Decidability of Epsilon Pushing PDA

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 pushdown automata (PDA). Stirling showed that strong bisimilarity is decidable for normed PDA. Sénizergues strengthened the result by demonstrating the decidability of the strong bisimilarity of the deterministic epsilon-popping PDA. Recently Jančar provided a different proof of the decidability of the strong bisimilarity of PDA in a more general setting. 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.

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) [9]. 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 [9]. 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 $\alpha = \alpha'$.

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

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 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 [21, 23]. His original proof is very long. Simplified proofs were discovered later by Sénizergues [24] himself and by Stirlng [31]. 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 [22, 25]. 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 [22, 25] are ϵ-popping. A silent transition $pX \stackrel{\epsilon}{\longrightarrow} q\alpha$ 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 $pX \stackrel{\epsilon}{\longrightarrow} q\alpha$.

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

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 $pX \stackrel{\epsilon}{\longrightarrow} \ldots \stackrel{\epsilon}{\longrightarrow} q\alpha$ and $q\alpha$ cannot do any silent transition. If $\alpha = \epsilon$ then we can redefine the semantics of pX by $pX \stackrel{\epsilon}{\longrightarrow} q\epsilon$; 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, 11] pointed out that as far as BPA and BPP are concerned the bisimulation equivalence à la Milner [19] and Park [20] is more tractable than the language equivalence. The best way to understand Senizergues' result proved in [22, 25] 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 [32] 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 [22, 25] 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) [28, 29] exploits the tableau method [12, 10]. Later he extended the tableau approach to the study of the unnormed PDA in an unpublished paper [30]. Stirling also provided a simplified account of Senizergues' proof [22, 25] using the process method [31]. The proofs in [22, 25, 31] 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 [17]. Srba proved that weak bisimilarity on nPDA's is undecidable [26]. 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 [17]. In fact they proved that the problems are Π_1^0-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 [18] are undecidable [35]. 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 [22] Non-Elementary [2]	Decidable [22] Non-Elementary [2]
$\simeq$	Undecidable [35]	Undecidable [35]
$\approx$	Σ_1^1-Complete [17] Undecidable [26]	Σ_1^1-Complete [17] Undecidable [26]

Recently Jančar has studied the strong bisimilarity problem of PDA and the DPDA problem in a more general framework called first order grammar, and provided a defender-refuter game argument for the proof of the decidability problems [13, 15, 16]. More recently Fu has taken a closer look at termination condition for equivalence checking algorithm for the strong bisimilarity of PDA [5].

The decidability of the strong bisimilarity and the undecidability of the weak bisimilarity still leave a number of questions unanswered. The motivation for this work is to establish a strictly 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. The former rules out nondeterminism to a great extent, whereas the latter does not deal with silent transitions. The contribution of this paper is two fold.

1. We prove that the branching bisimilarity on the normed ϵ-pushing PDA is decidable.
2. We propose a general approach that deals with silent transitions in equivalence checking algorithms, which could have applications in other models of process rewriting system.

In a separate paper we give a proof of the decidability of the branching bisimilarity for the ϵ-popping PDA. Together with the result of this paper it subsumes all previous decidability results about equivalence checking for PDA.

Section 2 fixes the syntax and the operational semantics of PDA. Section 3 reviews some properties of the branching bisimilarity. Section 4 discusses the finite branching property for the normed ϵ-pushing PDA. Section 5 introduces bisimulation trees. Section 6 explains how to reduce the growth of a pair of processes with long common suffix to the growth of a pair of processes with shorter common suffix. Section 7 presents a semi-decidable procedure for the branching bisimilarity and derives the decidability. Section 8 concludes.

2. PDA

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

- a finite state set $\mathcal{Q} = \{p_1, \ldots, p_{\mathfrak{q}}\}$ ranged over by o, p, q, r, s, t,
- a finite symbol set $\mathcal{V} = \{X_1, \ldots, X_{\mathfrak{n}}\}$ ranged over by X, Y, Z,
- a finite letter set $\mathcal{L} = \{a_1, \ldots, a_{\mathfrak{s}}\}$ 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 $\widetilde{\ell}$. 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 the same as $|\alpha|$. We shall write $A, B, C, D, L, M, N, O, P, Q$ 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.

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

We shall write $\xrightarrow{\widetilde{\ell}}$ for $\xrightarrow{\ell_1} \ldots \xrightarrow{\ell_k}$ if $\widetilde{\ell} = \ell_1 \ldots \ell_k$, $\Longrightarrow$ for the reflexive and transitive closure of $\xrightarrow{\tau}$, and $\overset{\tau}{\Longrightarrow}$ for the transitive closure of $\xrightarrow{\tau}$. If $P \xrightarrow{\widetilde{\ell}} P'$ for some $\widetilde{\ell}$ then P' is a *descendant* of P and P is an *ancestor* of P'. By definition P is a descendant of itself. A process P is *normed*, or P is an nPDA process, if $P \xrightarrow{\widetilde{\ell}} p\epsilon$ for some $\widetilde{\ell}$ and some 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.

3. Branching Bisimilarity

The definition of branching bisimilarity is due to van Glabbeek and Weijland [33]. For PDA's care should be given to processes of the form $p\epsilon$ in order to guarantee that the bisimilarity is a congruence relation [28].

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 $\mathcal{R}^{-1} = \{(Q,P) \mid (P,Q) \in \mathcal{R}\}$ 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 due to the third condition of Definition 1.

A technical lemma that plays an important role in the study of branching bisimilarity is the following Computation Lemma [33, 4].

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 usage in DPDA in which all ϵ-transitions are state-preserving. We write $(\to^*)\to^+$ for the (reflexive and) transitive closure of $\to$. The notation $P \not\to$ stands for the fact that $P \not\simeq P'$ for all P' such that $P \xrightarrow{\tau} P'$. Let $\jmath$ and its decorated versions range over $\mathcal{L} \cup \{\iota\}$. It is necessary to introduce 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 observation considerably simplifies bisimulation argument. Notice also that $P\alpha \xrightarrow{\epsilon} P'\alpha$ whenever $P \xrightarrow{\epsilon} P'$.

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

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

It follows from the congruence property that $\|P\|(h) = \|Q\|(h)$ whenever $P \simeq Q$. Let $\ker(P) = \{h \mid \|P\|(h) \neq \bot\}$. The *strong norm* of P, denoted by $\|P\|_s$, is defined as follows: For each $h \in \ker(P)$ the value $\|P\|_s(h)$ is the least k such that $P \xrightarrow{\ell_1} \ldots \xrightarrow{\ell_k} p_h\epsilon$ for some $\ell_1, \ldots, \ell_k$; and $\|P\|_s(h) = \bot$ for each $h \notin \ker(P)$. We shall use the following convention.

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

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.

Following Stirling [29] we introduce two types of equationally defined stack symbols for PDA. A *simple constant* U is defined by a family of process equalities

$$p_1U = M_1,\ p_2U = M_2,\ \ldots,\ p_{\mathfrak{q}}U = M_{\mathfrak{q}}. \tag{2}$$

We think of the equalities in (2) as grammar equalities. In other words we regard p_iU and M_i as the same syntactical object for every $i \in [\mathfrak{q}]$. We write $U(i)$ for M_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 (generalized) stack should be understood as a finite string composed of stack symbols and/or constants. For example $XUYU'ZV$ is a stack. From now on we will use the small Greek letters for (generalized) stacks. When calculating the length of a stack or the size of a process a simple constant counts as one symbol. Simple constants should *not* be nested, meaning that in (2) the processes M_1 through $M_{\mathfrak{q}}$ may contain *no* simple constants.

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. For the sake of algorithmic study we impose the following condition: For each $i \in [\mathfrak{q}]$ the process $V(i)$ may contain at most one occurrence of simple constant, and the occurrence must be in the last symbol of $V(i)$; and if the last symbol of a stack is a recursive constant, then only the second last symbol of the stack can be a simple constant.

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 remain unaffected.

4. Finite Branching Property

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 raise any 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 induction. 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 $qX \xrightarrow{\tau} q_1X_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} q_kX_k$ is strictly upper bounded by $\mathfrak{nq}$.

Lemma 3. *If P contains no constants and $Q \simeq P$, then $|Q| \leq \mathfrak{m}|P|$.*

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

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 4. *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 $k < \mathfrak{qm}\mathfrak{r}(\mathfrak{m}+1)^{\mathfrak{q}}$.*

Proof. Suppose $qX\sigma \xrightarrow{\epsilon} q_1Z_1\delta_1\sigma$. Let $q_1Z_1\delta_1\sigma \to^* \xrightarrow{j_1^1} \ldots \to^* \xrightarrow{j_{j_1}^1} r_1\epsilon\sigma \to^* \xrightarrow{j_{j_1+1}^1} \ldots \to^* \xrightarrow{j_{j_{k_1}}^1} p_{h_1}\epsilon$ be a transition sequence of minimal length that empties the stack, where $k_1 = \min\|q_1Z_1\delta_1\sigma\|$. Clearly $j_1 \leq \mathfrak{rm}$. Suppose $q_1Z_1\delta_1\sigma \to^* q_2Z_2\delta_2\delta_1\sigma$ such that $\mathfrak{rm} < |Z_2\delta_2\delta_1| \leq \mathfrak{r}(\mathfrak{m}+1)$. Let $q_2Z_2\delta_2\delta_1\sigma \to^* \xrightarrow{j_1^2} \ldots \to^* \xrightarrow{j_{j_2}^2} r_2\epsilon\sigma \to^* \xrightarrow{j_{j_2+1}^2} \ldots \to^* \xrightarrow{j_{j_{k_2}}^2} p_{h_2}\epsilon$ be a

transition sequence with a minimal k 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

$$q_1Z_1\delta_1\sigma \rightarrow^* q_2Z_2\delta_2\delta_1\sigma \rightarrow^* \ldots \rightarrow^* q_{i+1}Z_{i+1}\delta_{i+1}\delta_i\ldots\delta_1\sigma \rightarrow^* \ldots \rightarrow^* q_{\mathfrak{q}+1}Z_{\mathfrak{q}+1}\delta_{\mathfrak{q}+1}\delta_{\mathfrak{q}}\ldots\delta_1\sigma$$

with $\mathfrak{r}\mathfrak{m}(\mathfrak{m}+1)^{i-1} < |Z_{i+1}\delta_{i+1}\delta_i\ldots\delta_1| \leq \mathfrak{r}(\mathfrak{m}+1)^i$ 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

$$q_iZ_i\delta_i\ldots\delta_1\sigma \rightarrow^* \stackrel{j^i_1}{\longrightarrow} \ldots \rightarrow^* \stackrel{j^i_{j_i}}{\longrightarrow} \rightarrow^* r_i\epsilon\sigma \rightarrow^* \stackrel{j^i_{j_i+1}}{\longrightarrow} \ldots \rightarrow^* \stackrel{j^i_{j_{k_i}}}{\longrightarrow} \rightarrow^* p_{h_i}\epsilon$$

where $k_i = \min\|q_iZ_i\delta_i\ldots\delta_1\sigma\|$. Since there are only $\mathfrak{q}$ states, there must be some t_1, t_2 such that $0 < t_1 < t_2 \leq \mathfrak{q}+1$ 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 $qX\sigma \rightarrow^* q'\gamma\sigma$ then $|\gamma| \leq \mathfrak{r}(\mathfrak{m}+1)^{\mathfrak{q}}$. Since there is no ϵ-loop the bound $k < \mathfrak{q}\mathfrak{m}\mathfrak{r}(\mathfrak{m}+1)^{\mathfrak{q}}$ follows. □

Using the finite branching property guaranteed by Lemma 4 it is standard to prove the following.

Proposition 5. *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 $Q \stackrel{a}{\longrightarrow} Q'$ then $P \stackrel{\tau}{\longrightarrow} \ldots \stackrel{\tau}{\longrightarrow} P_j \stackrel{a}{\longrightarrow} P' \simeq_k Q'$ for some $P_1,\ldots,P_j,P'$ such that $P_i \simeq_k Q$ for all $i \in [j]$.
2. If $Q \stackrel{\tau}{\longrightarrow} Q'$ then either $P \simeq_k Q'$ or some $P_1,\ldots,P_j,P'$ exist such that $P \stackrel{\tau}{\longrightarrow} \ldots \stackrel{\tau}{\longrightarrow} P_j \stackrel{\tau}{\longrightarrow} P' \simeq_k Q'$ 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 4 one shows that $\bigcap_{i\geq 0} \simeq_i$ is $\simeq$. So $P \not\simeq Q$ can be checked by checking $P \not\simeq_i Q$ for $i > 0$. □

5. 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 silent transition sequences starting from P. Recall that our nPDA$^{\epsilon+}$ does not admit silent loop action sequence and according to Lemma 4 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. For that purpose we introduce a weaker version of ϵ-tree. A finite set T_P of prefix closed silent transition sequences from P is a *quasi ϵ-tree* of P if the following is valid: If a silent transition $P' \stackrel{\tau}{\longrightarrow} P''$ *leaves* T_P, meaning that P' is a node of T_P and P'' is not a node of T_P, then $P' \not\simeq P''$. A quasi ϵ-tree is *nontrivial* if it contains at least two nodes. The next lemma follows immediately from Proposition 5.

Lemma 6. *Given a finite set of silent transition sequences from P, it is semi-decidable to check if it is a quasi ϵ-tree.*

The root of $\mathfrak{B}$ is (P, Q), and the following are valid for every node (M, N) of $\mathfrak{B}$.

1. (M, N) is a descendant of (P, Q), meaning that M is a descendant of P and N is a descendant of Q.
2. If $M = N$, the node (M, N) is an *i-leaf*. An i-leaf does not have any child.
3. If (M, N) coincides with the label of an ancestor, the node (M, N) is an *r-leaf*. An r-leaf does not have any child.
4. If (M, N) does not have any outgoing $\simeq$-edge, the following are valid:
   (a) $M \not\to$ and $N \not\to$.
   (b) If $M \xrightarrow{J} M'$, then $(M, N) \xrightarrow{J} (M', N')$ for some N' such that $N \xrightarrow{J} N'$.
   (c) If $N \xrightarrow{J} N'$, then $(M, N) \xrightarrow{J} (M', N')$ for some M' such that $M \xrightarrow{J} M'$.
5. If (M, N) has an outgoing $\simeq$-edge, the following are valid:
   (a) There are a quasi ϵ-tree T_M of M and a quasi ϵ-tree T_N of N such that at least one of T_M and T_N is nontrivial. Moreover $(M, N) \xrightarrow{\simeq} (M', N')$ if and only if M' is a leaf of T_M and N' is a leaf of T_N.
   (b) If $(M, N) \xrightarrow{\simeq} (M', N')$, $M \Longrightarrow L \Longrightarrow M'$ and $N \Longrightarrow O \Longrightarrow N'$, then the following are valid:
      i. If $L \xrightarrow{J} L'$, then $(M', N') \xrightarrow{J} (L', N'')$ for some N'' such that $N' \xrightarrow{J} N''$.
      ii. If $O \xrightarrow{J} O'$, then $(M', N') \xrightarrow{J} (M'', O')$ for some M'' such that $M' \xrightarrow{J} M''$.

Figure 1: Bisimulation Tree Property

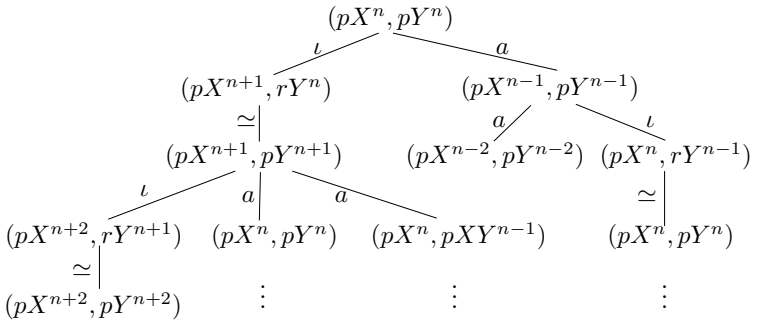
A *bisimulation tree* $\mathfrak{B}$ *for* (P, Q) is a finite branching rooted tree such that each node is labeled by a pair (M, N) of nPDA$^{\epsilon+}$ processes and a directed edge is labeled by an element of $\mathcal{L} \cup \{\iota\} \cup \{\simeq\}$. For convenience we identify a node with its label and refer to an edge labeled by say $\simeq$ as a $\simeq$-edge. Accordingly we write $(M, N) \xrightarrow{\simeq} (M', N')$ for example for a $\simeq$-edge from (M, N) to (M', N'). We write $\mathfrak{B}(P, Q)$ for a bisimulation tree for (P, Q). The definitional property of $\mathfrak{B}(P, Q)$ is stated in Fig. 1. Condition 4 and condition 5 in Fig. 1 remind one of the disjointness condition of DPDA. If one swaps the processes of every label of a bisimulation tree $\mathfrak{B}$ for (P, Q) one obtains a bisimulation tree for (Q, P), denoted by $\mathfrak{B}^{-1}$.

Let's see an example of bisimulation tree.

Example 7. *Suppose an* nPDA$^{\epsilon+}$ *has the following semantic rules.*

1. $pX \xrightarrow{\tau} pX^2$, $pX \xrightarrow{a} p\epsilon$; $pY \xrightarrow{\tau} rY$, $pY \xrightarrow{a} p\epsilon$;
2. $rY \xrightarrow{\tau} sY$, $rY \xrightarrow{a} pX$; $sY \xrightarrow{\tau} pY^2$, $sY \xrightarrow{a} pY$.

It is easy to verify that $pX^n \simeq pY^n$. *A part of the infinite bisimulation tree for* (pX^n, pY^n) *is shown below.*



The definition of bisimulation tree appears unusual at first sight since it refers to the relation $\not\simeq$. Our decidability proof boils down to showing that $\simeq$ is semi-decidable. This is done by a procedure that enumerates finite representations of all the bisimulation trees. The reference to $\not\simeq$ poses no problem in view of Proposition 5.

Given a bisimulation tree $\mathfrak{B}$, define $\overline{\mathfrak{B}}$ as follows:

1. $(M, N) \in \overline{\mathfrak{B}}$ if (M, N) is a node in $\mathfrak{B}$.
2. $(L, O) \in \overline{\mathfrak{B}}$ if $(M, N) \xrightarrow{\simeq} (M', N')$ and $M \Longrightarrow L \Longrightarrow M'$ and $N \Longrightarrow O \Longrightarrow N'$.

Let $\mathcal{I}$ be the identity relation.

Lemma 8. *$\overline{\mathfrak{B}} \cup \mathcal{I}$ is a branching bisimulation.*

It follows immediately from Lemma 8 and the definition of quasi ϵ-tree that $(M, N) \xrightarrow{\simeq} (M', N')$ implies both $M \simeq M'$ and $N \simeq N'$. In a bisimulation tree we have made it explicit if a silent transition is state-preserving or a change-of-state internal action. A bisimulation tree makes a branching bisimulation look as much like a strong bisimulation as possible. This property greatly simplifies the composition of two bisimulation trees.

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 $(M, N) \xrightarrow{\simeq} (M', N')$, it is $k+1$ if $(M, N) \xrightarrow{J} (M', N')$. 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. A *k-bisimulation tree* is obtained from a bisimulation tree by cutting off all the nodes of height $k + 1$.

Suppose $\mathfrak{B}(P, Q)$ is a bisimulation tree for (P, Q) and $\mathfrak{B}(Q, O)$ is a bisimulation tree for (Q, O). We grow a tree $\mathfrak{B}(P, Q); \mathfrak{B}(Q, O)$ in the following manner:

- The root is (P, O).
- If $(L, M) \xrightarrow{\simeq} (L', M')$ is at level k in $\mathfrak{B}(P, Q)$ and $(M, N) \xrightarrow{\simeq} (M', N')$ is at level k in $\mathfrak{B}(Q, O)$, then $(L, N) \xrightarrow{\simeq} (L', N')$ is at level k in $\mathfrak{B}(P, Q); \mathfrak{B}(Q, O)$.
- If $(L, M) \xrightarrow{\simeq} (L', M)$ is at level k in $\mathfrak{B}(P, Q)$ and (M, N) in $\mathfrak{B}(Q, O)$ does not have any out-going $\simeq$-edges, then $(L, N) \xrightarrow{\simeq} (L', N)$ is at level k in $\mathfrak{B}(P, Q); \mathfrak{B}(Q, O)$.
- If (L, M) in $\mathfrak{B}(P, Q)$ does not have any out-going $\simeq$-edges and $(M, N) \xrightarrow{\simeq} (M, N')$ is at level k in $\mathfrak{B}(Q, O)$, then $(L, N) \xrightarrow{\simeq} (L, N')$ is at level k in $\mathfrak{B}(P, Q); \mathfrak{B}(Q, O)$.
- If $(L, M) \xrightarrow{J} (L', M')$ is from level k to level $k + 1$ in $\mathfrak{B}(P, Q)$ and $(M, N) \xrightarrow{J} (M', N')$ is from level k to level $k + 1$ in $\mathfrak{B}(Q, O)$, then $(L, N) \xrightarrow{J} (L', N')$ is from level k to level $k + 1$ in $\mathfrak{B}(P, Q); \mathfrak{B}(Q, O)$.

It is routine to prove the following lemma.

Lemma 9. *$\mathfrak{B}(P, Q); \mathfrak{B}(Q, O)$ is a bisimulation tree for (P, O).*

6. Characteristic Tree

Suppose V is a recursive constant and $p_i\sigma \simeq V(i)\sigma$ for all $i \in [\mathfrak{q}]$. A *bisimulation tree for $(P\sigma, Q\sigma)$ over σ with regards to V* is grown in the following manner: (i) A node $(M\sigma, N\sigma)$ with $|M|, |N| > 0$ is grown in the same way as it is grown in a bisimulation tree for $(M\sigma, N\sigma)$. (ii) A node $(p_i\sigma, N\sigma)$, respectively $(N\sigma, p_i\sigma)$ with $|N| > 0$ is renamed to $(V(i)\sigma, N\sigma)$, respectively $(N\sigma, V(i)\sigma)$, and the latter is grown. In this case $|V(i)| > 0$ must be valid. (iii) A node $(p_i\sigma, p_j\sigma)$ is renamed to $(V(i)\sigma, V(j)\sigma)$ and either $|V(i)| > 0 < |V(j)|$ or $V(i) = p_k\epsilon = V(j)$ for some $k \in [\mathfrak{q}]$. The role of V is to decompose the growth of a bisimulation tree to the growth of smaller trees as it were.

Now suppose $P\sigma \simeq Q\sigma$ such that $|P| > 0$ and $|Q| > 0$ and V_0 is a recursive constant such that $p_i\sigma \simeq V_0(i)\sigma$ for all $i \in [\mathfrak{q}]$. We hope to extend V_0 to a recursive constant V such that $p_i\sigma \simeq V(i)\sigma$ for all $i \in [\mathfrak{q}]$ and that one can grow a bisimulation tree for $(P\sigma, Q\sigma)$ over σ with regards to V. A *characteristic tree for $(P\sigma, Q\sigma)$ over σ extending V_0* is grown like a bisimulation tree for $(P\sigma, Q\sigma)$. 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 $\mathcal{G} = \{G_i\}_{i\in[\mathfrak{q}]}$ of processes such that

$$p_1\sigma \simeq G_1\sigma,\ p_2\sigma \simeq G_2\sigma,\ \ldots,\ p_{\mathfrak{q}}\sigma \simeq G_{\mathfrak{q}}\sigma. \tag{4}$$

Initially $G_i = V_0(i)$ for all $i \in [\mathfrak{q}]$. The second is an equivalence relation $\mathcal{E}$ on $[\mathfrak{q}]$. We write $i \in \mathcal{E}$ if $\langle i, i\rangle \in \mathcal{E}$. Initially $\mathcal{E}$ is the reflexive and transitive closure of $\{\langle i, j\rangle \mid V_0(i) = j \wedge i \in [\mathfrak{q}]\}$. We will write $\langle i\rangle$ for the equivalence class of i. The third is a recursive constant V defined by the grammar equalities

$$p_1V = G_1V,\ p_2V = G_2V,\ \ldots,\ p_{\mathfrak{q}}V = G_{\mathfrak{q}}V. \tag{5}$$

We will update $\mathcal{G}$ and $\mathcal{E}$ dynamically while we grow the tree level by level. The definition of V is updated accordingly. Since all silent transitions are ϵ-pushing, the following construction never gets stuck along $\simeq$-edges/ι-edges. The growth of a node labeled $(M\sigma, N\sigma)$ such that $|M| > 0$ and $|N| > 0$ is the same as in the growth of a bisimulation tree for $(M\sigma, N\sigma)$. 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 $|N| > 0$ respectively $N = p_j\epsilon$ 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 some levels both $\mathcal{G}$ and $\mathcal{E}$ must stabilize and $PV \simeq QV$ must hold. We complete the definition of V by letting $V(i) = p_{\min\langle i\rangle}$ for every $i \in \mathcal{E}$. We call V the recursive constant *generated* by the characteristic tree extending V_0. The recursive constant V extends V_0, notation $V_0 \preccurlyeq V$, in the sense that $|V_0(i)| > 0$ implies $V(i) = V_0(i)$ for all $i \in [q]$. If V_0 is defined by $V_0(i) = p_i\epsilon$ for all $i \in [\mathfrak{q}]$, we call V the recursive constant generated by the characteristic tree.

For normed PDA it should not be surprising that 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 qV such that $V(q)\uparrow$.

Lemma 10. *For every p_i there is some $\widetilde{\ell}$ such that $p_iV \xrightarrow{\widetilde{\ell}} p_hV$ for some p_hV such that $V(h)\uparrow$.*

Proof. Suppose the statement of the lemma were false for some p_i. Then $V(p_i) = L_i$ for some L_i such that $|L_i| > 0$. Now p_jV is defined whenever $p_iV \xrightarrow{\widetilde{\ell_1}} p_jV$. That is $V(p_j) = L_j$ for some L_j such that $|L_j| > 0$. It follows that $p_i\sigma \simeq L_i\sigma \xrightarrow{\widetilde{\ell_1}} p_j\sigma \simeq L_j\sigma$. Similarly p_kV is defined whenever $p_jV \xrightarrow{\widetilde{\ell_2}} p_kV$. So $V(p_k) = L_k$ for some L_k such that $|L_k| > 0$. By definition $p_j\sigma \simeq L_j\sigma \xrightarrow{\widetilde{\ell_2}} p_k\sigma \simeq L_k\sigma$. 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. □

The correlation between (4) and (5) has important consequence.

Lemma 11. *Suppose the recursive constant V is defined by $\{p_iV = L_iV\}_{i\in[\mathfrak{q}]}$ and $p_i\sigma \simeq L_i\sigma$ for all $i \in [\mathfrak{q}]$. If $|P| > 0$ and $|Q| > 0$ 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. □

For fixed P, Q there could be an infinite number of σ such that $P\sigma \simeq Q\sigma$. Each of these σ generates a recursive constant. As the next lemma shows the number of the recursive constants generated this way only depends on P, Q.

Lemma 12. *For fixed P, Q the recursive constants generated by characteristic trees for $(P\sigma, Q\sigma)$ over any σ such that $P\sigma \simeq Q\sigma$ are finite in number.*

Proof. The initial V, defined by $V(i) = p_i\epsilon$ for all $i \in [\mathfrak{q}]$, 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 4 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 σ. Since P, Q are fixed, we are done by induction. □

In the rest of the paper we shall be only concerned with recursive constants generated by the characteristic trees for $(P\sigma, Q\sigma)$ satisfying $|P| \leq \mathfrak{m} + 1$ and $|Q| \leq \mathfrak{m} + 1$. It follows from the above lemma that there is a constant $\mathfrak{h}_0$ such that every recursive constant is generated by an $\mathfrak{h}_0$-characteristic tree. An $\mathfrak{h}_0$-characteristic tree is obtained from a characteristic tree by cutting off all nodes of height $k + 1$.

Let $\mathfrak{B}_\sigma(P, Q)$ be a characteristic tree for $(P\sigma, Q\sigma)$ over σ that generates V. Let $\mathfrak{B}_\sigma(P, Q)\{V/\sigma\}$ denote the tree obtained by substituting V for σ in the labels of $\mathfrak{B}_\sigma(P, Q)$. The following is a useful consequence of Lemma 11.

Corollary 13. *$\mathfrak{B}_\sigma(P, Q)\{V/\sigma\}$ is a bisimulation tree for (PV, QV).*

Proof. Lemma 11 says that $P'\sigma \xrightarrow{\iota} P''\sigma$ implies $P'V \xrightarrow{\iota} P''V$. So the definition of $\mathfrak{B}_\sigma(P, Q)$ and the equalities in (5) guarantee that the relation $\mathfrak{B}_\sigma(P, Q)\{V/\sigma\}$ meets the properties stated in Fig. 1. □

Conversely let $\mathfrak{B}(PV, QV)$ be a bisimulation tree for (PV, QV). Let $\mathfrak{B}(PV, QV)\{\sigma/V\}$ be obtained from $\mathfrak{B}(PV, QV)$ by substituting σ for V.

Lemma 14. *Suppose that V is the recursive constant generated by a characteristic tree for $(P\sigma, Q\sigma)$ over σ. Then $\mathfrak{B}(PV, QV)\{\sigma/V\}$ is a characteristic tree for $(P\sigma, Q\sigma)$ over σ.*

Proof. If (MV, NV) is a node in $\mathfrak{B}(PV, QV)$ then $(M\sigma, N\sigma)$ is a node in $\mathfrak{B}(PV, QV)\{\sigma/V\}$; otherwise V could be further updated. It remains to show that $MV \xrightarrow{\iota} M'V$ implies $M\sigma \xrightarrow{\iota} M'\sigma$. But if $M\sigma \to M'\sigma$, then $M'\sigma \simeq N\sigma$. On the other hand $M'V \not\simeq MV \simeq NV$ by Lemma 8. That would mean that V could be further updated. This is again a contradiction. We conclude from definition that $\mathfrak{B}(PV, QV)\{\sigma/V\}$ is a characteristic tree for $(P\sigma, Q\sigma)$ over σ. □

The significance of Lemma 14 is that algorithmically it reduces the construction of a characteristic tree for $(P\sigma, Q\sigma)$ over σ to the construction of a bisimulation tree for (PV, QV). The latter is a simpler task in view of Lemma 12.

7. A Tree Generation Algorithm

A bisimulation tree is in general infinite. For an infinite bisimulation tree we look for a finite tree from which the infinite tree can be recovered. The literature suggests two crucial ideas to help obtain a finite representation of bisimulation trees. The first is to control the size of the prefixes M, N in a node of the form $(M\sigma, N\sigma)$. This might increase the size of the common suffix. The second is to replace a long common suffix by a shorter one. The latter has been described in Section 6. We now explain the former technique.

A pair (P, Q) is *constrained* if either $|P| < \mathfrak{m} + 1$ or $|Q| < \mathfrak{m} + 1$; it is *unconstrained* otherwise. Suppose $A \simeq B$ and (A, B) is unconstrained. The pair (A, B) can be presented as $(pX\alpha\sigma, M\sigma'\sigma)$ such that $|pX\alpha\sigma| > \mathfrak{m} + 1$, $|M| = \mathfrak{m}$, $|\sigma'\sigma| > 1$ and that M does not contain any constant. For any $i \in \ker(pX)$ suppose that $pX\alpha\sigma \to^* \xrightarrow{J_1} \ldots \to^* \xrightarrow{J_k} p_i\alpha\sigma$ where $k = \|pX\|(i) < \mathfrak{m}$. By Lemma 4 the length of a simulating sequence $M\sigma'\sigma \to^* \xrightarrow{J_1} \ldots \to^* \xrightarrow{J_k} M_i\sigma'\sigma$ is bounded by $\mathfrak{q}\mathfrak{m}(\mathfrak{m} + 1)^{\mathfrak{q}}\mathfrak{m} + \mathfrak{m} < \mathfrak{q}\mathfrak{m}(\mathfrak{m} + 1)^{(\mathfrak{q}+1)}$ and the suffix $\sigma'\sigma$ is kept intact throughout the simulation. It then follows easily that $0 < |M_i| < \mathfrak{q}\mathfrak{m}^2(\mathfrak{m} + 1)^{(\mathfrak{q}+1)}$. Let the simple constant U be defined by

$$p_iU = \begin{cases} M_i, & \text{if } i \in \ker(pX), \\ p_i\epsilon, & \text{if } i \notin \ker(pX). \end{cases} \tag{6}$$

By the definition and the assumption one has that

$$pXU\sigma'\sigma \simeq pX\alpha\sigma \simeq M\sigma'\sigma. \tag{7}$$

The introduction of the simple constant allows one to extend the common suffix σ of the bisimilar pair $(pX\alpha\sigma, M\sigma'\sigma)$ to the common suffix $\sigma'\sigma$ of the bisimilar pair $(pXU\sigma'\sigma, M\sigma'\sigma)$. This is the only use of simple constants in this paper. We therefore impose the following constraint on all simple constants: For $M_1, \ldots, M_\mathfrak{q}$ in (2) the inequality

$$|M_i| < \mathfrak{q}\mathfrak{m}\mathfrak{r}^2(\mathfrak{m}+1)^{(\mathfrak{q}+1)} \tag{8}$$

holds for every $i \in [\mathfrak{q}]$. Hence the following.

Lemma 15. *The number of the simple constants is bounded by* $\mathfrak{q} \cdot \mathfrak{n}^{\mathfrak{q}\mathfrak{m}\mathfrak{r}^2(\mathfrak{m}+1)^{(\mathfrak{q}+1)}}$.

We are in a position to strengthen Lemma 3.

Lemma 16. *If $Q \simeq P$ then $|Q|$ is bounded by a function computable from $|P|$ and $\mathfrak{h}_0$.*

Proof. It follows from Lemma 10 that for every $i \in [\mathfrak{q}]$ there is a minimal length transition sequence $p_iV \xrightarrow{\widetilde{\ell_1}} p_jV \xrightarrow{\widetilde{\ell_2}} \ldots \xrightarrow{\widetilde{\ell_q}} p_hV$ for some undefined p_hV. By the minimality no p_kV appears twice in the sequence. Thus $|\widetilde{\ell_1}\widetilde{\ell_2}\ldots\widetilde{\ell_q}|$ is bounded by a computable function of $\mathfrak{h}_0$ because $V(q)$ is bounded by a computable function of $\mathfrak{h}_0$ for every $q \in [\mathfrak{q}]$. If the last symbol of P is a recursive constant use the above observation. If P does not contain any recursive constant it is easy to bound $\|P\|(i)$ using (8). In any case there is a sequence $P \xrightarrow{\widetilde{\ell}} r\epsilon$ for some $\widetilde{\ell}$ and state r such that $|\widetilde{\ell}|$ can be bounded by a function of $\mathfrak{h}_0$. A computable bound on $|Q|$ is then easy to derive from the equality $Q \simeq P$. □

We now make use of the two ideas to give a finite presentation of a bisimulation tree. A nondeterministic algorithm works as follows: Suppose (A, B) is an input pair. If either A or B is constrained, then the other has a computable size bound by Lemma 16. In this case we simply grow the pair (A, B) for one step. If (A, B) is unconstrained, say $(A, B) = (pX\alpha\sigma, M\sigma'\sigma)$ with $|M| = \mathfrak{m}$, we grow an $(\mathfrak{m}-1)$-bisimulation tree for (A, B), called a *U-tree* for (A, B). In the U-tree we suspend the growth of all nodes of the form $(p_i\alpha\sigma, M_i\sigma'\sigma)$. We call $(p_i\alpha\sigma, M_i\sigma'\sigma)$ a *sink node* and a node $(L\alpha\sigma, N\sigma'\sigma)$ at level $\mathfrak{m}-1$ a *nonsink node* if $|L| > 0$. It is worth remarking that the left hand side of a sink node $(p_i\alpha\sigma, M_i\sigma'\sigma)$ is strictly smaller in size than the left hand side of the root (A, B) of the U-tree. This suggests that we should grow all the sink nodes inductively. For each $i \in \ker(pX)$ we fix a sink node $(p_i\alpha\sigma, M_i\sigma'\sigma)$. A simple constant U is then defined like in (6). We say that U is generated by the U-tree. By definition growing the nonsink node $(L\alpha\sigma, N\sigma'\sigma)$ is equivalent to growing the pair $(LU\sigma'\sigma, N\sigma'\sigma)$. We call $(LU\sigma'\sigma, N\sigma'\sigma)$ the *u-alias* of $(L\alpha\sigma, N\sigma'\sigma)$. By (8) the prefix pair (LU, N) is computationally bounded. For each u-alias we apply the construction defined in Section 6. That is we construct an $\mathfrak{h}_0$-characteristic tree for $(LU\sigma'\sigma, N\sigma'\sigma)$ over $\sigma'\sigma$, generating a recursive constant V. We will call it a *V-tree*. In the V-tree a node of the form $(p\sigma'\sigma, G\sigma'\sigma)$ or of the form $(G\sigma'\sigma, p\sigma'\sigma)$ with $|G| > 0$ is called a *recursive node*. We grow all the recursive nodes recursively. At the $\mathfrak{h}_0$-th level of the V-tree there are possibly nodes of the form $(C\sigma'\sigma, D\sigma'\sigma)$ where $|C| > 0$ and $|D| > 0$. We call such a node a *balance node*. We call (CV, DV) the *v-alias* of $(C\sigma'\sigma, D\sigma'\sigma)$. Since C, D are bounded in size by a computable function of $\mathfrak{h}_0$, there are only a finite number of v-alias. So we grow the v-aliases inductively. This algorithm is nondeterministic because in the buildup of the U-trees and the V-trees it has to make many choices.

In the algorithm outlined above, we can control the size of the nodes in such a way that almost every path terminates in either an i-leaf or an r-leaf. The only place a path may not terminate appears in the following scenario: A nonsink node $(L_0\alpha_0\sigma'_0, N_0\sigma''_0\sigma'_0)$ is generated in a U-tree, and a V-tree for the u-alias $(L_0U_0\sigma_0, N_0\sigma_0)$ is produced, where $\sigma_0 = \sigma''_0\sigma'_0$. Then a U-tree for a recursive node $(p\sigma_0, G\sigma_0)$ or $(G\sigma_0, p\sigma_0)$ in the V-tree is grown, giving rise to a new nonsink node $(L_1\alpha_1\sigma'_1, N_1\sigma''_1\sigma'_1)$ whose u-alias is $(L_1U_1\sigma_1, N_1\sigma_1)$, where $\sigma_1 = \sigma''_1\sigma'_1$. It is important to notice the fact that either σ_0 is a suffix of σ_1 or σ_1 is a suffix of σ_0. It is possible that the alternating production of U-trees and V-trees is repeated infinitely often, leading to an infinite sequence

$$(L_0U_0\sigma_0, N_0\sigma_0), (L_1U_1\sigma_1, N_1\sigma_1), \ldots, (L_iU_i\sigma_i, N_i\sigma_i), \ldots. \tag{9}$$

If for every i there is some $j > i$ such that $|\sigma_j| \leq |\sigma_i|$ then a repeat must occur in (9). Otherwise there must exist some i_0 such that for every $j > i_0$ the common suffix σ_{i_0} is a proper suffix of σ_j. In other words there is a consecutive subsequence of (9) that is of the following form

$$(L_{i_0}U_{i_0}\sigma_{i_0}, N_{i_0}\sigma_{i_0}), (L_{i_0+1}U_{i_0+1}\delta_{i_0+1}\sigma_{i_0}, N_{i_0+1}\delta_{i_0+1}\sigma_{i_0}), \ldots, (L_{i_0+k}U_{i_0+k}\delta_{i_0+k}\sigma_{i_0}, N_{i_0+k}\delta_{i_0+k}\sigma_{i_0}), \ldots. \tag{10}$$

We call (10) a *positive u-alias sequence*. We can construct a characteristic tree for $(L_{i_0+k}U_{i_0+k}\delta_{i_0+k}\sigma_{i_0}, N_{i_0+k}\delta_{i_0+k}\sigma_{i_0})$ over σ_{i_0} for all $k \geq 0$. Let V_{i_0} be the recursive constant generated by an $\mathfrak{h}_0$-characteristic tree for $(L_{i_0}U_{i_0}\sigma_{i_0}, N_{i_0}\sigma_{i_0})$ over σ_{i_0}. If $L_{i_0+\mathfrak{h}_0}U_{i_0+\mathfrak{h}_0}\delta_{i_0+\mathfrak{h}_0}V_{i_0} \simeq N_{i_0+\mathfrak{h}_0}\delta_{i_0+\mathfrak{h}_0}V_{i_0}$, then a characteristic tree for $(L_{i_0+\mathfrak{h}_0}U_{i_0+\mathfrak{h}_0}\delta_{i_0+\mathfrak{h}_0}\sigma_{i_0}, N_{i_0+\mathfrak{h}_0}\delta_{i_0+\mathfrak{h}_0}\sigma_{i_0})$ over σ_{i_0} can be obtained from a bisimulation tree for $(L_{i_0+\mathfrak{h}_0}U_{i_0+\mathfrak{h}_0}\delta_{i_0+\mathfrak{h}_0}V_{i_0}, N_{i_0+\mathfrak{h}_0}\delta_{i_0+\mathfrak{h}_0}V_{i_0})$. We call the latter the *d-alias* of the former. If $L_{i_0+\mathfrak{h}_0}U_{i_0+\mathfrak{h}_0}\delta_{i_0+\mathfrak{h}_0}V_{i_0} \not\simeq N_{i_0+\mathfrak{h}_0}\delta_{i_0+\mathfrak{h}_0}V_{i_0}$, then we grow a characteristic tree over σ_{i_0} *extending* V_{i_0} for the node $(L_{i_0+\mathfrak{h}_0}U_{i_0+\mathfrak{h}_0}\delta_{i_0+\mathfrak{h}_0}\sigma_{i_0}, N_{i_0+\mathfrak{h}_0}\delta_{i_0+\mathfrak{h}_0}\sigma_{i_0})$ to generate a recursive constant V_1. Now $|\delta_{i_0+\mathfrak{h}_0}|$ is bounded by a computable function of $\mathfrak{h}_0$. Thus there is a constant $\mathfrak{h}_1$ such that an $\mathfrak{h}_1$-characteristic tree for $(L_{i_0+\mathfrak{h}_0}U_{i_0+\mathfrak{h}_0}\delta_{i_0+\mathfrak{h}_0}\sigma_{i_0}, N_{i_0+\mathfrak{h}_0}\delta_{i_0+\mathfrak{h}_0}\sigma_{i_0})$ over σ_{i_0} generates V_1. If $L_{i_0+\mathfrak{h}_0+\mathfrak{h}_1}U_{i_0+\mathfrak{h}_0+\mathfrak{h}_1}\delta_{i_0+\mathfrak{h}_0+\mathfrak{h}_1}V_1 \simeq N_{i_0+\mathfrak{h}_0+\mathfrak{h}_1}\delta_{i_0+\mathfrak{h}_0+\mathfrak{h}_1}V_1$, then

$$(L_{i_0+\mathfrak{h}_0+\mathfrak{h}_1}U_{i_0+\mathfrak{h}_0+\mathfrak{h}_1}\delta_{i_0+\mathfrak{h}_0+\mathfrak{h}_1}V_1, N_{i_0+\mathfrak{h}_0+\mathfrak{h}_1}\delta_{i_0+\mathfrak{h}_0+\mathfrak{h}_1}V_1) \tag{11}$$

is deemed the d-alias of $(L_{i_0+\mathfrak{h}_0+\mathfrak{h}_1}U_{i_0+\mathfrak{h}_0+\mathfrak{h}_1}\delta_{i_0+\mathfrak{h}_0+\mathfrak{h}_1}\sigma_{i_0}, N_{i_0+\mathfrak{h}_0+\mathfrak{h}_1}\delta_{i_0+\mathfrak{h}_0+\mathfrak{h}_1}\sigma_{i_0})$. Otherwise we continue in the same way. By induction $|\delta_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_i}|$ is bounded by a computable function of $\mathfrak{h}_0, \mathfrak{h}_1, \ldots, \mathfrak{h}_i$. There must be a recursive constant $\mathfrak{h}_{i+1}$ such that an $\mathfrak{h}_{i+1}$-characteristic tree for $(L_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_i}U_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_i}\delta_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_i}\sigma_{i_0}, N_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_i}\delta_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_i}\sigma_{i_0})$ over σ_{i_0} extending V_i generates a recursive constant V_{i+1}. Since there are $\mathfrak{q}$ entries in a recursive constant, we will get some d-alias in no more than $\mathfrak{q}$ rounds. The size of any d-alias is bounded by a function of $\mathfrak{h}_0, \ldots, \mathfrak{h}_\mathfrak{q}$. The reason that we pin down in the sequence (10) the nonsink nodes with indexes $i_0 + \mathfrak{h}_0, i_0 + \mathfrak{h}_0 + \mathfrak{h}_1, \ldots, i_0 + \mathfrak{h}_0 + \mathfrak{h}_1 + \ldots + \mathfrak{h}_\mathfrak{q}$ rather than the nonsink nodes with indexes $i_0 + 1, i_0 + 2, \ldots, i_0 + \mathfrak{q}$ is that the d-alias $(L_{i_0+2}U_{i_0+2}\delta_{i_0+2}V_{i_0+1}, N_{i_0+2}\delta_{i_0+2}V_{i_0+1})$ for instance may stay well above the leaves of the characteristic tree for $(L_{i_0+1}U_{i_0+1}\delta_{i_0+1}\sigma_{i_0}, N_{i_0+1}\delta_{i_0+1}\sigma_{i_0})$ over σ_{i_0} generating V_{i_0+1}, rendering an inductive argument not very smooth. The node (11) is by definition definitely below the leaves of the $\mathfrak{h}_1$-characteristic tree for $(L_{i_0+\mathfrak{h}_0}U_{i_0+\mathfrak{h}_0}\delta_{i_0+\mathfrak{h}_0}\sigma_{i_0}, N_{i_0+\mathfrak{h}_0}\delta_{i_0+\mathfrak{h}_0}\sigma_{i_0})$ over σ_{i_0}.

We are now ready to spell out the details of a nondeterministic algorithm GT_ϵ that upon receiving a pair of input (A, B) produces a finite representation of a bisimulation tree for (A, B) in a successful run if $A \simeq B$. If $A \not\simeq B$ no execution is successful. The definition of GT_ϵ is given in Fig. 2. It is a recursive algorithm. The following lemma is about the existence of a terminating execution of the algorithm when the input processes are bisimilar.

Lemma 17. *If $A \simeq B$, there are $\mathfrak{h}_0, \mathfrak{h}_1, \ldots, \mathfrak{h}_\mathfrak{q}$ satisfying $\mathfrak{h}_0 < \mathfrak{h}_1 < \ldots < \mathfrak{h}_\mathfrak{q}$ such that $GT_\epsilon(A, B, \mathfrak{h}_0, \mathfrak{h}_1, \ldots, \mathfrak{h}_\mathfrak{q})$ terminates successfully in at least one execution path.*

Proof. We need to argue that with $\mathfrak{h}_0, \mathfrak{h}_1, \ldots, \mathfrak{h}_\mathfrak{q}$ properly chosen in a successful run the tree recursively generated by the algorithm is always finite. A path cannot contain an infinite number of v-aliases because v-aliases are bounded in size. A path cannot contain an infinite number of u-aliases because that would induce an infinite sequence of d-aliases, contradicting to the fact that the d-aliases are finite in number. There cannot be an infinite consecutive sequence of sink nodes because the left hand side of a sink node in a U-tree is strictly smaller in size than the left hand size of the root of the U-tree. So if a path is long enough it must contain many constrained nodes. It follows from Lemma 16 that such a path must terminate in an r-leaf. □

Theorem 18. *The branching bisimilarity of the ϵ-pushing PDA is decidable.*

Proof. We only have to prove that $\simeq$ is semi-decidable. Given an input pair (A, B), guess constants $\mathfrak{h}_0, \mathfrak{h}_1, \ldots, \mathfrak{h}_\mathfrak{q}$ such that $\mathfrak{h}_0 < \mathfrak{h}_1 < \ldots < \mathfrak{h}_\mathfrak{q}$, and apply GT_ϵ to $(A, B, \mathfrak{h}_0, \mathfrak{h}_1, \ldots, \mathfrak{h}_\mathfrak{q})$. If the algorithm successfully outputs a finite tree, a bisimulation tree for (A, B) can be generated. The idea is to grow the finite tree, the output of the algorithm, to a full-fledged bisimulation tree for the input pair. This is done in a co-inductive manner. We will describe a procedure that grows a bisimulation tree level by level. The growth of a node may well depend on part of the tree grown at earlier stages. The i-leaves and the r-leaves are easy. The way to grow a constrained node has been explained in the algorithm. The key is to grow the nodes that have aliases. The growth of the aliases are done independently in parallel.

INPUT: A pair (A, B) of processes, and numbers $\mathfrak{h}_0, \mathfrak{h}_1, \ldots, \mathfrak{h}_\mathfrak{q}$ such that $\mathfrak{h}_0 < \mathfrak{h}_1 < \ldots < \mathfrak{h}_\mathfrak{q}$.

ALGORITHM GT_ϵ:

1. If (A, B) is an i-node or an r-node, stop.
2. If $|A| \leq \mathfrak{m} + 1$ or $|B| \leq \mathfrak{m} + 1$, guess a quasi ϵ-tree T_A of A and a quasi ϵ-tree T_B of B, whose height is bounded by $\mathfrak{q}\mathfrak{m}\mathfrak{r}(\mathfrak{m}+1)^\mathfrak{q}$. Check if T_A and T_B are legal. [Notice that nontermination is introduced here.] If both are legal, do the following.
   (a) For every leaf M of T_A and every leaf N of T_B guess a finite set of transitions $\left\{(M,N) \xrightarrow{b_i} (M_i, N_i)\right\}_{i\in I}$, and verify that the set of edges $\left\{M \xrightarrow{b_i} M_i, N \xrightarrow{b_i} N_i\right\}_{i\in I}$ form a 1-bisimulation tree for (M, N). If it does form a 1-bisimulation tree, apply GT_ϵ to every (M_i, N_i); abort if otherwise. [In all recursive invocations of GT_ϵ the parameters $\mathfrak{h}_0, \mathfrak{h}_1, \ldots, \mathfrak{h}_\mathfrak{q}$ remain the same.]
   (b) For every leaf M of T_A and every node N of T_B that is not a leaf guess a finite set of transitions $\left\{(M,N) \xrightarrow{b_i} (M_i, N_i)\right\}_{i\in I}$, and check that for every transition $N \xrightarrow{b} N'$ there is a transition $M \xrightarrow{b} M'$ such that $(M,N) \xrightarrow{b} (M', N') \in \left\{(M,N) \xrightarrow{b_i} (M_i, N_i)\right\}_{i\in I}$. If the check passes, apply GT_ϵ to every (M_i, N_i); otherwise abort.
   (c) For every node M of T_A that is not a leaf and every leaf N of T_B carry out the symmetric construction.
3. If $|A| > \mathfrak{m} + 1$ and $|B| > \mathfrak{m} + 1$, we write $A = pX\alpha$ and $B = M\sigma$ with $|M| = \mathfrak{m}$. Guess a U-tree for $(pX\alpha, M\sigma)$ so that a normed constant U is defined. Verify that the internal nodes of the guessed U-tree satisfy the branching bisimulation property. If the verification is unsuccessful, abort; otherwise do the following.
   (a) Apply GT_ϵ to every sink node.
   (b) For the u-alias $(LU\sigma, N\sigma)$ of every nonsink node of the U-tree, abort if there is a positive u-alias sequence longer than $\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_\mathfrak{q}+1$ that ends in $(LU\sigma, N\sigma)$; otherwise nondeterministically do (3(b)i) or (3(b)ii).
      i. Guess an $\mathfrak{h}_0$-characteristic tree for $(LU\sigma, N\sigma)$. If any of the internal nodes of the guessed tree fails the branching bisimulation property, abort; otherwise do the following.
         A. Apply GT_ϵ to every recursive node.
         B. Apply GT_ϵ to the v-alias of every balance node.
      ii. If there is a positive u-alias sequence of length $\mathfrak{h}_0 + \mathfrak{h}_1 + \ldots + \mathfrak{h}_h + 1$ ending in $(LU\sigma, N\sigma)$, where $h < \mathfrak{q}$, we write $(L_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_h} U_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_h} \delta_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_h} \sigma_{i_0}, M_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_h} \delta_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_h} \sigma_{i_0})$ for $(LU\sigma, N\sigma)$. Guess recursive constants $V_1, \ldots, V_h$ such that $V_1 \preccurlyeq \ldots \preccurlyeq V_h$ and the inequality $|V_j(i)| < \mathfrak{h}_j$ is valid for all $j \leq h$ and all $i \in [\mathfrak{q}]$. Check that for every $j < h$ there is an $\mathfrak{h}_j$-bisimulation tree for $(L_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_j} U_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_j} \delta_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_j} \sigma_{i_0}, M_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_j} \delta_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_j} \sigma_{i_0})$ over σ_{i_0} extending V_j and generating V_{j+1}; and also check that there is an $\mathfrak{h}_{h+1}$-bisimulation tree over σ_{i_0} with regards to V_h for $(L_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_{h+1}} U_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_{h+1}} \delta_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_{h+1}} \sigma_{i_0}, M_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_{h+1}} \delta_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_{h+1}} \sigma_{i_0})$. [It is explained after the proof of Theorem 18 on page 14 how the checks are done.] If all the checks are successful, apply GT_ϵ to $(L_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_{h+1}} U_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_{h+1}} \delta_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_{h+1}} V_h, M_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_{h+1}} \delta_{i_0+\mathfrak{h}_0+\mathfrak{h}_1+\ldots+\mathfrak{h}_{h+1}} V_h)$; otherwise the algorithm aborts. [Without loss of generality we have assumed that $\forall i \in [\mathfrak{q}].|V_j(i)| < \mathfrak{h}_j$.]

Figure 2: Nondeterministic Algorithm GT_ϵ.

- For a nonsink node $(L\alpha, N\sigma)$, we first grow the u-alias $(LU\sigma, N\sigma)$. We then grow $(L\alpha, N\sigma)$ by copying the growth of $(LU\sigma, N\sigma)$ until a path reaches to a node of the form $(p_i\alpha, N')$ for some i. We then start to compose the bisimulation tree for the sink node $(p_i\alpha, M_i\sigma)$ with the bisimulation tree for $(M_i\sigma, N')$ level by level.

- For every u-alias $(LU\sigma, N\sigma)$ the algorithm has produced an $\mathfrak{h}_0$-characteristic tree for it over σ, generating a recursive constant V. A balance node $(C\sigma, D\sigma)$ in the characteristic tree is grown as follows: Grow the v-alias (CV, DV) of the balance node. Then grow a characteristic tree for $(C\sigma, D\sigma)$ over σ by duplicating the bisimulation tree for (CV, DV). Finally a bisimulation tree for $(C\sigma, D\sigma)$ is grown by composing the characteristic tree for $(C\sigma, D\sigma)$ with the bisimulation trees for the recursive nodes of the form $(p_i\sigma, G_i\sigma)$ or $(G_i\sigma, p_i\sigma)$. This kind

of composition may need be carried out infinitely often. Notice that the recursive nodes $(p_i\sigma, G_i\sigma)$ or $(G_i\sigma, p_i\sigma)$ appear at levels higher up in the bisimulation tree that is being constructed.

- Let $(L_{i_0+\mathfrak{b}_0+\mathfrak{b}_1+\ldots+\mathfrak{b}_{h+1}}U_{i_0+\mathfrak{b}_0+\mathfrak{b}_1+\ldots+\mathfrak{b}_{h+1}}\delta_{i_0+\mathfrak{b}_0+\mathfrak{b}_1+\ldots+\mathfrak{b}_{h+1}}V_h, M_{i_0+\mathfrak{b}_0+\mathfrak{b}_1+\ldots+\mathfrak{b}_{h+1}}\delta_{i_0+\mathfrak{b}_0+\mathfrak{b}_1+\ldots+\mathfrak{b}_{h+1}}V_h)$ be the d-alias of

$$(L_{i_0+\mathfrak{b}_0+\mathfrak{b}_1+\ldots+\mathfrak{b}_{h+1}}U_{i_0+\mathfrak{b}_0+\mathfrak{b}_1+\ldots+\mathfrak{b}_{h+1}}\delta_{i_0+\mathfrak{b}_0+\mathfrak{b}_1+\ldots+\mathfrak{b}_{h+1}}\sigma_{i_0}, M_{i_0+\mathfrak{b}_0+\mathfrak{b}_1+\ldots+\mathfrak{b}_{h+1}}\delta_{i_0+\mathfrak{b}_0+\mathfrak{b}_1+\ldots+\mathfrak{b}_{h+1}}\sigma_{i_0}). \tag{12}$$

  Grow a bisimulation tree for the d-alias, from which we get a characteristic tree for (12) over σ_{i_0}. We then grow a bisimulation tree for (12) by composing the characteristic tree with the bisimulation trees for the recursive nodes of the form $(p_i\sigma_{i_0}, V_h(i)\sigma_{i_0})$ or $(V_h(i)\sigma_{i_0}, p_i\sigma_{i_0})$ that appear higher up in the bisimulation tree being constructed.

The existence of a bisimulation tree for (A, B) is now evident. We are done by applying Lemma 8. □

Now that we know how to construct level by level a bisimulation tree for a node in the output tree of a successful run of the algorithm GT_ϵ, we are in a position to explain the remark made in (3(b)ii) of the algorithm GT_ϵ. The crucial thing that need be checked is that for all $j \leq h$ and all $k \in [\mathfrak{q}]$ such that $|V_j(k)| > 0$ a bisimulation tree for $(p_j\sigma_{i_0}, V_j(k)\sigma_{i_0})$ can be grown. For the base case the algorithm has produced an $\mathfrak{b}_0$-characteristic tree over σ_{i_0} for $(L_{i_0}U_{i_0}\sigma_{i_0}, M_{i_0}\sigma_{i_0})$ generating V_{i_0}. By the construction we know that for every k such that $|V_{i_0}(k)| > 0$ the pair $(p_j\sigma_{i_0}, V_j(k)\sigma_{i_0})$ or the pair $(V_j(k)\sigma_{i_0}, p_j\sigma_{i_0})$ is a recursive node. By the construction defined in the above proof we know how to grow the node into a bisimulation tree level by level. Now consider $(L_{i_0+\mathfrak{b}_0}U_{i_0+\mathfrak{b}_0}\delta_{i_0+\mathfrak{b}_0}\sigma_{i_0}, M_{i_0+\mathfrak{b}_0}\delta_{i_0+\mathfrak{b}_0}\sigma_{i_0})$ in the tree constructed by the algorithm. We can grow a bisimulation tree $\mathfrak{B}$ for $(L_{i_0+\mathfrak{b}_0}U_{i_0+\mathfrak{b}_0}\delta_{i_0+\mathfrak{b}_0}\sigma_{i_0}, M_{i_0+\mathfrak{b}_0}\delta_{i_0+\mathfrak{b}_0}\sigma_{i_0})$ level by level. We now use $\mathfrak{B}$ to grow an $\mathfrak{b}_1$-characteristic tree $\mathfrak{C}$ for $(L_{i_0+\mathfrak{b}_0}U_{i_0+\mathfrak{b}_0}\delta_{i_0+\mathfrak{b}_0}\sigma_{i_0}, M_{i_0+\mathfrak{b}_0}\delta_{i_0+\mathfrak{b}_0}\sigma_{i_0})$ over σ_{i_0} extending V_{i_0}. This is done using the construction defined in the second paragraph of Section 6. There is however a fundamental difference. The construction defined in Section 6 are semantical. Every pair of processes appearing as a node in a bisimulation tree are in fact bisimilar. There *is* a bisimulation tree for the node. Here the bisimilarity of the two processes appearing in a node is what we are trying to establish. We must use a proof theoretical version of the semantical construction. For proof theoretical construction we can only rely on the fact that we are building up a bisimulation tree for the node level by level by composing and duplicating the bisimulation trees which we have started building up at earlier stages. Suppose $V_{\mathfrak{b}_0}$ is the current recursive constant generated at some point during the growth of $\mathfrak{C}$. Suppose $(p_i\sigma_{i_0}, N_i\sigma_{i_0})$, where $|N_i| > 0$, is a node to be grown. If $|V_{\mathfrak{b}_0}(i)| = 0$ the construction is the same as what has been described in Section 6. But if $|V_{\mathfrak{b}_0}(i)| > 0$, then we compose the bisimulation tree for $(V_{\mathfrak{b}_0}(i)\sigma_{i_0}, p_i\sigma_{i_0})$ with the bisimulation tree for $(p_i\sigma_{i_0}, N_i\sigma_{i_0})$ to obtain a bisimulation tree $\mathfrak{B}'$ for $(V_{\mathfrak{b}_0}(i)\sigma_{i_0}, N_i\sigma_{i_0})$. We then use $\mathfrak{B}'$ to construct the rest of the characteristic tree rooted at $(V_{\mathfrak{b}_0}(i)\sigma_{i_0}, N_i\sigma_{i_0})$. Similarly for a node of the form $(p_i\sigma_{i_0}, p_j\sigma_{i_0})$, where $|V_{\mathfrak{b}_0}(i)| > 0$ and $|V_{\mathfrak{b}_0}(j)| > 0$, we compose the bisimulation tree for $(p_i\sigma_{i_0}, p_j\sigma_{i_0})$ with the bisimulation tree for $(V_{\mathfrak{b}_0}(i)\sigma_{i_0}, p_i\sigma_{i_0})$ on the left and the bisimulation tree for $(p_j\sigma_{i_0}, V_{\mathfrak{b}_0}(j)\sigma_{i_0})$ on the right to obtain a bisimulation tree $\mathfrak{B}''$ for $(V_{\mathfrak{b}_0}(i)\sigma_{i_0}, V_{\mathfrak{b}_0}(j)\sigma_{i_0})$. We then use $\mathfrak{B}''$ to grow the rest of the characteristic tree for $(V_{\mathfrak{b}_0}(i)\sigma_{i_0}, V_{\mathfrak{b}_0}(j)\sigma_{i_0})$. If eventually the recursive constant $V_{\mathfrak{b}_0}$ generated by the $\mathfrak{b}_0$-characteristic tree $\mathfrak{C}$ is the same as the recursive constant V_1 guessed by the algorithm, the check is successful. The other checks can be carried out by induction. Finally we also need to check if an $\mathfrak{b}_{h+1}$-bisimulation tree over σ_{i_0} with regards to V_h can be grown for $(L_{i_0+\mathfrak{b}_0+\mathfrak{b}_1+\ldots+\mathfrak{b}_h}U_{i_0+\mathfrak{b}_0+\mathfrak{b}_1+\ldots+\mathfrak{b}_h}\delta_{i_0+\mathfrak{b}_0+\mathfrak{b}_1+\ldots+\mathfrak{b}_h}\sigma_{i_0}, M_{i_0+\mathfrak{b}_0+\mathfrak{b}_1+\ldots+\mathfrak{b}_h}\delta_{i_0+\mathfrak{b}_0+\mathfrak{b}_1+\ldots+\mathfrak{b}_h}\sigma_{i_0})$. This can be done by using a proof theoretical version of the construction defined in the beginning of Section 6. Since all the involved trees have height bound, the nondeterminism introduces only a finite number of possibilities.

8. Conclusion

The result of the paper and the results of Jančar and Srba [17] are summarized in the following table. The new result is significant in the light that the corresponding problem for the ϵ-pushing PDA remains in analytic hierarchy [34].

	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 present work. We have attempted to prove the present result by using tableau system as is done in Stirling's work. It turned out however that due to the presence of silent transitions, proof based on a tableau system is not easy to handle. The difficulty is two fold. Firstly in the presence of silent transitions the k-bisimilarity, as introduced in the proof of Proposition 5, is very subtle. It is a powerful tool to establish negative results. It is 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, Stirling's proof is not repeatable. Secondly an alternative would be to construct branching bisimulations from a tableau, bypassing the 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. All these problems can be avoided by introducing negative information in terms of $\not\simeq$. This crucial observation has led us to the definition of bisimulation tree for branching bisimilairty.

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 [6]. The result of [6] and the result of this paper together accomplish the goal set up in the beginning of the paper.

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 [3]. 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. So instead of rewriting processes, one rewrites trees.

Jančar has studied first order grammar [13] and provided a quite different proof for the decidability of the strong bisimilarity of nPDA [15]. In the full paper he also outlined an idea of how to extend his proof to take care of silent transitions. 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 [27]. 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 [14], a consequence of which is that the strong bisimilarity proved decidable by Sénizergues in [22] is Ackermann-hard. It would be interesting to look for tighter upper and lower bounds on the complexity of 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, 61772336) 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] D. Caucal. Graphes canoniques de graphes algébriques. *Informatique théorique et Applications*, 24:339–352, 1990.
[4] Y. Fu. Checking Equality and Regularity for Normed BPA with Silent Moves. *ICALP'13*, Lecture Notes in Computer Science 7966, 238–249, 2013.
[5] Y. Fu. Termination Condition for Equivalence Checking Algorithm of PDA. 2018. `https://basics.sjtu.edu.cn/~yuxi/`.
[6] Y. Fu and Q. Yin. Decidability of Epsilon Popping PDA. 2018.
[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] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Publishing Company, 1979.
[10] 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.
[11] H. Hüttel. Undecidable Equivalences for Basic Parallel Processes. In *Theoretical Aspects of Computer Software*, Lecture Notes in Computer Science 789, 454–464, 1994.

[12] H. Hüttel and C. Stirling. Actions Speak Louder than Words: Proving Bisimilarity for Context-Free Processes. In *LICS'91*, 376–386, 1991.
[13] P. Jančar. Decidability of DPDA Language Equivalence via First-Order Grammars. In *LICS'12*, 415–424. IEEE Computer Society, 2012.
[14] P. Jančar. Equivalences of Pushdown Systems are Hard. *Foundations of Software Science and Computation*, 1–28, 2014.
[15] 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.
[16] P. Jančar. Bisimulation Equivalence of First Order Grammars. arXiv:1405.7923, 2014.
[17] P. Jančar and J. Srba. Undecidability of Bisimilarity by Defender's Forcing. *Journal of ACM*, 55(1), 2008.
[18] R. Mayr. Process Rewrite Systems. *Information and Computation*, 156:264–286, 2000.
[19] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
[20] D. Park. Concurrency and Automata on Infinite Sequences. In *TCS'81*, Lecture Notes in Computer Science 104, 167–183. Springer, 1981.
[21] 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.
[22] G. Sénizergues. Decidability of Bisimulation Equivalence for Equational Graphs of Finite Out-Degree. In *FOCS'98*, 120–129. IEEE, 1998.
[23] G. Sénizergues. L(a)=L(b)? Decidability Results from Complete Formal Systems. *Theoretical Computer Science*, 251(1-2):1–166, 2001.
[24] G. Sénizergues. L(a)=L(b)? A Simplified Decidability Proof. *Theoretical Computer Science*, 281(1):555–608, 2002.
[25] G. Sénizergues. Decidability of Bisimulation Equivalence for Equational Graphs of Finite Out-Degree. *SIAM Journal of Computing*, 34(5):1025–1106, 2005.
[26] J. Srba. Undecidability of Weak Bisimilarity for Pushdown Processes. In *CONCUR'02*, Lecture Notes in Computer Science 2421, 579–593. Springer-Verlag, 2002.
[27] Stirling. Deciding DPDA Equivalence is Primitive Recursive. In *ICALP'02*, Lecture Notes in Computer Science 2380, 821–832. Springer, 2002.
[28] C. Stirling. Decidability of Bisimulation Equivalence for Normed Pushdown Processes. In *CONCUR'96*, Lecture Notes in Computer Science, 217–232. Springer-Verlag, 1996.
[29] C. Stirling. Decidability of Bisimulation Equivalence for Normed Pushdown Processes. *Theoretical Computer Science*, 195(2):113–131, 1998.
[30] C. Stirling. Decidability of Bisimulation Equivalence for Pushdown Processes. 2000.
[31] C. Stirling. Decidability of DPDA Equivalence. *Theoretical Computer Science*, 255(1-2):1–31, 2001.
[32] R. van Glabbeek and W. Weijland. Branching Time and Abstraction in Bisimulation Semantics. In *Information Processing'89*, 613–618. North-Holland, 1989.
[33] R. van Glabbeek and W. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of ACM*, 3:555–600, 1996.
[34] Q. Yin. Branching Bisimilarity of Epsilon Pushing PDA Remains in Analytic Hierarchy. 2018.
[35] 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.