The Decidability of ϵ-Pushing PDA

Yuxi Fu and Qiang Yin

BASICS, Department of Computer Science, Shanghai Jiao Tong University

Abstract. Sénizergues has proved that language equivalence is decidable for disjoint ϵ-deterministic PDA. Stirling has showed that strong bisimilarity is decidable for PDA. On the negative side Srba demonstrated that the weak bisimilarity is undecidable for normed PDA. Later Jančar and Srba established the undecidability of the weak bisimilarity for disjoint ϵ-pushing PDA and disjoint ϵ-popping PDA. In the present paper it is shown that the branching bisimilarity of the normed ϵ-pushing PDA is decidable and that the branching bisimilarity of the ϵ-pushing PDA remains Σ_1^1-complete.

1 Introduction

> "Is it recursively unsolvable to determine if $L_1 = L_2$ for arbitrary deterministic languages L_1 and L_2?"
>
> – Ginsburg and Greibach, 1966

The above question was raised in Ginsburg and Greibach's 1966 paper [4] titled Deterministic Context Free Languages. The equality referred to in the above 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) [7]. 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

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

The transition rule $pX \stackrel{a}{\longrightarrow} q\alpha$ 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 $pX \stackrel{\epsilon}{\longrightarrow} q\alpha$ 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 [7]. 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) [4].

A deterministic pushdown automaton enjoys disjointness and determinism properties. The determinism property is the combination of A-determinism and ϵ-determinism. 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 $pX \stackrel{a}{\longrightarrow} q\alpha$ and $pX \stackrel{a}{\longrightarrow} q'\alpha'$ then $q = q'$ and $\alpha = \alpha'$.
ϵ-Determinism. If $pX \stackrel{\epsilon}{\longrightarrow} q\alpha$ and $pX \stackrel{\epsilon}{\longrightarrow} q'\alpha'$ 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 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 [20,22]. His original proof is very long. Simplified proofs were soon discovered by Sénizergues [23] himself and by Stirlng [30]. After the positive answer of Sénizergues, one wonders if the strong constraints (disjointness+A-determinism+ϵ-determinism) can be relaxed. The first such extension was given by Sénizergues himself [21]. 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 [21] 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.

ϵ-Popping PDA. A PDA is ϵ-popping if $|\alpha| \leq 1$ whenever $pX \xrightarrow{\epsilon} q\alpha$.
ϵ-Pushing PDA. 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 [5,9] pointed out that as far as BPA and BPP are concerned the bisimulation equivalence à la Milner [18] and Park [19] is more tractable than the language equivalence. The best way to understand Senizergues' result 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 [31] 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 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) [26,27] exploits the tableau method [10,8]. Later he extended the tableau approach to the study of the unnormed PDA [29]. Stirling also provided a simplified account of Senizergues' proof [21] using the process method [30]. The proof in [30], as well as the one in [21], is interesting in that it turns the language equivalence of disjoint ϵ-deterministic PDA to the strong 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 [15]. Srba proved that weak bisimilarity on nPDA's is undecidable [24]. 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 [15]. 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 [17] are undecidbale [33]. This general result implies that the branching bisimilarity on nPDA is undecidable. The idea of Defender's Forcing can also be used to prove complexity bound. An example is Benedikt,

	PDA	**nPDA**
$\sim$	Decidable [21,29] Non-Elementary [1]	Decidable [26,27] Non-Elementary [1]
$\simeq$	Undecidable [33]	Undecidable [33]
$\approx$	Σ^1_1-Complete [15] Undecidable [24]	Σ^1_1-Complete [15] Undecidable [24]

Fig. 1. Decidability of PDA

	ϵ-Popping **nPDA/PDA**	ϵ-Pushing **nPDA**	ϵ-Pushing **PDA**
$\simeq$	?	?	?
$\approx$	Π^0_1-Complete [15]	Π^0_1-Complete [15]	Σ^1_1-Complete [15]

Fig. 2. More on Decidability of PDA

Moller, Kiefer and Murawski's proof that the strong bisimilarity on PDA is non-elementary [1]. A summary of the decidability/undecidability results mentioned above is given in Fig. 1 and Fig. 2, where $\sim$ stands for the strong bisimilarity, $\simeq$ the branching bisimilarity, and $\approx$ the weak bisimilarity.

The decidability of the strong bisimilarity and the undecidability of the weak bisimilarity still leaves a number of questions unanswered. A conservative extension of the language equivalence for DPDA is neither the strong bisimilarity nor the weak bisimilarity. It is not the former because language equivalence ignores silent transitions. It is not the latter since the whole point of introducing the disjointness and ϵ-determinism conditions is to force all silent transitions to preserve equivalence. To investigate the possibility of extending the decidability result of DPDA, one should really start with the branching bisimilarity. This is what we are going to do in this paper. Since Senizergues' result can be stated as saying that the branching bisimilarity on the disjoint ϵ-deterministic PDA is decidable, we will look at the situations in which either the disjointness condition is dropped and/or the ϵ-determinism condition is weakened/removed.

The contributions of this paper are summarized as follows.

1. Technically we will prove that The main results are the following.
   - the branching bisimilarity on the ϵ-pushing nPDA is decidable, and that
   - the branching bisimilarity on the ϵ-pushing PDA is Σ^1_1-complete.
2. At the model theoretical level we propose a model that strictly extends the classical PDA model. The new model gets rid of the notion of stack in favour of a structural definition of processes. The structural definition helps simply the proofs of our results significantly.

The rest of the paper is organised as follows. Section 2 introduces an extended PDA model. Section 3 reviews the basic properties of the branching bisimilarity. Section 4 confirms that the finite branching property hold for both the normed ϵ-pushing PDA. Section 5 establishes the decidability of the normed ϵ-pushing PDA. Section 6 applies the Defender's Forcing technique to show that ϵ-nondeterminism is highly undecidable. Section 7 concludes.

2 PDA and its Extension

A *pushdown automaton* (or simply PDA) $\varGamma = (\mathcal{Q}, \mathcal{V}, \mathcal{L}, \mathcal{R})$ consists of

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

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 convention in language theory a silent action will be denoted by ϵ. The set $\mathcal{A} = \mathcal{L} \cup \{\epsilon\}$ of actions is ranged over by ℓ. The set $\mathcal{A}^*$ of action sequence is ranged over by ℓ^*. The set $\mathcal{V}^*$ of strings of symbols is ranged over by small Greek letters. By overloading notation the empty string is also denoted by ϵ. We identify both $\epsilon\alpha$ and $\alpha\epsilon$ to α syntactically. The length of α is denoted by $|\alpha|$.

A *pushdown process*, or *PDA process*, is an interactive object with a syntactical tree structure. To emphasize the structural aspect, our pushdown processes are defined with the help of *simple constants*. For a PDA with $\mathfrak{q}$ states a simple constant is a $\mathfrak{q}$-ary tuple of PDA processes. The inductive definition is given below.

$$\begin{aligned} P &:= \mathbf{0} \mid p\epsilon \mid pXC_{[\mathfrak{q}]}, \\ C_{[\mathfrak{q}]} &:= (P_1, \ldots, P_{\mathfrak{q}}). \end{aligned}$$

A process is either the *nil* process $\mathbf{0}$, or an *accepting* process $p\epsilon$, or a *sequential* process $pXC_{[\mathfrak{q}]}$. If $C_{[\mathfrak{q}]} = (P_1, \ldots, P_{\mathfrak{q}})$ then we impose the equality $p_i\epsilon C_{[\mathfrak{q}]} = P_i$. Throughout this paper the equality symbol "=" stands for grammar equality. So $p_i\epsilon C_{[\mathfrak{q}]}$ is syntactically identified to P_i. For simplification we often omit the subscript in $C_{[\mathfrak{q}]}$. To make evident the relationship between the PDA defined in the standard fashion and our PDA we introduce the auxiliary notation $p\alpha C_{[\mathfrak{q}]}$ as well as the notation $p\alpha$. Here is the structural definition.

$$\begin{aligned} p_i\epsilon C &= P_i, \quad \text{if } C = (P_1, \ldots, P_{\mathfrak{q}}), \\ pX\beta C &= pX(p_1\beta C, \ldots, p_{\mathfrak{q}}\beta C), \\ p\alpha &= p\alpha(p_1\epsilon, \ldots, p_{\mathfrak{q}}\epsilon) \end{aligned}$$

In this way a standard PDA process $p\alpha$ can be seen as an abbreviation of a pushdown process in our model.

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 structural rules:

$$\frac{pX \xrightarrow{\ell} q\alpha \in \mathcal{R}}{pX \xrightarrow{\ell} q\alpha} \qquad \frac{pX \xrightarrow{\ell} q\alpha}{pXC \xrightarrow{\ell} q\alpha C} \tag{1}$$

We shall use the standard notations $\xrightarrow{\ell^*}$ and $\Longrightarrow$ and $\stackrel{\ell^*}{\Longrightarrow}$. A process P *accepts* a word w if $P \stackrel{w}{\Longrightarrow} p\epsilon$ for some p. A process P is *normed*, or P is an nPDA process, if $P \xrightarrow{\ell^*} p\epsilon$ for some ℓ^*, p. A PDA $\varGamma = (\mathcal{Q}, \mathcal{V}, \mathcal{L}, \mathcal{R})$ is normed, or $\varGamma$ is an nPDA, if pX is normed for all $p \in \mathcal{Q}$ and all $X \in \mathcal{V}$. The notation PDA$^{\epsilon-}$ will refer to the variant of PDA with ϵ-popping transitions, and nPDA$^{\epsilon+}$ to the variant of nDPA with ϵ-pushing transitions.

2.1 Recursive Constant

To help study the recursive behavours of PDA processes, it is convenient to introduce a special class of constants called recursive constants. To define these constants we find it convenient to work in an extended PDA model. Formally the set of the *extended PDA terms* and the set of the constants admitted by a PDA Γ are generated from the following BNF:

$$\begin{aligned} P &:= \mathbf{0} \mid p\epsilon \mid l \mid pXC_{[n]}, \\ C_{[n]} &:= (P_1, \ldots, P_n) \mid V_{[n]}. \end{aligned}$$

Instead of having only $\mathfrak{q}$-ary constants as in the PDA model, in the extended model we have n-ary *term constant* for all $n \geq 0$. An *n-ary* term constant $C_{[n]}$ is either an n-tuple of terms $(P_1, \ldots, P_n)$ or an *n-ary* recursive constant $V_{[n]}$. An alternative notation for $(P_1, \ldots, P_n)$ is $(P_i)_{i\in[n]}$. The notation $V_{[n]}$ stands for an n-ary recursive constant. An n-ary recursive constant is also an n-tuple. For each $i \in [n]$ we write $C_{[n]}(i)$ for its i-th component. A nullary constant is identified with $\mathbf{0}$ syntactically. The notation $p_i\epsilon C_{[n]}$ is identified to $C_{[n]}(i)$ if $i \in [n]$; it is identified to $p_i\epsilon$ otherwise. We omit the subscript in $C_{[n]}$ when no confusion may arise. In the above definition, l ranges over the set $\mathbb{N}$ of *positive* integer and $[n]$ denotes the set $\{1, 2, \ldots, n\}$. We will call l a *selector*. The reason that we use the terminologies "term" and "term constant" is that they may contain selectors.

The main purpose of introducing selectors is to facilitate the definition of a meta operation. The *composition* between a term P and an n-ary term constant $C_{[n]}$, notation $P{\cdot}C_{[n]}$, is obtained by simultaneously substituting $C_{[n]}(1), \ldots, C_{[n]}(n)$ respectively for the selectors $1, \ldots, n$ appearing in P. The composition must be well typed in the sense that $sl(P) \subseteq [n]$, where $sl(P)$ is the set of the selectors appearing in P. When there are consecutive applications of the composition operation, association is to the left. So $P{\cdot}C{\cdot}C'$ is $(P{\cdot}C){\cdot}C'$. By definition $P{\cdot}C = P$ if $sl(P) = \emptyset$. According to definition $(pX){\cdot}C = pX$ and $(p\epsilon){\cdot}C = p\epsilon$. This makes sense since none of the descendants of pX contains any selector. From now on we will omit the composition operator "$\cdot$" when no confusion arises. Whenever we write pXC we should understand that pXC is a sequential process.

Definition 1. *A* recursive constant $V_{[n]}$ *is defined by an equality of the form* $V_{[n]} = (P_1, \ldots, P_n)V_{[n]}$, *more explicitly* $V_{[n]} = (P_1, \ldots, P_n){\cdot}V_{[n]}$, *such that the following statements are valid for each* $i \in [n]$*:*

1. $sl(P_i) \subseteq [n]$.
2. P_i *is a simple term. A term is said to be* simple *if it does not contain any recursive constants.*

We say that $V_{[n]}$ *is undefined at* i*, notation* $V_{[n]}(i)\uparrow$*, if* $V_{[n]}(i) = i$.

The single equality $V_{[n]} = (P_1, \ldots, P_n)V_{[n]}$ stands for n grammar equalities $V_{[n]}(1) = P_1V_{[n]}, \ldots, V_{[n]}(n) = P_nV_{[n]}$. The constant $V_{[n]}$ can be imagined as a stack that has recursive behaviour. The simplest n-ary recursive constant $I_{[n]}$ is defined by $I_{[n]} = (1, \ldots, n)I_{[n]}$. According to our definition of the meta operation, $P_iV_{[n]}$ does not contain any occurrence of selector. Consequently $sl(V_{[n]}) = \emptyset$.

The semantics of the extended PDA terms is also defined by the rules given in (1).

Definition 2. *A term (constant) is* finite *if all recursive constants it contains are of the form* $V_{[n]}(i)$ *with* $V_{[n]}(i)\uparrow$*. An extended PDA process* P *is an extended PDA term such that* $sl(P) = \emptyset$*. A constant* $(P_1, \ldots, P_n)$ *in an extended PDA is a term constant such that* $\bigcup_{i\in[n]} sl(P_i) = \emptyset$*.*

From now on PDA refers to the extended PDA unless otherwise specified. Accordingly PDA processes means the extended PDA processes, and a PDA constant is a term constant that does not contain any selectors.

The terminologies 'simple constant' and 'recursive constant' are introduced in Stirling's work [26]. The constants introduced in this paper are more general and are more convenient when describing decomposition property. In the sequel we will write L, M, N, O, P, Q, R, S for processes, $A_{[n]}, B_{[n]}, C_{[n]}, D_{[n]}$ for constants, $U_{[n]}$ for simple constant, and $V_{[n]}$ for recursive constant.

2.2 Decomposition

At a more intuitive level a process can be identified to a finite-branching labeled tree with an internal node labeled by pX for some $p \in \mathcal{Q}, X \in \mathcal{V}$ and a leaf labeled by either the nil process $\mathbf{0}$ or an accepting process. For the purpose of this paper we need to talk about decomposition of a process/constant at k-th level. Let P be a process and $k > 0$. The *decomposition* of P at the k-th level consists of a *finite* term $P\restriction_k$, called a *k-prefix* of P, and a constant $\downharpoonleft^k P$, called a *k-residue* of P. To help define the decomposition we define the *residue set* $\Re_k(P)$ by the following.

$$\begin{aligned}
\Re_k(\mathbf{0}) &= \emptyset, \\
\Re_k(p\epsilon) &= \emptyset, \\
\Re_k(V_{[n]}(i)) &= \emptyset, \quad \text{if } V_{[n]}(i)\uparrow, \\
\Re_k(pX(P_1, \ldots, P_n)) &= \begin{cases} \{P_1, \ldots, P_n\}, & \text{if } k = 1, \\ \bigcup_{i \in [n]} \Re_{k-1}(P_i), & \text{if } k > 1. \end{cases}
\end{aligned}$$

The operation $\Re_k(\_)$ can be applied to constants by defining $\Re_k((P_1, \ldots, P_n)) = \bigcup_{i \in [n]} \Re_k(P_i)$ and $\Re_k(V_{[n]}) = \bigcup_{i \in [n]} \Re_k(V_{[n]}(i))$. It should be clear that a residue set is finite. Suppose $\Re_k(P)$ contains m elements. Let $\jmath$ be a bijection from $[m]$ to $\Re_k(P)$. The function associates a unique number to each element of $\Re_k(P)$. Given such a bijection $\jmath$, the k-residue of P is defined by

$$\downharpoonleft^k P = (\jmath(1), \ldots, \jmath(m)).$$

The k-prefix $P\restriction_k$ of P is defined as follows, where $\jmath^{-1}$ is the inverse function of $\jmath$.

$$\begin{aligned}
\mathbf{0}\restriction_k &= \mathbf{0}, \\
p\epsilon\restriction_k &= p\epsilon, \\
V_{[n]}(i)\restriction_k &= V_{[n]}(i), \quad \text{if } V_{[n]}(i)\uparrow, \\
(pX(P_1, \ldots, P_n))\restriction_k &= \begin{cases} pX(\jmath^{-1}(P_1), \ldots, \jmath^{-1}(P_n)), & \text{if } k = 1, \\ pX(P_1\restriction_{k-1}, \ldots, P_n\restriction_{k-1}), & \text{if } k > 1. \end{cases}
\end{aligned}$$

The operation $(\_)\restriction_k$ can be applied to a constant, which is defined in the following obvious way:

$$\begin{aligned}
(P_1, \ldots, P_n)\restriction_k &= (P_1\restriction_k, \ldots, P_n\restriction_k), \\
V_{[n]}\restriction_k &= (V_{[n]}(1)\restriction_k, \ldots, V_{[n]}(n)\restriction_k).
\end{aligned}$$

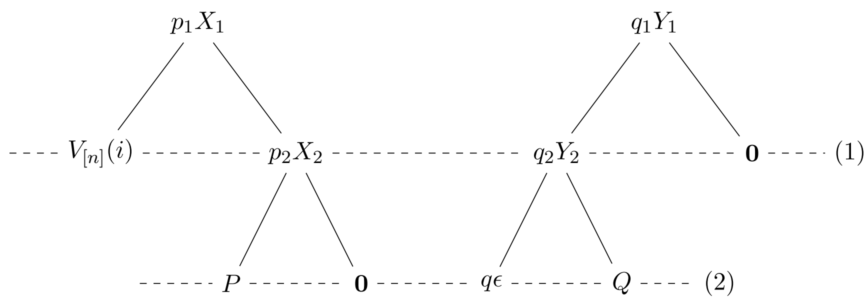
By definition $P = (P\restriction_k)(\downharpoonleft^k P)$ for some bijection $\jmath$. We can decompose a constant D to $(D\restriction_k)(\downharpoonleft^k D)$ in a similar fashion. We shall not mention any bijection when we talk about a decomposition.

The k-prefix of a process P or a constant D is at most of size k. The *size* of a finite term is defined by the following induction: (i) $|\mathbf{0}| = |p\epsilon| = |l| = 0$, (ii) $|V_{[n]}(i)| = 0$ if $V_{[n]}(i)\uparrow$, and (iii) $|pX(P_1, \ldots, P_n)| = 1 + \max_{1 \le i \le n}\{|P_i|\}$. The size of a finite term constant $(P_1, \ldots, P_n)$ is $\max\{|P_1|, \ldots, |P_n|\}$.

Lemma 1. *For each process P and constant D the sets $\bigcup_{k>0} \downharpoonleft^k P$ and $\bigcup_{k>0} \downharpoonleft^k D$ are finite.*

Proof. The set of the sub-terms of a term is finite. □

Consider a finite constant $C = (p_1X_1(V_{[n]}(i), p_2X_2(P, \mathbf{0})), q_1Y_1(q_2Y_2(q\epsilon, Q), \mathbf{0}))$ and a recursive constant $V_{[n]}$ defined in a PDA such that $V_{[n]}(i)\uparrow$. Diagrammatically C can be depicted as the following two trees, where the diagrams for P, Q are not given.



Two decompositions are given below.

(1) $C\upharpoonleft_1 = (p_1X_1(1, 2), q_1Y_1(3, 4))$ and $\downharpoonleft^1 C = (V_{[n]}(i), p_2X_2(P, \mathbf{0}), q_2Y_2(q\epsilon, Q), \mathbf{0})$;

(2) $C\upharpoonleft_2 = (p_1X_1(V_{[n]}(i), p_2X_2(1, 2)), q_1Y_1(q_2Y_2(3, 4), \mathbf{0}))$ and $\downharpoonleft^2 C = (P, \mathbf{0}, q\epsilon, Q)$.

The decompositions are indicated by the dashed lines in the above diagrams.

3 Branching Bisimilarity

The definition of branching bisimilarity is due to van Glebbeek and Weijland [32]. Our definition of branching bisimilarity for PDA is similar to Stirling's definition given in [26].

Definition 3. *A binary relation $\mathcal{R}$ on PDA terms is a branching simulation if the following statements are valid for $\mathcal{R}$:*

1. *If $P\mathcal{R}Q \xrightarrow{a} Q'$ then $P \Longrightarrow P'' \xrightarrow{a} P'\mathcal{R}Q'$ and $P''\mathcal{R}Q$ for some P', P''.*
2. *If $P\mathcal{R}Q \xrightarrow{\epsilon} Q'$ then either $P\mathcal{R}Q'$ or $P \Longrightarrow P'' \xrightarrow{\epsilon} P'\mathcal{R}Q'$ for some Q', Q'' such that $P''\mathcal{R}Q$.*
3. *If $P\mathcal{R}Q = p\epsilon$ then $P' \Longrightarrow p\epsilon$ whenever $P \Longrightarrow P'$.*
4. *If $P\mathcal{R}Q = l$ then $P' \Longrightarrow l$ whenever $P \Longrightarrow P'$.*
5. *If $P\mathcal{R}Q = V(i)\uparrow$ then $P' \Longrightarrow V(i)$ whenever $P \Longrightarrow P'$.*
6. *If $P\mathcal{R}Q = \mathbf{0}$ then $P' \Longrightarrow \mathbf{0}$ whenever $P \Longrightarrow P'$.*

The relation is a branching bisimulation *if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are branching simulation. The* branching bisimilarity $\simeq$ *is the largest branching bisimulation. Two n-ary (term) constants $C_{[n]}, D_{[n]}$ are branching bisimilar, notation $C_{[n]} \simeq D_{[n]}$, if $C_{[n]}(i) \simeq D_{[n]}(i)$ for all $i \in [n]$.*

We write $\simeq_{\mathrm{nPDA}^{\epsilon+}}$ for example for the branching bisimilarity on nPDA$^{\epsilon+}$ processes.

Proposition 1. *The relation $\simeq$ is both an equivalence relation and a congruence relation.*

Proof. It is well-known that the properties 1 and 2 of Definition 3 are transitive. Now suppose $\mathbf{0} \simeq R \simeq Q \Longrightarrow Q'$. Then $R \Longrightarrow R' \simeq P'$ for some R' by condition 2. It follows that $P' \Longrightarrow P'' \simeq \mathbf{0}$ for some P''. By condition 6, $R' \Longrightarrow \mathbf{0}$. Hence $P'' \Longrightarrow \mathbf{0}$. Conclude that 6 is transitive. Similarly properties 3, 4 and 5 are transitive. The congruence property is easy to prove using the ϵ-pushing property and properties 3 through 6. □

Definition 3 forces $V(i) \not\simeq \mathbf{0}$, which might appear too strong. It does however simplify the equivalence checking algorithm. Notice that conditions 4 through 6 do not have any effect on our results since the translation of the classical PDA into our PDA involves neither recursive constants nor selectors nor $\mathbf{0}$. Consequently two classical PDA processes are branching bisimilar if and only if their translations are equivalent in the sense of Definition 3.

A technical lemma that plays an important role in the study of branching bisimilarity is the Computation Lemma [32,3].

Lemma 2. *If $P_0 \xrightarrow{\epsilon} P_1 \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} P_k \simeq P_0$, then $P_0 \simeq P_1 \simeq \ldots \simeq P_k$.*

A silent transition $P \xrightarrow{\epsilon} P'$ is *state-preserving*, notation $P \to P'$, if $P \simeq P'$. It is a *change-of-state*, notation $P \xrightarrow{\iota} P'$, if $P \not\simeq P'$. We write $\to^*$ 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 $P \xrightarrow{\epsilon} P'$. 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 $Q_0 \xrightarrow{\epsilon} Q_1 \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} Q_k \xrightarrow{\jmath} Q_{k+1}$, then $Q_0 \to Q_1 \to \ldots \to Q_k$. This property of the branching bisimilarity will be used extensively.

The next lemma is basic to our decidability algorithm.

Lemma 3. *In* nPDA$^{\epsilon+}$ *the problem $\{P \mid \exists Q.(|Q| = 0) \wedge (P \simeq Q)\}$ is decidable.*

Proof. In nPDA$^{\epsilon+}$, $P \simeq Q$ for some $|Q| = 0$ if and only if $P = Q$. This is because that in nPDA$^{\epsilon+}$ silent transitions do not decrease the size of terms. □

Given a PDA process P, the *norm* of P, denoted by $\|P\|$, is a function from $\mathbb{N}$ to $\mathbb{N} \cup \{\perp\}$, where $\perp$ stands for undefinedness, such that the following holds:

- $\|P\|(h) = \perp$ if and only if there does not exist any ℓ^* such that $P \xrightarrow{\ell^*} p_h\epsilon$.
- $\|P\|(h)$ is the least number i such that $\exists \jmath_1 \ldots \jmath_i.\ P \to^* \xrightarrow{\jmath_1} \ldots \to^* \xrightarrow{\jmath_i} \to^* p_h\epsilon$.

The set $\mathrm{def}\,\|P\| = \{h \mid \|P\|(h) \neq \perp\}$ is finite. A process P is *normed* if $\mathrm{def}\,\|P\| \neq \emptyset$. It is *unnormed* otherwise. For normed process P we introduce the following notations.

$$\min \|P\| = \min\{\|P\|(h) \mid h \in \mathrm{def}\,\|P\|\},$$
$$\max \|P\| = \max\{\|P\|(h) \mid h \in \mathrm{def}\,\|P\|\}.$$

We shall use the following convention in the rest of the paper.

$$\mathfrak{r} = \max \left\{ |\eta| \;\middle|\; pX \xrightarrow{\ell} q\eta \in \mathcal{R} \text{ for some } p, q \in \mathcal{Q}, X \in \mathcal{V} \right\},$$
$$\mathfrak{m} = \max \left\{ \max \|pX\| \mid p \in \mathcal{Q}, X \in \mathcal{V} \right\}.$$

The values $\mathfrak{r}$ and $\mathfrak{m}$ can be effectively calculated. By definition $\|pX\|(i) \leq \mathfrak{m}$ for all p, X and all $i \in \mathrm{def}\,\|pX\|$.

3.1 Bisimulation Game

In the proofs to be given later we need to use the game theoretical interpretation of bisimulation. A *bisimulation game* [28,15] for a pair of processes (P_0, P_1), called a *configuration*, is played between Attacker and Defender in an alternating fashion. It is played according to the following rules: Suppose (P_0, P_1) is the current configuration.

- $|P_i| > 0$ or $P_i = \mathbf{0}$ for each $i \in \{0, 1\}$.
  1. Attacker picks up some P_i, where $i \in \{0, 1\}$, to start with and chooses some $P_i \xrightarrow{\ell} P_i'$.
  2. Defender must respond in the following manner:
     (a) Do nothing. This option is available if $\ell = \epsilon$.
     (b) Choose a transition sequence $P_{1-i} \xrightarrow{\epsilon} P_{1-i}^1 \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} P_{1-i}^{k-1} \xrightarrow{\ell} P_{1-i}^k$.
  3. If case 2(a) happens the new configuration is (P_i', P_{1-i}). If case 2(b) happens Attacker chooses one of $\{(P_i, P_{1-i}^1), \ldots, (P_i, P_{1-i}^{k-1}), (P_i', P_{1-i}^k)\}$ as the new configuration.
  4. The game continues with the new configuration.
- $P_i = p\epsilon$ or $P_i = l$ or $P_i = V_{[n]}(j)$ with $V_{[n]}(j)\uparrow$ for either $i = 0$ or $i = 1$.
  1. Attacker chooses some $P_{i-1} \Longrightarrow P_{i-1}'$ for some P_{i-1}'.
  2. Defender must respond with $P_{i-1}' \Longrightarrow P_i$.

Attacker wins a bisimulation game if Defender gets stuck in the game. Defender wins a bisimulation game if Attacker cannot win the game. Attacker/Defender has a winning strategy if it can win no matter how its opponent plays. The effectiveness of the bisimulation game is enforced by the following lemma.

Lemma 4. *$P \simeq Q$ if and only if Defender has a winning strategy for the bisimulation game starting with the configuration (P, Q).*

The above lemma is the basis for game theoretical proofs of process equality. It is also the basis for game constructions using Defender's Forcing.

4 Finite Branching Property

Generally bisimilarity is undecidable for models with infinite branching transitions. For the branching bisimilarity the finite branching property can be described by the following statement:

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

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 definition that the members of a clique are branching bisimilar. We can remove a maximal clique $\mathcal{S}$ in two steps.

1. Remove every rule of the form $pX \xrightarrow{\epsilon} qY$ such that $pX, qY \in \mathcal{S}$.
2. For each $pX \in \mathcal{S}$ introduce the rule $pX \xrightarrow{\lambda} P$ whenever $qY \in \mathcal{S}$ that is distinct from pX and the rule $qY \xrightarrow{\lambda} P$ has not been removed in the previous 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. 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 $qX \xrightarrow{\epsilon} q_1X_1 \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} q_kX_k$ is less than $\mathfrak{nq}$.

Lemma 5. *In* nPDA$^{\epsilon+}$, $|\alpha| \leq \min \|p\alpha C\|$ *holds for all* $p\alpha C$.

Proof. Only an external action can remove a symbol from an nPDA$^{\epsilon+}$ process. $\square$

Using the simple property stated in Lemma 5, one can show that there is a constant bound for the length of the state-preserving transitions in nPDA$^{\epsilon+}$.

Lemma 6. *If* $qXC \to q_1\beta_1C \to \ldots \to q_k\beta_kC$ *for an* nPDA$^{\epsilon+}$ *process* qXC, *then* $k < \mathfrak{qnr}(\mathfrak{m}+1)^{\mathfrak{q}}$.

Proof. Now suppose $qXC \to q_1Z_1\delta_1C$. Let $k_1 = \min \|q_1Z_1\delta_1C\|$ and let

$$q_1Z_1\delta_1C \to^* \xrightarrow{\jmath^1_1} \ldots \to^* \xrightarrow{\jmath^1_{j_1}} \to^* r_1\epsilon C \to^* \xrightarrow{\jmath^1_{j_1+1}} \ldots \to^* \xrightarrow{\jmath^1_{j_{k_1}}} \to^* p_{h_1}\epsilon$$

be a transition sequence of minimal length that empties the stack. Clearly $j_1 \leq \mathfrak{rm}$. Now suppose $q_1Z_1\delta_1C \to^* q_2Z_2\delta_2\delta_1C$ such that

$$\mathfrak{rm} < |Z_2\delta_2\delta_1| \leq \mathfrak{r}(\mathfrak{m}+1). \tag{2}$$

Let $k_2 = \min \|q_2Z_2\delta_2\delta_1C\|$ and let

$$q_2Z_2\delta_2\delta_1C \to^* \xrightarrow{\jmath^2_1} \ldots \to^* \xrightarrow{\jmath^2_{j_2}} \to^* r_2\epsilon C \to^* \xrightarrow{\jmath^2_{j_2+1}} \ldots \to^* \xrightarrow{\jmath^2_{j_{k_2}}} \to^* p_{h_2}\epsilon$$

be a transition sequence of minimal length that empties the stack. One must have $j_2 > j_1$ according to (2). By iterating the above argument one gets from

$$\begin{aligned} q_1Z_1\delta_1C &\to^* q_2Z_2\delta_2\delta_1C \\ &\to^* \ldots \\ &\to^* q_{i+1}Z_{i+1}\delta_{i+1}\delta_i \ldots \delta_1C \\ &\to^* \ldots \\ &\to^* q_{\mathfrak{q}+1}Z_{\mathfrak{q}+1}\delta_{\mathfrak{q}+1}\delta_{\mathfrak{q}} \ldots \delta_1C \end{aligned}$$

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

$$q_iZ_i\delta_i \ldots \delta_1C \to^* \xrightarrow{\jmath^i_1} \ldots \to^* \xrightarrow{\jmath^i_{j_i}} \to^* r_i\epsilon C \to^* \xrightarrow{\jmath^i_{j_i+1}} \ldots \to^* \xrightarrow{\jmath^i_{j_{k_i}}} \to^* p_{h_i}\epsilon$$

where $k_i = \min \|q_iZ_i\delta_i \ldots \delta_1C\|$. 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 $q_{t_1}Z_{t_1}\delta_{t_1} \ldots \delta_1C \simeq q_{t_2}Z_{t_2}\delta_{t_2} \ldots \delta_1C$. We conclude that if $qXC \to^* q'\gamma C$ then $|\gamma| < \mathfrak{r}(\mathfrak{m}+1)^{\mathfrak{q}}$. It follows from our convention that $k < \mathfrak{qnr}(\mathfrak{m}+1)^{\mathfrak{q}}$. $\square$

A proof of the following corollary can be read off from the above proof.

Corollary 1. *Suppose P is an* nPDA$^{\epsilon+}$ *process. There is a computable bound on the size of any* nPDA$^{\epsilon+}$ *process $p\alpha$ such that $p\alpha \simeq P$.*

Using Lemma 6 one can define for nPDA$^{\epsilon+}$ the approximation relation $\simeq_i$, the branching bisimilarity up to depth $i \geq 0$.

Definition 4. *Let $\simeq_0$ be the total relation on* nPDA$^{\epsilon+}$ *terms. The relation $\simeq_{k+1}$ on* nPDA$^{\epsilon+}$ *terms is defined as follows: $P \simeq_{k+1} Q$ if the following statements are valid:*

1. *If $Q \xrightarrow{a} Q'$ then $P \xrightarrow{\tau} P_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_j \xrightarrow{a} P' \simeq_k Q'$ for some $P_1, \ldots, P_j, P'$ such that $P_i \simeq_k Q$ for all $i \in [j]$.*
2. *If $Q \xrightarrow{\epsilon} Q'$ then either $P \simeq_k Q'$ or $P \xrightarrow{\tau} P_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_j \xrightarrow{\epsilon} P' \simeq_k Q'$ for some $P_1, \ldots, P_j, P'$ such that $P_i \simeq_k Q$ for all $i \in [j]$.*
3. *If $|Q| = 0$ then $P' \Longrightarrow Q$ whenever $P \Longrightarrow P'$.*
4. *Symmetric statements of 1 through 3.*

The infinite approximation $\simeq_0 \supseteq \simeq_1 \supseteq \simeq_2 \supseteq \ \ldots$ approaches to $\simeq$. The proof of the next lemma is standard using Lemma 6.

Lemma 7. *The intersection $\bigcap_{i \geq 0} \simeq_i$ coincides with $\simeq$ on* nPDA$^{\epsilon+}$ *processes.*

The following theorem follows from Lemma 7 and the fact that $\not\simeq_i$ is decidable for all $i \geq 0$.

Theorem 1. *The relation $\not\simeq_{\mathrm{nPDA}^{\epsilon+}}$ is semidecidable.*

5 Decidability of nPDA$^{\epsilon+}$

The basic idea of our proof of the decidability of nPDA$^{\epsilon+}$ is drawn from Stirling's proof for the strong bisimilarity on PDA [26,27,29]. To explain the key technical tool of Stirling's proof, it is helpful to recall the proof of the semidecidability of the strong bisimilarity of BPA [10]. To check if $X\alpha \sim Y\beta$, we decompose the goal $X\alpha = Y\beta$ into say subgoals $\alpha = \gamma\beta$ and $X\gamma\beta = Y\beta$ derivable from the bisimulation property. The latter can be simplified to $X\gamma = Y$ by cancellation. Now the size of γ is small as it were because γ is derived in a computationally bounded number of steps. It follows that the subgoal $X\gamma = Y$ is small and the subgoal $\alpha = \gamma\beta$ is smaller than $X\alpha = Y\beta$ in the sense that α is smaller than $X\alpha$ and the size of γ is under control. Using this 'smallness' property we can build a finite tree of subgoals, called a tableau, in an organized fashion. A semidecidable procedure is then designed by enumerating all finite tableaux and checking if any one of them giving rise to a strong bisimulation. The unnormed BPA processes enjoy the following weak cancellation property: If there is an infinite family of pairwise nonbisimilar BPA processes $\{\delta_i\}_{i \in \mathbb{N}}$ such that $\alpha\delta_i \sim \beta\delta_i$ for all $i \in \mathbb{N}$, then $\alpha \sim \beta$. This weak cancellation guarantees that in a tree of subgoals there cannot be a path containing an infinite number of subgoals $\{\alpha\delta_i \sim \beta\delta_i\}_{i \in \mathbb{N}}$, where $\alpha \not\sim \beta$, without producing equivalent subgoals. It follows from König Lemma that only finite tableaux are produced. So the same semidecidable procedure works for the unnormed BPA.

Given BPA processes α, β with $\alpha \not\sim \beta$, we say that $\{\gamma_i\}_{i \in I}$ is a *minimal set of fixpoints* for α, β if the following hold:

- For each $i \in I$ the process γ_i is a fixpoint for α, β, i.e. $\alpha\gamma_i \sim \beta\gamma_i$.

– For all $i, j \in I$ if $i \neq j$ then $\gamma_i \not\sim \gamma_j$.
– $\alpha\gamma \sim \beta\gamma$ if and only if $\alpha\gamma_i \sim \beta\gamma_i$ for some $i \in I$.

Both the strong and the weak cancellation properties of BPA can be reiterated in the following more enlightening manner.

Lemma 8. *Let α, β be BPA processes. If $\alpha \not\sim \beta$ then the minimal set of fixpoints for α, β is finite.*

The property described in Lemma 8, called the finite representation property in this paper, is the prime reason for the semidecidability of $\sim$ on BPA. Stirling's remarkable observation is that the property described in Lemma 8 is also valid for the strong bisimilarity on PDA. What is subtle about PDA is that the fixpoints are stacks rather than processes due to the nonstructural definition of PDA processes. In fact they must be extended stacks if they are able to code up recursive behaviours, hence the recursive constants.

What we will prove in this section is that the property described in Lemma 8 continues to be valid for the branching bisimilarity on nPDA$^{\epsilon+}$ and that the cancellation property stated in the lemma is sufficient for us to design a semidecidable procedure for the equivalence checking problem.

Throughout this section we assume the following on recursive constants: For every recursive constant $V_{[n]}$ and every $i \in [n]$, $V_{[n]}(i) = i$ whenever $V_{[n]}$ is a selector.

5.1 Finite Representation

Lemma 9. *If $pXA \simeq MD$ and $|M| = \mathfrak{m}$, then there is a simple constant $U_{[\mathfrak{q}]}$ such that $|U_{[\mathfrak{q}]}| < \mathfrak{q}\mathfrak{n}\mathfrak{r}^2(\mathfrak{m}+1)^{(\mathfrak{q}+1)}$ and $A(i) \simeq U_{[\mathfrak{q}]}(i)D$ for all $i \in \mathrm{def}\|pX\|$.*

Proof. If $i \notin \mathrm{def}\|pX\|$ we let $U_{[\mathfrak{q}]}(i)$ be $\mathbf{0}$. Otherwise let

$$pXA \to^* \xrightarrow{\jmath_1} \to^* \ldots \to^* \xrightarrow{\jmath_k} \to^* A(h) \tag{3}$$

be a sequence reaching $A(h)$ with minimal k. Since $\simeq$ is closed under composition, k cannot be greater than $\mathfrak{m}$. The action sequence (3) must be bisimulated by MD in the following manner:

$$MD \to^* \xrightarrow{\jmath_1} Q_1 D \to^* \xrightarrow{\jmath_2} Q_2 D \ldots \to^* \xrightarrow{\jmath_k} Q_h D. \tag{4}$$

Since M is thick enough as it were, D remains intact throughout the transitions in (4). By Lemma 6 the length of the sequence (4) is bounded by $\mathfrak{q}\mathfrak{n}\mathfrak{r}(\mathfrak{m}+1)^{\mathfrak{q}}\mathfrak{m} < \mathfrak{q}\mathfrak{n}\mathfrak{r}(\mathfrak{m}+1)^{(\mathfrak{q}+1)}$. It follows that $|Q_h| < \mathfrak{q}\mathfrak{n}\mathfrak{r}^2(\mathfrak{m}+1)^{(\mathfrak{q}+1)}$. Let $U_{[\mathfrak{q}]}(h) = Q_h$. It is clear that $A(i) \simeq U_{[\mathfrak{q}]}(i)D$ for all $i \in \mathrm{def}\|pX\|$. □

We now establish for nPDA$^{\epsilon+}$ the finite representation property. In the following lemma the equivalence (5) is the fixpoint property while the equivalence (6) is the minimality property.

Lemma 10. *Let P, Q be finite nPDA$^{\epsilon+}$ terms satisfying $|P| > 0$ and $|Q| > 0$, $sl(P), sl(Q) \subseteq [n]$ and $\{C_{[n]} \mid PC_{[n]} \simeq QC_{[n]}\} \neq \emptyset$. A finite set of recursive constant $\left\{V^k_{[n]} = (L^k_1, \ldots, L^k_n)V^k_{[n]}\right\}_{k \in K}$ exists such that*

$$PV^k_{[n]} \simeq QV^k_{[n]} \tag{5}$$

for all $k \in K$ and for each $D_{[n]}$ satisfying $PD_{[n]} \simeq QD_{[n]}$ there is some $k \in K$ rendering true the following equivalence.

$$D_{[n]} \simeq (L^k_1, \ldots, L^k_n)D_{[n]}. \tag{6}$$

Proof. Suppose $PD_{[n]} \simeq QD_{[n]}$. We will construct $V^k_{[n]}$ by induction such that at each step (6) is maintained. Let V^0 be $I_{[n]}$. Thus V^0 is defined by $V^0 = (1, \ldots, n)V^0$. The finite constant $(1, \ldots, n)$ trivially validates (6). If it also satisfies (5), we are done. Otherwise we refine V^0 to some V^1 by the following induction. Suppose $V^d = (L^d_1, \ldots, L^d_n)V^d$ has been constructed such that

$$\begin{aligned} &PV^d \not\simeq QV^d, \\ &D_{[n]} \simeq (L^d_1, \ldots, L^d_n)D_{[n]}. \end{aligned} \tag{7}$$

Let m be the least number such that $PV^d \not\simeq_m QV^d$. We refine V^d to V^{d+1} by exploring the mismatch between the following equality and inequality:

$$PD_{[n]} \;\simeq\; QD_{[n]}, \tag{8}$$

$$PV^d \not\simeq_m QV^d. \tag{9}$$

It follows from (9) that some transition $PV^d \xrightarrow{\jmath} P'V^d$ exists such that for all transition sequence $QV^d \xrightarrow{\epsilon} O_1V^d \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} O_eV^d \xrightarrow{\jmath} O'V^d$ at least one of the following inequalities is valid:

$$\begin{aligned} PV^d &\not\simeq_{m-1} O_1V^d, \\ &\ldots \\ PV^d &\not\simeq_{m-1} O_{e-1}V^d, \\ PV^d &\not\simeq_{m-1} O_eV^d, \\ P'V^d &\not\simeq_{m-1} O'V^d. \end{aligned} \tag{10}$$

According to (8) however the transition $PD_{[n]} \xrightarrow{\jmath} P'D_{[n]}$ must be matched by some transition sequence $QD_{[n]} \xrightarrow{\epsilon} Q_1D_{[n]} \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} Q_{e'}D_{[n]} \xrightarrow{\jmath} Q'D_{[n]}$ such that $PD_{[n]} \simeq Q_1D_{[n]}$, $\ldots$, $PD_{[n]} \simeq Q_{e'}D_{[n]}$ and

$$P'D_{[n]} \simeq Q'D_{[n]}. \tag{11}$$

Suppose for this particular transition sequence, the valid inequality of (10) is

$$P'V^d \not\simeq_{m-1} Q'V^d. \tag{12}$$

The above construction takes us from (8,9) to (11,12). By repeating the construction, we eventually get the following equality and inequality for some O with $|O| = 0$ and some L:

$$OD_{[n]} \;\simeq\; LD_{[n]}, \tag{13}$$

$$OV^d \not\simeq_{m'} LV^d. \tag{14}$$

Without loss of generality, we may assume that L may not perform any state-preserving silent transitions. We continue the construction by examining the shape of L.

1. $|L| > 0$. In this case O must be a selector, say i; otherwise (13) would be violated. Let $V^{d+1} = (L^d_1, \ldots, L^d_{i-1}, L, \overline{L^d_{i+1}}, \ldots, \overline{L^d_n})V^{d+1}$, where for each $j \in \{i+1, \ldots, n\}$ the process $\overline{L^d_j}$ is defined as follows: If there are $j_1 > \ldots > j_g > i$ such that $V^d(j) = j_1$, $V^d(j_1) = j_2$, $\ldots$, $V^d(j_g) = i$, then $\overline{L^d_j} = L$; otherwise $\overline{L^d_j} = L^d_j$. Now V^{d+1} trivially validates the equivalence (7).

2. $|L| = 0$ and $O = L$. This is impossible because of (14).
3. $|L| = 0$ and $O \neq L$ and neither O nor L is a selector. This is impossible because of (13).
4. O is a selector and L is not a selector. Carry out the update defined in Case 1.
5. $O = i \in [n]$ and $L = j \in [n]$. If $i < j$ then let

$$V^{d+1} = (L_1^d, \ldots, L_{j-1}^d, i, L_{j+1}^d, \ldots, L_n^d)V^{d+1}.$$

Otherwise let

$$V^{d+1} = (L_1^d, \ldots, L_{i-1}^d, j, L_{i+1}^d, \ldots, L_n^d)V^{d+1}.$$

Clearly V^{d+1} validates (7). Notice that (14) implies $i \neq j$.

The construction must stop after at most $\frac{n(n-1)}{2}$ steps. Eventually we get some $V = (L_1, \ldots, L_n)V$. Modify the definition of V as follows: For each $i \in [n]$ let $V(i) = i$ if $V(i)$ is a number. What we get is the required $V_{[n]}^k$. Starting with $I_{[n]}$ there are only finitely many such $V_{[n]}^k$ one can construct in $\frac{n(n-1)}{2}$ steps due to the finite branching property. We are done. $\square$

Lemma 9 allows one to create common suffix by introducing a constant, whereas Lemma 10 helps to substitute a recursive constant for the suffix. We get a more useful result if we combine these two lemmas.

Lemma 11. *Suppose $pXAC_{[n]} \simeq MC_{[n]}$ for some $A, C_{[n]}$ such that $|M| \geq \mathfrak{m}$. A finite family $\left\{V_{[n]}^k = (L_1^k, \ldots, L_n^k)V_{[n]}^k\right\}_{k \in K}$ of recursive constant exists such that for every pair $(A, D_{[n]})$ satisfying $sl(A) \subseteq [n]$ and $pXAD_{[n]} \simeq MD_{[n]}$, there is some $k \in K$ rendering true the following.*

$$pXAV_{[n]}^k \simeq MV_{[n]}^k, \tag{15}$$
$$D_{[n]} \simeq (L_1^k, \ldots, L_n^k)D_{[n]}. \tag{16}$$

Proof. By the proof of Lemma 9 there is a finite set $\left\{U^j = (G_1^j, \ldots, G_{\mathfrak{q}}^j)\right\}_{j \in J}$ such that for each pair $A, D_{[n]}$ with $sl(A) \subseteq [n]$ and $pXAD_{[n]} \simeq MD_{[n]}$ there is some $U^j = (G_1^j, \ldots, G_{\mathfrak{q}}^j)$ validating the following.

$$pXU^jD_{[n]} \simeq MD_{[n]}, \tag{17}$$
$$A(i)D_{[n]} \simeq G_i^jD_{[n]} \text{ for every } i \in \mathrm{def}\|pX\|. \tag{18}$$

For each $j \in J$ let ∇^j be the set of pairs $A, D_{[n]}$ that satisfy $sl(A) \subseteq [n]$ and $pXAD_{[n]} \simeq MD_{[n]}$ and (17) and (18). It follows from (17) and Lemma 10 that there is a finite family of recursive constants $\left\{V_{[n]}^k = (L_1^k, \ldots, L_n^k)V_{[n]}^k\right\}_{k \in K}$ such that for each pair $A, D_{[n]}$ in ∇^j there is some $k \in K$ rendering true the following.

$$pXU^jV_{[n]}^k \simeq MV_{[n]}^k, \tag{19}$$
$$D_{[n]} \simeq (L_1^k, \ldots, L_n^k)D_{[n]}. \tag{20}$$

It remains to show $pXAV_{[n]}^k \simeq MV_{[n]}^k$, and by (19) it is sufficient to show

$$A(i)V_{[n]}^k \simeq G_i^jV_{[n]}^k \text{ for every } i \in \mathrm{def}\|pX\|. \tag{21}$$

Decmp$^{\epsilon+}$	$\dfrac{rXA = MD}{\{A(i) = G_iD\}_{i\in \mathrm{def}\,\|rX\|} \quad rXUD = MD}$	$0 < \|M\| \leq \mathfrak{m}$; $U = (G_1, \ldots, G_{\mathfrak{q}})$, $\|U\| < \mathfrak{q}\mathfrak{n}\mathfrak{r}^2(\mathfrak{m}+1)^{(\mathfrak{q}+1)}$.
Cancel$^{\epsilon+}$	$\dfrac{NCD_{[n]} = MD_{[n]}}{\{L_iD_{[n]} = D_{[n]}(i)\}_{i\in[n]} \quad NCV_{[n]} = MV_{[n]}}$	$\|N\| > 0$, $\|M\| > 0$; $V_{[n]} = (L_i)_{i\in[n]}\, V_{[n]}$.
Match$^{\epsilon+}$	$\dfrac{P = Q}{P_1 = Q_1 \ \ldots \ P_l = Q_l}$	$\{P_1 = Q_1, \ldots, P_l = Q_l\}$ is a match for $P = Q$.

Fig. 3. Tableau Rules for nPDA$^{\epsilon+}$

Now for each $i \in \mathrm{def}\|pX\|$, consider the bisimulation game of $A(i)V^k_{[n]} \simeq G^j_i V^k_{[n]}$. The Defender simply copycats the Defender's strategy of the game (18), invoking the Defender's strategy of the game (20) whenever necessary. What we have described is a winning strategy for the Defender. We conclude that (21) is valid. □

What Lemma 11 says is that the order of an application of Lemma 9 followed by an immediate application of Lemma 10 can be swapped without sacrificing the finite representation property. The reordering is important in guaranteeing the termination of of our equivalence checking algorithms.

5.2 Tableau System

A straightforward way to prove bisimilarity between two processes is to construct a finite binary relation containing the pair of processes that can be extended to a bisimulation. Such a finite relation is called a bisimulation base, originally due to Caucal [2]. The tableau approach [10,8] can be seen as an effective way of generating a bisimulation base. Lemma 9 and Lemma 11 suggest the first two tableau rules for nPDA$^{\epsilon+}$ given in Figure 3. To define the third tableau rule, we need the notion of *match*. A match for an equality $P = Q$ is a *finite* set $\{P_i = Q_i\}_{i=1}^k$ containing those and only those equalities accounted for in the following statements:

1. For each transition $P \xrightarrow{\ell} P'$, one of the following holds:
   - $\ell = \epsilon$ and $P' = Q \in \{P_i = Q_i\}_{i=1}^k$;
   - there is a transition sequence $Q \xrightarrow{\epsilon} Q_1 \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} Q_n \xrightarrow{\ell} Q'$ such that $\{P = Q_1, \ldots, P = Q_n, P' = Q'\} \subseteq \{P_i = Q_i\}_{i=1}^k$.
2. For each transition $Q \xrightarrow{\ell} Q'$, one of the following holds:
   - $\ell = \epsilon$ and $P = Q' \in \{P_i = Q_i\}_{i=1}^k$;
   - there is a transition sequence $P \xrightarrow{\epsilon} P_1 \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} P_n \xrightarrow{\ell} P'$ such that $\{P_1 = Q, \ldots, P_n = Q, P' = Q'\} \subseteq \{P_i = Q_i\}_{i=1}^k$.

We remark that a match could be empty.

Decmp$^{\epsilon+}$ decomposes the left hand side of the goal into a form that matches the right hand side, creating a common suffix. Cancel$^{\epsilon+}$ harnesses the complexity by introducing a recursive subgoal. The Match$^{\epsilon+}$ rule is standard. We note a simple observation that if both side of a goal has a recursive constant V as their tail, then V either persist in its subgoals or it is replaced by another recursive constant.

Lemma 12. *All the rules defined in Fig. 3 are both sound and backward sound with respect to $\simeq$.*

Proof. The soundness of Decmp$^{\epsilon+}$ and Cancel$^{\epsilon+}$ is guaranteed by Lemma 9 and Lemma 11 respectively. The soundness and the backward soundness of Match$^{\epsilon+}$ are by definition. The backward soundness of Decmp$^{\epsilon+}$ is by Proposition 1. To show that Cancel$^{\epsilon+}$ is backward sound, we use the fact that if in PC the constant C is not exposed, then no silent transitions from PC can expose C. Suppose there is a recursive constant $V_{[n]} = (L_1, \ldots, L_n)V_{[n]}$ such that

$$L_i D_{[n]} \simeq D_{[n]}(i) \quad \forall i \in [n] \tag{22}$$

$$NCV_{[n]} \simeq MV_{[n]} \tag{23}$$

We show the following equivalence using a game theoretical argument.

$$NCD_{[n]} \simeq MD_{[n]} \tag{24}$$

In the branching bisimulation game of (24) no matter how Attacker plays Defender mimics a winning strategy of the game (23). Whenever a configuration of the form $(kD_{[n]}, PD_{[n]})$ is reached, the game of (23) must reach the configuration $(L_k V_{[n]}, PV_{[n]})$ such that

$$L_k V_{[n]} \simeq PV_{[n]}. \tag{25}$$

Defender then applies the strategy that composes the winning strategies of (25) and (22), which is sound by transitivity (Proposition 1). In this way the game either continues forever or reaches a configuration that Attacker cannot play. What we have described is a winning strategy for Defender of the game of (24).

5.3 Subtableau

A subtableau is a building block for tableau. Its chief role is to help to reduce a goal to a *finite* number of subgoals of controlled size. A goal can be reduced in many ways. It is important that there are only a finite number of subtableaux that can be constructed for every goal. A subtableau is manufactured using a strategy that applies Dcmp$^{\epsilon+}$ and Cancel$^{\epsilon+}$ in an orderly manner. Notice that Match$^{\epsilon+}$ is *never* used in the construction of any *sub*tableaux. The strategy is described by the *non*deterministic algorithm defined in Fig. 4. The construction of a subtableau must meet the following condition:

(‡) For each pair of finite terms P, Q a *unique* constant V is introduced such that $PV = QV$.

The construction of a branch of a subtableau ends in either a leaf, or a small goal, or a potentially successful node. A node labelled $P = Q$ is a leaf if $|P| = 0 \vee |Q| = 0$. A leaf labeled by $P = Q$ is successful if $P \simeq Q$ and is unsuccessful if $P \not\simeq Q$. A node labelled $P = Q$ is a small goal if the size of both P and Q is bounded by some number that is computable from the definition of the PDA and the recursive constants nondeterministically guessed at the beginning of the algorithm. Fig. 5 gives a diagrammatic illustration of the algorithm. Our definition of leaves is justified by Lemma 3.

The key question about the construction of a subtableau using the above strategy is if it always terminates. This is answered by the next lemma.

Lemma 13. *Every subtableau is finite.*

Guess a finite set of recursive constants.

1. Apply $\mathrm{Decmp}^{\epsilon+}$ to $rXA = MD$. We get two classes of subgoals.
   (a) $A(i) = G_i D$. If either $|A(i)| = 0$ or $|G_i D| = 0$, it is a leaf; otherwise there are two cases.
      i. A is a recursive constant and there is an ancestor of the form $A(i) = G'_i D'$. In this case reduce $A(i) = G_i D$ to the subgoal $G'_i D' = G_i D$. Now $G'_i D' = G_i D$ must be of the form $G_1 D'' = G_2 D''$, where D'' is the common suffix of $G'_i D'$ and $G_i D$.
         A. $|G_1| \leq \mathfrak{m}$. Reduce $G_1 D'' = G_2 D''$ to $G_2 D'' = G_1 D''$. Go to Step 1(a)iB.
         B. $|G_2| \leq \mathfrak{m}$. Let $M_1 = (G_2 D'')\lceil_{\mathfrak{m}}$ and $D_1 = \rfloor^{\mathfrak{m}}(G_2 D'')$. Accordingly $G_1 D'$ is decomposed as some $r' X' C_1 D_1$. Go to Step 2.
         C. If neither Case 1(a)iA nor Case 1(a)iB applies, then repeatedly apply $\mathrm{Decmp}^{\epsilon+}$ until Case 1(a)iA or Case 1(a)iB applies.
      ii. Otherwise repeat Step 1 inductively.
   (b) $rXUD = MD$. Go to Step 2.
2. Apply $\mathrm{Cancel}^{\epsilon+}$ to $rXCD = MD$, where C has a computable bound. Two types of subgoals are generated.
   (a) $L_i D = D(i)$. Go to Step 3.
   (b) $rXCV = MV$. This is a small goal.
3. $L_i D = D(i)$. There are following subcases.
   (a) $|L_i| = 0$. This is a leaf.
   (b) $|L_i| > 0$ and $|D| \leq \mathfrak{m}$. We take $L_i D = D(i)$ as a small goal.
   (c) $|L_i| > 0$ and $|D| > \mathfrak{m}$ and $L_i D = D(i)$ does not coincide with any of its ancestors. Guess a decomposition of $D(i)$, say $D(i) = M^0 D'$, such that $0 < |M^0| \leq \mathfrak{m}$. Let D^2 be defined by $D^2(j) = D\lceil_{\mathfrak{m}}(j)$ if $j \neq i$ and $D^2(i) = M^0$. Suppose $D = D^2 D^1$. It is clear that $D(i) = M^0 D^1$. Now apply $\mathrm{Cancel}^{\epsilon+}$ to $L_i D^2 D^1 = M^0 D^1$. We get two types of subgoal.
      i. $L_i D^2 V' = M^0 V'$. This is a small goal.
      ii. $L^1_j D^1 = D^1(j)$. Repeat Step 3 inductively.
   (d) $|L_i| > 0$ and D coincides with one of its ancestors. This is a small goal.

Fig. 4. A Nondeterministic Algorithm for Constructing Subtableaux in $\mathrm{nPDA}^{\epsilon+}$

Proof. We argue that each of the three steps in the algorithm can only be executed for a finite number of times.

– Step 1 is crucial for the termination of the algorithm. If the left hand side can be reduced to a process of size 0, then it is a leaf; otherwise Case 1(a)i must apply. Case 1(a)iA is reduced to Case 1(a)iB. The termination of the latter depends on the termination of Step 2. Recursive invocation of 1(a)iC must terminate because the left hand side keeps shrinking. The crucial observation is that since the distance between $A(i) = G_i D$ and $A(i) = G'_i D'$ has a computable bound, both $|G_i|$ and $|G'_i|$ have computable bound. So the number of steps 1(a)iC executes is computationally bounded. It follows that when Step 2 is invoked with $r' X' C_1 D_1 = M_1 D_1$, the size of C_1 is computationally bounded.
  Let's work out the computable bounds claimed in the above paragraph. Let $\mathfrak{c}$ be the maximum arity of the recursive constants defined in our PDA. Suppose $\{V_{[n_k]} = (L^1_{n_k}, \ldots, L^{n_k}_{n_k}) V_{[n_k]}\}_{k \in K}$ is the finite set of recursive constants. Let $\mathfrak{v}$ be defined as follows:

$$\mathfrak{v} = \max\left\{ |L^j_{n_k}| \mid k \in K,\ j \in [n_k] \right\}.$$

  The distance between $A(i) = G_i D$ and $A(i) = G'_i D'$ is bounded by $\mathfrak{c}\mathfrak{v}$.
  1. By the side condition of $\mathrm{Decmp}^{\epsilon+}$ an application of $\mathrm{Decmp}^{\epsilon+}$ decreases the size of the right hand side of a goal by no more than $\mathfrak{m}$; it increases the size of the right hand side of a goal by no more than $\mathfrak{q}\mathfrak{n}\mathfrak{r}^2(\mathfrak{m}+1)^{(\mathfrak{q}+1)}$. It follows that $|G_1| < \mathfrak{m}\mathfrak{c}\mathfrak{v}$ and $|G_2| < \left(\mathfrak{q}\mathfrak{n}\mathfrak{r}^2(\mathfrak{m}+1)^{(\mathfrak{q}+1)}\right)\mathfrak{c}\mathfrak{v}$.

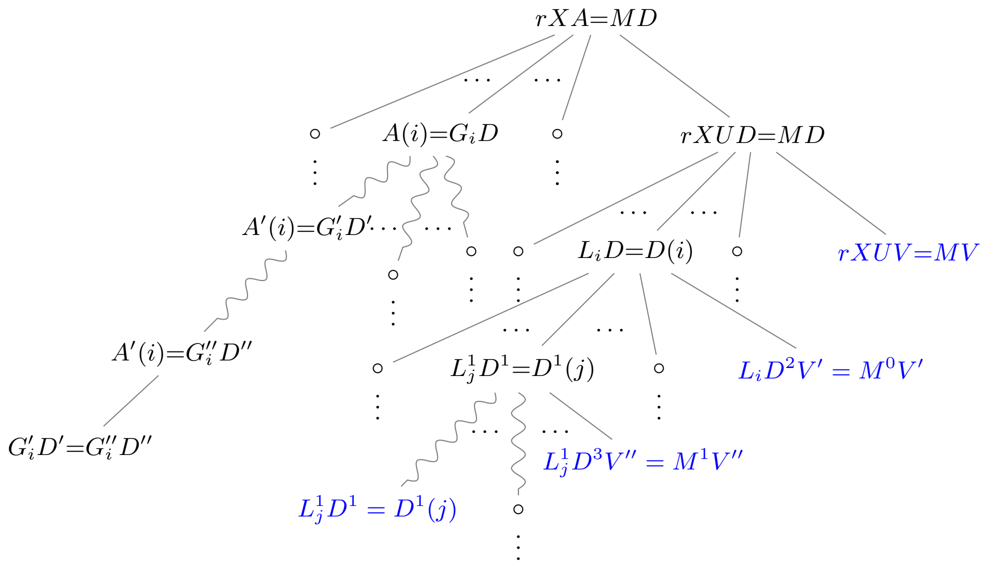


Fig. 5. Subtableau Construction for nPDA$^{\epsilon+}$

2. Consequently the number of times 1(a)iC is excuted is bounded by $\left(\mathfrak{q}\mathfrak{n}\mathfrak{r}^2(\mathfrak{m}+1)^{(\mathfrak{q}+1)}\right)\mathfrak{c}\mathfrak{v}$. When a subgoal of the form $r'X'C_1D_1 = M_1D_1$ is reached, where $|M_1| \leq \mathfrak{m}$, the size of C' is loosely bounded by

$$\left(\mathfrak{q}\mathfrak{n}\mathfrak{r}^2(\mathfrak{m}+1)^{(\mathfrak{q}+1)}\right)\mathfrak{c}\mathfrak{v} + \left(\left(\mathfrak{q}\mathfrak{n}\mathfrak{r}^2(\mathfrak{m}+1)^{(\mathfrak{q}+1)}\right)\mathfrak{c}\mathfrak{v}\right)^2. \tag{26}$$

- Step 2 is algorithmically simple. The subgoal $rXCV = MV$ is small because $|M|$ is bounded by $\mathfrak{m}$ and $|C|$ is bounded by the number in (26). The termination of Step 2 depends on the termination of Step 3.
- Step 3 cannot repeat infinitely often. In 3a, if L_i is not a selector, then $L_iD = L_i$ and the node is a leaf. If L_i is a selector, it must be i, and consequently the leaf must be $D(i) = D(i)$, which is successful. In 3b, the size of the subgoal is computationally bounded. So it is small. In case 3(c)i a small goal is introduced. Notice that the size of L_iD^2 is bounded by $\mathfrak{m}+\mathfrak{v}$. In case 3(c)ii if the size of the right hand side continues to decrease during the successive recursive call of Step 3, eventually either Case 3a applies, or Case 3b applies, or Case 3d applies.

So every branch of a subtableau ends. By König Lemma the subtableau is finite. □

5.4 Tableau

We are now in a position to explain how to produce a tableau for a goal $P = Q$ satisfying $|P| > 0$ and $|Q| > 0$. To start with we construct a subtableau for $P = Q$. For each leaf of the subtableau that is neither successful nor unsuccessful, we apply Match$^{\epsilon+}$. If it turns out that Match$^{\epsilon+}$ is not applicable, then the leaf is unsuccessful. If Match$^{\epsilon+}$ is applicable and the resulting match is empty, then the leaf is successful. Otherwise we repeat the subtableau construction for each subgoal of Match$^{\epsilon+}$ under the condition that the guessing step in the beginning of the nondeterministic algorithm is bypassed. In other words, all the subtableaux share the same guess of the recursive constants. In this way we obtain a *tableau* for $P = Q$. The construction of the tableau ends on a leaf F of a subtableau if either it is a successful/unsuccessful leaf of the subtableau or it is potentially successful in that it coincides with the leaf F' of an ancestor subtableau with F' staying in the path from the root of the tableau to F.

Definition 5. *A tableau is* successful *if its leaves are either successful or potentially successful.*

The following lemma follows immediately from the proof of Lemma 13.

Lemma 14. *Every tableau is finite.*

Lemma 14 guarantees that a tableau is either successful or unsuccessful.

Lemma 15. *If* $|P|, |Q| > 0$*, then* $P \simeq Q$ *if and only if* $P = Q$ *has a successful tableau.*

Proof. If $P \simeq Q$, then a successful tableau can be constructed by enumeration using the algorithm described in Fig. 4. The correctness of the construction is guaranteed by Lemma 10, Lemma 11, Lemma 12 and Lemma 14. To prove the converse implication, assume that all the leaves of a successful tableau for $P = Q$ are sound for $\simeq_k$. If a potentially successful leaf is sound for $\simeq_k$, it must be sound for $\simeq_{k+1}$. This is because all the rules are backward sound and there is at least one application of Match$^{\epsilon+}$ between a subtableau and its parent subtableau. It follows from induction that all the equalities appearing in the tableau are sound for $\simeq_k$ for all $k > 0$. We are done by applying Lemma 7. ☐

Lemma 15 provides the following semidecidable procedure for checking $\simeq$ on nPDA$^{\epsilon+}$ processes: Given input P, Q, check if $|P| > 0$ and $|Q| > 0$ using Lemma 3. If the answer is negative, we can easily decide if $P \simeq Q$. Otherwise we enumerate all the tableaux for $P = Q$ and at the same time check if any of them is successful. Together with Theorem 1 we get the main result of the section.

Theorem 2. *The branching bisimilarity on* nPDA$^{\epsilon+}$ *processes is decidable.*

6 High Undecidability of ϵ-Nondeterminism

In this section we show that the branching bisimilarity is highly undecidable on PDA$^{\epsilon+}$. This is done by a reduction from a Σ_1^1-complete problem. A *nondeterministic Minsky counter machine* $\mathcal{M}$ with two counters c_1, c_2 is a program of the form $1 : I_1;\ 2 : I_2;\ \ldots;\ n{-}1 : I_{n-1};\ n :$ halt, where for each $i \in \{1, \ldots, n-1\}$ the instruction I_i is in one of the following forms, assuming $1 \le j, k \le n$ and $e \in \{1, 2\}$.

- $c_e := c_e + 1$ and then goto j.
- if $c_e = 0$ then goto j; otherwise $c_e := c_e - 1$ and then goto k.
- 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 use the following fact from [6].

Proposition 2. rec-NMCM *is* Σ_1^1*-complete.*

Following [15] we 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 following form.

– $i : c_e := *$ and then goto j.

The effect of this instruction is to set c_e by a nondeterministically chosen number and then go to I_j. Every instruction "$i : I_i$" of $\mathcal{M}$ is then replaced by two instructions in $\mathcal{M}'$, with respective labels $2i{-}1$ and $2i$.

– $1 : I_1$ is replaced by
  $1 : c_3 := *$ and goto 2;
  $2 : I_1$.
– $i : I_i$, where $i \in \{2, \ldots, n\}$, is replaced by
  $2i - 1$: if $c_3 = 0$ then goto $2n$; otherwise $c_3 := c_3 - 1$ and goto $2i$;
  $2i : I_i$
– Inside each I_i, where $i \in \{1, \ldots, n\}$, every occurrence of "goto j" is replaced by "goto $2j - 1$".

It is easy to see that $\mathcal{M}'$ has an infinite computation if and only if $\mathcal{M}$ has an infinite computation that executes the instruction I_1 infinitely often. Our goal is to construct a PDA$^{\epsilon+}$ system $\mathcal{G} = \{\mathcal{Q}, \mathcal{L}, \mathcal{V}, \mathcal{R}\}$ in which we can define two processes $p_1X\bot$ and $q_1X\bot$ that render true the following equivalence.

$$p_1X\bot \simeq q_1X\bot \text{ if and only if } \mathcal{M}' \text{ has an infinite computation.}$$

The system $\mathcal{G} = \{\mathcal{Q}, \mathcal{L}, \mathcal{V}, \mathcal{R}\}$ contains the following key elements:

– Two states $p_i, q_i \in \mathcal{Q}$ are introduced for each instruction I_i.
– $\mathcal{L} = \{a, b, c, c_1, c_2, c_3, f, f'\}$.
– Three stack symbols $C_1, C_2, C_3 \in \mathcal{V}$ are introduced for the three counters respectively. A bottom symbol $\bot \in \mathcal{V}$ is also introduced.

Our construction borrows ideas from [16,15,33], making use of the game characterization of branching bisimulation and Defender's Forcing technique. A configuration of $\mathcal{M}'$ that consists of instruction label i and counter values $(c_1, c_2, c_3) = (n_1, n_2, n_3)$ is represented by the game configuration $(p_iXC_1^{n_1}C_2^{n_2}C_3^{n_3}\bot, q_iXC_1^{n_1}C_2^{n_2}C_3^{n_3}\bot)$. In the rest of the section we shall complete the definition of $\mathcal{G}$ and explain its working mechanism.

6.1 Test on Counter

We need some rules to carry out testing on the counters. In the rules given in Fig. 6, j and e range over the set $\{1, 2, 3\}$. These rules are straightforward. The following proposition summarizes the correctness requirement on the equality test, the successor and predecessor tests, and the zero test. Its routine proof is omitted.

Proposition 3. *Let $\alpha = C_1^{n_1}C_2^{n_2}C_3^{n_3}$ and $\beta = C_1^{m_1}C_2^{m_2}C_3^{m_3}$. The following statements are valid.*

1. *$t\alpha\bot \simeq t\beta\bot$ if and only if $n_e = m_e$ for $e = 1, 2, 3$.*
2. *$t(3, *)\alpha\bot \simeq t'(3, *)\beta\bot$ if and only if $n_e = m_e$ for $e = 1, 2$.*
3. *$t(e, +)\alpha\bot \simeq t'(e, +)\beta\bot$ if and only if $n_e + 1 = m_e$ and $n_j = m_j$ for $j \neq e$.*
4. *$t(e, -)\alpha\bot \simeq t'(e, -)\beta\bot$ if and only if $n_e = m_e + 1$ and $n_j = m_j$ for $j \neq e$.*
5. *$t(e, 0)\alpha\bot \simeq t'(e, 0)\beta\bot$ if and only if $n_j = m_j$ for $j = 1, 2, 3$ and $n_e = 0$.*
6. *$t(e, 1)\alpha\bot \simeq t'(e, 1)\beta\bot$ if and only if $n_j = m_j$ for $j = 1, 2, 3$ and $n_e > 0$.*
7. *$p\alpha\bot \simeq p\alpha\bot\beta$ for all $p \in \mathcal{Q}$ and all $\alpha, \beta \in \mathcal{V}^*$.*

– $tC_1 \xrightarrow{c_1} t,\ tC_2 \xrightarrow{c_2} t,\ tC_3 \xrightarrow{c_3} t$;

– $t(e,+)C_j \xrightarrow{c_j} t(e,+)$ if $j < e$, $t(e,+)C_j \xrightarrow{c_e} tC_j$ if $j \geq e$, $t(e,+)\bot \xrightarrow{c_e} t\bot$;
$t'(e,+)C_j \xrightarrow{c_j} t$;

– $t(e,*)C_1 \xrightarrow{c_1} t(e,*)$, $t(e,*)C_2 \xrightarrow{c_2} t(e,*)$, $t(e,*)C_3 \xrightarrow{b} t\bot$;
$t'(e,*)C_1 \xrightarrow{c_1} t(e,*)$, $t'(e,*)C_2 \xrightarrow{c_2} t(e,*)$, $t'(e,*)C_3 \xrightarrow{b} t\bot$;

– $t(e,-)C_j \xrightarrow{c_j} t$;
$t'(e,-)C_j \xrightarrow{c_j} t'(e,-)$ if $j < e$, $t'(e,-)C_j \xrightarrow{c_e} tC_j$ if $j \geq e$, $t'(e,-)\bot \xrightarrow{c_e} t\bot$;

– $t(e,0)C_j \xrightarrow{c_j} t(e,0)$ if $j \neq e$, $t(e,0)C_e \xrightarrow{f} t(e,0)$;
$t'(e,0)C_j \xrightarrow{c_j} t'(e,0)$ if $j \neq e$, $t'(e,0)C_e \xrightarrow{f'} t(e,0)$;

– $t(e,1)C_j \xrightarrow{c_j} t(e,1)$ if $j < e$, $t(e,1)C_e \xrightarrow{c_e} t$, $t(e,1)C_j \xrightarrow{f} t$ if $j > e$; $t(e,1)\bot \xrightarrow{f} t\bot$;
$t'(e,1)C_j \xrightarrow{c_j} t'(e,1)$ if $j < e$, $t'(e,1)C_e \xrightarrow{c_e} t$, $t'(e,1)C_j \xrightarrow{f'} t$ if $j > e$; $t'(e,1)\bot \xrightarrow{f'} t\bot$;

– $p\bot \xrightarrow{b} t\bot$ for every $p \in \{t, t', t(e,+), t'(e,+), t(e,-), t'(e,-), t(e,0), t'(e,0), t(e,1), t'(e,1)\}$.

Fig. 6. Test on Counter

6.2 Operation on Counter

There are three basic operations on counters, the increment operation, the decrement operation and the nondeterministic assignment operation. Our task is to encode these operations in the branching bisimulation game $\mathcal{G}$. To do that we use a technique from [33], which is a refinement of Defender's Forcing technique [15], taking into account of the subtlety of the branching bisimulation. The idea can be explained using the following system.

1. $P \xrightarrow{a} P'$, $P \xrightarrow{\epsilon} Q_0$. The latter is the only silent transition of P.
2. $Q \xrightarrow{\epsilon} Q_0$. This is the only transition Q may perform. Hence $Q \simeq Q_0$.
3. $Q_0 \simeq \overline{Q}$ whenever $Q_0 \Longrightarrow \overline{Q}$.

Condition 1 and condition 2 guarantee that $P \simeq Q$ if and only if $P \simeq Q_0$. So the effectiveness of the Defender's Forcing the copycat rules $P \xrightarrow{\epsilon} Q_0$, $Q \xrightarrow{\epsilon} Q_0$ intend to achieve depends on how we define Q_0. Condition 3 is forced upon us by the previous two conditions. A standard approach to meet the requirement 3 is to make sure that everything that has been done to derive $Q_0 \Longrightarrow \overline{Q}$ can be undone. In our setting this is accomplished by starting all over again with the help of the bottom symbol $\bot$. Once we know that condition 3 is indeed satisfied, the argument for the correctness of the bisimulation game can be simplified in the following sense: In the game of (P, Q) Attacker would play $P \xrightarrow{a} P'$. Defender's optimal response must be of the following form

$$Q \xrightarrow{\epsilon} Q_0 \xrightarrow{\epsilon} Q_1 \xrightarrow{\epsilon} Q_2 \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} Q_k \xrightarrow{a} Q'.$$

For both players only the configuration (P', Q') need be checked.

With the above remark in mind we turn to the part of the game that implements the basic operations. Let e range over $\{1, 2, 3\}$, o over $\{+, -, *\}$, and j over $\{1, \ldots, 2n\}$. For each triple (e, o, j) we introduce the rules given in Fig. 7. The following lemma identifies some useful state preserving silent transitions.

Lemma 16. *$P \simeq g(e,o,j)X\bot$ for all P such that $g(e,o,j)X\bot \Longrightarrow P$. Similarly $Q \simeq g'(e,o,j)X\bot$ for all Q such that $g'(e,o,j)X\bot \Longrightarrow Q$.*

– $u(e,o,j)X \xrightarrow{a} u_1(e,o,j)X$, $u(e,o,j)X \xrightarrow{\epsilon} r'(e,o,j)X$;
$u'(e,o,j)X \xrightarrow{\epsilon} r'(e,o,j)X$;

– $r'(e,o,j)X \xrightarrow{\epsilon} g'(e,o,j)X\bot$;
$g'(e,o,j)X \xrightarrow{\epsilon} g'(e,o,j)X_3$;
$g'(e,o,j)X_3 \xrightarrow{\epsilon} g'(e,o,j)X_3C_3$, $g'(e,o,j)X_3 \xrightarrow{\epsilon} g'(e,o,j)X_2$;
$g'(e,o,j)X_2 \xrightarrow{\epsilon} g'(e,o,j)X_2C_2$, $g'(e,o,j)X_2 \xrightarrow{\epsilon} g'(e,o,j)X_1$;
$g'(e,o,j)X_1 \xrightarrow{\epsilon} g'(e,o,j)X_1C_1$, $g'(e,o,j)X_1 \xrightarrow{\epsilon} r'(e,o,j)X$;

– $g'(e,o,j)X_1 \xrightarrow{a} u'_1(e,o,j)X$;

– $u_1(e,o,j) \xrightarrow{a} u_2(e,o,j)X$, $u_1(e,o,j)X \xrightarrow{c} t(e,o)$;
$u'_1(e,o,j) \xrightarrow{a} u'_2(e,o,j)X$, $u'_1(e,o,j)X \xrightarrow{c} t'(e,o)$;

– $u_2(e,o,j)X \xrightarrow{\epsilon} r(e,o,j)X$;
$u'_2(e,o,j)X \xrightarrow{\epsilon} r(e,o,j)X$, $u'_2(e,o,j)X \xrightarrow{a} u'_3(e,o,j)X$;

– $r(e,o,j)X \xrightarrow{\epsilon} g(e,o,j)X\bot$; $g(e,o,j)X \xrightarrow{\epsilon} g(e,o,j)X_3$;
$g(e,o,j)X_3 \xrightarrow{\epsilon} g(e,o,j)X_3C_3$, $g(e,o,j)X_3 \xrightarrow{\epsilon} g(e,o,j)X_2$;
$g(e,o,j)X_2 \xrightarrow{\epsilon} g(e,o,j)X_2C_2$, $g(e,o,j)X_2 \xrightarrow{\epsilon} g(e,o,j)X_1$;
$g(e,o,j)X_1 \xrightarrow{\epsilon} g(e,o,j)X_1C_1$, $g(e,o,j)X_1 \xrightarrow{\epsilon} r(e,o,j)X$;

– $g(e,o,j)X_1 \xrightarrow{a} u_3(e,o,j)X$;

– $u_3(e,o,j)X \xrightarrow{a} p_jX$, $u_3(e,o,j)X \xrightarrow{c} t$;
$u'_3(e,o,j)X \xrightarrow{a} q_jX$, $u'_3(e,o,j)X \xrightarrow{c} t$.

Fig. 7. Operation on Counter

Proof. Suppose $g(e,o,j)X\bot \Longrightarrow P$. Then $P \Longrightarrow g(e,o,j)X\bot\alpha$ for some α. By (7) of Proposition 3 one has $g(e,o,j)X\bot \simeq g(e,o,j)X\bot\alpha$. Consequently $g(e,o,j)X\bot \simeq P$. □

The next lemma states the soundness property of the rules defined in Fig. 7, in which we write $\mathbf{1}^1$, $\mathbf{1}^2$ and $\mathbf{1}^3$ respectively for $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$.

Lemma 17. *Suppose $\alpha = C_1^{m_1}C_2^{m_2}C_3^{m_3}$. The following statements are valid.*

1. *In the bisimulation of $(u(e,+,j)X\alpha\bot, u'(e,+,j)X\alpha\bot)$, Defender, respectively Attacker, has a strategy to win or at least push the game to (P,Q) such that $P \simeq p_jXC_1^{n_1}C_2^{n_2}C_3^{n_3}\bot$ and $Q \simeq q_jXC_1^{n_1}C_2^{n_2}C_3^{n_3}\bot$ and $(n_1,n_2,n_3) = (m_1,m_2,m_3)+\mathbf{1}^e$.*
2. *If $m_e > 0$ then in the bisimulation game of $(u(e,-,j)X\alpha\bot, u'(e,-,j)X\alpha\bot)$, Defender, respectively Attacker, has a strategy to win or at least push the game to (P,Q) such that $P \simeq p_jXC_1^{n_1}C_2^{n_2}C_3^{n_3}\bot$ and $Q \simeq q_jXC_1^{n_1}C_2^{n_2}C_3^{n_3}\bot$ and $(n_1,n_2,n_3) = (m_1,m_2,m_3)-\mathbf{1}^e$.*
3. *Suppose $n \geq m_3$. In the bisimulation game of $(u(3,*,j)X\alpha\bot, u'(3,*,j)X\alpha\bot)$, Defender has a strategy to win or at least push the game to (P,Q) such that $P \simeq p_jXC_1^{n_1}C_2^{n_2}C_3^{n_3}\bot$ and $Q \simeq q_jXC_1^{n_1}C_2^{n_2}C_3^{n_3}\bot$ and $(n_1,n_2,n_3) = (m_1,m_2,m_3) + (n-m_3)\cdot\mathbf{1}^3$.*

Proof. We prove the first statement. The proof for the other two is similar. Let $\beta = C_1^{n_1}C_2^{n_2}C_3^{n_3}$ such that $(n_1,n_2,n_3) = (m_1,m_2,m_3)+\mathbf{1}^e$. In what follows we describe Defender and Attacker's step-by-step optimal strategy in the bisimulation game of $(u(e,+,j)X\alpha\bot, u'(e,+,j)X\alpha\bot)$.

(i) By Defender's Forcing, Attacker plays $u(e,+,j)X\alpha\bot \xrightarrow{a} u_1(e,+,j)X\alpha\bot$. Defender responds with

$$u'(e,+,j)X\alpha\bot \xRightarrow{\epsilon} g'(e,+,j)X_1\beta\bot\alpha\bot \xrightarrow{a} u'_1(e,+,j)X\beta\bot\alpha\bot.$$

According to Lemma 16, Attacker's optimal move is to continue the game from

$$(u_1(e,+,j)X\alpha\bot, u_1'(e,+,j)X\beta\bot\alpha\bot).$$

(ii) It follows from Proposition 3 that $t(e,+)X\alpha\bot \simeq t'(e,+)X\beta\bot\alpha\bot$. If Attacker plays an action labeled c, Defender wins. Attacker's optimal move is to play an action labeled a. Defender then follows suit, and the game reaches the configuration $(u_2(e,+,j)X\alpha\bot, u_2'(e,+,j)X\beta\bot\alpha\bot)$.

(iii) Attacker's next move is $u_2'(e,+,j)X\beta\bot\alpha\bot \xrightarrow{a} u_3'(e,+,j)X\beta\bot\alpha\bot$. This is optimal by Proposition 3. Defender responds with

$$u_2(e,+,j)X\alpha\bot \stackrel{\epsilon}{\Longrightarrow} g(e,+,j)X_1\beta\bot\alpha\bot \xrightarrow{a} u_3(e,+,j)X\beta\bot\alpha\bot.$$

By an argument similar to the one given in (i) Attacker would choose

$$(u_3(e,+,j)X\beta\bot\alpha\bot, u_3'(e,+,j)X\beta\bot\alpha\bot)$$

as the next configuration.

(iv) If Attacker plays an action labeled c, Defender wins by Proposition 3. So Attacker's best bet is to play an action labeled by a. The game reaches the configuration $(p_jX\beta\bot\alpha\bot, q_jX\beta\bot\alpha\bot)$.

The above argument shows that the configuration $(p_jX\beta\bot\alpha\bot, q_jX\beta\bot\alpha\bot)$ is optimal for both Attacker and Defender. We are done. $\square$

6.3 Control Flow

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

1. The following rules are introduced in the game $\mathcal{G}$ for an instruction of the form "$i : c_e := c_e + 1$ and then goto j".
$$p_iX \xrightarrow{a} u(e,+,j)X, \quad q_iX \xrightarrow{a} u'(e,+,j)X.$$
2. For each instruction of the form "$i : c_e := *$ and then goto j" the following two rules are added to $\mathcal{R}$.
$$p_iX \xrightarrow{a} u(e,*,j)X, \quad q_iX \xrightarrow{a} u'(e,*,j)X.$$
3. For each instruction of the form "i : goto j or goto k", we have the following.
   - $p_iX \xrightarrow{a} p_i^1X,\ p_iX \xrightarrow{a} q_i^1X,\ p_iX \xrightarrow{a} q_i^2X;$
     $q_iX \xrightarrow{a} q_i^1X,\ q_iX \xrightarrow{a} q_i^2X;$
   - $p_i^1X \xrightarrow{a} p_jX,\ p_i^1X \xrightarrow{a} p_kX;$
     $q_i^1X \xrightarrow{a} q_jX,\ q_i^1X \xrightarrow{a} p_kX;$
     $q_i^2X \xrightarrow{a} p_jX,\ q_i^2X \xrightarrow{a} q_kX.$

   These rules embody precisely the idea of Defender's Forcing [15]. It is Defender who makes the choice.
4. For each instruction of the form
$$\text{“}i : \text{if } c_e = 0 \text{ then goto } j\text{; otherwise } c_e = c_e - 1 \text{ and then goto } k\text{”}$$
we construct a system defined by the following rules.

- $p_i X \xrightarrow{a} p_i(e,0,j)X$, $p_i X \xrightarrow{c} p_i(e,1,k)X$;
  $q_i X \xrightarrow{a} q_i(e,0,j)X$, $q_i X \xrightarrow{c} q_i(e,1,k)X$;
- $p_i(e,0,j)X \xrightarrow{a} v_1(e,0,j)X$, $p_i(e,1,k)X \xrightarrow{a} v_1(e,1,k)X$;
  $p_i(e,0,j)X \xrightarrow{a} v_2(e,0,j)X$, $p_i(e,1,k)X \xrightarrow{a} v_2(e,1,k)X$;
  $p_i(e,0,j)X \xrightarrow{a} v_3(e,0,j)X$, $p_i(e,1,k)X \xrightarrow{a} v_3(e,1,k)X$;
- $q_i(e,0,j)X \xrightarrow{a} v_2(e,0,j)X$, $q_i(e,1,k)X \xrightarrow{a} v_2(e,1,k)X$;
  $q_i(e,0,j)X \xrightarrow{a} v_3(e,0,j)X$, $q_i(e,1,k)X \xrightarrow{a} v_3(e,1,k)X$;
- $v_1(e,0,j)X \xrightarrow{a} t(e,1)X$, $v_1(e,0,j)X \xrightarrow{a} p_j X$;
  $v_2(e,0,j)X \xrightarrow{a} t'(e,1)X$, $v_2(e,0,j)X \xrightarrow{a} p_j X$;
  $v_3(e,0,j)X \xrightarrow{a} t(e,1)X$, $v_3(e,0,j)X \xrightarrow{a} q_j X$;
- $v_1(e,1,k)X \xrightarrow{a} t(e,0)X$, $v_1(e,1,k)X \xrightarrow{a} u(e,-,k)X$;
  $v_2(e,1,k)X \xrightarrow{a} t'(e,0)X$, $v_2(e,1,k)X \xrightarrow{a} u(e,-,k)X$;
  $v_3(e,1,k)X \xrightarrow{a} t(e,0)X$, $v_3(e,1,k)X \xrightarrow{a} u'(e,-,k)X$.

  The idea of the above encoding is that Attacker must claim either "$c_e = 0$" or "$c_e > 0$". Defender can check the claim and wins if Attacker lies. If Attacker has not lied, Defender can force Attacker to do what Defender wants.

5. For "$2n$: halt", we add the rules

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

   So Attacker wins if the game ever terminates.

This completes the definition of $\mathcal{G}$.

With the help of Proposition 3 and Lemma 17, it is a routine to prove the next lemma.

Lemma 18. *$\mathcal{M}'$ has an infinite computation if and only if $p_1 X\bot \simeq q_1 X\bot$.*

Branching bisimilarity on PDA$^{\epsilon+}$ is in Σ^1_1 for the following reason: For any PDA$^{\epsilon+}$ processes P and Q, $P \simeq Q$ 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 3. Together with the reduction justified by Lemma 18 we derive the main result of the section.

Theorem 3. *Branching bisimilarity on PDA$^{\epsilon+}$ is Σ^1_1-complete.*

It has been proved in [33] that the branching bisimilarity is undecidable on normed PDA. The reduction defined in the above can be constructed for nPDA too. This is because in nPDA the stack can be reset by popping off all the symbols in the stack using ϵ-popping transitions and creating new stack content using ϵ-pushing transitions, achieving the same effect as the bottom symbol $\bot$ has achieved in PDA$^{\epsilon+}$. The details are omitted.

Theorem 4. *Branching bisimilarity on nPDA is Σ^1_1-complete.*

7 Conclusion

The structural definition of PDA plays an important role in simplifying our proof. After proving the main results of this paper in the beginning of 2014, we became aware of the relationship between our definition of PDA and Jančar's notion of first order grammar [11]. In our opinion Jančar's approach is an abstraction of the issue at a more basic level. Recently Jančar has provided a quite different proof for the decidability of the strong bisimilarity of first order grammar [13]. In the full paper he also outlined an idea of how to extend his proof to take care of silent transitions.

Stirling proved that the language equivalence of DPDA is primitive recursive [25]. Benedikt, Goller, Kiefer and Murawski showed that the strong bisimilarity on nPDA is non-elementary [1]. More recently Jančar observed that the strong bisimilarity of first-order grammar is Ackermann-hard [12], a consequence of which is that the strong bisimilarity proved decidable by Sénizergues in [21] is Ackermann-hard. It is an interesting research direction to look for tighter upper and lower bounds on the branching bisimilarity of nPDA$^{\epsilon+}$.

Acknowledgement. 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. M. Benedikt, S. Moller, S. Kiefer, and A. Murawski. Bisimilarity of Pushdown Automata is Nonelementary. In *LICS'13*, pages 488–498, 2013.
2. O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on Infinite Structures. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 545–623. North-Holland, 2001.
3. Y. Fu. Checking Equality and Regularity for Normed BPA with Silent Moves. *ICALP'13*, Lecture Notes in Computer Science 7966, 238–249, 2013.
4. S. Ginsburg and S. Greibach. Deterministic Context Free Languages. *Information and Control*, 9:620–648, 1966.
5. J. Groote and H. Hüttel. Undecidable Equivalences for Basic Process Algebra. *Information and Computation*, 115:354–371, 1994.
6. D. Harel. Effective Transformations on Infinite Trees, with Applications to High Undecidability, Dominoes, and Fairness. *J. ACM*, 33:224–248, 1986.
7. J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Publishing Company, 1979.
8. H. Hüttel. Silence is Golden: Branching Bisimilarity is Decidable for Context Free Processes. In *CAV'91*, pages 2–12. Lecture Notes in Computer Science 575, Springer, 1992.
9. H. Hüttel. Undecidable Equivalences for Basic Parallel Processes. In *Theoretical Aspects of Computer Software*, Lecture Notes in Computer Science 789, pages 454–464, 1994.
10. H. Hüttel and C. Stirling. Actions Speak Louder than Words: Proving Bisimilarity for Context-Free Processes. In *LICS'91*, pages 376–386, 1991.
11. P. Jančar. Decidability of DPDA Language Equivalence via First-Order Grammars. In *LICS'12*, page 415–424. IEEE Computer Society, 2012.
12. P. Jančar. Equivalences of Pushdown Systems are Hard. *Foundations of Software Science and Computation*, pages 1–28, 2014.
13. 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, pages 232–243, 2014.
14. P. Jančar. Bisimulation Equivalence of First Order Grammars. arXiv:1405.7923, 2014.
15. P. Jančar and J. Srba. Undecidability of Bisimilarity by Defender's Forcing. *Journal of ACM*, 55(1), 2008.
16. E. Mayr. Undecidability of Weak Bisimulation Equivalence for 1-Counter Processes. In *ICALP'03*, Lecture Notes in Computer Science 2719, page 570–583. Springer, 2003.
17. R. Mayr. Process Rewrite Systems. *Information and Computation*, 156:264–286, 2000.
18. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
19. D. Park. Concurrency and Automata on Infinite Sequences. In *TCS'81*, Lecture Notes in Computer Science 104, pages 167–183. Springer, 1981.
20. G. Sénizergues. The Equivalence Problem for Deterministic Pushdown Automata is Decidable. In *ICALP'97*, Lecture Notes in Computer Science 1256, pages 671–681. Springer-Verlag, 1997.
21. G. Sénizergues. Decidability of Bisimulation Equivalence for Equational Graphs of Finite Out-Degree. In *FOCS'98*, pages 120–129. IEEE, 1998.

22. G. Sénizergues. L(a)=L(b)? Decidability Results from Complete Formal Systems. *Theoretical Computer Science*, 251(1-2):1–166, 2001.
23. G. Sénizergues. L(a)=L(b)? A Simplified Decidability Proof. *Theoretical Computer Science*, 281(1):555–608, 2002.
24. J. Srba. Undecidability of Weak Bisimilarity for Pushdown Processes. In *CONCUR'02*, Lecture Notes in Computer Science 2421, pages 579–593. Springer-Verlag, 2002.
25. Stirling. Deciding DPDA Equivalence is Primitive Recursive. In *ICALP'02*, Lecture Notes in Computer Science 2380, pages 821–832. Springer, 2002.
26. C. Stirling. Decidability of Bisimulation Equivalence for Normed Pushdown Processes. In *CONCUR'96*, Lecture Notes in Computer Science, pages 217–232. Springer-Verlag, 1996.
27. C. Stirling. Decidability of Bisimulation Equivalence for Normed Pushdown Processes. *Theoretical Computer Science*, 195(2):113–131, 1998.
28. C. Stirling. The Joy of Bisimulation. In *MFCS'98*, Lecture Notes in Computer Science 1450, pages 142–151. Springer, 1998.
29. C. Stirling. Decidability of Bisimulation Equivalence for Pushdown Processes. 2000.
30. C. Stirling. Decidability of DPDA Equivalence. *Theoretical Computer Science*, 255(1-2):1–31, 2001.
31. R. van Glabbeek and W. Weijland. Branching Time and Abstraction in Bisimulation Semantics. In *Information Processing'89*, pages 613–618. North-Holland, 1989.
32. R. van Glabbeek and W. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of ACM*, 3:555–600, 1996.
33. 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, pages 363–374, 2014.