Bisimulation Decomposition for PDA Decidability

Yuxi Fu[a,*], Qiang Yin[b]

[a]*BASICS, Shanghai Jiao Tong University, Shanghai, China*
[b]*BDBC, Beihang University, Beijing, China*

Abstract

Sénizergues proved that language equivalence is decidable for disjoint ϵ-deterministic PDA. Stirling showed that strong bisimilarity is decidable for PDA. On the negative side Srba pointed out that the weak bisimilarity is undecidable for normed PDA. Jančar and Srba demonstrated the undecidability of the weak bisimilarity for disjoint ϵ-pushing PDA and disjoint ϵ-popping PDA. In this paper it is proved that the branching bisimilarity of the normed ϵ-pushing PDA is decidable and the branching bisimilarity of the ϵ-pushing PDA is Σ^1_1-complete.

Keywords: PDA, branching bisimulation, decidability

1. Introduction

"Is it recursively unsolvable to determine if $L_1 = L_2$ for arbitrary deterministic languages L_1 and L_2"? The question was raised in Ginsburg and Greibach's 1966 paper [7] titled Deterministic Context Free Languages. The equality referred to in the quotation is the language equivalence between context free grammars. It is well known that the context free languages are precisely those accepted by pushdown automata (PDA) [10]. A PDA extends a finite state automaton with a memory stack. It accepts an input string whenever the memory stack is empty. The operational semantics of a PDA is defined by a finite set of rules of the following form

$$pX \xrightarrow{a} q\alpha \text{ or } pX \xrightarrow{\epsilon} q\alpha.$$

The transition rule $pX \xrightarrow{a} q\alpha$ reads "If the PDA is in state p with X being the top symbol of the stack, then it can accept an input letter a, pop off X, place the string α of stack symbols onto the top of the stack, and turn into state q". The rule $pX \xrightarrow{\epsilon} q\alpha$ describes a silent transition that has nothing to do with any input letter. It was proved early on that language equivalence between pushdown automata is undecidable [10]. A natural question asks what restrictions one may impose on the PDA's so that language equivalence becomes decidable. Ginsburg and Greibach studied deterministic context free languages. These are the languages accepted by deterministic pushdown automata (DPDA) [7].

A DPDA enjoys disjointness and determinism properties. These conditions are defined as follows:

Disjointness. For all state p and all stack symbol X, if pX can accept a letter then it cannot perform a silent transition, and conversely if pX can do a silent transition then it cannot accept any letter.

A-Determinism. If $pX \xrightarrow{a} q\alpha$ and $pX \xrightarrow{a} q'\alpha'$ then $q = q'$ and $\alpha = \alpha'$.

ϵ-Determinism. If $pX \xrightarrow{\epsilon} q\alpha$ and $pX \xrightarrow{\epsilon} q'\alpha'$ then $q = q'$ and $\alpha = \alpha'$.

These are strong constraints from an algorithmic point of view. It turns out however that the language problem is still difficult even for this simple class of PDA's. One indication of the difficulty of the problem is that there is no size

[*]Corresponding author
Email addresses: `fu-yx@cs.sjtu.edu.cn` (Yuxi Fu), `yinqiang@buaa.edu.cn` (Qiang Yin)

bound for equivalent DPDA configurations. It is easy to design a DPDA such that two configurations pY and pX^nY accept the same language for all n.

It was Sénizergues who proved after 30 years that the problem is decidable [23, 25]. His original proof is very long. Simplified proofs were soon discovered by Sénizergues [26] himself and by Stirlng [33]. After the positive answer of Sénizergues, one wonders if the strong constraints (disjointness+A-determinism+ϵ-determinism) can be relaxed. The first such relaxation was given by Sénizergues himself [24]. He showed that strong bisimilarity on the collapsed graphs of the disjoint ϵ-deterministic pushdown automata is also decidable. In the collapsed graphs all ϵ-transitions are absorbed. This result suggests that *A-non*determinism is harmless as far as decidability is concerned. The silent transitions considered in [24] are ϵ-popping. A silent transition $pX \xrightarrow{\epsilon} q\alpha$ is ϵ-popping if $\alpha = \epsilon$. In this paper we shall use a slightly more liberal definition of this terminology.

A PDA is *ϵ-popping* if $|\alpha| \leq 1$ whenever $pX \xrightarrow{\epsilon} q\alpha$.

A PDA is *ϵ-pushing* if $|\alpha| \geq 1$ whenever $pX \xrightarrow{\epsilon} q\alpha$.

A disjoint ϵ-deterministic PDA can be converted to an equivalent disjoint ϵ-popping PDA in the following manner. Without loss of generality we may assume that the disjoint ϵ-deterministic PDA does not admit any infinite sequence of silent transitions. Suppose $pX \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} q\alpha$ and $q\alpha$ cannot do any silent transition. If $\alpha = \epsilon$ then we can redefine the semantics of pX by $pX \xrightarrow{\epsilon} q\epsilon$; otherwise we can remove pX in favour of qZ with Z being the first symbol of α. So under the disjointness condition ϵ-popping condition is weaker than ϵ-determinism.

A paradigm shift from a language viewpoint to a process algebraic viewpoint helps see the issue in a more productive way. Groote and Hüttel [8, 12] pointed out that as far as BPA and BPP are concerned the bisimulation equivalence à la Milner [21] and Park [22] is more tractable than the language equivalence. The best way to understand Senizergues' result proved in [24] is to recast it in terms of bisimilarity. Disjointness and ϵ-determinism imply that all silent transitions preserve equivalence. It follows that the branching bisimilarity [34] of the disjoint ϵ-deterministic PDA's coincides with the strong bisimilarity on the collapsed graphs of these PDA's. So what Senizergues has proved in [24] is that the branching bisimilarity on the disjoint ϵ-deterministic PDA's is decidable.

The process algebraic approach allows one to use the apparatus from the process theory to study the equivalence checking problem for PDA. Stirling's proof of the decidability of the strong bisimilarity for normed PDA (nPDA) [29, 30] exploits the tableau method [13, 11]. Later he extended the tableau approach to the study of the unnormed PDA in an unpublished paper [32]. Stirling also provided a simplified account of Senizergues' proof [24] using the process method [33]. The proofs in [24, 33] are interesting in that they turn the language equivalence of disjoint ϵ-deterministic PDA to bisimilarity of correlated models. Another advantage of bisimulation equivalence is that it admits a nice game theoretical interpretation. This has been exploited in the proofs of negative results using the technique of Defender's Forcing [18]. Srba proved that weak bisimilarity on nPDA's is undecidable [27]. Jančar and Srba improved this result by showing that the weak bisimilarity on the disjoint nPDA's with only ϵ-popping transitions, respectively ϵ-pushing transitions, is already undecidable [18]. In fact they proved that the problems are Π^0_1-complete. Recently Yin, Fu, He, Huang and Tao have proved that the branching bisimilarity for all the models above either the normed BPA or the normed BPP in the hierarchy of process rewriting system [20] are undecidable [36]. This general result implies that the branching bisimilarity on nPDA is undecidable. Defender's Forcing can be used to study complexity bound. An example is Benedikt, Göller, Kiefer and Murawski's proof that the strong bisimilarity on PDA is non-elementary [2]. A summary of the (un)decidability results mentioned above is given in the following table, where $\sim$ stands for the strong bisimilarity, $\simeq$ the branching bisimilarity, and $\approx$ the weak bisimilarity.

	PDA	**nPDA**
$\sim$	Decidable [24] Non-Elementary [2]	Decidable [24] Non-Elementary [2]
$\simeq$	Undecidable [36]	Undecidable [36]
$\approx$	Σ^1_1-Complete [18] Undecidable [27]	Σ^1_1-Complete [18] Undecidable [27]

The decidability of the strong bisimilarity and the undecidability of the weak bisimilarity still leaves a number of questions unanswered. The prime motivation for this work is to establish a stronger result that would subsume both the decidability results in the language theme and the decidability results in the process algebraic line. This is desirable since these two classes of results are incompatible, neither implies the other. A conservative extension of the language equivalence for DPDA is not the strong bisimilarity because language equivalence ignores silent transitions. It is not the weak bisimilarity since the whole point of introducing the disjointness and ϵ-determinism conditions is to force all silent transitions to preserve equivalence. Our basic idea is to look at the decidability issue of the ϵ-pushing and ϵ-popping PDA's. Their decidability would subsume the two classes of incompatible decidability results.

The contributions of this paper are summarized as follows.

1. We prove that the branching bisimilarity on the normed ϵ-pushing PDA is decidable, and that the branching bisimilarity on the ϵ-pushing PDA is Σ^1_1-complete.
2. We propose a bisimulation decomposition approach, for decidability proof of bisimulation equivalence that involves silent transitions, that could have applications in other models of process rewriting system.

There is a wide range of applications of pushdown automata/systems in the study of programming language theory. Although the decidability result proved in this paper is for a class of PDA's whose internal transitions are constrained, it nonetheless points out a direction for system design. In practice the branching bisimilarity is more useful than the weak bisimilarity. So the decidability result is not that restrictive.

The rest of the paper is organised as follows. Section 2 fixes the syntax and the semantics of PDA. Section 3 reviews the basic properties of the branching bisimilarity. Section 4 introduces two classes of stack constants that will play an important role in describing the decidability algorithm. Section 5 discusses the finite branching property for the normed ϵ-pushing PDA. Section 6 defines bisimulation trees. Section 7 studies techniques to decompose bisimulation trees into finite trees. Section 8 explains how bisimulation trees can be recovered from finite trees, the main decidability result then follows. Section 9 proves the high undecidability of the general ϵ-pushing PDA. Section 10 concludes.

2. PDA

A *pushdown automaton* (PDA for short) $\mathcal{P} = (\mathcal{Q}, \mathcal{V}, \mathcal{L}, \mathcal{R})$ consists of

- a state set $\mathcal{Q} = \{p_1, \ldots, p_{\mathrm{q}}\}$ ranged over by o, p, q, r, s, t,
- a symbol set $\mathcal{V} = \{X_1, \ldots, X_{\mathrm{n}}\}$ ranged over by X, Y, Z,
- a letter set $\mathcal{L} = \{a_1, \ldots, a_{\mathrm{s}}\}$ ranged over by a, b, c, d, and
- a finite set $\mathcal{R}$ of transition rules.

If we think of a PDA as a process we may interpret a letter in $\mathcal{L}$ as an action label. The set $\mathcal{L}^*$ of words is ranged over by u, v, w. Following the process algebraic convention a silent action will be denoted by τ. The set $\mathcal{A} = \mathcal{L} \cup \{\tau\}$ of actions is ranged over by ℓ. The set $\mathcal{A}^*$ of action sequence is ranged over by ℓ^*. The set $\mathcal{V}^*$ of finite strings of symbols is ranged over by small Greek letters. The empty string is denoted by ϵ. We write $\alpha\delta$ for the concatenation of α and δ. Since concatenation is associative no parenthesis is necessary when we write $\alpha\delta\gamma$. The length of α is denoted by $|\alpha|$.

The syntax of a *PDA process* is $p\alpha$, where $p \in \mathcal{Q}$ is a state and $\alpha \in \mathcal{V}^*$ is called a *stack*. The size of $p\alpha$, denoted by $|p\alpha|$, is defined by $|\alpha|$. We shall write L, M, N, O, P, Q for PDA processes. If $P = p\alpha$ then $P\delta$ stands for the PDA process $p\alpha\delta$. If $\delta = \epsilon$, then $P\delta$ is nothing but P. The transition set $\mathcal{R}$ of a PDA contains rules of the form $pX \xrightarrow{\ell} q\alpha$. The semantics of the PDA processes is defined by the following rule.

$$\frac{pX \xrightarrow{\ell} q\alpha \in \mathcal{R}}{pX\delta \xrightarrow{\ell} q\alpha\delta} \tag{1}$$

We shall write $\xrightarrow{\ell^*}$ for $\xrightarrow{\ell_1} \ldots \xrightarrow{\ell_k}$ if $\ell^* = \ell_1 \ldots \ell_k$, $\Longrightarrow$ for the reflexive and transitive closure of $\xrightarrow{\tau}$, and $\stackrel{\tau}{\Longrightarrow}$ for the transitive closure of $\xrightarrow{\tau}$. We say that P' is a descendant of P if $P \xrightarrow{\ell^*} P'$ for some ℓ^*. A process P is *normed*, or P is an nPDA process, if $P \xrightarrow{\ell^*} p\epsilon$ for some ℓ^*, p. It is *unnormed* otherwise. A PDA $\mathcal{P} = (\mathcal{Q}, \mathcal{V}, \mathcal{L}, \mathcal{R})$ is *normed*, or $\mathcal{P}$ is an nPDA, if pX is normed for all $p \in \mathcal{Q}$ and all $X \in \mathcal{V}$. The notation (n)PDA$^{\epsilon+}$ will refer to the variant of (n)PDA with ϵ-pushing transitions. Unless explicitly specified we shall focus exclusively on nPDA$^{\epsilon+}$.

3. Branching Bisimilarity

The definition of branching bisimilarity is due to van Glabbeek and Weijland [35]. For PDA's care should be given to processes of the form $p\epsilon$ to guarantee that the bisimilarity is a congruence relation [29].

Definition 1. *A binary relation $\mathcal{R}$ on* nPDA$^{\epsilon+}$ *processes is a branching simulation if the following statements are valid whenever $P\mathcal{R}Q$:*

1. *If $P \xrightarrow{a} P'$ then there are some Q', Q'' such that $Q \Longrightarrow Q'' \xrightarrow{a} Q'$ and $P\mathcal{R}Q''$ and $P'\mathcal{R}Q'$.*
2. *If $P \xrightarrow{\tau} P'$ then either $Q \Longrightarrow Q'$ and $P\mathcal{R}Q'$ and $P'\mathcal{R}Q'$ for some Q' or $Q \Longrightarrow Q'' \xrightarrow{\tau} Q'$ and $P\mathcal{R}Q''$ and $P'\mathcal{R}Q'$ for some Q', Q''.*
3. *If $P = p\epsilon$ then $Q \Longrightarrow p\epsilon$.*

The relation $\mathcal{R}$ is a branching bisimulation *if both $\mathcal{R}$ and $\mathcal{R}^{-1} = \{(y, x) \mid (x, y) \in \mathcal{R}\}$ are branching simulations. The* branching bisimilarity $\simeq$ *is the largest branching bisimulation.*

The branching bisimulations are closed under set theoretical union. Let $\mathcal{R}_1, \mathcal{R}_2$ be branching bisimulations. It is proved in [1] that the composition $\mathcal{R}_1;\mathcal{R}_2$, defined by $\{(O, Q) \mid \exists P.(O, P) \in \mathcal{R}_1 \wedge (P, Q) \in \mathcal{R}_2\}$, is a branching bisimulation. Consequently $\simeq$ is an equivalence. Moreover $\simeq$ is also a congruence.

A technical lemma that plays an important role in the study of branching bisimilarity is the following Computation Lemma [35, 5].

Lemma 2. *If $P_0 \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_k \simeq P_0$, then $P_0 \simeq \ldots \simeq P_k$.*

A silent transition $P \xrightarrow{\tau} P'$ is *state-preserving*, denoted by $P \xrightarrow{\epsilon} P'$ or $P \to P'$, if $P \simeq P'$. It is a *change-of-state*, notation $P \xrightarrow{\iota} P'$, if $P \not\simeq P'$. This use of the notation ϵ is consistent with the fact that in DPDA all ϵ-transitions are state-preserving. We write $(\to^*)$ $\to^+$ for the (reflexive and) transitive closure of $\xrightarrow{\epsilon}$. The notation $P \not\twoheadrightarrow$ stands for the fact that $P \not\simeq P'$ for all P' such that $P \xrightarrow{\tau} P'$. Let $\jmath$ range over $\mathcal{L} \cup \{\iota\}$. We will find it necessary to use the notation $\xrightarrow{\jmath}$. The transition $P \xrightarrow{\jmath} P'$ refers to either $P \xrightarrow{a} P'$ for some $a \in \mathcal{L}$ or $P \xrightarrow{\iota} P'$. Lemma 2 implies that if $P_0 \xrightarrow{\jmath} P_1$ is bisimulated by $Q_0 \xrightarrow{\tau} Q_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} Q_k \xrightarrow{\jmath} Q_{k+1}$, then $Q_0 \xrightarrow{\epsilon} Q_1 \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} Q_k$. This is a very useful property. Another useful property of congruence is that $P\alpha \xrightarrow{\epsilon} P'\alpha$ whenever $P \xrightarrow{\epsilon} P'$.

Given a PDA process P, the *norm* of P *over* σ, denoted by $||P||_\sigma$, is a function from $[\mathfrak{q}] = \{1, \ldots, \mathfrak{q}\}$ to $\mathbb{N} \cup \{\perp\}$, where $\mathbb{N}$ is the set of natural numbers and $\perp$ stands for undefinedness, such that the following hold for every $h \in [\mathfrak{q}]$:

- $||P||_\sigma(h) = \perp$ if there is no ℓ^* such that $P\sigma \xrightarrow{\ell^*} p_h\sigma$.
- $||P||_\sigma(h)$ is the least number i such that $P\sigma \to^* \xrightarrow{\jmath_1} \ldots \to^* \xrightarrow{\jmath_i} \to^* p_h\sigma$ for some $\jmath_1 \ldots \jmath_i$.

It follows from the congruence property that $||P||_\sigma(h) = ||Q||_\sigma(h)$ whenever $P \simeq Q$. Let $\ker ||P||$ be the finite set $\{h \mid ||P||(h) \neq \perp\}$. For nPDA$^{\epsilon+}$ process P we introduce the following notations.

$$\begin{aligned}\min ||P||_\sigma &= \min\{||P||_\sigma(h) \mid h \in \ker ||P||\},\\ \max ||P||_\sigma &= \max\{||P||_\sigma(h) \mid h \in \ker ||P||\}.\end{aligned}$$

We will omit the subscript σ if $\sigma = \epsilon$, and we call $||P||$ the *norm* of P. The *strong norm* $||P||_s$ of P is defined as follows: For each $h \in \ker ||P||$ the value $||P||_s(\mathrm{h})$ is the least k such that $P \xrightarrow{\ell_1} \ldots \xrightarrow{\ell_k} p_h\epsilon$ for some $\ell_1, \ldots, \ell_k$. We shall use the following convention in the rest of the paper.

$$\begin{aligned}\mathfrak{r} &= \max\left\{|\eta| \;\middle|\; pX \xrightarrow{\ell} q\eta \in \mathcal{R} \text{ for some } p, q \in \mathcal{Q}, X \in \mathcal{V}\right\},\\ \mathfrak{m} &= \max\{\max\{||pX||_s(h) \mid h \in \ker ||P||\} \mid p \in \mathcal{Q}, X \in \mathcal{V}\} + 1.\end{aligned}$$

It follows from definition that $||pX||(i) < \mathfrak{m}$ for all p, X, all $i \in \ker ||pX||$. It is obvious how to compute $\mathfrak{r}$. The value $||pX||_s(h)$ for $h \in [\mathfrak{q}]$, and the value $\mathfrak{m}$ as well, can be effectively calculated using a dynamic programming algorithm.

4. Constant

Following Stirling [30] we introduce two types of equationally defined stack symbols for a PDA. A *simple constant* U is defined by a family of process equalities

$$p_1U = o_1\eta_1,\ p_2U = o_2\eta_2,\ \ldots,\ p_{\mathfrak{q}}U = o_{\mathfrak{q}}\eta_{\mathfrak{q}}. \tag{2}$$

We think of the equalities in (2) as grammar equalities. In other words we regard p_iU and $o_i\eta_i$ as the same syntactical object for every $i \in [\mathfrak{q}]$. We write $U(i)$ for $o_i\eta_i$, where $i \in [\mathfrak{q}]$. We shall write U and its decorated versions for simple constants. In the rest of the paper we take simple constants as first class citizens. Accordingly a (simple) stack should be understood as a finite string composed of stack symbols and/or simple constants. For example $XUYU'ZV$ is a stack. Simple constants should be defined in a hierarchical way. They should not be mutually dependent. With this extension we should understand that in (2) the stacks η_1 through $\eta_{\mathfrak{q}}$ may contain simple constants.

We will insist throughout the paper that in a process of the form $p\alpha U\beta$ the size of $U(i)$ is greater than 0 for every $i \in \ker(p\alpha)$. In any case this can be checked algorithmically. From now on we will use the small Greek letters for (simple) stacks. When calculating the length of a stack or the size of a process a simple constant counts as one symbol.

A *recursive constant* V is defined by a family of process equalities

$$p_1V = L_1V,\ p_2V = L_2V,\ \ldots,\ p_{\mathfrak{q}}V = L_{\mathfrak{q}}V. \tag{3}$$

Again we see the equalities in (3) as grammar equalities. We write $V(i)$ for L_i, where $i \in [\mathfrak{q}]$. We say that V is undefined at $i \in [\mathfrak{q}]$, notation $V(i)\uparrow$, if $V(i) = p_i\epsilon$. We will identify a stack say $\alpha V\beta$ with αV. In other words we ignore all symbols after a recursive constant since operationally they are irrelevant. We shall write V and its decorated versions for recursive constants. In (3) the stacks in L_1 through $L_{\mathfrak{q}}$ may contain simple constants but not recursive constants. Unlike the simple constants, the recursive constants are auxiliary. Their sole role is to help define characteristic trees in Section 7.1, Section 7.2 and Section **??**. *Recursive constants will not appear in any goals in the decidability algorithm.*

In the presence of the grammar equalities defined in (2) and (3) the operational semantics of pU and pV is well defined by the rules given in (1). The bisimulation semantics need be enriched. In Definition 1 the following clause must be incorporated.

4) If $P = p_iV$ and $V(i)\uparrow$ then $Q = p_iV$.

The equivalence and congruence properties are unaffected.

Suppose $pX\alpha\sigma \simeq M\delta\sigma$ such that $|M| = \mathfrak{m}$ and $|\delta| > 0$ and that $pX\alpha\sigma \rightarrow^{*}\xrightarrow{J_1} \ldots \rightarrow^{*}\xrightarrow{J_{\|pX\|_{\alpha\sigma}(i)}} p_i\alpha\sigma$. Suppose further that M does not contain any simple constants. By Lemma 6 to be established in next section, the length of a simulating sequence $M\delta\sigma \rightarrow^{*}\xrightarrow{J_1} \ldots \rightarrow^{*}\xrightarrow{J_{\|pX\|_{\alpha\sigma}(i)}} o_i\eta_i\delta\sigma$ is bounded by $\mathfrak{q}\mathfrak{n}\mathfrak{r}(\mathfrak{m}+1)^{\mathfrak{q}}\mathfrak{m} < \mathfrak{q}\mathfrak{n}\mathfrak{r}(\mathfrak{m}+1)^{(\mathfrak{q}+1)}$ and the suffix $\delta\sigma$ is kept intact throughout the simulation. It follows easily that $|\eta_i| < \mathfrak{q}\mathfrak{n}\mathfrak{r}^2(\mathfrak{m}+1)^{(\mathfrak{q}+1)}$. Let the simple constant U be defined by

$$p_iU = \begin{cases} o_i\eta_i, & \text{if } i \in \ker\|pX\|, \\ p_i\epsilon, & \text{if } i \notin \ker\|pX\|. \end{cases} \tag{4}$$

It should be clear that $|o_i\eta_i| > 0$ if $i \in \ker\|pX\|$ and that

$$pXU\delta\sigma \simeq pX\alpha\sigma \simeq M\delta\sigma.$$

The use of the simple constant allows us to extend the common suffix σ of the bisimilar pair $(pX\alpha\sigma, M\delta\sigma)$ to the common suffix $\delta\sigma$ of the bisimilar pair $(pXU\delta\sigma, M\delta\sigma)$. This is essentially the only use of simple constants in this paper. We therefore impose the following constraint on all simple constants: In (2) the inequality

$$|\eta_i| < \mathfrak{q}\mathfrak{n}\mathfrak{r}^2(\mathfrak{m}+1)^{(\mathfrak{q}+1)}$$

holds for every $i \in [\mathfrak{q}]$. The following lemma then follows.

Lemma 3. *The number of simple constants admitted in a PDA is finite.*

5. Finite Branching Property

Generally bisimilarity is undecidable for models with infinite branching transitions. For $\simeq_{\text{nPDA}^{\epsilon+}}$ the finite branching property boils down to the following statement.

> For each P there is a finite set of processes $\{P_i\}_{i\in I}$ such that $P' \simeq P_i$ for some $i \in I$ whenever $P \to^* P'$.

We prove in this section that nPDA$^{\epsilon+}$ enjoys the finite branching property. Before doing that we need be assured that silent transition cycles of nPDA$^{\epsilon+}$ processes do not render a problem. There is in fact an effective procedure to remove such a silent transition cycle. A clique $\mathcal{S}$ is a subset of $\{pX \mid p \in \mathcal{Q},\text{ and } X \in \mathcal{V}\}$ such that for every two distinct members pX, qY of $\mathcal{S}$ there is a silent transition sequence from pX to qY. It follows from Lemma 2 that the members of a clique are branching bisimilar. We can remove a maximal clique $\mathcal{S}$ in two steps.

1. Remove all rules of the form $pX \xrightarrow{\tau} qY$ such that $pX, qY \in \mathcal{S}$.
2. For each $pX \in \mathcal{S}$ introduce the rule $pX \xrightarrow{\lambda} P$ whenever there is some $qY \in \mathcal{S}$ that is distinct from pX and the rule $qY \xrightarrow{\lambda} P$ has not been removed in the first step.

In the new nPDA$^{\epsilon+}$ there is no circular silent transition sequence involving any member of $\mathcal{S}$ due to the maximality of $\mathcal{S}$. The legitimacy of transformation is guaranteed by Lemma 2. In this way we can remove all cycles by repetition. From now on we assume that such circularity does not occur in our nPDA$^{\epsilon+}$. Consequently for an nPDA$^{\epsilon+}$ with $\mathfrak{n}$ variables and $\mathfrak{q}$ states the length of a silent transition sequence of the form $qX \xrightarrow{\tau} q_1X_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} q_kX_k$ is upper bounded by $\mathfrak{n}\mathfrak{q}$.

Lemma 4. *$|P| \le \min\|P\|$ holds for all P.*

Proof. In nPDA$^{\epsilon+}$ only external actions remove symbols from a stack. Silent actions never does that. □

Corollary 5. *If P contains no simple constants and $Q \simeq P$, then $|Q| \le \mathfrak{m}|P|$.*

Proof. By Lemma 4 the size of Q is bounded by $\min\|Q\| = \min\|P\| \le \mathfrak{m}|P|$. □

Using the above corollary it is easy to establish the finite branching property for nPDA$^{\epsilon+}$. There is however a stronger result stating that a *constant* bound exists for the length of the state-preserving transitions in an nPDA$^{\epsilon+}$.

Lemma 6. *Suppose $qX\sigma \xrightarrow{\epsilon} q_1\gamma_1\sigma \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} q_k\gamma_k\sigma$ for an* nPDA$^{\epsilon+}$ *process $qX\sigma$. Then $k < \mathfrak{q}\mathfrak{n}\mathfrak{r}(\mathfrak{m}+1)^{\mathfrak{q}}$.*

Proof. Suppose $qX\sigma \xrightarrow{\epsilon} q_1Z_1\delta_1\sigma$. Let

$$q_1Z_1\delta_1\sigma \to^* \xrightarrow{J^1_1} \ldots \to^* \xrightarrow{J^1_{j_1}} \to^* r_1\epsilon\sigma \to^* \xrightarrow{J^1_{j_1+1}} \ldots \to^* \xrightarrow{J^1_{j_{k_1}}} \to^* p_{h_1}\epsilon$$

be a transition sequence of minimal length that empties the stack, where $k_1 = \min\|q_1Z_1\delta_1\sigma\|$. Clearly $j_1 \le \mathfrak{r}\mathfrak{m}$. Suppose $q_1Z_1\delta_1\sigma \to^* q_2Z_2\delta_2\delta_1\sigma$ such that $\mathfrak{r}\mathfrak{m} < |Z_2\delta_2\delta_1| \le \mathfrak{r}(\mathfrak{m}+1)$. Let $Q_2 = q_2Z_2\delta_2\delta_1\sigma$ and $k_2 = \min\|Q_2\|$, and let

$$Q_2 \to^* \xrightarrow{J^2_1} \ldots \to^* \xrightarrow{J^2_{j_2}} \to^* r_2\epsilon\sigma \to^* \xrightarrow{J^2_{j_2+1}} \ldots \to^* \xrightarrow{J^2_{j_{k_2}}} \to^* p_{h_2}\epsilon$$

be a transition sequence of minimal length that empties the stack. One must have $j_2 > j_1$ according to the size bound on $Z_2\delta_2\delta_1$. By iterating the above argument one gets from

$$\begin{array}{rcl} q_1Z_1\delta_1\sigma & \to^* & q_2Z_2\delta_2\delta_1\sigma \\ & \to^* & \ldots \\ & \to^* & q_{i+1}Z_{i+1}\delta_{i+1}\delta_i\ldots\delta_1\sigma \\ & \to^* & \ldots \\ & \to^* & q_{\mathfrak{q}+1}Z_{\mathfrak{q}+1}\delta_{\mathfrak{q}+1}\delta_{\mathfrak{q}}\ldots\delta_1\sigma \end{array}$$

with $\mathfrak{rm}(\mathfrak{m}+1)^{i-1} < |Z_{i+1}\delta_{i+1}\delta_i \ldots \delta_1| \le \mathfrak{r}(\mathfrak{m}+1)^i$ for all $i \in [\mathfrak{q}]$, some states $r_1, \ldots, r_{\mathfrak{q}+1}$, some numbers $k_1 < \ldots < k_{\mathfrak{q}+1}$ and $h_1, \ldots, h_{\mathfrak{q}+1}$. For each $i \in [\mathfrak{q}+1]$ there is some transition sequence

$$Q_i \to^* \xrightarrow{j^i_1} \ldots \to^* \xrightarrow{j^i_{j_i}} \to^* r_i\epsilon\sigma \to^* \xrightarrow{j^i_{j_i+1}} \ldots \to^* \xrightarrow{j^i_{jk_i}} \to^* p_{h_i}\epsilon$$

where $Q_i = q_i Z_i \delta_i \ldots \delta_1 \sigma$ and $k_i = \min \|Q_i\|$. Since there are only $\mathfrak{q}$ states, there must be some t_1, t_2 such that $0 < t_1 < t_2 \le \mathfrak{q}+1$ and $r_{t_1} = r_{t_2}$. It follows from the minimality that $j_{k_{t_1}} - j_{t_1} = j_{k_{t_2}} - j_{t_2}$. But $j_{t_2} > j_{t_1}$. Consequently $j_{k_{t_1}} < j_{k_{t_2}}$. This inequality contradicts to the fact that $q_{t_1} Z_{t_1} \delta_{t_1} \ldots \delta_1 \sigma \simeq q_{t_2} Z_{t_2} \delta_{t_2} \ldots \delta_1 \sigma$. We conclude that if $qX\sigma \to^* q'\gamma\sigma$ then $|\gamma| \le \mathfrak{r}(\mathfrak{m}+1)^{\mathfrak{q}}$. Since there is no ϵ-loop the bound $k < \mathfrak{qmr}(\mathfrak{m}+1)^{\mathfrak{q}}$ follows. □

Using the finite branching property guaranteed by Lemma 6 it is standard to prove the following.

Proposition 7. *The relation $\not\simeq$ on* nPDA$^{\epsilon+}$ *processes is semidecidable.*

Proof. Let $\simeq_0$ be the total relation. The symmetric relation $\simeq_{k+1}$ is defined as follows: $P \simeq_{k+1} Q$ if the following statements are valid:

1. If $Q \xrightarrow{a} Q'$ then $P \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_j \xrightarrow{a} P' \simeq_k Q'$ for some $P_1, \ldots, P_j, P'$ such that $P_i \simeq_k Q$ for all $i \in [j]$.
2. If $Q \xrightarrow{\tau} Q'$ then either $P \simeq_k Q'$ or some $P_1, \ldots, P_j, P'$ exist such that $P \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_j \xrightarrow{\tau} P' \simeq_k Q'$ and $P_i \simeq_k Q$ for all $i \in [j]$.
3. If $P = p\epsilon$ then $Q = p\epsilon$.

The approximation $\simeq_0 \supseteq \simeq_1 \supseteq \simeq_2 \supseteq \ \ldots$ approaches to $\simeq$. By standard argument using Lemma 6 one shows that $\bigcap_{i \ge 0} \simeq_i$ is $\simeq$. So $P \not\simeq Q$ can be checked by checking $P \not\simeq_i Q$ for $i > 0$. □

6. Bisimulation Tree

Intuitively a bisimulation tree for (P, Q) is a stratified presentation of a branching bisimulation for (P, Q) in which one ignores all intermediate state-preserving silent transitions. This reminds one of the collapsed graphs due to Sénizergues. To see how the collapse is done semantically, let us define the *ϵ-tree* of a process P, denoted by $T_\epsilon(P)$, to be the tree consisting of all state-preserving transition sequences starting from P. Recall that our nPDA$^{\epsilon+}$ does not admit silent loop action sequence and according to Lemma 6 there is a bound, computable from the definition of nPDA$^{\epsilon+}$, on the length of state-preserving silent transition sequences, hence the finiteness of the ϵ-trees. We say that the τ-tree $T_\epsilon(P)$ is *trivial* if it contains only one node; otherwise it is *nontrivial*. In a collapsed presentation of a bisimulation for (P, Q) the root P is related to the root Q, and every leaf of $T_\epsilon(P)$ is related to every leaf of $T_\epsilon(Q)$. The internal nodes of the two trees are not explicitly included in the collapsed presentation. These nodes and the state-preserving silent transitions related to them are implicit. However in order to recover a branching bisimulation from the collapsed presentation, every external action or change-of-state silent action of any internal node, say P', of $T_\epsilon(P)$ must be matched by every leaf of $T_\epsilon(Q)$. We remark that since a leaf of $T_\epsilon(Q)$ cannot perform any state-preserving silent transition the action of P' must be bisimulated by a single step action of the leaf.

We need to turn the above semantic intuition into a definition in terms of the operational semantics. We shall not introduce a formal treatment of rooted trees. For proof of decidability our level of formality should be sufficient. Formally a *bisimulation tree $\mathfrak{B}$ for (P, Q)* is a finite branching rooted tree such that each of its nodes is labeled by a pair (M, N) of nPDA$^{\epsilon+}$ processes and a directed edge is labeled by a member of $\mathcal{L} \cup \{\tau\} \cup \{\epsilon\}$. The root is labeled by (P, Q), and the following properties hold for every node labeled by say (M, N).

1. M is descendant of P and N is a descendant of Q.
2. If there is an edge labeled ϵ from the node labeled (M, N) to a node labeled (M', N') then either $M \overset{\tau}{\Longrightarrow} M'$ and $N \Longrightarrow N'$ or $M \Longrightarrow M'$ and $N \overset{\tau}{\Longrightarrow} N'$, and the following are valid.
   (a) There is no edge labeled ϵ from the node labeled (M, N) to a node labeled by (M'', N'') such that either $(M \Longrightarrow M'' \overset{\tau}{\Longrightarrow} M') \vee (M' \overset{\tau}{\Longrightarrow} M'')$ or $(N \Longrightarrow N'' \overset{\tau}{\Longrightarrow} N') \vee (N' \overset{\tau}{\Longrightarrow} N'')$.

(b) If there are ϵ labeled edges from the node labeled (M, N) to nodes labeled by (M', N') and (M'', N'') respectively, then there are ϵ labeled edges from the node labeled (M, N) to nodes labeled by (M', N'') and (M'', N') respectively.
(c) Every edge from the node labeled (M, N) is labeled by ϵ.
(d) There is no ϵ labeled edge from the node labeled (M', N').

3. Suppose there is an ϵ labeled edge from the node labeled (L, O) to a node labeled (M, N). A silent transition $P_0 \xrightarrow{\tau} P_1$, respectively $Q_0 \xrightarrow{\tau} Q_1$, is *implicit* with regards to (L, O) if there is an ϵ labeled edge from the node labeled (L, O) to a node labeled by some (M', N') such that $P_0 \xrightarrow{\tau} P_1$, respectively $Q_0 \xrightarrow{\tau} Q_1$, appears in a silent transition sequence from L to M', respectively from O to N'. The third requirement states that for the ϵ labeled edge from the node labeled (L, O) to the node labeled (M, N) the following are valid.
(a) If $L \Longrightarrow L' \Longrightarrow M$ and $L' \xrightarrow{\lambda} L''$ is not implicit with regards to (L, O), then some N' exists such that $N \xrightarrow{\lambda} N'$ and there is a λ labeled edge from the node labeled (M, N) to a node labeled (L'', N').
(b) If $O \Longrightarrow O' \Longrightarrow N$ and $O' \xrightarrow{\lambda} O''$ is not implicit with regards to (L, O), then some M' exists such that $M \xrightarrow{\lambda} M'$ and there is a λ labeled edge from the node labeled (M, N) to a node labeled (M', O'').

We write $\mathfrak{B}(P, Q)$ for a bisimulation tree for (P, Q).

Condition (2c) and condition (2d) of the above definition reminds one of the disjointness condition. It is easy to see that if one reverses the order of the labels of a bisimulation tree $\mathfrak{B}(P, Q)$ for (P, Q) one obtains a bisimulation tree for (Q, P), denoted by $\mathfrak{B}^{-1}(P, Q)$. In what follows we often refer to a node by its label. Accordingly we write $(M, N) \xrightarrow{\tau} (M', N')$ for example for a τ-edge from a node labeled (M, N) to a node labeled (M', N').

Let's see some bisimulation trees. Suppose an nPDA$^{\epsilon+}$ has the following semantic rules.

$$pX \xrightarrow{\tau} p'X,\ p'X \xrightarrow{a} r\epsilon,\ pX \xrightarrow{a} r\epsilon;$$

$$qX \xrightarrow{\tau} q'X,\ q'X \xrightarrow{a} r\epsilon,\ qX \xrightarrow{a} r\epsilon.$$

The following are two distinct bisimulation trees, T_1 and T_2, for the process pair (pX, qX).

$$\begin{tikzcd}
(T_1). & (pX,qX) \arrow[dl, "a"'] \arrow[dr, "\tau"] & & (T_2). & (pX,qX) \arrow[d, "\epsilon"'] \\
(r\epsilon,r\epsilon) & & (p'X,q'X) \arrow[d, "a"'] & & (p'X,q'X) \arrow[d, "a"'] \\
& & (r\epsilon,r\epsilon) & & (r\epsilon,r\epsilon)
\end{tikzcd}$$

In this example $pX \simeq p'X$ and $qX \simeq q'X$. So the ϵ edge in T_2 is justified. We can construct a different bisimulation tree T_1 without making use of the fact $pX \to p'X$ and $qX \to q'X$.

Given a bisimulation tree $\mathfrak{B}$ for (P, Q). Define $\overline{\mathfrak{B}}$ inductively as follows.

1. If (M, N) is a label in $\mathfrak{B}$ then $(M, N) \in \overline{\mathfrak{B}}$.
2. If $(M, N) \xrightarrow{\epsilon} (M', N')$, then $(M'', N'') \in \overline{\mathfrak{B}}$ for each pair (M'', N'') such that $M \Longrightarrow M'' \Longrightarrow M'$ and $N \Longrightarrow N'' \Longrightarrow N'$.

In the above example, $\overline{T_1} = \{(pX, qX), (p'X, q'X), (r\epsilon, r\epsilon)\}$ and $\overline{T_2} = \{(pX, qX), (p'X, q'X), (p'X, qX), (pX, q'X), (r\epsilon, r\epsilon)\}$ are two bisimulations.

A branching bisimulation *rooted at* (M, N) is a branching bisimulation $\mathcal{R}$ such that $(M, N) \in \mathcal{R}$ and if $(M', N') \in \mathcal{R}$ then M' is a descendant of M and N' is a descendant of N. The trivial proof of the property stated next justifies the slightly complicated definition of bisimulation tree.

Lemma 8. *$\overline{\mathfrak{B}}$ is a branching bisimulation rooted at (P, Q).*

It follows immediately from Lemma 8 that $(M, N) \xrightarrow{\epsilon} (M', N')$ implies either $M \to^{+} M'$ and $N \to^{*} N'$ or $M \to^{*} M'$ and $N \to^{+} N'$.

We assign a level number to every node of a bisimulation tree. The level number of the root is 0. Suppose the level number of (M, N) is k. Then the level number of (M', N') is k if $(M, N) \xrightarrow{\epsilon} (M', N')$, it is $k+1$ if $(M, N) \xrightarrow{\tau} (M', N')$ or $(M, N) \xrightarrow{a} (M', N')$. We say that a bisimulation tree is generated or *grown* level by level if for each $k \geq 0$ none of its nodes at the $(k+1)$-th level is generated before all nodes at the k-th level have been generated. The *k-subtree* of a tree consists of all the nodes of the latter whose level number is no more than k.

From a proof perspective we think of a pair (P, Q) as a *goal*. The bisimulation tree for (P, Q) is a proof that the goal can be established in the sense that $P \simeq Q$. If $P \simeq Q$ there is a canonical bisimulation tree for (P, Q) in which every τ-edge represents a change-of-state silent transition. However there could be bisimulation trees for (P, Q) in which a τ-edge is actually a state-preserving silent transition; see the T_1 in the above example. We will refer to a bisimulation for a pair (P, Q) satisfying $P \simeq Q$ a *bisimulation tree of* $P \simeq Q$. In this case we also say that $P \simeq Q$ is a goal.

In the rest of the paper we assume that *simple constants will not appear in the right process of any goal.*

7. Decomposing Bisimulation Trees to Finite Trees

Bisimulation trees of bisimilar nPDA$^{\epsilon+}$ processes are in general infinite. The proof of decidability looks for periodic properties among the infinite trees. In this section we look for bisimulation trees of bisimilar nPDA$^{\epsilon+}$ processes that share common suffix. The idea is that every goal can be turned into such a goal together with a set of reduced subgoals, where by a reduced subgoal we mean that either the size of the subgoal has been strictly decreased or the size of the prefixes is computationally bounded. The general proof framework of this section follows that of Stirling [31]. Our presentation is in terms of trees for branching bisimulation rather than tableaux systems for strong bisimulation.

7.1. Recursive Subgoal

Suppose $P\sigma \simeq Q\sigma$ with $|P| > 0$ and $|Q| > 0$. A *bisimulation tree of* $P\sigma \simeq Q\sigma$ *over* σ is grown like a bisimulation tree of $P\sigma \simeq Q\sigma$. The major difference is that the suffix σ should remain intact throughout the construction of the tree. A transition $O\sigma \xrightarrow{a} O''$ for example is admitted in the buildup of the bisimulation tree only if $|O| > 0$, in which case O'' must be of the form $O'\sigma$. In the following construction we maintain three parameters modified dynamically.

- The first is a set $\mathcal{G} = \{G_i\}_{i \in [\mathrm{q}]}$ of processes such that

$$p_1\sigma \simeq G_1\sigma,\ p_2\sigma \simeq G_2\sigma,\ \ldots,\ p_{\mathrm{q}}\sigma \simeq G_{\mathrm{q}}\sigma \tag{5}$$

  Initially $G_i = p_i\epsilon$ for all $i \in [\mathrm{q}]$.

- The second is an equivalence relation $\mathcal{E}$ on $[\mathrm{q}]$. We write $i \in \mathcal{E}$ if $\langle i, i \rangle \in \mathcal{E}$. Initially $\mathcal{E} = \{\langle i, i \rangle \mid i \in [\mathrm{q}]\}$.

- The third is a recursive constant V defined by the grammar equalities

$$p_1V = L_1V,\ p_2V = L_2V,\ \ldots,\ p_{\mathrm{q}}V = L_{\mathrm{q}}V, \tag{6}$$

  where $L_1, \ldots, L_{\mathrm{q}}$ are processes defined by

$$L_i = \begin{cases} G_i, & \text{if } i \notin \mathcal{E}, \\ p_j\epsilon, & \text{if } i \in \mathcal{E} \text{ and } j = \min\{j \mid (i, j) \in \mathcal{E}\}. \end{cases}$$

  Initially $L_i = p_i\epsilon$ for all $i \in [\mathrm{q}]$.

We shall update $\mathcal{G}$ and $\mathcal{E}$ dynamically while we build up the tree level by level. The definition of the recursive constant V is updated accordingly. The correlation between (5) and (6) has important consequence.

Lemma 9. *Suppose V is defined by $\{p_iV = L_iV\}_{i \in [\mathrm{q}]}$ and $p_i\sigma \simeq L_i\sigma$ for all $i \in [\mathrm{q}]$. If $|P|, |Q| > 0$ then $PV \simeq QV$ implies $P\sigma \simeq Q\sigma$.*

Proof. Let $\mathcal{R}$ be the relation $\{(P\sigma, Q\sigma) \mid PV \simeq QV \wedge |P| > 0 \wedge |Q| > 0\} \cup \simeq$. We prove that $(\simeq;\mathcal{R};\simeq) \cup \simeq$ is a branching bisimulation. Suppose $M \simeq P\sigma\mathcal{R}Q\sigma \simeq N$ and $M \xrightarrow{a} M'$. Then $P\sigma \xrightarrow{\epsilon} P_1\sigma \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} P_i\sigma \xrightarrow{a} P'\sigma$ bisimulates $M \xrightarrow{a} M'$ for some $P_1, \ldots, P_i, P'$. By the ϵ-pushing property $|P_1| > 0, \ldots, |P_i| > 0$. Thus $PV \xrightarrow{\tau} P_1V \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_iV \xrightarrow{a} P'V$. Since $PV \simeq QV$ this action sequence must be bisimulated by some $QV \xrightarrow{\tau} Q_1V \xrightarrow{\tau} \ldots \xrightarrow{\tau} Q_jV \xrightarrow{a} Q'V$ for some $Q_1, \ldots, Q_j, Q'$. Also $Q\sigma \xrightarrow{\tau} Q_1\sigma \xrightarrow{\tau} \ldots \xrightarrow{\tau} Q_j\sigma \xrightarrow{a} Q'\sigma$ must be bisimulated by $N \xrightarrow{\tau} N_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} N_k \xrightarrow{a} N'$ for some $N_1, \ldots, N_k, N'$. It is easy to see that $(M, N_g) \in \simeq;\mathcal{R};\simeq$ for all $g \in [k]$. To finish the proof we carry out a case analysis.

- $|P'| > 0$ and $|Q'| > 0$. Clearly $(M', N') \in \simeq;\mathcal{R};\simeq$
- $P' = p_h\epsilon$. If $p_hV = L_hV$ such that $|L_h| > 0$ then $L_hV \simeq p_hV \simeq Q'V$. Thus $M' \simeq p_h\sigma \simeq L_h\sigma\mathcal{R}Q'\sigma \simeq N'$. If $V(h) = p_i\epsilon$ such that $V(i)\uparrow$, then by the definition of bisimulation $Q' = p_{h'}\epsilon$ for some h' such that $V(h') = p_i\epsilon$. Then $M' \simeq p_h\sigma \simeq p_i\sigma\mathcal{R}p_i\sigma \simeq p_{h'}\sigma \simeq N'$.

The third case is symmetric to the second one. □

Let's define how to grow a bisimulation tree of $P\sigma \simeq Q\sigma$ over σ level by level with the initial $\mathcal{G}, \mathcal{E}, V$. Two things are worth spelling out. Firstly we will consider all bisimilar pairs between the descendants of $P\sigma$ and the descendants of $Q\sigma$ with intact σ. Secondly since all transitions are ϵ-pushing, the following construction never gets stuck along ϵ-edges. The growth of a node labeled by $(p_i\sigma, N\sigma)$, for $N \neq p_i\epsilon$, is defined as follows.

1. $i \notin \mathcal{E}$. Relabel the node by $(G_i\sigma, N\sigma)$.
2. $i \in \mathcal{E}$, and $|N| > 0$ or $N = p_j\epsilon$ for some $j \notin \mathcal{E}$. Update $\mathcal{G}$ by letting $G_h = N$, respectively $G_h = G_j$, for every h in the equivalence class of i. Remove the equivalence class of i from $\mathcal{E}$. Relabel the node by $(N\sigma, N\sigma)$, respectively $(G_j\sigma, G_j\sigma)$.
3. $i \in \mathcal{E}$ and $N = p_j\epsilon$ for some $j \in \mathcal{E}$. Update $\mathcal{E}$ by joining the equivalence class of i with that of j, relabel the node by $(p_{\min\{i,j\}}\sigma, p_{\min\{i,j\}}\sigma)$.

The growth of a node labeled $(N\sigma, p_i\sigma)$ is symmetric. Each time $\mathcal{G}$ or $\mathcal{E}$ has been modified, we check if the semantic equivalence $PV \simeq QV$ holds. If $PV \not\simeq QV$ then there must be a least i such that the construction of the bisimulation tree for (PV, QV) gets stuck at the i-th level. When this happens further update of $\mathcal{G}$ and/or $\mathcal{E}$ can be carried out. After a finite number of levels both $\mathcal{G}$ and $\mathcal{E}$ must stabilize and $PV \simeq QV$ must hold.

We call the final $\mathcal{G} = \{r_i\gamma_i\}_{i\in[\mathrm{q}]}$ a *recursive guard*, and

$$r_i\gamma_i\sigma \simeq p_i\sigma \tag{7}$$

with $\gamma_i \neq \epsilon$ a *recursive subgoal*. Notice that the right process in (7) does not contain any simple constants if $Q\sigma$ does not contain any simple constants. We say that the goal $P\sigma \simeq Q\sigma$ over σ has the recursive guard $\mathcal{G}$ and that it is decomposed into the subgoals $r_1\gamma_1\sigma \simeq p_1\sigma, \ldots, r_{\mathrm{q}}\gamma_{\mathrm{q}}\sigma \simeq p_{\mathrm{q}}\sigma$. Let k be the minimal number such that the recursive guard $\mathcal{G}$ of (P, Q) over σ is generated by the k-subtree of the bisimulation tree of $P\sigma \simeq Q\sigma$ over σ. We call this subtree the *generating subtree* for $\mathcal{G}$.

The recursive constant V is normed in the sense that for every state p there is a finite sequence of actions of pV that terminates on some undefined qV.

Lemma 10. *For every p_i there is some ℓ^* such that $p_iV \xrightarrow{\ell^*} p_hV$ for some undefined p_hV.*

Proof. Assume that the statement of the lemma is false for some p_i. Then $V(p_i) = r_i\gamma_i$ for some $r_i\gamma_i$ such that $|\gamma_i| > 0$. Now p_jV is defined whenever $p_iV \xrightarrow{\ell_1^*} p_jV$. That is $V(p_j) = r_j\gamma_j$ for some $r_j\gamma_j$ such that $|\gamma_j| > 0$. It follows that $p_i\sigma \simeq r_i\gamma_i\sigma \xrightarrow{\ell_1^*} p_j\sigma \simeq r_j\gamma_j\sigma$. Similarly p_kV is defined whenever $p_jV \xrightarrow{\ell_2^*} p_kV$. So $V(p_k) = r_k\gamma_k$ for some $r_k\gamma_k$ such that $|\gamma_k| > 0$. By definition $p_j\sigma \simeq r_j\gamma_j\sigma \xrightarrow{\ell_2^*} p_k\sigma \simeq r_k\gamma_k\sigma$. It should now be clear by the definition of bisimulation that $p_i\sigma$ cannot do any finite sequence of actions to reach any $p_h\epsilon$, contradicting to the fact that $p_i\sigma$ is normed. □

- $p_1X \xrightarrow{a} p_1,\ p_1X \xrightarrow{\tau} p_3X$;
- $p_2X \xrightarrow{a} p_1,\ p_2X \xrightarrow{\tau} p_3X$;
- $p_1Y \xrightarrow{a} p_3,\ p_2Y \xrightarrow{a} p_3$;
- $p_3X \xrightarrow{a} p_2$.
- $V = (p_1\epsilon, p_2\epsilon, p_3\epsilon)$.

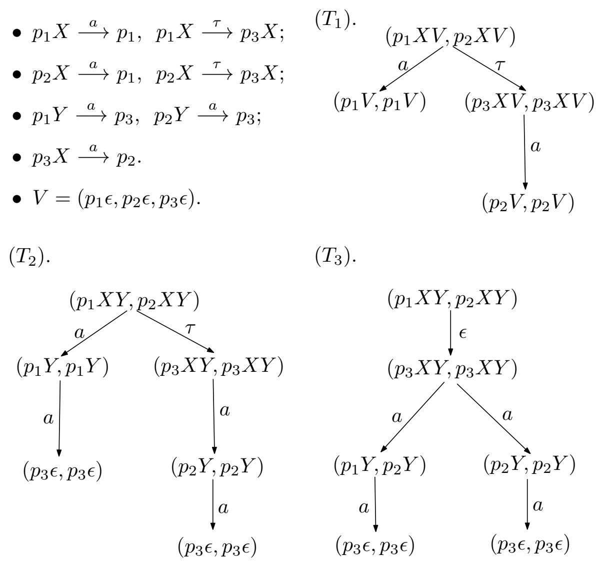

Figure 1: Construction of Bisimulation Tree

A recursive guard for P, Q is a recursive guard of $P\sigma \simeq Q\sigma$ over some σ. The following observation is crucial to the decidability argument.

Lemma 11. *The number of recursive guards for P, Q is finite.*

Proof. The initial V does not depend on any suffix σ. Suppose at a particular stage there is a minimal i such that the construction of the bisimulation tree for (PV, QV) gets stuck at level i. Due to Lemma 6 there are only finitely many ways to update V and the maximal number of ways to do that is dependent of P and Q and is independent of any suffix. We are done by induction. $\square$

7.1.1. Horizontal Composition

Now construct a bisimulation tree $\mathfrak{B}(PV, QV)$ of $PV \simeq QV$. We should not get stuck in the construction for otherwise V could be further updated. Using the grammar equalities in (6) one realizes that this is also a bisimulation tree of $PV \simeq QV$ over V. By Lemma 9 one gets from the latter tree a bisimulation tree of $P\sigma \simeq Q\sigma$ over σ if V is replaced by σ. This tree is in general finer than any bisimulation tree of $P\sigma \simeq Q\sigma$ over σ in the sense that two nodes in the former may be collapsed into one node in the latter. For the PDA defined in Fig 1, T_2 and T_3 are bisimulation trees of $p_1XY \simeq p_2XY$ (over ϵ), while T_1 is the bisimulation tree of $p_1XV \simeq p_2XV$.

The recursive constant V defined in the above construction is not unique since in the middle of the construction any effort to build up the bisimulation tree of $PV \simeq QV$ over V may get stuck at the i-level for different pairs. Suppose a different set of choices produces a different recursive constant V'. Since we have considered all bisimilar pairs between the descendants of $P\sigma$ and the descendants of $Q\sigma$ with intact σ, it must be the case that $r_i\gamma_iV' \simeq p_iV'$ for all $i \in [\mathfrak{q}]$. We derive from Lemma 9 that $LV' \simeq NV'$ implies $LV \simeq NV$ whenever $|L|, |N| > 0$. Using symmetric argument we derive that $LV \simeq NV$ implies $LV' \simeq NV'$ whenever $|L|, |N| > 0$. In other words (PV, QV) and (PV', QV') have essentially the same bisimulation tree. This observation allows us to introduce the following definition. The *characteristic tree* of $P\sigma \simeq Q\sigma$ over σ, denoted by $\chi_\sigma(P, Q)$, is the bisimulation tree of $P\sigma \simeq Q\sigma$ over σ obtained from the bisimulation tree of $PV \simeq QV$ by substituting σ for V. We assume that every characteristic tree is presented with the associated subgoals.

We now explain how to make use of the characteristic tree $\chi_\sigma(P, Q)$. Suppose $\{G_i\sigma \simeq p_i\sigma\}_{i\in[\mathfrak{q}]}$ are the set of subgoals generated in the construction of $\chi_\sigma(P, Q)$ and for each $i \in [\mathfrak{q}]$ let $\mathcal{B}(G_i\sigma, p_i\sigma)$ be a branching bisimulation rooted at $(G_i\sigma, p_i\sigma)$. Let $\mathcal{B} = \bigcup_{k\in\omega} \mathcal{B}_k$, where $\mathcal{B}_k$ is defined as follows:

1. $\mathcal{B}_0 = \bigcup_{i\in[\mathfrak{q}]} \mathcal{B}(G_i\sigma, p_i\sigma)$.

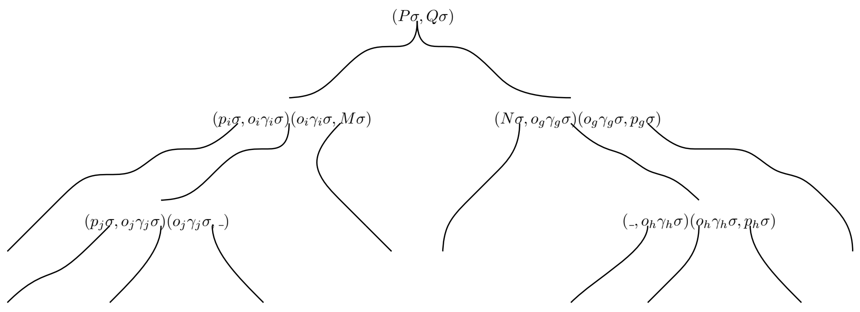
$(P\sigma, Q\sigma)$
$(p_i\sigma, o_i\gamma_i\sigma)(o_i\gamma_i\sigma, M\sigma)$
$(N\sigma, o_g\gamma_g\sigma)(o_g\gamma_g\sigma, p_g\sigma)$
$(p_j\sigma, o_j\gamma_j\sigma)(o_j\gamma_j\sigma, \_)$
$(\_, o_h\gamma_h\sigma)(o_h\gamma_h\sigma, p_h\sigma)$

Figure 2: Horizontal Composition

2. $\mathcal{B}_{k+1} = \mathcal{B}_k \cup \mathcal{B}_0; \mathcal{B}_k$.

Since bisimulations are closed under union and composition [1], the relations $\mathcal{B}_k$ for $k \geq 0$ and the relation $\mathcal{B}$ are branching bisimulations. Define $\mathcal{B}(P\sigma, Q\sigma)$ by

$$\mathcal{B}(P\sigma, Q\sigma) \quad = \quad (\mathcal{B}^{-1} \cup \mathcal{I}); \overline{\chi_\sigma(P, Q)}; (\mathcal{B} \cup \mathcal{I}),$$

where $\mathcal{I}$ is the identity relation on the nPDA$^{\epsilon+}$ processes.

Lemma 12. *$\mathcal{B}(P\sigma, Q\sigma)$ is a branching bisimulation.*

Proof. The relation $\overline{\chi_\sigma(P, Q)}$ is not a branching bisimulation of nPDA$^{\epsilon+}$ for the following reason: It may contain a pair $(r_i\gamma_i\sigma, M\sigma)$ obtained from $(p_i\sigma, M\sigma)$ or a pair $(N\sigma, r_i\gamma_i\sigma)$ obtained from $(N\sigma, p_i\sigma)$. But notice that for each $i \in [\mathfrak{q}]$ the relation $\mathcal{B}^{-1}$ contains a branching bisimulation for $p_i\sigma \simeq r_i\gamma_i\sigma$ and the relation $\mathcal{B}$ contains a branching bisimulation for $r_i\gamma_i\sigma \simeq p_i\sigma$. Therefore the pairs of the form $(p_i\sigma, M\sigma)$ and of the form $(N\sigma, p_i\sigma)$ for $i \in [\mathfrak{q}]$ enjoy the bisimulation property by composition. A diagrammatic illustration of the composition is given in Figure 2. The middle part is $\overline{\chi_\sigma(P, Q)}$. The left and right parts are contained in $\mathcal{B}^{-1}$ and $\mathcal{B}$ respectively. We will call this type of composition *horizontal composition*. □

We conclude that to prove $P\sigma \simeq Q\sigma$ we only need to construct the characteristic tree $\chi_\sigma(P, Q)$ and the bisimulation trees for the subgoals.

7.2. Simple Subgoal

In this subsection we take a look at decomposition of large goals. Recall that according to Corollary 5 the size of the left side of a goal is constrained by the size of the right side of the goal. Two bisimilar processes with large enough size is of the form

$$pX\alpha\sigma \simeq M\delta\sigma \tag{8}$$

such that $|M| = \mathfrak{m}$ and $|\delta\sigma| > 0$. In Section 4 we have introduced a simple constant, defined in (4), that turns the goal $pX\alpha\sigma \simeq M\delta\sigma$ into the goal

$$pXU\delta\sigma \simeq M\delta\sigma. \tag{9}$$

We get for every $i \in \ker \|pX\|$ the subgoal

$$p_i\alpha\sigma \simeq s_i\eta_i\delta\sigma. \tag{10}$$

We call (10) a *simple subgoal* and the family $\{s_i\eta_i\}_{i\in\ker\|pX\|}$ a *simple guard*. Notice that since $M\delta\sigma$ does not contain any simple constants, $s_i\eta_i\delta\sigma$ does not contain any simple constants.

For further simplification we firstly decompose a goal of the form (8) satisfying $|M| = \mathfrak{m}$ to a goal of the form (9) and simple subgoals of the form (10). We then apply the decomposition described in Section 7.1 to (9). Now let V be the recursive constant generated in the construction of $\chi_{\delta\sigma}(pXU, M)$. We remark that the recursive subgoals are of the type

$$r_i\gamma_i\delta\sigma \simeq p_i\delta\sigma. \tag{11}$$

By Lemma 3 there are only finitely many simple constants. For each such constant there are finitely many recursive constants by Lemma 11. Hence the following.

Lemma 13. *Suppose $|M| = \mathfrak{m}$. There are only finitely many recursive guards for fixed pX and M.*

To avoid having to spell out the simple constant, we shall write $\chi_{\alpha,\delta\sigma}(pX, M)$ for $\chi_{\delta\sigma}(pXU, M)$. We say that $\chi_{\alpha,\delta\sigma}(pX, M)$ and $\chi_{\alpha',\delta'\sigma'}(pX, M)$ are of same *type* if they are obtained from the same bisimulation tree $\mathfrak{B}(pXUV, MV)$ of $pXUV \simeq MV$; in other words they have the same characteristic tree. In this case we also say that $\chi_{\alpha,\delta\sigma}(pX, M)$ and $\chi_{\alpha',\delta'\sigma'}(pX, M)$ are *duplicates* of each other.

For each $h \in \ker \|pX\|$ let C_s^h be a branching bisimulation rooted at the pair in (10). For $i \in [\mathfrak{q}]$ let C_r^i be a branching bisimulation rooted at the pair in (11). Let $C_s = \bigcup_{h\in\ker\|pX\|} C_s^h$ and $C = \bigcup_{k\in\omega} C_k$, where C_k is defined inductively by the following two clauses.

1. $C_0 = \bigcup_{i\in[\mathfrak{q}]} C_r^i$.
2. $C_{k+1} = C_k \cup C_0; C_k$.

Define $C(pX\alpha\sigma, M\delta\sigma)$ by $\left(C_s \cup C^{-1} \cup \mathcal{I}\right); \overline{\chi_{\alpha,\delta\sigma}(pX, M)}; (C_s^{-1} \cup C \cup \mathcal{I})$. Using again the fact that bisimulations are closed under composition and union one sees that C_k for all $k \geq 0$ and C are branching bisimulations.

Lemma 14. *$C(pX\alpha\sigma, M\delta\sigma)$ is a branching bisimulation.*

Proof. The argument is similar to the one for Lemma 12. The treatment of a pair of the form $(s_i\eta_i\delta\sigma, N)$ obtained from the pair $(p_i\alpha\sigma, N)$ is by straightforward composition. □

7.3. Generic Tree

In this subsection we complete the process of cutting down the size of bisimulation trees by introducing conditions for nodes not to grow. Here are two obvious conditions.

1. If the label of a node is the same label as an ancestor, the node stops to grow.
2. If the label of a node is a pair of identical processes, the node stops to grow. For example $(p\epsilon, p\epsilon)$ is such a label.

The lemmas of Section 7.1 and Section 7.2 imply that there is a bound $\mathfrak{c}$ such that the following property holds:

> For every goal $pX\alpha\sigma \simeq M\delta\sigma$ satisfying $|M| = \mathfrak{m}$ and $|\delta\sigma| \geq \mathfrak{c}$, there is a goal $pX\alpha'\sigma' \simeq M\delta'\sigma'$ such that $0 < |\delta'\sigma'| < \mathfrak{c}$ and that $\chi_{\alpha',\delta'\sigma'}(pX, M)$ is of the same type as $\chi_{\alpha,\delta\sigma}(pX, M)$.

Recall that the construction of $\chi_{\alpha,\delta\sigma}(pX, M)$ generates both simple subgoals and recursive subgoals. The situation is simpler when we construct the characteristic tree for a recursive subgoal because no more simple subgoals need be generated. Suppose $r\gamma\sigma \simeq p\sigma$ is a recursive subgoal. If $|\sigma| > \mathfrak{m}$ we can write σ as $\mu\sigma'$ such that $|\mu| = \mathfrak{m}$. The recursive subgoal becomes $r\gamma\mu\sigma' \simeq p\mu\sigma'$. Since there is a bound on γ, there is only a finite set of pairs $(r\gamma\mu, p\mu)$. If we construct the characteristic tree of $r\gamma\mu\sigma' \simeq p\mu\sigma'$ over σ', we get recursive subgoals of the form $r'\gamma'\sigma' \simeq p'\sigma'$. Continue to construct the characteristic tree of $r'\gamma'\sigma' \simeq p'\sigma'$. We get new recursive subgoals whose right hand sides are of even smaller size. From an algorithmic viewpoint this provides a termination condition. So we distinguish the case $|\delta| = 0$ from the case $|\delta| > 0$. Consequently we consider three types of goals.

1. The goals $L \simeq M$ are such that $|M| \leq \mathfrak{m}$. In this case grow a bisimulation tree of $L \simeq M$ as is defined in Section 6. When a node labeled (L', M') is generated such that $|M'| \geq \mathfrak{m} + \mathfrak{c}$, the node stops growing. We call such trees *type-I generic trees*.
2. The goals $pX\alpha\sigma \simeq M\delta\sigma$ are such that $|M| = \mathfrak{m}$ and $|\delta| > 0$ and $|\delta\sigma| < \mathfrak{c}$. Grow the characteristic tree of $pX\alpha\sigma \simeq M\delta\sigma$ over $\delta\sigma$ with the following additional constraint: If a node labeled (L', M') is generated such that $|M'| \geq \mathfrak{m} + \mathfrak{c}$, the node stops growing. These are *type-II generic trees*.
3. The goals $pX\alpha\sigma \simeq M\sigma$ are such that $|M| = \mathfrak{m}$ and $|pX\alpha| < \mathfrak{c} + \mathfrak{m}$ and $|\sigma| < \mathfrak{c}$. Grow the characteristic tree of $pX\alpha\sigma \simeq M\sigma$ over σ with the following additional constraint: If a node labeled (L', M') is generated such that $|M'| \geq \mathfrak{m} + \mathfrak{c}$, the node stops growing. These are *type-III generic trees*.

In a generic tree we will call a leaf (L, M) such that $|M| \geq \mathfrak{m} + \mathfrak{c}$ a *large leaf*. The other leaves are *small* leaves. For the above construction to make sense, we should choose the bound $\mathfrak{c}$ such that

1. it is larger than the size of all the simple and recursive guards generated in the constructions, and
2. it is larger than the height of all the generating subtrees.

We call the label of the root of a generic tree a *generic goal*. We assume that every generic tree is presented with the associated subgoals, and we sometimes confuse a generic goal with the generic tree. By definition and Corollary 5 the set of generic goals as well as the set of generic trees is finite. Moreover we have the following important fact.

Lemma 15. *Every generic tree is finite.*

Proof. The generic trees of the first type is obviously finite. In the light of Corollary 5 if in a path no large leaf is ever generated, the path end with a small leaf that is labeled either by a pair of identical processes or by a pair that appears twice in the path. We are done by applying König Lemma. □

7.3.1. Vertical Composition

Suppose $q_1X_1\alpha_1\sigma_1 \simeq M_1\delta_1\sigma_1, \ldots, q_gX_g\alpha_g\sigma_g \simeq M_g\delta_g\sigma_g$ are the generic goals and let the corresponding generic trees be denoted by $T_{q_1X_1\alpha_1\sigma_1\simeq M_1\delta_1\sigma_1}, \ldots, T_{q_gX_g\alpha_g\sigma_g\simeq M_g\delta_g\sigma_g}$. We say that a goal $q_iX_i\alpha\sigma \simeq M_i\delta\sigma$ with $|\delta\sigma| \geq \mathfrak{c}$ is of type i if $|\delta_i| > 0$ and $\chi_{\alpha,\delta\sigma}(q_iX_i, M_i)$ and $\chi_{\alpha_i,\delta_i\sigma_i}(q_iX_i, M_i)$ are of the same type, and that a goal $q_iX_i\alpha\sigma \simeq M_i\sigma$ with $|\sigma| \geq \mathfrak{c}$ is of type i if $\chi_\sigma(q_iX_i\alpha_i, M_i)$ and $\chi_{\sigma_i}(q_iX_i\alpha_i, M_i)$ are of the same type. For each $i \in [g]$ let

$$\mathcal{B}_i = \bigcup_{q_iX_i\alpha\sigma\simeq M_i\delta\sigma \text{ is of type } i} \overline{T_{q_iX_i\alpha_i\sigma_i\simeq M_i\delta_i\sigma_i}}\{\alpha/\alpha_i, \delta/\delta_i, \sigma/\sigma_i\},$$

where the relation $\overline{T_{q_iX_i\alpha_i\sigma_i\simeq M_i\delta_i\sigma_i}}\{\alpha/\alpha_i, \delta/\delta_i, \sigma/\sigma_i\}$ is obtained from $\overline{T_{q_iX_i\alpha_i\sigma_i\simeq M_i\delta_i\sigma_i}}$ by substituting α for α_i, δ for δ_i and σ for σ_i. Notice that if $\delta_i = \epsilon$ then $\overline{T_{q_iX_i\alpha_i\sigma_i\simeq M_i\delta_i\sigma_i}}\{\alpha/\alpha_i, \delta/\delta_i, \sigma/\sigma_i\}$ is $\overline{T_{q_iX_i\alpha_i\sigma_i\simeq M_i\delta_i\sigma_i}}\{\sigma/\sigma_i\}$. Let

$$\mathcal{BB} = \bigcup_{i\in[g]} \mathcal{B}_i.$$

Finally let $\mathcal{BB}^* = \left(\mathcal{I} \cup \mathcal{BB} \cup \mathcal{BB}^{-1}\right)^*$.

Proposition 16. *$\mathcal{BB}^*$ is a branching bisimulation.*

Proof. In general the generic tree $T_{q_iX_i\alpha_i\sigma_i\simeq M_i\delta_i\sigma_i}$ does not give rise to a bisimulation of $(q_iX_i\alpha_i\sigma_i, M_i\delta_i\sigma_i)$. For one thing the large leafs of the tree are not taken care of. This problem can be addressed by *vertical composition*. Suppose $(q_jX_j\alpha\sigma, M_j\delta\sigma)$ is a large leaf of $T_{q_iX_i\alpha_i\sigma_i\simeq M_i\delta_i\sigma_i}$ of type j. We can graft the duplicate $T_{q_jX_j\alpha_j\sigma_j\simeq M_j\delta_j\sigma_j}\{\alpha/\alpha_j, \delta/\delta_j, \sigma/\sigma_j\}$ of $T_{q_jX_j\alpha_j\sigma_j\simeq M_j\delta_j\sigma_j}$ to the large leaf. This vertical composition can be carried out ad infinitum. What we get eventually is a decomposable version $\mathcal{D}$ of the characteristic tree $\chi_{\alpha_i,\delta_i\sigma_i}(q_iX_i, M_i)$. Notice that $\overline{\mathcal{D}}$ is a subset of $\mathcal{BB}^*$.

To turn the relation $\overline{\mathcal{D}}$ into a branching bisimulation of nPDA$^{\epsilon+}$, we need to compose it with bisimulations generated from the subgoals. This is the horizontal composition described in the proof of Lemma 12. Now to construct a bisimulation for a subgoal we need to do further vertical and horizontal compositions. So in essence all the vertical compositions are carried out simultaneously and level by level, and the horizontal compositions are interleaved with the vertical compositions.

Every pair in $\mathcal{BB} \cup \mathcal{BB}^{-1}$ appears in a branching bisimulation contained in $\mathcal{BB}^*$. Consequently $\mathcal{BB}^*$ must be a branching bisimulation. □

7.4. Decomposition via Generic Tree

Let $\mathfrak{A} = \{T_{(q_1Y_1\alpha_1\sigma_1, M_1\delta_1\sigma_1)}, \ldots, T_{(q_hY_h\alpha_h\sigma_h, M_h\delta_h\sigma_h)}\}$ be a set of generic trees. Given a goal (P, Q) we would like to check if (P, Q) can be decomposed in terms of the generic trees in $\mathfrak{A}$. The algorithm Decomposition$_\mathfrak{A}$ is described in Figure 3. If the input satisfies $|Q| < \mathfrak{m} + \mathfrak{c}$, it simply checks if $T_{(P,Q)} \in \mathfrak{A}$. Otherwise it checks if the input has a characteristic tree that is a duplicate of some element of $\mathfrak{A}$. The checks have to be carried out in an inductive fashion.

```
The input is $\mathfrak{A} = \{T_{(q_1Y_1\alpha_1\sigma_1, M_1\delta_1\sigma_1)}, \ldots, T_{(q_hY_h\alpha_h\sigma_h, M_h\delta_h\sigma_h)}\}$.
1. If $|Q| < \mathfrak{m} + \mathfrak{c}$, return true and halt whenever $T_{(P,Q)} \in \mathfrak{A}$, and return false and halt whenever $T_{(P,Q)} \notin \mathfrak{A}$.
2. If $|Q| \geq \mathfrak{m} + \mathfrak{c}$, let $(P, Q)$ be $(pX\alpha\sigma, M\delta\sigma)$ such that $|M| = \mathfrak{m}$, and do the following.
   (a) If $|\delta| > 0$ and either $pX \neq q_iY_i$ for all $i \in [h]$ or $M \neq M_i$ for all $i \in [h]$, return false and halt. Otherwise guess some $j \in [h]$ such that $pX = q_jY_j$ and $M = M_j$, and do the following.
       i. For each simple subgoal $(p_i\alpha_j\sigma_j, s_i\eta_i\delta_j\sigma_j)$ of the tree $T_{(q_jY_j\alpha_j\sigma_j, M_j\delta_j\sigma_j)} \in \mathfrak{A}$, continue if DECOMPOSITION$_{\mathfrak{A}}$ $(p_i\alpha\sigma, s_i\eta_i\delta\sigma) = true$.
       ii. For each recursive subgoal $(r_i\gamma_i\delta_j\sigma_j, p_i\delta_j\sigma_j)$ of the tree $T_{(q_jY_j\alpha_j\sigma_j, M_j\delta_j\sigma_j)} \in \mathfrak{A}$, continue if DECOMPOSITION$_{\mathfrak{A}}$ $(r_i\gamma_i\delta\sigma, p_i\delta\sigma) = true$.
   (b) If $|\delta| = 0$ and either $pX\alpha \neq q_iY_i\alpha_i$ for all $i \in [h]$ or $M \neq M_i$ for all $i \in [h]$ such that $\delta_i = \epsilon$, return false and halt. Otherwise guess some $j \in [h]$ such that $pX\alpha = q_jY_j\alpha_j$ and $M = M_j$, and do the following.
       i. For each recursive subgoal $(r_i\gamma_i\sigma_j, p_i\sigma_j)$ of the tree $T_{(q_jY_j\alpha_j\sigma_j, M_j\delta_j\sigma_j)} \in \mathfrak{A}$ with $\delta_j = \epsilon$, continue if DECOMPOSITION$_{\mathfrak{A}}$ $(r_i\gamma_i\sigma, p_i\sigma) = true$.
   (c) Return true.
```

Figure 3: DECOMPOSITION$_{\mathfrak{A}}$ (P, Q)

```
1. Guess a set $\mathfrak{A}$ of generic trees bounded by $\mathfrak{c}$.
2. For every $T_{(L,N)} \in \mathfrak{A}$ with $|N| \leq \mathfrak{m}$, if it fails the bisimulation property, report a failure and halt.
3. For every $T_{(pX\alpha\sigma, M\delta\sigma)} \in \mathfrak{A}$ such that $|M| = \mathfrak{m}$ and $0 < |\delta\sigma| < \mathfrak{c}$, do the following.
   (a) If $T_{(pX\alpha\sigma, M\delta\sigma)}$ fails the bisimulation property, report a failure and halt.
   (b) For every recursive subgoal $(L, N)$ generated by $T_{(pX\alpha\sigma, M\delta\sigma)}$, continue if DECOMPOSITION$_{\mathfrak{A}}$ $(L, N) = true$.
   (c) For every simple subgoal $(L, N)$ generated by $T_{(pX\alpha\sigma, M\delta\sigma)}$, continue if DECOMPOSITION$_{\mathfrak{A}}$ $(L, N) = true$.
   (d) For every large leaf $(L, N)$ of $T_{(pX\alpha\sigma, M\delta\sigma)}$, continue if DECOMPOSITION$_{\mathfrak{A}}$ $(L, N) = true$.
4. Output $\mathfrak{A}$.
```

Figure 4: PREBASE $(\mathfrak{c})$

All the subgoals should also be decomposed in terms of the elements of $\mathfrak{A}$. We say that a goal (P, Q) is *decomposable with regards to* $\mathfrak{A}$ if DECOMPOSITION$_{\mathfrak{A}}$ (P, Q) returns true.

The algorithm must terminate. In the recursive call in Step (2(a)i) the size of $p\alpha\sigma$ is strictly smaller than the size of $pX\alpha\sigma$. The induction ends with processes of the form $q\epsilon$. So Step (2(a)i) can only be executed for a finite number of times, and only finitely many recursive subgoals can be introduced by Step (2(a)i). In the recursive call in Step (2(a)ii) the size of $p_i\delta\sigma$ is strictly smaller than the size of $M\delta\sigma$. So Step (2(a)ii) cannot be executed infinitely often. For the same reason Step (2(b)i) cannot be executed infinitely often. It is important to notice that the decomposition of the recursive subgoals does not introduce any new simple subgoals.

7.5. Generation of Generic Tree

We shall show that the generic trees are the building blocks for the bisimulation trees. Every bisimulation can be seen as composed of generic trees in an inductive fashion. This should provide a semidecidable procedure for checking branching bisimilarity for nPDA$^{\epsilon+}$ by enumeration. It should be emphasized that it is insufficient to know the *existence* of the set of generic trees. We need to be able to enumerate size increasing sets of generic trees that approach to the set of *all* generic trees. Given a bound $\mathfrak{c}$ we can guess a set $\mathfrak{A} = \{T_{(q_1Y_1\alpha_1\sigma_1, M_1\delta_1\sigma_1)}, \ldots, T_{(q_hY_h\alpha_h\sigma_h, M_h\delta_h\sigma_h)}\}$ and check if they are all generic trees bounded by $\mathfrak{c}$. More explicitly we need to check the following.

1. All members of $\mathfrak{A}$ satisfy the bisimulation property.
2. All large leaves can be decomposed in terms of the trees in $\mathfrak{A}$.
3. All subgoals generated inductively can be decomposed in terms of the trees in $\mathfrak{A}$.

We call $\mathfrak{A}$ a *prebase* if it satisfies the above three conditions. The nondeterministic algorithm PREBASE$(\mathfrak{c})$ is defined in Figure 4. Since the size of the right hand side of every internal node of every member of $\mathfrak{A}$ is bounded by $\mathfrak{m} + \mathfrak{c}$, The input $\mathfrak{A} = \{T_{(q_1Y_1\alpha_1\sigma_1, M_1\delta_1\sigma_1)}, \ldots, T_{(q_hY_h\alpha_h\sigma_h, M_h\delta_h\sigma_h)}\}$ is a prebase.

1. Let $\mathfrak{P} = \emptyset$.
2. For $i = 1$ to h, do the following.
   (a) Let $T = T_{(q_iY_i\alpha_i\sigma_i, M_i\delta_i\sigma_i)}$. Mark all the large leaves of T as unsuccessful. Repeat the following for every unsuccessful large leaf.
      i. If there is a large leaf of T of type say j, grow a duplicate of $T_{(q_jY_j\alpha_j\sigma_j, M_j\delta_j\sigma_j)}$ at this large leaf; and if moreover there is a node in the path to the root that is also a large leaf of type j, mark all the large leaves of the duplicate as successful.
      ii. Check if every subgoal generated in Step (2(a)i) is decomposable with regards to $\mathfrak{A}$. If any of the subgoals is not decomposable, report failure and halt.
   (b) Let $\mathfrak{P} = \mathfrak{P} \cup \{T\}$.
3. Output $\mathfrak{P}$.

Figure 5: Base ($\mathfrak{A}$)

and consequently the size of the right hand sides of the leaves are bounded by $\mathfrak{m} + \mathfrak{c} + \mathfrak{r}$. Lemma 6 and Corollary 5 imply that there is a bound on the size of $\mathfrak{A}$ if the algorithm does not report any failure. Thus the termination of PreBase($\mathfrak{c}$) follows from the termination of Decomposition$_{\mathfrak{A}}$ and the fact that the decomposition of a recursive subgoal does not generate any simple subgoal. We remark that even if PreBase($\mathfrak{c}$) terminates successfully, it does not mean that PreBase($\mathfrak{c}$) has found out all generic trees. What it has found is a set of generic trees enjoying some closure property.

8. Composing Bisimulation Trees from Finite Trees

We now show how to compose an infinite bisimulation tree from generic trees. Suppose $\mathfrak{T}$ is a generic tree in a prebase. If we can grow $\mathfrak{T}$ into a bisimulation tree we can grow every tree that is of the same type as $\mathfrak{T}$ into a bisimulation tree. It is tempting to think that every generic tree in a prebase can be grown into a bisimulaiton tree. As it turns out we need additional periodic structures to recover the bisimulation property of the composed tree from the bisimulation property of the component trees.

8.1. Bisimulation Base

Let $\mathfrak{A} = \{T_{(q_1Y_1\alpha_1\sigma_1, M_1\delta_1\sigma_1)}, \ldots, T_{(q_hY_h\alpha_h\sigma_h, M_h\delta_h\sigma_h)}\}$ be a prebase. For $i \in [h]$ we can try to grow $T = T_{(q_iY_i\alpha_i\sigma_i, M_i\delta_i\sigma_i)}$ in the following manner: For every large leaf of T if it is of the same type as some $T_{(q_{i'}Y_{i'}\alpha_{i'}\sigma_{i'}, M_{i'}\delta_{i'}\sigma_{i'})}$, graft a duplicate of $T_{(q_{i'}Y_{i'}\alpha_{i'}\sigma_{i'}, M_{i'}\delta_{i'}\sigma_{i'})}$ on the large leaf. If the duplicate of any subgoal generated by $T_{(q_{i'}Y_{i'}\alpha_{i'}\sigma_{i'}, M_{i'}\delta_{i'}\sigma_{i'})}$ is not decomposable with regards to $\mathfrak{A}$, report a failure. If every large leaf of T can be grown in this way, then due to the finite branching property and König Lemma every path of T reaches to either a node with identical processes or a leaf whose label repeats the label of one of its ancestors or a large leaf that is of the same type as some duplicate along the path to the root of T. In the first and the second cases the path ends with a small node. In the last case we graft a duplicate on the large leaf, and no leaf of the duplicate will ever be grown. It is easy to see that there is a computable bound, parameterised over c, on the height of the final tree. Consequently the size of subgoals generated during the generation of the final tree is controlled, and there is a computable bound, again parameterised over c, on the number of such subgoals. Using Decomposition$_{\mathfrak{A}}$ we can check if the subgoals are decomposable with regards to $\mathfrak{A}$. If all paths end successfully in this way we get a finite tree to be called a *production tree* of $(q_iY_i\alpha_i\sigma_i, M_i\delta_i\sigma_i)$. If every element of $\mathfrak{A}$ can be extended to a production tree, the set $\mathfrak{A}$ is a *bisimulation base*. The algorithm Base ($\mathfrak{A}$) defined in Figure 6 reports a failure if $\mathfrak{A}$ is not a bisimulation base and outputs the set of production trees otherwise.

Let's explain the idea using the left diagram in Figure 6. The two shaded duplicates are of type j for some $j \in [h]$, and there is no other duplicate of type j along the path to the root. Moreover there are no other two duplicates of any other type along the path to the root. Under these conditions none of the leaf of the bottom shaded duplicate will be grown any more. One may image that a duplicate, with all its subgoals, is a mega node. A path in a production tree ends with a mega node that has appeared in the above and the two mega nodes are of the same type as it were.

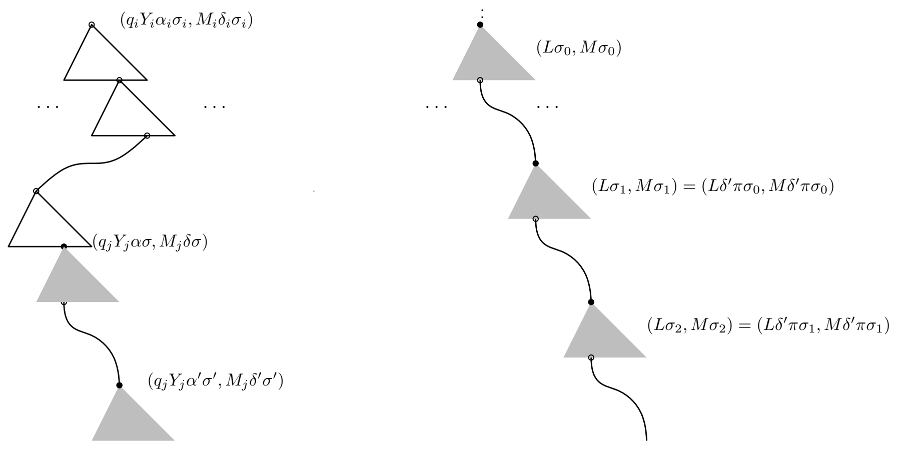
$(q_iY_i\alpha_i\sigma_i, M_i\delta_i\sigma_i)$
$(q_jY_j\alpha\sigma, M_j\delta\sigma)$
$(q_jY_j\alpha'\sigma', M_j\delta'\sigma')$
$(L\sigma_0, M\sigma_0)$
$(L\sigma_1, M\sigma_1) = (L\delta'\pi\sigma_0, M\delta'\pi\sigma_0)$
$(L\sigma_2, M\sigma_2) = (L\delta'\pi\sigma_1, M\delta'\pi\sigma_1)$

Figure 6: A Production Tree and its Unfolding

8.2. Soundness

Let $\mathfrak{P}$ be the set of production trees defined from a set of generic trees $\mathfrak{A} = \{T_{(q_1Y_1\alpha_1\sigma_1, M_1\delta_1\sigma_1)}, \ldots, T_{(q_hY_h\alpha_h\sigma_h, M_h\delta_h\sigma_h)}\}$. We refer to the production trees as $\mathfrak{P}_1, \ldots, \mathfrak{P}_h$. The soundness of $\mathfrak{P}$ means that every tree in $\mathfrak{P}$, and every tree in $\mathfrak{A}$ as well, can be extended to a bisimulation tree.

Proposition 17. *$q_iY_i\alpha_i\sigma_i \simeq M_i\delta_i\sigma_i$ for every $i \in [h]$.*

Proof. The basic idea is of course to grow the production tree $\mathfrak{P}_i$ to a bisimulation tree of $q_iY_i\alpha_i\sigma_i \simeq M_i\delta_i\sigma_i$. This is more complicated than the constructions we have seen before because we need to grow all the subgoals simultaneously in order to carry out horizontal composition. The definition of production tree allows one to do vertical composition. But unlike the situation in the proof of Lemma 15, vertical composition depends on the horizontal compositions carried out at previous level. We must explain, starting from the production tree $\mathfrak{P}_i$, how bisimulation rooted at a leaf is constructed. What is tricky in the present situation is that bisimulation of a node must be constructed level by level.

- If the leaf is labeled by a pair of identical processes, the construction is obvious.
- If the leaf has the same label as one of its ancestors, the construction has already been done.
- In the path from the leaf to the root there are two nodes $(q_jY_j\alpha\sigma, M_j\delta\sigma)$ and $(q_jY_j\alpha'\sigma', M_j\delta'\sigma')$ that are of the same type, say j, and are the large leaves of two different duplicates of some $T_{(q_kY_k\alpha_k\sigma_k, M_k\delta_k\sigma_k)}$. Suppose $(q_jY_j\alpha\sigma, M_j\delta\sigma)$ is the ancestor and $(q_jY_j\alpha'\sigma', M_j\delta'\sigma')$ is the descendant. This is the situation described by the left diagram of Figure 6. Let $\sigma_0 = \delta\sigma$. The nodes in the upper shaded duplicate are of the form $(L\sigma_0, M\sigma_0)$. In the bottom shaded duplicate the suffix σ' must be $\pi\sigma_0$ for some π. Let $\sigma_1 = \delta'\sigma' = \delta'\pi\sigma_0$. A node $(L\sigma_0, M\sigma_0)$ in the upper shaded duplicate corresponds to the node $(L\sigma_1, M\sigma_1)$ in the bottom shaded duplicate in the corresponding position. The key to our construction is to use the fact that the subgoals generated in the two shaded duplicates are in one-one correspondence in the sense that a subgoal
$$(r_i\gamma_i\sigma_0, p_i\sigma_0)$$
corresponds to the subgoal
$$(r_i\gamma_i\sigma_1, p_i\sigma_1).$$
Clearly we can construct the bisimulation rooted at $(q_jY_j\alpha'\sigma', M_j\delta'\sigma')$ in precisely the same way as we construct the bisimulation rooted at $(q_jY_j\alpha\sigma, M_j\delta\sigma)$ by composing with bsimulations for respective subgoals horizontally. The bisimulation of $(q_jY_j\alpha'\sigma', M_j\delta'\sigma')$ is grown by horizontal composition. During the growth of the bisimulation tree we will reach to the third duplicate $(q_jY_j\alpha''\sigma'', M_j\delta'\sigma'')$ of $T_{(q_kY_k\alpha_k\sigma_k, M_k\delta_k\sigma_k)}$. The nodes in

1. Check if $P \not\simeq Q$, and at the same time run in parallel PreBase($\mathfrak{m}+1$), PreBase($\mathfrak{m}+2$), ...
2. If $P \not\simeq Q$ then answer 'no' and halt.
3. If for some $\mathfrak{c} > \mathfrak{m}$, PreBase($\mathfrak{c}$) terminates successfully with a prebase $\mathfrak{A}$ of generic trees, run Base($\mathfrak{A}$). If Base($\mathfrak{A}$) outputs a bisimulation base then answer 'yes' and halt if Decomposition$_{\mathfrak{A}}$ (P, Q) returns *true*.

Figure 7: EqCheck (P, Q)

this duplicate are of the form $(L\sigma_2, M\sigma_2)$ with $\sigma_2 = \delta'\sigma'' = \delta'\pi\sigma_1$. See the right diagram of Figure 6. We now grow the bisimulation tree of the node $(L\sigma_2, M\sigma_2)$, which is

$$(L\delta'\pi\sigma_1, M\delta'\pi\sigma_1), \tag{12}$$

by simulating the growth of the bisimulation tree of the node $(L\sigma_1, M\sigma_1)$, which is

$$(L\delta'\pi\sigma_0, M\delta'\pi\sigma_0). \tag{13}$$

Notice that the node $(L\delta'\pi\sigma_0, M\delta'\pi\sigma_0)$ is at a level higher than the level of the node $(L\delta'\pi\sigma_1, M\delta'\pi\sigma_1)$. We only have to define how the growth of a node of the form $(p_i\sigma_0, M'\sigma_0)$ in the bisimulation of $(L\delta'\pi\sigma_0, M\delta'\pi\sigma_0)$ is simulated by the growth of the node $(p_i\sigma_1, M'\sigma_1)$ in the bisimulation of $(L\delta'\pi\sigma_1, M\delta'\pi\sigma_1)$. The following account provides the intuition although it is precise since bisimulation must take care of intermediate states. Suppose $(p_i\sigma_0, M'\sigma_0) \xrightarrow{\epsilon} (P', M''\sigma_0)$. Then for any $(r_i\gamma_i\sigma_0, p_i\sigma_0) \xrightarrow{\epsilon} (L'\sigma_0, P')$ we have $(r_i\gamma_i\sigma_0, M'\sigma_0) \xrightarrow{\epsilon} (L'\sigma_0, M''\sigma_0)$ by composition. Since the characteristic trees of $(q_jY_j\alpha\sigma, M_j\delta\sigma)$ and $(q_jY_j\alpha'\sigma', M_j\delta'\sigma')$ are duplicate of each other, $(r_i\gamma_i\sigma_1, M'\sigma_1) \xrightarrow{\epsilon} (L'\sigma_1, M''\sigma_1)$. Thus for any $(p_i\sigma_1, r_i\gamma_i\sigma_1) \xrightarrow{\epsilon} (Q', L'\sigma_1)$ we have $(p_i\sigma_1, M'\sigma_1) \xrightarrow{\epsilon} (Q', M''\sigma_1)$ by composition. Similarly if $(p_i\sigma_0, M'\sigma_0)$ has a τ-edge/a-edge, we can grow the τ-edge/a-edge of $(p_i\sigma_1, M'\sigma_1)$ in the same fashion. The point here is that the simulation must be carried out level by level. The pair (12) in the bisimulation rooted at $(q_jY_j\alpha'\sigma', M_j\delta'\sigma')$ will be seen as the pair $(L\delta'\pi\delta'\pi\sigma_0, M\delta'\pi\delta'\pi\sigma_0)$ in the bisimulation rooted at $(q_jY_j\alpha\sigma, M_j\delta\sigma)$, which will be further simulated by the corresponding pair down in the former bisimulation.

What we have described in the above is the vertical composition. This part of the tree satisfies the bisimulation property by composition and duplication. To construct the whole tree $\mathfrak{T}_i$ we need to carry out horizontal composition with the bisimulation trees of the subgoals. Since every subgoal is decomposable, it is either a generic goal, whose bisimulation tree is described in the above, or is of the same type as some generic goal. In the latter case a bisimulation tree of the subgoal can be constructed by simulating the bisimulation tree of the generic goal in precisely the same manner described in the above. We are in a situation similar to the one in the proof of Proposition 16. We can grow all the relevant trees simultaneously by vertical composition and at the meantime do the horizontal composition level by level. The mutual dependence is inductive. □

The above proof implies immediately the following.

Corollary 18. *If a goal (P, Q) is decomposable with regards to a bisimulation base, then $P \simeq Q$.*

8.3. Decidability

We are ready to give a decision algorithm. The algorithm EqCheck is defined in Figure 7. The termination of the algorithm is clear in the light of Proposition 7. We have effectively proved the main result of the paper.

Theorem 19. *The relation $\simeq_{\mathrm{nPDA}^{\epsilon+}}$ is decidable.*

9. High Undecidability of ϵ-Nondeterminism

In this section we show that branching bisimilarity is highly undecidable on PDA$^{\epsilon+}$ and on nPDA. The branching bisimilarity introduced in this paper requires that two bisimilar processes must agree on the states when they terminate (the third item of Definition 1). As we will see later, this is not a substantial requirement from the point of view of decidability. Let us first recall the definition of the van-Glabbeek-Weijland branching bisimulation [35] .

Definition 20. *A binary relation $\mathcal{R}$ on* PDA *processes is a* van-Glabbeek-Weijland branching simulation *if the following statements are valid whenever $P\mathcal{R}Q$:*

1. *If $P \xrightarrow{a} P'$ then there are some Q', Q'' such that $Q \Longrightarrow Q'' \xrightarrow{a} Q'$ and $P\mathcal{R}Q''$ and $P'\mathcal{R}Q'$.*
2. *If $P \xrightarrow{\tau} P'$ then either $Q \Longrightarrow Q'$ and $P\mathcal{R}Q'$ and $P'\mathcal{R}Q'$ for some Q' or $Q \Longrightarrow Q'' \xrightarrow{\tau} Q'$ and $P\mathcal{R}Q''$ and $P'\mathcal{R}Q'$ for some Q', Q''.*

The relation $\mathcal{R}$ is a van-Glabbeek-Weijland branching bisimulation *if both $\mathcal{R}$ and $\mathcal{R}^{-1} = \{(y,x) \mid (x,y) \in \mathcal{R}\}$ are van-Glabbeek-Weijland branching simulations. The* van-Glabbeek-Weijland branching bisimilarity $\approxeq$ *is the largest branching bisimulation on* PDA *processes. The relation $\approxeq$ is an equivalence. Compared with $\simeq$, the relation $\approxeq$ dose not distinguish processes like $p\epsilon$ and $q\epsilon$ where $p \neq q$. Consequently $\approxeq$ is not a congruence. Note that Computation Lemma is also valid for $\approxeq$. We denote by $\approxeq_{\mathrm{PDA}^{\epsilon+}}$, $\approxeq_{\mathrm{nPDA}}$ and $\approxeq_{\mathrm{nPDA}^{\epsilon+}}$ the largest van-Glabbeek-Weijland branching bisimilarity on* $\mathrm{PDA}^{\epsilon+}$, nPDA *and* $\mathrm{nPDA}^{\epsilon+}$*processes.*

The following proposition shows that $\approxeq$ and $\simeq$ on PDA are equivalent under many-one reduction.

Proposition 21. *Given a* PDA *system $\mathcal{P} = (\mathcal{Q}, \mathcal{V}, \mathcal{L}, \mathcal{R})$ and two processes P and Q, we can construct in polynomial time a new* PDA *system $\mathcal{P}_i = (\mathcal{Q}_i, \mathcal{V}_i, \mathcal{L}_i, \mathcal{R}_i)$ along with a pair of processes P_i, Q_i for $1 \leq i \leq 4$ such that the following conditions hold.*

1. *If $\mathcal{P}$ is a* $\mathrm{PDA}^{\epsilon+}$ *system, then both $\mathcal{P}_1$ and $\mathcal{P}_2$ are* $\mathrm{PDA}^{\epsilon+}$ *systems.*
2. *$P \approxeq Q$ iff $P_1 \simeq Q_1$; and $P \simeq Q$ iff $P_2 \approxeq Q_2$.*
3. *If $\mathcal{P}$ is a* nPDA *(resp.* $\mathrm{nPDA}^{\epsilon+}$*) system, then both $\mathcal{P}_3$ and $\mathcal{P}_4$ are* nPDA *(resp.* $\mathrm{nPDA}^{\epsilon+}$*) systems.*
4. *$P \approxeq Q$ iff $P_3 \simeq Q_3$; and $P \simeq Q$ iff $P_4 \approxeq Q_4$.*

Proof. For $1 \leq i \leq 4$, the PDA system $\mathcal{P}_i = (\mathcal{Q}_i, \mathcal{V}_i, \mathcal{L}_i, \mathcal{R}_i)$ and the processes P_i and Q_i are defined as follows.

1. $\mathcal{Q}_1 = \mathcal{Q}$, $\mathcal{V}_1 = \mathcal{V} \uplus \{Z_1\}$, $\mathcal{L}_1 = \mathcal{L}$ and $\mathcal{R}_1 = \mathcal{R} \uplus \mathcal{R}'_1$, where $\mathcal{R}'_1$ contains a rule $pZ_1 \xrightarrow{\tau} pZ_1$ for each $p \in \mathcal{Q}$. $P_1 = PZ_1$ and $Q_1 = QZ_1$.
2. $\mathcal{Q}_2 = \mathcal{Q}$, $\mathcal{V}_2 = \mathcal{V} \uplus \{Z_2\}$, $\mathcal{L}_2 = \mathcal{L} \uplus \{d_1, d_2, \ldots, d_{|\mathcal{Q}|}\}$ and $\mathcal{R}_2 = \mathcal{R} \uplus \mathcal{R}'_2$, where $\mathcal{R}'_2$ contains a rule $p_iZ_2 \xrightarrow{d_i} p_iZ_2$ for each $p_i \in \mathcal{Q}$. $P_2 = PZ_2$ and $Q_2 = QZ_2$.
3. $\mathcal{Q}_3 = \mathcal{Q}$, $\mathcal{V}_3 = \mathcal{V} \uplus \{Z_3\}$, $\mathcal{L}_3 = \mathcal{L} \uplus \{d\}$ and $\mathcal{R}_3 = \mathcal{R} \uplus \mathcal{R}'_3$. Let p_1 be a specific state in $\mathcal{Q}$. $\mathcal{R}'_3$ contains a rule $pZ_1 \xrightarrow{d} p_1$ for each $p \in \mathcal{Q}$. $P_3 = PZ_3$ and $Q_3 = QZ_3$.
4. $\mathcal{Q}_4 = \mathcal{Q}$, $\mathcal{V}_4 = \mathcal{V} \uplus \{Z_4\}$, $\mathcal{L}_4 = \mathcal{L} \uplus \{d_1, d_2, \ldots, d_{|\mathcal{Q}|}\}$ and $\mathcal{R}_4 = \mathcal{R} \uplus \mathcal{R}'_4$, where $\mathcal{R}'_4$ contains a rule $p_iZ_4 \xrightarrow{d_i} p_i$ for each $p_i \in \mathcal{Q}$. $P_4 = PZ_4$ and $Q_4 = QZ_4$.

It is easy to verify that the above systems and processes satisfy the requirements.

Corollary 22. *The relation $\approxeq_{\mathrm{nPDA}^{\epsilon+}}$ is decidable.*

By Proposition 21 in order to show the high undecidability of $\simeq_{\mathrm{PDA}^{\epsilon+}}$ and $\simeq_{\mathrm{nPDA}}$, it is sufficient to show the high undecidability of $\approxeq_{\mathrm{PDA}^{\epsilon+}}$ and $\approxeq_{\mathrm{nPDA}}$. As a result we will focus on $\approxeq_{\mathrm{PDA}^{\epsilon+}}$ and $\approxeq_{\mathrm{nPDA}}$ in the rest of this section. We prefer to working on the van-Glabbeek-Weijland branching bisimilarity for its clean game characterization. We will first introduce this game characterization along with a technique we called *semantic forcing*. We then construct a reduction from a Σ^1_1-complete problem to $\approxeq_{\mathrm{PDA}^{\epsilon+}}$. We also show the reduction can be adapted for $\approxeq_{\mathrm{nPDA}}$. For any $\mathrm{PDA}^{\epsilon+}$ processes P and Q, $P \approxeq Q$ if and only if there exists a set of pairs that contains (P, Q) and satisfies the first order arithmetic definable conditions prescribed in Definition 20. Thus the relation $\approxeq_{\mathrm{PDA}^{\epsilon+}}$ is in Σ^1_1. The relation $\approxeq_{\mathrm{nPDA}}$ is also in Σ^1_1 for the same reason. The main result of the section is stated as follows.

Theorem 23. *The relations $\approxeq_{\mathrm{PDA}^{\epsilon+}}$, $\simeq_{\mathrm{PDA}^{\epsilon+}}$, $\approxeq_{\mathrm{nPDA}}$ and $\simeq_{\mathrm{nPDA}}$ are Σ^1_1-complete.*

9.1. Branching Bisimulation Game and Semantic Forcing

A *branching bisimulation game* [31, 18] for a pair of processes (P_0, P_1), called a *configuration*, is played between Attacker and Defender in rounds. Each round has 3 steps: Attacker chooses a move; Defender then responds to match Attacker's move; and last Attacker set the configuration of the next round according to Defender's response. A round of branching bisimulation game is defined as follows, assuming (P_0, P_1) is the current configuration.

1. Attacker picks up $i \in \{0, 1\}$, ℓ, and P'_i to play $P_i \xrightarrow{\ell} P'_i$.
2. Defender responds with $P_{1-i} \Longrightarrow P''_{1-i} \xrightarrow{\ell} P'_{1-i}$ for some P''_{1-i} and and P'_{1-i}. Defender can also play an empty response when $\ell = \tau$ and we stipulate $P'_{1-i} = P_{1-i}$ if Defender plays an empty response.
3. If Defender plays an empty response, then Attacker set (P'_i, P'_{1-i}) as the configuration of the next round; otherwise Attacker chooses either (P'_i, P'_{1-i}) or (P_i, P''_{1-i}) as the configuration of the next round.

Attacker wins a branching bisimulation game if Defender gets stuck in the game. Defender wins a branching bisimulation game if Attacker cannot win the game. Attacker/Defender has a winning strategy if Attacker/Defender can win no matter how the other one plays. The following result is standard.

Lemma 24. *$P \approxeq Q$ if and only if Defender has a winning strategy in the branching bisimulation game of (P, Q).*

It's critical to the correctness of branching bisimulation game that Attacker is able to challenge the pair (P'_i, P''_{1-i}) when necessary. But this introduces an extra burden when one design or justify a branching bisimulation game. To deal with this we use a technique called *semantic forcing* which was first used implicitly in [36]. The idea is that if Defender response with a transition sequence $P_{1-i} \Longrightarrow P''_{1-i} \xrightarrow{\ell} P'_{1-i}$ and it holds that $P_{1-i} \approxeq P''_{1-i}$ then it should be safe to disable the Attacker's option to challenge the pair (P_i, P''_{1-i}). More specifically, we assume that there is a predefined set of configurations $\mathcal{E}$ such that $\mathcal{E} \subseteq \approxeq$ and we refine the third step by the following one.

3. The next round configuration is (P'_i, P'_{1-i}) automatically if either $P_{1-i} \approxeq P''_{1-i}$ or Defender plays an empty response. Otherwise Attacker chooses either (P'_i, P'_{1-i}) or (P_i, P''_{1-i}) as the next round configuration.

A game defined in this way is referred to as a branching bisimulation game with semantic forcing of $\mathcal{E}$. The effectiveness of semantic forcing is justified by the following lemma.

Lemma 25. *$P \approxeq Q$ if and only if there is some $\mathcal{E} \subseteq \approxeq$ such that Defender has a winning strategy in the branching bisimulation game of (P, Q) with semantic forcing of $\mathcal{E}$.*

Proof. By Lemma 24, $P \approxeq Q$ implies Defender has a winning strategy in the branching bisimulation game with semantic forcing of $\emptyset$. We now prove that if Defender has a winning strategy in the branching bisimulation game of (P, Q) with semantic forcing of some $\mathcal{E} \subseteq \approxeq$, then $P \approxeq Q$. It is sufficient to show that the relation $\approxeq; \mathcal{B}; \approxeq$ is a branching bisimulation, where $\mathcal{B}$ is the set of configurations that Defender has a winning strategy in a branching bisimulation game with semantic forcing $\mathcal{E}$. Suppose $P \approxeq P_0\mathcal{B}Q_0 \approxeq Q$ and $P \xrightarrow{\ell} P'$. There are three cases to consider.

1. $\ell = \tau$ and $P' \approxeq P_0$. We have $P' \approxeq; \mathcal{B}; \approxeq Q$.
2. $\ell \neq \tau$ and there are P''_0 and P'_0 such that $P_0 \Longrightarrow P''_0 \xrightarrow{\ell} P'_0$ with $P \approxeq P''_0$ and $P' \approxeq P'_0$. We show that $Q \Longrightarrow Q_1 \xrightarrow{\ell} Q_2$ for some Q_1 and Q_2 with $P \approxeq P''_0\mathcal{B}; \approxeq Q_1$ and $P' \approxeq P'_0\mathcal{B}; Q_2$. As $(P_0, Q_0) \in \mathcal{B}$ and $P_0 \Longrightarrow P''_0$, by the definition of branching bisimulation game there is some Q''_0 such that $Q \Longrightarrow Q''_0$ and $(P''_0, Q''_0) \in \mathcal{B}$. It follows that $Q \Longrightarrow Q''$ and $Q''_0 \approxeq Q''$. Now consider Defender's winning strategy in the case that Attacker plays $P''_0 \xrightarrow{\ell} P'_0$ at configuration (P''_0, Q''_0). We have three subcases.
   (a) Defender plays $Q''_0 \xrightarrow{\ell} Q'_0$ for some Q'_0. We have $(P'_0, Q'_0) \in \mathcal{B}$. The transition $Q''_0 \xrightarrow{\ell} Q'_0$ must be matched by Q'' via $Q'' \Longrightarrow Q_1 \xrightarrow{\ell} Q_2$ for some Q_1, Q_2. And we have $Q''_0 \approxeq Q_1$ and $Q'_0 \approxeq Q_2$.
   (b) Defender plays $Q''_0 \Longrightarrow Q^1_0 \xrightarrow{\ell} Q^2_0$ and it holds that $(Q''_0, Q^1_0) \in \mathcal{E}$. In this case we have $(P'_0, Q^2_0) \in \mathcal{B}$ and $Q''_0 \approxeq Q^1_0$. By transitivity of $\approxeq$ we have $Q^1_0 \approxeq Q''$. Then there are Q_1 and Q_2 such that $Q'' \Longrightarrow Q_1 \xrightarrow{\ell} Q_2$, $Q^1_0 \approxeq Q_1$ and $Q^2_0 \approxeq Q_2$.

(c) Defender plays $Q_0'' \Longrightarrow Q_0^1 \xrightarrow{\ell} Q_0^2$. We must have $(P_0'', Q_0^1) \in \mathcal{B}$ and $(P_0', Q_0^2) \in \mathcal{B}$. By definition of branching bisimulation we have Q_1 and Q_2 such that $Q'' \Longrightarrow Q_1 \xrightarrow{\ell} Q_2$, $Q_0^1 \simeq Q_1$ and $Q_0^2 \simeq Q_2$.

3. $\ell = \tau$ and there are P_0'' and P_0' such that $P_0 \Longrightarrow P_0'' \xrightarrow{\ell} P_0'$ with $P \simeq P_0''$ and $P' \simeq P_0'$. The argument is similar.

By the proof of Lemma 25, any relation $\mathcal{E}$ such that $\mathcal{E} \subseteq \simeq$ is effective for semantic forcing. This implies that as long as we can demonstrate $P_{1-i} \simeq P_{1-i}''$ in Defender's response sequence $P_{1-i} \Longrightarrow P_{1-i}'' \xrightarrow{\ell} P_{1-i}'$, it is safe to apply semantic forcing. In the sequel we shall use semantic forcing without referring to a specific $\mathcal{E}$.

9.2. The Reduction

A *nondeterministic Minsky counter machine* $\mathcal{M}$ with two counters c_1, c_2 is a program "$1 : I_1;\ 2 : I_2;\ \ldots;\ n{-}1 : I_{n-1};\ n :$ `halt`", where for each $i \in \{1, \ldots, n-1\}$ the instruction I_i is of one of the three types. Assuming $1 \le j, k \le n$ and $e \in \{1, 2\}$, the first type is an increment instruction, which is of the from "$c_e := c_e{+}1$ `and then goto` j"; the second type is a decrement instruction, which is of the form "`if` $c_e = 0$ `then goto` j, `otherwise` $c_e := c_e{-}1$ `and then goto` k"; the third type is a nondeterministic jump instruction, which is of the form "`goto` j `or goto` k". The problem rec-NMCM asks if $\mathcal{M}$ has an infinite computation on $(c_1, c_2) = (0, 0)$ such that I_1 is executed infinitely often. We shall construct a reduction from rec-NMCM to $\simeq_{\mathrm{PDA}^{\epsilon+}}$ and use the following fact [9].

Proposition 26. rec-NMCM *is* Σ_1^1*-complete.*

Following [18] we first transform a nondeterministic Minsky counter machine $\mathcal{M}$ with two counters c_1, c_2 into a machine $\mathcal{M}'$ with three counters c_1, c_2, c_3. The machine $\mathcal{M}'$ makes use of a new nondeterministic instruction of the form: "$c_3 := *$ `and then goto` j". The effect of this instruction is to set c_3 by a nondeterministically chosen number and then go to I_j. The transformation from $\mathcal{M}$ to $\mathcal{M}'$ is done in two steps. We first replace every instruction "$i : I_i$" of $\mathcal{M}$ by two instructions in $\mathcal{M}'$, with respective labels $2i{-}1$ and $2i$. The instruction "$1 : I_1$" is replaced by "$1 : c_3 := *$ `and then goto` 2; $2 : I_1$"; and for $i \in \{2, \ldots, n\}$, the instruction "$i : I_i$" is replaced by "$2i-1$: `if` $c_3 = 0$ `then goto` $2n$, `otherwise` $c_3 := c_3{-}1$ `and then goto` $2i$; $2i : I_i$". We then replace every occurrence of "`goto` j" by "`goto` $2j{-}1$" inside each I_i, where $i \in \{1, \ldots, n\}$. Let $\mathcal{M}'$ be the resulting program "$1 : I_1'; 2 : I_2'; \ldots; (2n{-}1) : I_{2n-1}'; 2n :$ `halt`". It is easy to see that $\mathcal{M}'$ has an infinite computation on $(c_1, c_2, c_3) = (0, 0, 0)$ if and only if $\mathcal{M}$ has an infinite computation on $(c_1, c_2) = (0, 0)$ that executes the instruction I_1 infinitely often.

Our goal is to construct a PDA$^{\epsilon+}$ system $\mathcal{P} = (\mathcal{Q}, \mathcal{V}, \mathcal{L}, \mathcal{R})$ in which we can define two processes P and Q that render true the following equivalence.

$$P \simeq Q \iff \mathcal{M}' \text{ has an infinite computation.} \tag{14}$$

In order to get (14), we will use the game interpretation of branching bisimulation. The basic idea of the construction resembles the one used in [18, 19]. The computation of $\mathcal{M}'$ is encoded into the branching bisimulation game $\mathcal{G}$ of (P, Q). A computation step of $\mathcal{M}'$ is emulated by a finite number of rounds of branching bisimulation games. $\mathcal{M}'$ has an infinite run if and only if Defender has a winning strategy in $\mathcal{G}$. To smooth the analysis of $\mathcal{G}$, semantic forcing is employed in our construction. The key elements of the PDA$^{\epsilon+}$ system $\mathcal{P} = (\mathcal{Q}, \mathcal{V}, \mathcal{L}, \mathcal{R})$ are defined as follows.

- $\mathcal{Q} = \mathcal{Q}_I \uplus \mathcal{Q}_T \uplus \mathcal{Q}_U \uplus \mathcal{Q}_F$. $\mathcal{Q}_I$ is used to encode the labels of instructions. For each instruction "$i : I_i'$" from $\mathcal{M}'$, we introduce a pair of states p_i, q_i. We have $\mathcal{Q}_I = \{p_i, q_i \mid 1 \le i \le 2n\}$. $\mathcal{Q}_T$ is used to implement counter test in branching bisimulation games and is defined to be the set $\{t, t', t_e, z_e, z_e', \bar{z}_e, \bar{z}_e' \mid 1 \le e \le 3\}$. $\mathcal{Q}_U$ and $\mathcal{Q}_F$ are used to implement counter update operation and the control flow of $\mathcal{M}'$. The definitions of these two sets will be clear later.

- $\mathcal{V} = \{C_1, C_2, C_3, \bot, X, X_1, X_2, X_3\}$. The stack symbols C_1, C_2 and C_3 are introduced for the three counters c_1, c_2 and c_3 respectively. The stack symbol "$\bot$" is introduced to indicate stack bottom. The rule set $\mathcal{R}$ defined later will guarantee $p\gamma\bot\sigma \simeq p\gamma\bot$ for all $p \in \mathcal{Q}$ and $\gamma, \sigma \in \mathcal{V}^*$.

- $\mathcal{L} = \{a, b, c, c_1, c_2, c_3, f, f'\}$.

The two PDA$^{\epsilon+}$ processes P and Q for (14) are defined by

$$P = p_1 X\bot, \qquad Q = q_1 X\bot. \tag{15}$$

Notation. In the rest of this section we write $\mathbf{1}^1$, $\mathbf{1}^2$ and $\mathbf{1}^3$ for $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$ respectively; and we use $[\![(n_1,n_2,n_3)]\!]$ to represent a stack symbol sequence of the form $C_1^{n_1}C_2^{n_2}C_2^{n_3}\bot\sigma$, where $n_1, n_2, n_3 \geq 0$ and $\sigma \in \mathcal{V}^*$. A configuration of $\mathcal{M}'$ is represented by a tuple of the form (i, n_1, n_2, n_3), where i is the label of the instruction and n_1, n_2 and n_3 are the values of the counters c_1, c_2 and c_3 respectively. The construction will keep the following correspondence between the configurations of $\mathcal{M}'$ and $\mathcal{G}$. If $\mathcal{M}'$ reaches a configuration (i, n_1, n_2, n_3), then $\mathcal{G}$ reaches a configuration of the form $(p_i X[\![(n_1,n_2,n_3)]\!], q_i X[\![(n_1,n_2,n_3)]\!])$. Clearly (P, Q) corresponds to $(1,0,0,0)$, which is the initial configuration of $\mathcal{M}'$.

In the rest of this section we will complete the definition of $\mathcal{P}$ and explain its working mechanism. We first introduce rules for Q_T and show how to test the value of a counter. We then design branching bisimulation games to mimic the counter update operations. We finish our construction by assembling the counter test and counter update construction according to the control flow of $\mathcal{M}'$.

(I) Counter Test. We need to carry out equality test, successor test, and zero test on the counters. The rules to implement these operations are given in Figure 8, assuming $i, e \in \{1,2,3\}$. The correctness requirement of these rules are summarized in the following lemma. Its routine proof is omitted.

Lemma 27. *Let $e \in \{1,2,3\}$. The following statements are valid.*

1. $t[\![(n_1,n_2,n_3)]\!] \approxeq t[\![(m_1,m_2,m_3)]\!]$ *if and only if* $(n_1,n_2,n_3) = (m_1,m_2,m_3)$.
2. $t'[\![(n_1,n_2,n_3)]\!] \approxeq t'[\![(m_1,m_2,m_3)]\!]$ *if and only if* $(n_1,n_2) = (m_1,m_2)$.
3. $t[\![(n_1,n_2,n_3)]\!] \approxeq t_e[\![(m_1,m_2,m_3)]\!]$ *if and only if* $(n_1,n_2,n_3) + \mathbf{1}^e = (m_1,m_2,m_3)$.
4. $z_e[\![(n_1,n_2,n_3)]\!] \approxeq z'_e[\![(m_1,m_2,m_3)]\!]$ *if and only if* $(n_1,n_2,n_3) = (m_1,m_2,m_3)$ *and* $n_e = 0$.
5. $\bar{z}_e[\![(n_1,n_2,n_3)]\!] \approxeq \bar{z}'_e[\![(m_1,m_2,m_3)]\!]$ *if and only if* $(n_1,n_2,n_3) = (m_1,m_2,m_3)$ *and* $n_e > 0$.
6. $p\gamma\bot\sigma \approxeq p\gamma\bot$ *for all* $p \in Q$ *and* $\gamma, \sigma \in \mathcal{V}^*$.

1. $tC_1 \xrightarrow{c_1} t$,	$tC_2 \xrightarrow{c_2} t$,	$tC_3 \xrightarrow{c_3} t$,	$t\bot \xrightarrow{b} t\bot$;
2. $t'C_1 \xrightarrow{c_1} t'$,	$t'C_2 \xrightarrow{c_2} t'$,	$t'C_3 \xrightarrow{b} t\bot$,	$t'\bot \xrightarrow{b} t\bot$;
3. $t_eC_i \xrightarrow{c_i} t_e$ $(i < e)$,	$t_eC_i \xrightarrow{c_e} tC_e$ $(i \geq e)$,	$t_e\bot \xrightarrow{c_e} t\bot$;	
4. $z_eC_i \xrightarrow{c_i} z_e$ $(i < e)$,	$z_eC_i \xrightarrow{c_i} t$ $(i > e)$,	$z_eC_e \xrightarrow{f} t$	$z_e\bot \xrightarrow{b} t\bot$;
5. $z'_eC_i \xrightarrow{c_i} z'_e$ $(i < e)$,	$z'_eC_i \xrightarrow{c_i} t$ $(i > e)$,	$z'_eC_e \xrightarrow{f'} t$	$z'_e\bot \xrightarrow{b} t\bot$;
6. $\bar{z}_eC_i \xrightarrow{c_i} z_e$ $(i < e)$,	$\bar{z}_eC_i \xrightarrow{f} tC_i$ $(i > e)$,	$\bar{z}_eC_e \xrightarrow{c_e} t$	$\bar{z}_e\bot \xrightarrow{f} t\bot$;
7. $\bar{z}'_eC_i \xrightarrow{c_i} z'_e$ $(i < e)$,	$\bar{z}'_eC_i \xrightarrow{f'} tC_i$ $(i > e)$,	$\bar{z}'_eC_e \xrightarrow{c_e} t$	$\bar{z}'_e\bot \xrightarrow{f'} t\bot$;
8. For each $p \in Q \backslash Q_T$ we have $p\bot \xrightarrow{b} t\bot$.			

Figure 8: Rules for Counter Test

(II) Counter Update. There are three basic operations on counters, the increment operation, the decrement operation and the nondeterministic assignment operation. We encode each such operation into a branching bisimulation game. Let O be the set of triples defined by

$$O = \{(e,+,j), (e,-,j), (3,*,j) \mid 1 \leq e \leq 3 \wedge 1 \leq j \leq 2n\}.$$

For each triple $\tilde{o} \in O$, we add the elements of $\mathcal{U}(\tilde{o})$ to Q_U, where

$$\mathcal{U}(\tilde{o}) = \{u(\tilde{o}), u'(\tilde{o}), u_1(\tilde{o}), u'_1(\tilde{o}), u_2(\tilde{o}), u'_2(\tilde{o}), u_3(\tilde{o}), u'_3(\tilde{o}), g(\tilde{o}), g'(\tilde{o})\}.$$

Rules for $\mathcal{U}(\tilde{o})$ are given in Figure 9. In Figure 9, s_1, s_2, s_3 and s_4 are four placeholders. For each triple $\tilde{o} = (e, o, j)$, they are defined as follows.

(L1).	$u(\tilde{o})X \xrightarrow{a} u_1(\tilde{o})X,$	$u(\tilde{o})X \xrightarrow{\tau} g'(\tilde{o})X_3\bot;$	(R1).	$u'_2(\tilde{o})X \xrightarrow{a} u'_3(\tilde{o})X,$	$u'_2(\tilde{o})X \xrightarrow{\tau} g(\tilde{o})X\bot;$
		$u'(\tilde{o})X \xrightarrow{\tau} g'(\tilde{o})X_3\bot;$			$u_2(\tilde{o})X \xrightarrow{\tau} g(\tilde{o})X_3\bot;$
(L2).	$g'(\tilde{o})X_3 \xrightarrow{\tau} g'(\tilde{o})X_3C_3,$	$g'(\tilde{o})X_3 \xrightarrow{\tau} g'(\tilde{o})X_2;$	(R2).	$g(\tilde{o})X_3 \xrightarrow{\tau} g(\tilde{o})X_3C_3,$	$g(\tilde{o})X_3 \xrightarrow{\tau} g(\tilde{o})X_2;$
	$g'(\tilde{o})X_2 \xrightarrow{\tau} g'(\tilde{o})X_2C_2,$	$g'(\tilde{o})X_2 \xrightarrow{\tau} g'(\tilde{o})X_1;$		$g(\tilde{o})X_2 \xrightarrow{\tau} g(\tilde{o})X_2C_2,$	$g(\tilde{o})X_2 \xrightarrow{\tau} g(\tilde{o})X_1;$
	$g'(\tilde{o})X_1 \xrightarrow{\tau} g'(\tilde{o})X_1C_1,$	$g'(\tilde{o})X_1 \xrightarrow{\tau} g'(\tilde{o})X_3\bot,$		$g(\tilde{o})X_1 \xrightarrow{\tau} g(\tilde{o})X_1C_1,$	$g(\tilde{o})X_1 \xrightarrow{\tau} g(\tilde{o})X_3\bot;$
		$g'(\tilde{o})X_1 \xrightarrow{a} u'_1(\tilde{o})X;$			$g(\tilde{o})X_1 \xrightarrow{a} u_3(\tilde{o})X;$
(L3).	$u_1(\tilde{o})X \xrightarrow{a} u_2(\tilde{o})X,$	$u_1(\tilde{o})X \xrightarrow{c} \underline{s_1};$	(R3).	$u_3(\tilde{o})X \xrightarrow{a} \underline{s_3}X,$	$u_3(\tilde{o}) \xrightarrow{c} t;$
	$u'_1(\tilde{o})X \xrightarrow{a} u'_2(\tilde{o})X,$	$u'_1(\tilde{o})X \xrightarrow{c} \underline{s_2};$		$u_3(\tilde{o})X \xrightarrow{a} \underline{s_4}X,$	$u'_3(\tilde{o}) \xrightarrow{c} t;$

Figure 9: Rule Template for Counter Update

- If $o =$"+", then $s_1 = t_e$ and $s_2 = t$; if $o =$"−", then $s_1 = t$ and $s_2 = t_e$; if $o =$"∗", then $s_1 = s_2 = t'$.
- $s_3 = p_j$ and $s_4 = q_j$.

The correctness of the counter update operation is justified by the next lemma.

Lemma 28. *In the branching bisimulation game of* $(u(\tilde{o})X[\![(m_1, m_2, m_3)]\!], u'(\tilde{o})X[\![(m_1, m_2, m_3)]\!])$

1. *if* $\tilde{o} = (e, +, j)$*, then Defender, respectively Attacker, has a strategy to win or at least push the game to a configuration of the form* $(p_jX[\![(m_1, m_2, m_3) + \mathbf{1}^e]\!], q_jX[\![(m_1, m_2, m_3) + \mathbf{1}^e]\!])$*;*
2. *if* $\tilde{o} = (e, -, j)$ *and* $m_e > 0$*, then Defender, respectively Attacker, has a strategy to win or at least push the game to a configuration of the form* $(p_jX[\![(m_1, m_2, m_3) - \mathbf{1}^e]\!], q_jX[\![(m_1, m_2, m_3) - \mathbf{1}^e]\!])$*;*
3. *if* $\tilde{o} = (3, *, j)$ *and* $m \geq 0$*, then Defender has a strategy to win or at least push the game to a configuration of the form* $(p_jX[\![(m_1, m_2, m)]\!], q_jX[\![(m_1, m_2, m)]\!])$*.*

Proof. We prove the first statement. The proof for the other two is similar. In what follows we describe Defender and Attacker's step-by-step optimal strategy in the branching bisimulation game of $(u(\tilde{o})X\gamma, u'(\tilde{o})X\gamma')$, where $\tilde{o} = (e, +, j)$, $\gamma = C_1^{m_1}C_2^{m_2}C_3^{m_3}\bot\sigma$, $\gamma' = C_1^{m_1}C_2^{m_2}C_3^{m_3}\bot\sigma'$, and $\sigma, \sigma' \in \mathcal{V}^*$.

1. By rules of (L1) if Attacker chooses to perform an τ action, then Defender responds with an τ transition and the game reaches $(g'(\tilde{o})X_3\bot\gamma, g'(\tilde{o})X_3\bot\gamma')$. By Lemma 27, Defender has a winning strategy afterward.
2. Attacker's optimal choice is to play $u(\tilde{o})X\gamma \xrightarrow{a} u_1(\tilde{o})X\gamma$. By rules of (L2), Defender can respond with
$$u'(\tilde{o})X\gamma' \xrightarrow{\tau} g'(\tilde{o})X_3\bot\gamma' \Longrightarrow g'(\tilde{o})X_1C_1^{n_1}C_2^{n_2}C_3^{n_3}\bot\gamma' \xrightarrow{a} u'_1(\tilde{o})XC_1^{n_1}C_2^{n_2}C_3^{n_3}\bot\gamma' \qquad (16)$$
for some n_1, n_2, n_3. Defender has to choose n_1, n_2, n_3 such that $(n_1, n_2, n_3) = (m_1, m_2, m_3) + \mathbf{1}^e$. The reason is that Attacker can continue the game from $(u_1(\tilde{o})X\gamma, u'_1(\tilde{o})XC_1^{n_1}C_2^{n_2}C_3^{n_3}\bot\gamma')$ and then initiate counter test by rules of (L3) via an action c. By Lemma 27, $t_e\gamma \simeq tC_1^{n_1}C^{n_2}C^{n_3}\bot\gamma'$ iff $(n_1, n_2, n_3) = (m_1, m_2, m_3) + \mathbf{1}^e$. Observe that $u'(\tilde{o})X\gamma' \simeq g'(\tilde{o})X_3\bot\gamma'$ as the τ transition is the only action of $u'(\tilde{o})X\gamma'$. By Lemma 27, $g'(\tilde{o})XC_1^{n_1}C_2^{n_2}C_3^{n_3}\bot\gamma' \xrightarrow{\tau} g'(\tilde{o})X_3\bot C_1^{n_1}C_2^{n_2}C_3^{n_3}\bot\gamma' \simeq g'(\tilde{o})X_3\bot\gamma'$. By (16) and Computation Lemma we have $g'(\tilde{o})X_3\bot\gamma' \simeq g'(\tilde{o})X_1C_1^{n_1}C_2^{n_2}C_3^{n_3}\bot\gamma'$. As a result $u'(\tilde{o})X\gamma' \simeq g'(\tilde{o})X_1C_1^{n_1}C_2^{n_2}C_3^{n_3}\bot\gamma'$. By semantic forcing, Attacker has to continue the game from $(u_1(\tilde{o})X\gamma, u'_1(\tilde{o})X\gamma_1)$, where $\gamma_1 = C_1^{n_1}C_2^{n_2}C_3^{n_2}\bot\gamma'$.
3. Attacker would not play an action c as $t_e\gamma \simeq t\gamma_1$. Attacker's optimal choice is to play an action a and the game continues from $(u_2(\tilde{o})X\gamma, u'_2(\tilde{o})X\gamma_1)$.
4. The argument of the rest part is symmetric. The optimal strategy for Attacker and Defender is as follows. By rules of (R1) Attacker first plays $u'_2(\tilde{o})X\gamma_1 \xrightarrow{a} u'_3(\tilde{o})X\gamma_1$, then by rules of (R2) Defender has to respond with
$$u_2(\tilde{o})X\gamma \xrightarrow{\tau} g(\tilde{o})X_3\bot\gamma \Longrightarrow g(\tilde{o})X_1C_1^{n_1}C_2^{n_2}C_3^{n_3}\bot\gamma \xrightarrow{a} u_3(\tilde{o})XC_1^{n_1}C_2^{n_2}C_3^{n_3}\bot\gamma.$$

As $u_2(\tilde{o})X \approxeq g(\tilde{o})X_1C_1^{n_1}C_2^{n_2}C_3^{n_3}\bot\gamma$, by semantic forcing again the game continues from $(u_3(\tilde{o})X\gamma_2, u_3'(\tilde{o})X\gamma_1)$, where $\gamma_2 = C_1^{n_1}C_2^{n_2}C_3^{n_3}\bot\gamma$. By Lemma 27, $t\gamma_1 \approxeq t\gamma_2$. By rules of (R3) Attacker better plays an action a and the game reaches $(p_jX\gamma_2, q_jX\gamma_1)$. Note that γ_1, γ_2 are of the form $[\![(m_1, m_2, m_3) + \mathbf{1}^e]\!]$.

We are done. □

(III) Control Flow. We now encode the control flow of $\mathcal{M}'$ by the rules of the branching bisimulation game. We will introduce a number of rules for each instruction in $\mathcal{M}'$.

1. For an instruction of the form "$i : c_e := c_e + 1$ `and then goto` j" the following rules are added to $\mathcal{R}$.

$$p_iX \xrightarrow{a} u(e,+,j)X, \quad q_iX \xrightarrow{a} u'(e,+,j)X.$$

   These rules implement the counter update operation that increases the counter c_e by 1.
2. For an instruction of the form "$i : c_3 := *$ `and then goto` j" the following two rules are added to $\mathcal{R}$.

$$p_iX \xrightarrow{a} u(3,*,j)X, \quad q_iX \xrightarrow{a} u'(3,*,j)X.$$

   These rules implement the counter update operation that resets c_3 with a nondeterministically chosen number.
3. For an instruction of the form "i : `goto` j `or goto` k", we add p_i^1, q_i^1 and q_i^2 to Q_F and the following rules to $\mathcal{R}$.

$$\begin{array}{lll} p_iX \xrightarrow{a} p_i^1X, & p_iX \xrightarrow{a} q_i^1X, & p_iX \xrightarrow{a} q_i^2X; \\ & q_iX \xrightarrow{a} q_i^1X, & q_iX \xrightarrow{a} q_i^2X; \end{array}$$

$$\begin{array}{lll} p_i^1X \xrightarrow{a} p_jX, & q_i^1X \xrightarrow{a} q_j'X, & q_i^2X \xrightarrow{a} p_jX; \\ p_i^1X \xrightarrow{a} p_kX, & q_i^1X \xrightarrow{a} p_kX, & q_i^2X \xrightarrow{a} q_k'X. \end{array}$$

   These rules embody precisely the idea of Defender's Forcing [18]. It is Defender who makes the choice. From configuration $(p_iX\gamma, q_iX\gamma)$, Defender can force the game to reach either the configuration $(p_jX\gamma, q_jX\gamma)$ or the configuration $(p_kX\gamma, q_kX\gamma)$.
4. For an instruction of the form

   "i : `if` $c_e = 0$ `then goto` j, `otherwise` $c_e = c_e - 1$ `and then goto` k"

   The states $p(e,0,j)$, $q(e,0,j)$, $p(e,1,j)$, $q(e,1,j)$, $v_1(e,0,j)$, $v_2(e,0,j)$, $v_3(e,0,j)$, $v_1(e,1,j)$, $v_2(e,1,j)$, $v_3(e,1,j)$ are added to Q_F. The following rules are added to $\mathcal{R}$.

$$\begin{array}{llll} \text{(C).} & p_iX \xrightarrow{a} p(e,0,j)X, & p_iX \xrightarrow{c} p(e,1,k)X; & \\ & q_iX \xrightarrow{a} q(e,0,j)X, & q_iX \xrightarrow{c} q(e,1,k)X; & \end{array}$$

$$\begin{array}{llll} \text{(F0).} & p(e,0,j)X \xrightarrow{a} v_1(e,0,j)X, & p(e,0,j)X \xrightarrow{a} v_2(e,0,j)X, & p(e,0,j)X \xrightarrow{a} v_3(e,0,j)X; \\ & & q(e,0,j)X \xrightarrow{a} v_2(e,0,j)X, & q(e,0,j)X \xrightarrow{a} v_3(e,0,j)X; \\ & v_1(e,0,j)X \xrightarrow{c} \overline{z}(e), & v_2(e,0,j)X \xrightarrow{c} \overline{z}'(e), & v_3(e,0,j)X \xrightarrow{c} \overline{z}(e); \\ & v_1(e,0,j)X \xrightarrow{a} p_jX, & v_2(e,0,j)X \xrightarrow{a} p_jX, & v_3(e,0,j)X \xrightarrow{a} q_jX; \end{array}$$

$$\begin{array}{llll} \text{(F1).} & p(e,1,j)X \xrightarrow{a} v_1(e,1,j)X, & p(e,1,j)X \xrightarrow{a} v_2(e,1,j)X, & p(e,1,j)X \xrightarrow{a} v_3(e,1,j)X; \\ & & q(e,1,j)X \xrightarrow{a} v_2(e,1,j)X, & q(e,1,j)X \xrightarrow{a} v_3(e,1,j)X; \\ & v_1(e,1,j)X \xrightarrow{c} z(e), & v_2(e,1,j)X \xrightarrow{c} z'(e), & v_3(e,1,j)X \xrightarrow{c} z(e); \\ & v_1(e,1,j)X \xrightarrow{a} u(e,-,j)X, & v_2(e,1,j)X \xrightarrow{a} u(e,-,k)X, & v_3(e,1,j)X \xrightarrow{a} u'(e,-,k)X; \end{array}$$

Suppose $\mathcal{G}$ reaches a configuration $(p_i X\gamma, q_i X\gamma')$, where γ and γ' are of the form $[\![(n_1, n_2, n_3)]\!]$. By rules of (C) Attacker claims either "$n_e = 0$" via an action a or "$n_e > 0$"via an action c. Defender then responds with the same action and $\mathcal{G}$ reaches $(p(e, 0, j)X\gamma, q(e, 0, j)X\gamma')$ or $(p(e, 1, j)X\gamma, q(e, 1, j)X\gamma')$ respectively. Rules of (F0) and rules of (F1) compose two Defender's Forcing gadget. Defender can use them to punish Attacker if he lied about n_e; or to mimic the run of $\mathcal{M}'$ if Attacker is honest about n_e. When $n_e > 0$, there are two cases. If Attacker lied that "$n_e = 0$" and $\mathcal{G}$ reaches $(p(e, 0, j)X\gamma, q(e, 0, j)X\gamma')$, then by rules of (F0) Defender can force the game to reach the configuration $(\bar{z}_e\gamma, \bar{z}'_e\gamma')$. By Lemma 27 and the fact that "$n_e > 0$", we have $\bar{z}_e\gamma \approxeq \bar{z}'_e\gamma'$. Defender wins afterward. If Attacker claim "$n_e > 0$" and $\mathcal{G}$ reaches $(p(e, 1, j)X\gamma, q(e, 1, j)X\gamma')$, then Defender shall not force the configuration $(z_e\gamma, z'_e\gamma')$ as by Lemma 27 we have $z_e\gamma \not\approxeq z'_e\gamma'$. By rules of (F1) Defender's optimal choice is to do a counter decrement operation by forcing the configuration $(u(e, -, j)X\gamma, u'(e, -, j)X\gamma')$. When $n_e = 0$ the argument is similar.

5. For "$2n$: `halt`", we add the following two rules to $\mathcal{R}$.

$$p_{2n}X \xrightarrow{f} p_{2n}\bot, \quad q_{2n}X \xrightarrow{f'} q_{2n}\bot.$$

So if $\mathcal{M}'$ halts, then $\mathcal{G}$ reaches $(p_{2n}X\gamma, q_{2n}X\gamma')$ for some γ, γ' and Attacker wins immediately.

With the help of Lemma 27 and Lemma 28, we can show the correctness of our reduciton.

Proposition 29. *$\mathcal{M}'$ has an infinite computation if and only if $p_1X\bot \approxeq q_1X\bot$.*

A consequence of Proposition 29 is that the relation $\approxeq_{\mathrm{PDA}^{\epsilon+}}$ is Σ^1_1-complete. The relation $\approxeq_{\mathrm{nPDA}}$ has been shown to be undecidable in [36]. We can show $\approxeq_{\mathrm{nPDA}}$ is also Σ^1_1-complete by a slight modification to the system $\mathcal{P} = (\mathcal{Q}, \mathcal{V}, \mathcal{L}, \mathcal{R})$ as follows.

(M1). Introduce a new state r and for each $q \in \mathcal{Q}$ and $Z \in \mathcal{V}$ add a new rule $pZ \xrightarrow{b} r$ to $\mathcal{R}$.

(M2). For each $Z \in \mathcal{V}$ add a rule $rZ \xrightarrow{\tau} r$ to $\mathcal{R}$.

(M3). Remove all rules of the from $p\bot \xrightarrow{b} t\bot$ from $\mathcal{R}$.

(M1) and (M2) make sure the new PDA system is normed. (M1) (M2) and (M3) together keep the equivalence $p\gamma\bot\sigma \approxeq p\gamma\bot$ intact for all $p \in \mathcal{Q}$ and $\gamma, \sigma \in \mathcal{V}^*$. Moreover one can verify that Lemma 27, Lemma 28 and Proposition 29 are still valid under this new setting. The details are omitted.

10. Conclusion

The results of the paper and the results of Jančar and Srba [18] are summarized in the table given below. The new decidability result is significant compared to the fact that the corresponding problem for the ϵ-pushing PDA remains in the analytic hierarchy.

	ϵ-Pushing **nPDA**	ϵ-Pushing **PDA**
$\simeq$	Decidable	Σ^1_1-Complete
$\approx$	Π^0_1-Complete	Σ^1_1-Complete

Stirling's proof of the decidability of the strong bisimilarity of nPDA has strong influence on the present work. We have attempted to prove the present result by using tableau system as is done in Stirling's work, see [6] for a report It turned out that due to the presence of the silent transitions, proof based on a tableau system is not easy to handle. The difficulty is two fold. Firstly in the presence of silent actions the k-bisimilarity, as introduced in the proof of Proposition 7, is very subtle. It is a powerful tool to establish negative results. It is however a little tricky to use it to construct bisimulations. The reason is that transitivity can easily fail if one is not careful about the definition of $\simeq_k$. If transitivity fails, the proof of the backward soundness of tableau rules suffers. Without backward soundness of the tableau rules, Stirling's proof cannot be repeated. Secondly an alternative would be to construct branching bisimulations from a tableau, bypassing the use of k-bisimilarity. This cannot be done by generalizing the similar idea for the strong bisimilarity in a simple minded way. Every goal appearing in a tableau is the root of a branching bisimulation. Branching bisimulation of a goal in the conclusion of a tableau rule and that of a goal in the premises have different structure. That makes composition of these bisimulations difficult to define. The way out of the problem is Lemma 9. Using this idea one soon realizes that it would be simpler to work directly with the bisimulation trees. In this paper we have developed decomposition approach to branching bisimulations that in our opinion is better suited to deal with the branching structure in the presence of silent transitions.

The bisimulation decomposition approach can be applied to study the branching bisimilarity of the ϵ-popping PDA. The finite branching property is obviously valid for this model. The definition of the bisimulation and that of recursive constant have to be modified in the ϵ-popping setting. We hope to come back to the issue in another occasion. That would complete the picture this paper sets out to reveal.

In addition to the relationship to the tableau approach, the technique used in this paper can also be seen as a generalization of the bisimulation base method [4]. In Caucal's approach every process has a prime decomposition such that two processes are equivalent if their prime decompositions are equivalent according to a set of axioms. For PDA processes rewriting of processes is insufficient. We have to take into account of the tree structures induced by states. The tree structure carries additional proof information that can be verified on-the-fly. It would be interesting to see if the additional information is helpful in deriving better complexity bounds for similar problems.

Jančar introduced the notion of first order grammar [14] and provided a quite different proof for the decidability of the strong bisimilarity of nPDA [16]. In the full paper he also outlined an idea of how to extend his proof to take care of silent transitions. The extended PDA model introduced in [6] is similar to the first order grammar of Jančar. It is fair to say that the first order grammar offers a generalization of PDA that appears just right.

Stirling proved that the language equivalence of DPDA is primitive recursive [28]. Benedikt, Goller, Kiefer and Murawski showed that the strong bisimilarity on nPDA is non-elementary [2]. More recently Jančar observed that the strong bisimilarity of first-order grammar is Ackermann-hard [15], a consequence of which is that the strong bisimilarity proved decidable by Sénizergues in [24] is Ackermann-hard. It would be interesting to look for tighter upper and lower bounds on the branching bisimilarity of nPDA$^{\epsilon+}$.

Acknowledgment

We thank the members of BASICS for their interest. We are grateful to Prof. Jančar for his insightful discussion. The support from NSFC (61472239, ANR 61261130589, 91318301) is gratefully acknowledged.

References

[1] J. Baeten. Branching bisimilarity is an equivalence indeed. *Information Processing Letters* 58:141–147, 1996.

[2] M. Benedikt, S. Göller, S. Kiefer, and A. Murawski. Bisimilarity of Pushdown Automata is Nonelementary. In *LICS'13*, 488–498, 2013.

[3] O. Burkart, D. Caucal, F. Göller, and B. Steffen. Verification on Infinite Structures. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, 545–623. North-Holland, 2001.

[4] D. Caucal. Graphes canoniques de graphes algébriques. *Informatique théorique et Applications*, 24:339–352, 1990.

[5] Y. Fu. Checking Equality and Regularity for Normed BPA with Silent Moves. *ICALP'13*, Lecture Notes in Computer Science 7966, 238–249, 2013.

[6] Y. Fu and Q. Yin. Dividing Line between Decidable PDA's and Undecidable Ones. arXive, `https://arxiv.org/abs/1404.7015`, 2014.

[7] S. Ginsburg and S. Greibach. Deterministic Context Free Languages. *Information and Control*, 9:620–648, 1966.

[8] J. Groote and H. Hüttel. Undecidable Equivalences for Basic Process Algebra. *Information and Computation*, 115:354–371, 1994.

[9] D. Harel. Effective Transformations on Infinite Trees, with Applications to High Undecidability, Dominoes, and Fairness. *J. ACM*, 33:224–248, 1986.

[10] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Publishing Company, 1979.

[11] H. Hüttel. Silence is Golden: Branching Bisimilarity is Decidable for Context Free Processes. In *CAV'91*,2–12. Lecture Notes in Computer Science 575, Springer, 1992.

[12] H. Hüttel. Undecidable Equivalences for Basic Parallel Processes. In *Theoretical Aspects of Computer Software*, Lecture Notes in Computer Science 789, 454–464, 1994.

[13] H. Hüttel and C. Stirling. Actions Speak Louder than Words: Proving Bisimilarity for Context-Free Processes. In *LICS'91*, 376–386, 1991.

[14] P. Jančar. Decidability of DPDA Language Equivalence via First-Order Grammars. In *LICS'12*, 415–424. IEEE Computer Society, 2012.

[15] P. Jančar. Equivalences of Pushdown Systems are Hard. *Foundations of Software Science and Computation*, 1–28, 2014.

[16] P. Jančar. Bisimulation Equivalence of First Order Grammars. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *ICALP'14*, Lecture Notes in Computer Science 8573, 232–243, 2014.

[17] P. Jančar. Bisimulation Equivalence of First Order Grammars. arXiv:1405.7923, 2014.

[18] P. Jančar and J. Srba. Undecidability of Bisimilarity by Defender's Forcing. *Journal of ACM*, 55(1), 2008.

[19] R. Mayr. Undecidability of Weak Bisimulation Equivalence for 1-Counter Processes. In *ICALP'03*, Lecture Notes in Computer Science 2719, 570–583. Springer, 2003.

[20] R. Mayr. Process Rewrite Systems. *Information and Computation*, 156:264–286, 2000.

[21] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[22] D. Park. Concurrency and Automata on Infinite Sequences. In *TCS'81*, Lecture Notes in Computer Science 104, 167–183. Springer, 1981.

[23] G. Sénizergues. The Equivalence Problem for Deterministic Pushdown Automata is Decidable. In *ICALP'97*, Lecture Notes in Computer Science 1256, 671–681. Springer-Verlag, 1997.

[24] G. Sénizergues. Decidability of Bisimulation Equivalence for Equational Graphs of Finite Out-Degree. In *FOCS'98*, 120–129. IEEE, 1998.

[25] G. Sénizergues. L(a)=L(b)? Decidability Results from Complete Formal Systems. *Theoretical Computer Science*, 251(1-2):1–166, 2001.

[26] G. Sénizergues. L(a)=L(b)? A Simplified Decidability Proof. *Theoretical Computer Science*, 281(1):555–608, 2002.

[27] J. Srba. Undecidability of Weak Bisimilarity for Pushdown Processes. In *CONCUR'02*, Lecture Notes in Computer Science 2421, 579–593. Springer-Verlag, 2002.

[28] Stirling. Deciding DPDA Equivalence is Primitive Recursive. In *ICALP'02*, Lecture Notes in Computer Science 2380, 821–832. Springer, 2002.

[29] C. Stirling. Decidability of Bisimulation Equivalence for Normed Pushdown Processes. In *CONCUR'96*, Lecture Notes in Computer Science, 217–232. Springer-Verlag, 1996.

[30] C. Stirling. Decidability of Bisimulation Equivalence for Normed Pushdown Processes. *Theoretical Computer Science*, 195(2):113–131, 1998.

[31] C. Stirling. The Joy of Bisimulation. In *MFCS'98*, Lecture Notes in Computer Science 1450, 142–151. Springer, 1998.

[32] C. Stirling. Decidability of Bisimulation Equivalence for Pushdown Processes. 2000.

[33] C. Stirling. Decidability of DPDA Equivalence. *Theoretical Computer Science*, 255(1-2):1–31, 2001.

[34] R. van Glabbeek and W. Weijland. Branching Time and Abstraction in Bisimulation Semantics. In *Information Processing'89*, 613–618. North-Holland, 1989.

[35] R. van Glabbeek and W. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of ACM*, 3:555–600, 1996.

[36] Q. Yin, Y. Fu, C. He, M. Huang, and X. Tao. Branching Bisimilarity Checking for PRS. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *ICALP'14*, Lecture Notes in Computer Science 8573, 363–374, 2014.