p \alpha$ if and only if $\text{count}_i(\alpha) = 0$. ◀

► **Remark 12.** The implementation of counters and their zero check mechanism is the main technical difference of our reduction from the one for first-order grammars [9]. The reduction in [9] compares the increasing and decreasing operations of a counter directly, by skipping non-relevant operations. As will be seen in Section 4, the ability to skip non-relevant operations also makes first-order grammars more powerful than PDA.

Coverability Check. When the game reaches $(pY_{t_f}\alpha, pY'_{t_f}\alpha)$, Attacker can initiate a coverability check. Then Defender will choose a counter c_i and check if $\text{count}_i(\alpha) \geq m_i$. To this end, we introduce $d + 1$ pairs of stack symbols $(Z_1, Z'_1), \dots, (Z_{d+1}, Z'_{d+1})$ for counter choice. For each $1 \leq i \leq d$, we introduce $m_i + 1$ pairs of stack symbols $(Z_{i,0}, Z'_{i,0}), \dots, (Z_{i,m_i}, Z'_{i,m_i})$ for coverability check of counter c_i .

(1). We first add rules to check coverability of a specific counter c_i ($1 \leq i \leq d$). For the pair $(pZ_{i,n}, pZ'_{i,n})$, where $0 \leq n \leq m_i$, we introduce the following rules.

- $(pZ_{i,n}, pZ'_{i,n}) \xrightarrow{\text{DEF}} \{(pZ_{i,n-1}X_i^-, pZ'_{i,n-1}X_i^-), (p, q_i)\};$
- $pZ_{i,0} \xrightarrow{f} p, \quad pZ'_{i,0} \xrightarrow{b_{pop}} p.$

If $\text{count}_i(\alpha) < m_i$, then by Defender's forcing, Defender can push $\text{count}_i(\alpha)$ number of X_i^- to the stack and the game reaches $(p\gamma\alpha, p_i\gamma\alpha)$, where $\gamma = (X_i^-)^{\text{count}_i(\alpha)}$. By Lemma 11, Defender wins. If $\text{count}_i(\alpha) \geq m_i$, Defender's best strategy would lead the game into a configuration $(pZ_{i,0}\beta\alpha, pZ'_{i,0}\beta\alpha)$ where $\beta = (X_i^-)^{m_i}$. Attacker wins since $pZ_{i,0}\beta\alpha$ admits a fresh action f , while $pZ'_{i,0}\beta\alpha$ does not. As a result, we have the following lemma.

► **Lemma 13.** $pZ_{i,m_i}\alpha \sim pZ'_{i,m_i}\alpha$ if and only if $\text{count}_i(\alpha) < m_i$, where $\alpha \in \Gamma^*$.

(2). We next introduce the following rules to help Defender pick a specific counter to perform coverability check.

- $(pY_{t_f}, pY'_{t_f}) \xrightarrow{\text{ATT}} (pZ_1, pZ'_1);$
- $(pZ_i, pZ'_i) \xrightarrow{\text{DEF}} \{(pZ_{i,m_i}, pZ'_{i,m_i}), (pZ_{i+1}, pZ'_{i+1})\}$ for $i = 1, 2, \dots, d;$
- $pZ_{d+1} \xrightarrow{f} p, \quad pZ'_{d+1} \xrightarrow{b_{pop}} p.$

Suppose that Attacker initiates a coverability check and the game reaches $(pZ_1\alpha, pZ'_1\alpha)$. Defender can choose a specific counter to verify by Defender's forcing. Indeed if $\text{count}_i(\alpha) < m_i$ for some $1 \leq i \leq d$, Defender can force the game to $(pZ_{i,m_i}\alpha, pZ'_{i,m_i}\alpha)$. By Lemma 13, Defender wins. If $\text{count}_i(\alpha) \geq m_i$ for all $1 \leq i \leq d$, then Defender's best strategy is leading the game to configuration $(pZ_{d+1}\alpha, pZ'_{d+1}\alpha)$. Attacker wins via a fresh action f .

► **Proposition 14.** *The coverability problem of RPN (with d counters) can be reduced to the complement of the bisimilarity problem for PDA (with $d + 1$ states) in exponential time.*

Proof. We construct the reduction as described in the section and prove the property (\star) . (\Rightarrow) . If (t_f, m_1, \dots, m_d) is coverable from (t_s, n_1, \dots, n_d) , then there exists a configuration $(t_f, m'_1, \dots, m'_d) \geq (t_f, m_1, \dots, m_d)$, which is reachable from (t_s, n_1, \dots, n_d) . We describe a winning strategy for Attacker. Attacker simply follows the path from (t_s, n_1, \dots, n_d) to (t_f, m'_1, \dots, m'_d) . Since every decrease operation is valid, Defender will not ask for zero checks. The bisimulation game will reach $(pY_{t_f}\alpha, pY'_{t_f}\alpha)$ such that $\text{count}_i(\alpha) = m'_i$ for each $i \in \{1, 2, \dots, d\}$. Attacker starts a coverability check and wins the game.

141:10 Bisimulation Equivalence of Pushdown Automata Is Ackermann-Complete

(\Leftarrow). If (t_f, m_1, \dots, m_d) is not coverable from (t_s, n_1, \dots, n_d) , we describe a winning strategy for Defender. Defender just follows what Attacker does and asks for zero check if Attacker tries to decrease a zero counter. Whenever the game reaches $(pY_{t_f}\beta, pY'_{t_f}\beta)$ for some $\beta \in \Gamma^*$, there exists some $i \in \{1, 2, \dots, d\}$, $\text{count}_i(\beta) < m_i$. If Attacker asks for a coverability check, then Defender can choose to verify the value of counter c_i . By Lemma 13, Defender wins. If Attacker never initiates a coverability check, Defender also wins as it will never get stuck. \blacktriangleleft

By Theorem 7 and Proposition 14, we have our main result Theorem 4. Observe that it is safe to introduce rules $pY_s \xrightarrow{b_{pop}} p$ and $pY'_s \xrightarrow{b_{pop}} p$ for every $s \in \mathcal{S}$ since these transitions will never be chosen by Attacker (or Attacker loses immediately). As a result, the PDA \mathcal{A} is normed and our lower bound can be generalized to the normed case.

► **Corollary 15.** *The bisimilarity problem of normed PDA is ACKERMANN-complete and is F_{d-1} -hard if the number of control states is $d \geq 4$.*

3.3 A Polynomial Time Reduction

Although an elementary reduction suffices for our purpose, we show that the exponential time reduction can be actually revised to be polynomial. There are two exponential factors in the reduction presented in Section 3.2: (i) the size of the initial configuration of the bisimulation game can be exponentially large; and (ii) the number of rules for coverability check can be exponentially many. We eliminate both by incorporating a binary representation of counters.

Counter Binary Encoding. Let m be the maximal number occurs in the initial and final configuration of \mathcal{N} , and k the smallest number such that $2^k \geq m$. For every counter c_i , we introduce two new stack symbols $X_{i,j}^+$ and $X_{i,j}^-$ for each $j \in \{0, 1, \dots, k\}$. Intuitively, $X_{i,j}^+$ (resp. $X_{i,j}^-$) represents 2^j increasing (resp. decreasing) operations on counter c_i . Stack symbol X_i^0 is still used to represent operation $\text{RESET}(c_i)$. We then replace the initial game configuration by this binary representation. The function count_i can also be revised easily by taking the binary representation into account. We omit the details.

► **Example 16.** An initial RPN configuration $\sigma_1 = (s, 6, 3)$ can be encoded by a game configuration $(pY_s X_{1,2}^+ X_{1,1}^+ X_{2,1}^+ X_{2,0}^+, pY'_s X_{1,2}^+ X_{1,1}^+ X_{2,1}^+ X_{2,0}^+)$.

We next revise the rules to make zero check and coverability check work.

Zero Check. We introduce new rules for $X_{i,j}^+$ and $X_{i,j}^-$ for zero check, where $0 \leq j \leq k$.

$$\begin{array}{ll}
 \text{— } pX_{i,j}^+ \xrightarrow{b_{pop}b_{pop}} pX_{i,j-1}^+ \dots X_{i,0}^+, & pX_{i,0}^+ \xrightarrow{b_{pop}b_{pop}} p; \\
 \text{— } pX_{i,j}^- \xrightarrow{b_{pop}b_{pop}} pX_{i,j-1}^- \dots X_{i,0}^-, & pX_{i,0}^- \xrightarrow{b_{pop}b_{pop}} p; \\
 \text{— } q_i X_{i,j}^+ \xrightarrow{b_{pop}b_{pop}b_{pop}} q_i X_{i,j-1}^+ \dots X_{i,0}^+, & q_i X_{i,0}^+ \xrightarrow{b_{pop}b_{pop}b_{pop}} q_i; \\
 \text{— } q_i X_{i,j}^- \xrightarrow{b_{pop}} q_i X_{i,j-1}^- \dots X_{i,0}^-, & q_i X_{i,0}^- \xrightarrow{b_{pop}} q_i;
 \end{array}$$

Note that besides the rules introduced above, we keep the rule $q_i X_i^0 \xrightarrow{b_{pop}b_{pop}} p$ for $i \in \{1, \dots, d\}$. Lemma 11 is still valid and can be proved along the same line.

Coverability Check. Let m_i be the value of c_i in the final configuration σ_2 and k_i be the least number such that $2^{k_i} - 1 \geq m_i$. We replace the rules for $(pZ_{i,j}, pZ'_{i,j})$ ($0 \leq j \leq m_i$) by

- $(pZ_{i,m_i}, pZ'_{i,m_i}) \xrightarrow{ATT} (pW_{i,0}\beta, pW'_{i,0}\beta), \quad (pW_{i,k_i}, pW'_{i,k_i}) \xrightarrow{ATT} (p, q_i);$
- $(pW_{i,j}, pW'_{i,j}) \xrightarrow{DEF} \{(pW_{i,j+1}X_{i,j}^-, pW'_{i,j+1}X_{i,j}^-), (pW_{i,j+1}, pW'_{i,j+1})\} \quad (0 \leq j < k_i)$

Here $W_{i,0}, \dots, W_{i,k_i}$ are new stack symbols for coverability check and β consists of the binary stack symbols introduced above such that $\text{count}_i(\beta) = 2^{k_i} - m_i$.

We show that Lemma 13 still holds. Consider the bisimulation game for $(pZ_{i,m_i}\alpha, pZ'_{i,m_i}\alpha)$. Intuitively, the coverability check starts with increasing counter c_i with $2^{k_i} - m_i$. After that, Defender decreases c_i by a combination of k_i choices and the game reaches a configuration $(p\gamma\alpha, q_i\gamma\alpha)$. If $\text{count}_i(\alpha) < m_i$, Defender can pick γ such that $\text{count}_i(\gamma\alpha) = 0$. By Lemma 11, Defender wins. If $\text{count}_i(\alpha) \geq m_i$, then $\text{count}_i(\alpha) + (2^{k_i} - m_i) - (2^{k_i} - 1) > 0$. The counter c_i can never be decreased to zero and Attacker wins.

4 Relative Expressiveness of PDA

First-order grammars have close relationship with pushdown automata [7, 9, 10], as they are equivalent to PDA with deterministic and popping ε -rules. More specifically, any process T definable in a first-order grammar can be encoded into a PDA process P with deterministic and popping ε -rules, and vice versa. The processes T and P are weakly bisimilar. In the real-time case, i.e, when no ε -rules is allowed, FOG have strictly richer behaviours than PDA. This can be proved by considering the transition graph of real-time PDA and PDA with deterministic ε -rules (see, e.g., [3], Proposition 5.8).

Nevertheless, to build better intuition for this result and also demonstrate why the original reduction in [9] cannot be directly applied to real-time PDA, we give another proof in this section to show that FOG are strictly stronger than PDA, as far as bisimilarity is considered.

To avoid introducing the definition of FOG, we construct a special PDA with deterministic and popping ε -rules and show that it can define a process which cannot be bisimulated by any real-time PDA process. The special PDA \mathcal{A} contains the following rules:

- $pX_0 \xrightarrow{a} pX, \quad pX \xrightarrow{a} pXX, \quad pX \xrightarrow{b} pYX, \quad pY \xrightarrow{b} pYY;$
- $pY \xrightarrow{c} p_1Y, \quad pY \xrightarrow{d} p_2Y;$
- $p_1Y \xrightarrow{\varepsilon} p_1, \quad p_1X \xrightarrow{a} p_1, \quad p_2Y \xrightarrow{b} p_2, \quad p_2X \xrightarrow{\varepsilon} p_2.$

Observe that pX_0 can do actions $a^m b^n$ and record m and n by pushing $Y^m X^n$ into stack. The popping ε -rules enable the process to recover the information of m or n immediately (via doing actions a^m after c or actions b^n after d). We will show that such ε -rules are necessary in the sense that no real-time PDA is weakly bisimilar to \mathcal{A} . Intuitively, any real-time PDA process, after executing actions $a^m b^n$, cannot access both the information of m and n on the stack without undesired interference. The necessity of ε -rules is also the basic reason why the reduction in the hardness proof for first-order grammars [9] fails for real-time PDA.

► **Proposition 17.** *For any real-time PDA process O , $O \not\approx pX_0$.*

A direct consequence is that first-order grammars are strictly stronger than PDA. Indeed, following the transformation in [10], one can construct a process T in a first-order grammar such that (i) $T \approx pX_0$ and (ii) T has no ε transitions (the ε transitions are absorbed in FOG during transformation). As a result T cannot be related to any PDA process by bisimilarity.

► **Corollary 18.** *There exists a process T definable by a first-order grammar, such that for any real-time PDA process O , $O \not\sim T$.*

141:12 Bisimulation Equivalence of Pushdown Automata Is Ackermann-Complete

We prove Proposition 17 in the rest of this section. To do that, we need a useful pumping property of pushdown automata. We say a transition sequence $\xi : P_0 \xrightarrow{a_1} P_1 \xrightarrow{a_2} \dots \xrightarrow{a_k} P_k$ is *non-sinking* if $|P_i| \geq |P_0|$ for $1 \leq i \leq k$. Moreover, ξ is *pumpable* if (i) ξ is *non-sinking* and (ii) $P_0 = pZ\alpha$ and $P_k = pZ\beta\alpha$ for some $p \in \mathcal{Q}$, $Z \in \Gamma$, $\alpha \in \Gamma^*$, and $\beta \in \Gamma^+$.

► **Lemma 19.** *Suppose that P_0 is a PDA process defined in $\mathcal{A} = (\mathcal{Q}, \Gamma, \Sigma, \mathcal{R})$ and it holds that $|\alpha| \leq 2$ for each $(p, X, a, q, \alpha) \in \mathcal{R}$. If there exists a transition sequence $\xi : P_0 \xrightarrow{a_1} P_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} P_n$ such that (i) $n \geq |P_0| \cdot (|\mathcal{Q}||\Gamma| + 1)^{|\mathcal{Q}||\Gamma|+1}$ and (ii) $P_i \not\sim P_j$ ($\forall i \neq j, 0 \leq i, j \leq n$), then ξ contains a pumpable transition subsequence.*

Proof. Let $\xi' : Q_0 \xrightarrow{b_1} Q_1 \xrightarrow{b_2} \dots \xrightarrow{b_m} Q_m$ be a *non-sinking* transition sequence such that $Q_i \not\sim Q_j$ ($\forall i \neq j$). Denote by $\text{type}(\xi')$ the set of different pairs of state and top stack symbol that occur in ξ' . More specifically, $\text{type}(\xi') \stackrel{\text{def}}{=} \{(q, Z) : \text{there is a process } qZ\beta \text{ in } \xi'\}$. It is sufficient to prove the following property:

if $m \geq (|\text{type}(\xi')| + 1)^{|\text{type}(\xi')|+1}$, then ξ' contains a pumpable transition subsequence. (★★)

Our lemma is established by observing that ξ contains a non-sinking subsequence ξ' of length at least $(|\mathcal{Q}||\Gamma| + 1)^{|\mathcal{Q}||\Gamma|+1}$. Suppose otherwise, then in less than $|P_0| \cdot (|\mathcal{Q}||\Gamma| + 1)^{|\mathcal{Q}||\Gamma|+1}$ steps P_0 will reach a process $p'\varepsilon$ for some $p' \in \mathcal{Q}$. This contradicts with our assumption (i).

We next prove (★★) by induction on $|\text{type}(\xi')|$.

- $|\text{type}(\xi')| = 1$. It is trivial since ξ' is pumpable.
- $|\text{type}(\xi')| = k + 1$. By assumption, ξ' is of length at least $(k + 2)^{k+2}$. Since ξ' is non-sinking and any two processes in ξ' are not equal, there are no more than k other processes of the size $|Q_0|$. By removing these (shortest) processes, we get at most $k + 1$ transition subsequences. There must be a sequence (denoted as ξ'') of length at least $(k + 1)^{k+1}$. By assumption of \mathcal{A} , the stack height will increase at most by one in a transition. It follows that ξ'' is non-sinking. Let $Q_0 = qZ\alpha$. If $(q, Z) \in \text{type}(\xi'')$, say $Q_j = qZ\beta\alpha$, then $Q_0 \xrightarrow{b_1} \dots \xrightarrow{b_j} Q_j$ is pumpable. Otherwise $|\text{type}(\xi'')| = k$. By induction hypothesis, ξ'' contains a pumpable transition subsequence. ◀

We are ready to prove Proposition 17.

Proof of Proposition 17. Suppose otherwise and let O be a process of a real-time PDA $\mathcal{B} = (\mathcal{Q}, \Gamma, \Sigma, \mathcal{R})$ and $O \approx pX_0$. W.l.o.g., we assume that $|\alpha| \leq 2$ for each $(p, X, q, \alpha) \in \mathcal{R}$. Let $N = |\mathcal{Q}||\Gamma| + 1$ and $m \geq 1$. Consider the following transition sequence of pX_0

$$pX_0 \xrightarrow{a^m} pX^m \xrightarrow{b} pYX^m \xrightarrow{b} \dots \xrightarrow{b} pY^n X^m. \quad (2)$$

Since $O \approx pX_0$ and O is real-time, there exists $O_{m,i}$ ($0 \leq i \leq n$) such that

$$O \xrightarrow{a^m} O_{m,0} \xrightarrow{b} O_{m,1} \xrightarrow{b} \dots \xrightarrow{b} O_{m,n} \quad (3)$$

and $O_{m,i} \approx pY^i X^m$ for $0 \leq i \leq n$. Observe that $pY^i X^m \not\approx pY^j X^m$ ($\forall i \neq j$). Hence $O_{m,i} \not\sim O_{m,j}$ ($\forall i \neq j$). Let $n = |O_{m,0}|N^N$. By Lemma 19, we can find $q \in \mathcal{Q}$ and $Z \in \Gamma$ such that

- (i) $qZ\alpha = O_{m,s}$ and $qZ\beta\alpha = O_{m,t}$ for some $0 \leq s < t \leq n$, $\alpha \in \Gamma^*$ and $\beta \in \Gamma^+$; and
- (ii) the transition subsequence of $O_{m,s} \xrightarrow{b} O_{m,s+1} \xrightarrow{b} \dots \xrightarrow{b} O_{m,t}$ is pumpable.

Let $k = t - s$, we can repeat the pumpable transition sequence as many times as we want

$$qZ\alpha \xrightarrow{b^k} qZ\beta\alpha \xrightarrow{b^k} qZ\beta\beta\alpha \xrightarrow{b^k} \dots \quad (4)$$

As $qZ\alpha = O_{m,s} \approx pY^s X^m$, the above sequence must be simulated by

$$pY^s X^m \xrightarrow{b^k} pY^{s+k} X^m \xrightarrow{b^k} pY^{s+2k} X^m \xrightarrow{b^k} \dots \quad (5)$$

such that $qZ\beta^i\alpha \approx pY^{s+ki} X^m$ for $i \geq 0$.

For different m , we can have different transition sequences (4) and (5), possibly with different $s > 0$, $k > 0$, $q \in \mathcal{Q}$, $Z \in \Gamma$, $\alpha \in \Gamma^*$ and $\beta \in \Gamma^+$. By the pigeonhole principle, there are two different numbers $1 \leq m_1 < m_2 \leq N$ so that q and Z are the same in transition sequence (4). Assume w.l.o.g. that $qZ\alpha_1 = O_{m_1,s_1} \approx pY^{s_1} X^{m_1}$, $qZ\alpha_2 = O_{m_2,s_2} \approx pY^{s_2} X^{m_2}$ and $qZ \xrightarrow{b^{k_1}} qZ\beta_1$ for some $s_1, s_2, k_1 > 0$, $\alpha_1, \alpha_2 \in \Gamma^*$ and $\beta_1 \in \Gamma^+$. The actions $O_{m_1,s_1} \xrightarrow{b^{k_1 \cdot N}} qZ\beta_1^N \alpha_1$ must be simulated by $pY^{s_1} X^{m_1} \xrightarrow{b^{k_1 \cdot N}} pY^{s_1+k_1 \cdot N} X^{m_1}$. Similarly $O_{m_2,s_2} \xrightarrow{b^{k_1 \cdot N}} qZ\beta_1^N \alpha_2$ must be simulated by $pY^{s_2} X^{m_2} \xrightarrow{b^{k_1 \cdot N}} pY^{s_2+k_1 \cdot N} X^{m_2}$. We also have that $pY^{s_1+k_1 \cdot N} X^{m_1} \approx qZ\beta_1^N \alpha_1$ and $pY^{s_2+k_1 \cdot N} X^{m_2} \approx qZ\beta_1^N \alpha_2$.

Consider the following transition sequence of $pY^{s_2+k_1 \cdot N} X^{m_2}$

$$pY^{s_2+k_1 \cdot N} X^{m_2} \xrightarrow{c} p_1 Y^{s_2+k_1 \cdot N} X^{m_2} \xrightarrow{\varepsilon} p_1 X^{m_2} \xrightarrow{a^{m_1+1}} p_1 X^{m_2-m_1-1}. \quad (6)$$

It can be matched by $qZ\beta_1^N \alpha_2 \xrightarrow{ca^{m_1+1}} P_1$ for some P_1 . Since $qZ\beta_1^N \alpha_2$ is real-time and $N \geq m_1 + 1$, the actions ca^{m_1+1} from $qZ\beta_1^N \alpha_2$ only depend on $qZ\beta_1^N$. Thus $qZ\beta_1^N \alpha_1 \xrightarrow{ca^{m_1+1}} Q$ for some Q . This contradicts with $qZ\beta_1^N \alpha_1 \approx pY^{s_1+k_1 \cdot N} X^{m_1}$, since after transition $pY^{s_1+k_1 \cdot N} X^{m_1} \xrightarrow{c} p_1 Y^{s_1+k_1 \cdot N} X^{m_1}$, the longest non- ε action sequence is a^{m_1} . ◀

5 Conclusion

In this paper, we show that the bisimilarity of PDA is ACKERMANN-hard. Combining with the result in [10], we can conclude that the bisimilarity of PDA is ACKERMANN-complete. The result can be generalized to normed PDA. Besides, we give another proof on that the so-called real-time pushdown processes are strictly weaker than first-order grammars. Our work answers the question proposed by Jančar and Schmitz in [10].

When the number of control states of PDA is fixed as $d \geq 4$, bisimilarity is \mathbf{F}_{d-1} -hard. An obvious open problem is to close the complexity gap with the \mathbf{F}_{d+4} upper bound in [10].

References

- 1 Jos CM Baeten, Jan A Bergstra, and Jan Willem Klop. Decidability of bisimulation equivalence for process generating context-free languages. *Journal of the ACM*, 40(3):653–682, 1993. doi:10.1145/174130.174141.
- 2 Michael Benedikt, Stefan Göller, Stefan Kiefer, and Andrzej S Murawski. Bisimilarity of pushdown automata is nonelementary. In *Proc. LICS '13*, pages 488–498. IEEE, 2013. doi:10.1109/LICS.2013.55.
- 3 Didier Caucal. Bisimulation of context-free grammars and of pushdown automata. *Modal Logic and Process Algebra*, 53:85–106, 1995.
- 4 Bruno Courcelle. Recursive applicative program schemes. In *Formal Models and Semantics*, pages 459–492. Elsevier, 1990. doi:10.1016/B978-0-444-88074-1.50014-7.
- 5 Seymour Ginsburg and Sheila Greibach. Deterministic context free languages. *Information and Control*, 9(6):620–648, 1966. doi:10.1016/S0019-9958(66)80019-0.

- 6 J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Publishing Company, 1979.
- 7 Petr Jančar. Decidability of DPDA Language Equivalence via First-Order Grammars. In *Proc. LICS '12*, pages 415–424. IEEE, 2012. doi:10.1109/LICS.2012.51.
- 8 Petr Jančar. Bisimulation Equivalence of First-Order Grammars. In *Proc. ICALP '14*, LNCS, pages 232–243. Springer, 2014. doi:10.1007/978-3-662-43951-7_20.
- 9 Petr Jančar. Equivalences of pushdown systems are hard. In *Proc. FoSSaCS '14*, pages 1–28. Springer, 2014. doi:10.1007/978-3-642-54830-7_1.
- 10 Petr Jančar and Sylvain Schmitz. Bisimulation equivalence of first-order grammars is ACKERMANN-complete. In *Proc. LICS '19*, pages 1–12. IEEE, 2019. doi:10.1109/LICS.2019.8785848.
- 11 Petr Jančar and Jivří Srba. Undecidability of bisimilarity by Defender’s forcing. *Journal of the ACM (JACM)*, 55(1):5, 2008. doi:10.1145/1326554.1326559.
- 12 Stefan Kiefer. BPA bisimilarity is EXPTIME-hard. *Information Processing Letters*, 113(4):101–106, 2013. doi:10.1016/j.ip1.2012.12.004.
- 13 Antonín Kučera and Petr Jančar. Equivalence-checking on infinite-state systems: Techniques and results. *Theory and Practice of Logic Programming*, 6(201), 2006. doi:10.1017/S1471068406002651.
- 14 Antonín Kučera and Richard Mayr. On the complexity of checking semantic equivalences between pushdown processes and finite-state processes. *Information and Computation*, 208(7):772–796, 2010. doi:10.1016/j.ic.2010.01.003.
- 15 Robin Milner. *Communication and concurrency*, volume 84. Prentice-Hall, Inc., 1989.
- 16 Michio Oyamaguchi. The equivalence problem for real-time DPDAs. *Journal of the ACM (JACM)*, 34(3):731–760, 1987. doi:10.1145/28869.28881.
- 17 V Yu Romanovskii. The equivalence problem for real-time deterministic pushdown automata. *Cybernetics*, 22(2):162–175, 1986. doi:10.1007/BF01074776.
- 18 Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Transactions on Computation Theory (TOCT)*, 8(1):3, 2016. doi:10.1145/2858784.
- 19 Sylvain Schmitz. *Algorithmic Complexity of Well-Quasi-Orders*. Habilitation thesis, École Normale Supérieure Paris-Saclay, 2017.
- 20 Sylvain Schmitz. The parametric complexity of lossy counter machines. In *Proc. ICALP '19*, volume 132, pages 129:1–129:15. LZI, 2019. doi:10.4230/LIPIcs.ICALP.2019.129.
- 21 Philippe Schnoebelen. Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets. In *Proc. MFCS '10*, pages 616–628. Springer, 2010. doi:10.1007/978-3-642-15155-2_54.
- 22 Gérard Sénizergues. Decidability of bisimulation equivalence for equational graphs of finite out-degree. In *Proc. FOCS '98*, pages 120–129. IEEE, 1998. doi:10.1109/SFCS.1998.743435.
- 23 Gérard Sénizergues. $L(A)=L(B)$? Decidability results from complete formal systems. *Theoretical Computer Science*, 251(1-2):1–166, 2001. doi:10.1016/S0304-3975(00)00285-1.
- 24 Gérard Sénizergues. The bisimulation problem for equational graphs of finite out-degree. *SIAM Journal on Computing*, 34(5):1025–1106, 2005. doi:10.1137/S0097539700377256.
- 25 Colin Stirling. Decidability of bisimulation equivalence for normed pushdown processes. *Theoretical Computer Science*, 195(2):113–131, 1998. doi:10.1016/S0304-3975(97)00216-8.
- 26 Colin Stirling. Decidability of bisimulation equivalence for pushdown processes. Technical report, University of Edinburgh, 2000.
- 27 Colin Stirling. Decidability of DPDA equivalence. *Theor. Comput. Sci.*, 255(1-2):1–31, 2001. doi:10.1016/S0304-3975(00)00389-3.
- 28 Colin Stirling. Deciding DPDA equivalence is primitive recursive. In *Proc. ICALP '02*, pages 821–832. Springer, 2002. doi:10.1007/3-540-45465-9_70.