Decidability of Bisimulation Equivalence for Normed Pushdown Processes

Colin Stirling
Department of Computer Science
University of Edinburgh
Edinburgh EH9 3JZ, UK
email: cps@dcs.ed.ac.uk

Abstract

We prove that bisimulation equivalence is decidable for normed pushdown processes.

1 Introduction

In the classical theory of automata the expressive power of pushdown automata is matched by context-free grammars. Both accept the same family of languages, the context-free languages. Concurrency theory requires a more intensional exposition of behaviour (as language equivalence need not be preserved in the presence of communicating abstract machines). Many finer equivalences have been proposed. Bisimulation equivalence, due to Park and Milner, has received much attention.

Baeten, Bergstra and Klop proved that bisimulation equivalence is decidable for irredundant context-free grammars (without the empty production). Within process calculus theory these grammars correspond to normed BPA processes. Their proof relies on isolating a complex periodicity from the transition graphs of these processes. Simpler proofs of the result soon followed which expose algebraic structure.

Caucal and Monfort showed that normed pushdown processes (the process analogue of irredundant pushdown automata without ϵ-transitions) are strictly more expressive than normed BPA with respect to bisimulation equivalence. In this paper we prove that bisimulation equivalence is decidable for this richer family of processes. However the proof is not easy, and does not follow immediately from the techniques used for showing decidability of normed BPA. One indication of this is that although the proof of decidability of bisimulation equivalence for BPP (Basic Parallel Processes) is similar to that for BPA bisimulation equivalence is undecidable for the pushdown extension to BPP.

2 Normed pushdown processes

Ingredients for describing pushdown processes are a finite set of states $\mathcal{P} = \{p_1, \ldots, p_k\}$, a finite set of stack symbols $\Gamma = \{X_1, \ldots, X_m\}$, a finite set of actions $\mathcal{A} = \{a_1, \ldots, a_n\}$ and a finite family of basic transitions, each of the form $pX \xrightarrow{a} q\alpha$ where p and q are states, a is an action, X is a stack symbol and α is a sequence of stack symbols. A *pushdown process* is then any expression $p\alpha$, $p \in \mathcal{P}$ and $\alpha \in \Gamma^*$, whose behaviour (transition graph) is determined by the basic transitions together with the following prefix rule, where $\beta \in \Gamma^*$:

$$\text{if } pX \xrightarrow{a} q\alpha \text{ then } pX\beta \xrightarrow{a} q\alpha\beta$$

This account follows the presentation of Caucal [8]. It is a slight redescription of classical pushdown automata (without ϵ-transitions and final states), as for instance in [20], viewing them as generators instead of as acceptors.

Example 1 Let $\mathcal{P} = \{p, q, r, s\}$, $\Gamma = \{X\}$ and $\mathcal{A} = \{a, b, c, d\}$. The family of basic transitions is

$$\{pX \xrightarrow{a} pXX,\ pX \xrightarrow{c} q\epsilon,\ pX \xrightarrow{b} r\epsilon,\ qX \xrightarrow{d} sX,\ sX \xrightarrow{d} q\epsilon,\ rX \xrightarrow{d} r\epsilon\}$$

where ϵ is the empty stack sequence. The transition graph generated by pX is:

$$\begin{array}{lllllllll}
q\epsilon & \xleftarrow{d} & sX \xleftarrow{d} & qX & \xleftarrow{d} & sXX \xleftarrow{d} & qXX & \xleftarrow{d} & \ldots \\
\uparrow c & & & \uparrow c & & & \uparrow c & & \vdots \\
pX & \xrightarrow{a} & & pXX & \xrightarrow{a} & & pXXX & \xrightarrow{a} & \ldots \\
\downarrow b & & & \downarrow b & & & \downarrow b & & \vdots \\
r\epsilon & \xleftarrow{d} & & rX & \xleftarrow{d} & & rXX & \xleftarrow{d} & \ldots
\end{array}$$

For any $n \geq 0$ the transition $qXX^n \xrightarrow{d} sXX^n$ is derived from the basic transition $qX \xrightarrow{d} sX$ using the prefix rule when β is X^n. □

Example 1 illustrates how a pushdown process may generate an infinite state transition graph. In the following we are interested in comparing the behaviour of two pushdown processes. Without loss of generality, we can assume that they are built from the same ingredients $\mathcal{P}$, Γ, $\mathcal{A}$ and basic transitions (as the appropriate disjoint union, with respect to states and stack symbols, of two pushdown descriptions is a pushdown description). One notion of behaviour of $p\alpha$ is the language it generates, which is the set of words $\{w \in \mathcal{A}^* : \exists q \in \mathcal{P}.\, p\alpha \xrightarrow{w} q\epsilon\}$ where the extended transitions $\xrightarrow{w}$ are defined in the usual way. Acceptance is by empty stack instead of by final state (see [20]). The process pX of Example 1 generates the language $\{a^n b\, d^n : n \geq 0\} \cup \{a^n c\, d^{2n} : n \geq 0\}$. Any language generated by a pushdown process is context-free, and for each context-free language without the empty string there is a pushdown process which generates it. This remains true under the following restriction. A process $p\alpha$ is *normed* if every reachable process with a non-empty stack generates a non-empty language. (That is, for every $q\beta$ such that $\beta \neq \epsilon$ and $p\alpha \xrightarrow{v} q\beta$ for some v, there is a w and a state r, such that $q\beta \xrightarrow{w} r\epsilon$.) Normedness amounts to irredundancy in the pushdown automaton. In the rest of the paper we assume this restriction to normed pushdown processes.

Each context-free language without the empty string is also generated by a normed pushdown process whose state set $\mathcal{P}$ contains just one state. In which case $\mathcal{P}$ is superfluous, and the result is a normed context-free or BPA process (in Greibach normal form). The stack symbols Γ are the nonterminals and basic transitions have the form $X \xrightarrow{a} \alpha$. The prefix rule is: if $X \xrightarrow{a} \alpha$ then $X\beta \xrightarrow{a} \alpha\beta$. The language generated by a context-free process α is $\{w : \alpha \xrightarrow{w} \epsilon\}$. There is a standard transformation, see [20], which translates a normed pushdown process into a normed context-free process which generates the same language. With respect to languages, the expressive power of normed context-free processes is the same as normed pushdown processes.

Concurrency theory is built on a more intensional account of behavioural equivalence than that given by languages (as language equivalence need not be preserved in the presence of communicating abstract machines). A pivotal notion, due to Park and Milner, is bisimilarity which is finer than language equivalence on processes.

Definition 1 A binary relation $\mathcal{R}$ between processes is a bisimulation relation provided that whenever $(E, F) \in \mathcal{R}$, for all a

$$\begin{array}{l} \text{if } E \xrightarrow{a} E' \text{ then } \exists F'.\, F \xrightarrow{a} F' \text{ and } (E', F') \in \mathcal{R}, \text{ and} \\ \text{if } F \xrightarrow{a} F' \text{ then } \exists E'.\, E \xrightarrow{a} E' \text{ and } (E', F') \in \mathcal{R}. \end{array}$$

Two processes E and F are defined to be bisimulation equivalent, or bisimilar, written $E \sim F$, if there is a bisimulation relation $\mathcal{R}$ relating them.

The transformation from pushdown processes to context-free processes does not preserve bisimulation equivalence. In fact, with respect to bisimilarity normed pushdown processes constitute a richer family than normed context-free processes, as shown by Caucal and Monfort [11]. (Independently [1] proves a variant of this result when introducing the state operator into process algebra.) Their proof is very elegant, and utilises the canonical graph of a process which is the quotient of its transition graph with respect to bisimulation equivalence. For instance, the canonical graph for Example 1 fuses together the pairs of vertices labelled qX^n and rX^{2n} for any $n \geq 0$. Normed context-free processes are closed under canonical graphs in the sense that for each such canonical graph there is a normed context-free process whose transition graph is isomorphic to it. Normed pushdown processes fail to have this property. Caucal and Monfort show that Example 1 is a counterexample, as its canonical graph lacks the regular structure (as identified by Muller and Schupp [25]) that the transition graph of a pushdown process must possess: see [6] for further results about canonical graphs.

Burkart and Steffen provide additional insight into pushdown processes [4], by showing that, unlike context-free processes, they are closed under Hoare parallel composition with finite state processes (with respect to bisimilarity). Moreover they demonstrate that the family of pushdown processes is the smallest extension of context-free processes with this closure property.

Baeten, Bergstra and Klop proved that bisimulation equivalence is decidable for normed context-free processes [2, 3]. Simpler proofs were developed in [7, 15, 22, 17], and [19] showed that there is even a polynomial time decision procedure. The decidability result was generalized in [14] to encompass unnormed processes, and then refined in [5] to give upper bounds. Groote and Hüttel proved that other standard equivalences on processes (traces, failures, simulation, 2/3-bisimulation etc..,) are all undecidable [16]. Similar results were proved for Basic Parallel Processes, BPP, which are like context-free processes except that a process expression α is a multiset*. Christensen, Hirshfeld and Moller showed that bisimulation equivalence is decidable for normed BPP [12], and this result was generalised in [13] to include the unnormed case. Hüttel proved that the other equivalences are undecidable [21].

Decidability of bisimulation equivalence for normed pushdown processes is harder to show than for normed context-free processes. There are many reasons for this. First Baeten, Bergstra and Klop's method is not applicable, as normed pushdown transition graphs need not display the periodicity upon which their proof relies. Secondly, the structural methods in the simplifed proofs of decidability, which appeal to decomposition and congruence, are not immediately applicable, as it is not clear what are the components of a pushdown process: a context-free process $X_1 \ldots X_n$ is built

*The prefix rule becomes: if $X \xrightarrow{a} \alpha$ then $\delta X \beta \xrightarrow{a} \delta \alpha \beta$.

from the subprocesses X_i, but a process $pX_1 \dots X_n$ does not contain X_i as a pushdown component. Perhaps the clearest indication of the increased difficulty is that for the pushdown extension of BPP† bisimulation equivalence is undecidable, a result due to Hirshfeld (utilizing Jančar's technique for showing undecidability of bisimilarity for Petri nets [23]): for details see the excellent survey [24].

The proof presented below of decidability of bisimulation equivalence for normed pushdown processes consists of two semi-decision procedures (for which we are unable to provide a complexity measure). One half of the proof is easy, as bisimilarity is characterizable using approximants when processes, such as pushdown processes, are image-finite‡.

Definition 2 The family $\{\sim_n : n \geq 0\}$ is defined inductively as follows

$$\begin{array}{l} E \sim_0 F \text{ for all processes } E, F \\ E \sim_{n+1} F \text{ iff for each } a \in \mathcal{A} \\ \quad \text{if } E \xrightarrow{a} E' \text{ then } \exists F'.\, F \xrightarrow{a} F' \text{ and } E' \sim_n F', \text{ and} \\ \quad \text{if } F \xrightarrow{a} F' \text{ then } \exists E'.\, E \xrightarrow{a} E' \text{ and } E' \sim_n F' \end{array}$$

The following is a standard result.

Proposition 1 *If* E and F are image-finite then $E \sim F$ *iff* $\forall n \geq 0.\ E \sim_n F$.

For each $n \geq 0$, the relation $\sim_n$ is decidable for pushdown processes, and therefore bisimulation inequivalence is semi-decidable via the simple procedure which seeks the least i such that $p\alpha \not\sim_i q\beta$. Therefore we just need to establish the semi-decidability of bisimulation equivalence. The crux of this part of the proof is that there is a finite tableau proof of $p\alpha \sim q\beta$. As finite proofs can be enumerated, this amounts to a semi-decision procedure. The method generalises the technique developed by the author and Hüttel [22]. It relies upon exposure of structure within normed pushdown processes. We introduce a finer equivalence than bisimilarity to ensure a congruence, and we show that with respect to it pushdown processes can be taken apart, when extra stack symbols are introduced.

3 Congruence and cancellation

The following two key properties underpin decidability of bisimulation equivalence on normed context-free processes:

$$\begin{array}{ll} \text{Congruence :} & \text{if } \alpha \sim \beta \text{ then } \alpha\delta \sim \beta\delta \\ \text{Cancellation :} & \text{if } \alpha\delta \sim \beta\delta \text{ then } \alpha \sim \beta \end{array}$$

†When the prefix rule is: if $pX \xrightarrow{a} q\alpha$ then $p\delta X\beta \xrightarrow{a} q\delta\alpha\beta$.

‡E is image-finite if for each $w \in A^*$ the set $\{F : E \xrightarrow{w} F\}$ is finite.

Explicit use of these features can be seen in the decidability proof which uses tableaux [22]. (A more elegant proof of decidability of bisimilarity exploits the stronger property of unique prime decomposition, due to Hirshfeld [17].) In the more general case, when the restriction to being normed is lifted, congruence still holds but cancellation fails. However bisimulation equivalence remains decidable because of a weaker form of cancellation [14]: if there are infinitely many different δ (with respect to $\sim$) such that $\alpha\delta \sim \beta\delta$ then $\alpha \sim \beta$.

Neither of these key properties is true of normed pushdown processes. A counterexample to congruence is that although $qX \sim rXX$ in Example 1 of the previous section, $qXX \not\sim rXXX$. Example 1 also furnishes instances of failure of cancellation such as $rX^6 \sim qXXX$ but $rX^5 \not\sim qXX$. Moreover, the weaker version of cancellation also fails.

Two pushdown processes can be bisimilar without agreeing on their final states when the stack empties: for instance the bisimulation relation between qX and rXX of Example 1 contains the pair $(q\epsilon, r\epsilon)$. This explains why congruence (with respect to stacking) fails for pushdown processes. However by refining the definition of bisimulation equivalence we can ensure a congruence.

Definition 1 A binary relation $\mathcal{R}$ on pushdown processes is an a-bisimulation provided that whenever $(p\alpha, q\beta) \in \mathcal{R}$, for all $a \in \mathcal{A}$

$$
\begin{array}{l}
\text{if } \alpha = \epsilon \text{ then } \beta = \epsilon \text{ and } p = q, \text{ and} \\
\text{if } \beta = \epsilon \text{ then } \alpha = \epsilon \text{ and } p = q, \text{ and} \\
\text{if } p\alpha \xrightarrow{a} p'\alpha' \text{ then } \exists q'\beta'.\, q\beta \xrightarrow{a} q'\beta' \text{ and } (p'\alpha', q'\beta') \in \mathcal{R}, \text{ and} \\
\text{if } q\beta \xrightarrow{a} q'\beta' \text{ then } \exists p'\alpha'.\, p\alpha \xrightarrow{a} p'\alpha' \text{ and } (p'\alpha', q'\beta') \in \mathcal{R}.
\end{array}
$$

The "a" stands for "agreeing", as the first two clauses require final states to be the same when processes terminate. Two pushdown processes $p\alpha$ and $q\beta$ are a-bisimilar, written $p\alpha \equiv q\beta$, if there is an a-bisimulation relating them. Note that the earlier two processes qX and rXX are not a-bisimilar. Later, in this section, we shall relate bisimilarity and a-bisimilarity.

Not surprisingly a-bisimilarity is an equivalence relation, and it is also a congruence with respect to stacking.

Proposition 1 *The relation $\equiv$ is an equivalence relation.*

Proposition 2 *If* $p\alpha \equiv q\beta$ *then* $p\alpha\delta \equiv q\beta\delta$.

A-bisimilarity can also be characterized using approximants.

Definition 2 The family $\{\equiv_n :\ n \geq 0\}$ is defined inductively on pushdown processes as follows:

$p\epsilon \equiv_0 p\epsilon$ for any state p, and
$p\alpha \equiv_0 q\beta$ for any p, q, $\alpha \neq \epsilon$ and $\beta \neq \epsilon$.
$p\alpha \equiv_{n+1} q\beta$ iff $p\alpha \equiv_0 q\beta$, and for each $a \in \mathcal{A}$,
if $p\alpha \xrightarrow{a} p'\alpha'$ then $\exists q'\beta'.\, q\beta \xrightarrow{a} q'\beta'$ and $p'\alpha' \equiv_n q'\beta'$, and
if $q\beta \xrightarrow{a} q'\beta'$ then $\exists p'\alpha'.\, p\alpha \xrightarrow{a} p'\alpha'$ and $p'\alpha' \equiv_n q'\beta'$.

The base relation $\equiv_0$ does not include all pairs of pushdown processes. For example any pair $p\epsilon$, $q\beta$ (when $p \neq q$) is excluded.

Proposition 3 $p\alpha \equiv q\beta$ *iff* $\forall n \geq 0.\, p\alpha \equiv_n q\beta$.

Our next concern is to provide a mechanism for composing and decomposing pushdown processes. We extend pushdown descriptions by augmenting their stack symbols Γ with a finite family of stack constants $\mathcal{W}$. Assume that the state set of the pushdown description is $\mathcal{P} = \{p_1, \ldots, p_k\}$. Each new stack symbol $W \in \mathcal{W}$ has an associated definition

$$W \stackrel{\text{def}}{=} (q_1\delta_1, \ldots, q_k\delta_k)$$

where each $q_i \in \mathcal{P}$, and each δ_i is a sequence of stack elements, possibly including constants. However we assume that if $\delta_i \neq \epsilon$ then its first symbol belongs to Γ. The intention is that for each state $p_j \in \mathcal{P}$ the behaviour of p_jW is that of $q_j\delta_j$, the jth component of the definition of W[§]. An *extended* pushdown process is an expression $p\alpha$ where $\alpha = \epsilon$ or $\alpha \in \Gamma(\Gamma \cup \mathcal{W})^*$. Basic transitions are unaffected, remaining of the form $pX \xrightarrow{a} q\alpha$ where $X \in \Gamma$ and $\alpha \in \Gamma^*$. However the prefix rule is generalised to

$$\text{if } pX \xrightarrow{a} q\alpha \text{ then } pX\beta \xrightarrow{a} [q\alpha\beta]$$

where bracketing $[\ldots]$ is defined as follows:

$$\begin{array}{lcl} [p_i\epsilon] & = & p_i\epsilon, \text{ and} \\ [p_iX\beta] & = & p_iX\beta \text{ when } X \in \Gamma, \text{ and} \\ [p_iW\beta] & = & [q_i\delta_i\beta] \text{ when } W \stackrel{\text{def}}{=} (q_1\delta_1, \ldots, q_k\delta_k) \end{array}$$

Therefore if $p\alpha$ is an extended pushdown process and $w \in \mathcal{A}^*$ and $p\alpha \xrightarrow{w} q\beta$ then $q\beta$ is also an extended pushdown process.

A-bisimilarity remains a congruence in the presence of constants.

Proposition 4 *If* $pX\alpha \equiv qY\beta$ *then* $pX\alpha W \equiv qY\beta W$.

Constants also provide a handle for composing pushdown processes, as illustrated by the next two results whose proofs are routine.

[§]This selection notation $(\ldots)$ for pushdown processes is used in [26, 4].

Proposition 5 *If* $W \stackrel{\text{def}}{=} (q_1\delta_1\beta, \ldots, q_k\delta_k\beta)$ *and* $V \stackrel{\text{def}}{=} (q_1\delta_1, \ldots, q_k\delta_k)$ *then* $pX\alpha W \equiv pX\alpha V\beta$.

Proposition 6 *If* $W \stackrel{\text{def}}{=} (q_1\delta_1, \ldots, q_k\delta_k)$ *and* $V \stackrel{\text{def}}{=} (r_1\lambda_1, \ldots, r_k\lambda_k)$ *and for each* $i : 1 \leq i \leq k$, $q_i\delta_i \equiv r_i\lambda_i$ *then* $pX\alpha W \equiv pX\alpha V$.

It is time to examine the condition of being normed more carefully. The *norm* of a process $p\alpha$, when α may contain constants, is a k-tuple $(n_1, \ldots, n_k)$ where each $n_i \in \mathbb{N} \cup \{\perp\}$. The component n_i is either the length of a shortest word w such that $p\alpha \xrightarrow{w} p_i\epsilon$ or there is no such word and n_i is the undefined element $\perp$. We let $n(p\alpha)$ be the norm of $p\alpha$, and we let $n(p\alpha)_i$ be its ith component. The restriction to normed processes implies that at least one entry in a norm is different from $\perp$. Let $\max(p\alpha)$ be the maximum defined entry in $n(p\alpha)$ and $\min(p\alpha)$ be the least defined entry. Finally we let $D(p\alpha)$ be the set $\{i : n(p\alpha)_i \neq \perp\}$. We can now slightly refine the previous Proposition.

Proposition 7 *If* $W \stackrel{\text{def}}{=} (q_1\delta_1, \ldots, q_k\delta_k)$ *and* $V \stackrel{\text{def}}{=} (r_1\lambda_1, \ldots, r_k\lambda_k)$ *and for each* $i \in D(pX\alpha)$. $q_i\delta_i \equiv r_i\lambda_i$ *then* $pX\alpha W \equiv pX\alpha V$.

Two useful measures on a pushdown description are now defined. First M is just greater than the maximum norm of a stack symbol in Γ:

$$M \stackrel{\text{def}}{=} 1 + \max\{\max(pX) \,:\, p \in \mathcal{P} \text{ and } X \in \Gamma\}$$

And second G is the maximum length of a stack sequence on the right hand side of a basic transition (where $|\,|$ is "length of"):

$$G \stackrel{\text{def}}{=} \max\{|\alpha| \,:\, pX \xrightarrow{a} q\alpha \text{ is a basic transition}\}$$

The family of finite constants $\mathcal{W}$ is partitioned into two. First are *simple* constants. Each has a definition $U \stackrel{\text{def}}{=} (q_1\delta_1, \ldots, q_k\delta_k)$ where each $\delta_i \in \Gamma^+$ and $|\delta_i| \leq MG$: constants are not allowed in their definition, and neither is the empty stack. Up to renaming of constants, there are only finitely many different simple constants, because of the constraint on their length. We now come to the first substantive result, where simple constants are used to provide a format for cancellation.

Lemma 1 *If* $pX\alpha \equiv q\beta\delta$ *and* $\beta \in \Gamma^+$ *and* $\max(pX) < \min(q\beta)$ *and* $|\beta| \leq M$ *then there is a simple* $U \stackrel{\text{def}}{=} (q_1\gamma_1, \ldots, q_k\gamma_k)$ *such that*

1. $pXU\delta \equiv q\beta\delta$, *and*

2. *for all* $i \in D(pX)$. $[p_i\alpha] \equiv q_i\gamma_i\delta$.

Proof: Suppose $pX\alpha \equiv q\beta\delta$ and $\beta \in \Gamma^+$ and $\max(pX) < \min(q\beta)$ and $|\beta| \leq M$. For each $i \in D(pX)$ consider a shortest w_i such that $pX \xrightarrow{w_i} p_i\epsilon$. Therefore, $pX\alpha \xrightarrow{w_i} [p_i\alpha]$. Since $pX\alpha \equiv q\beta\delta$ we know that $q\beta\delta \xrightarrow{w_i} q_i\lambda_i$ and $[p_i\alpha] \equiv q_i\lambda_i$. However, as $\max(pX) < \min(q\beta)$ it follows that $\lambda_i = \gamma_i\delta$ and $q\beta \xrightarrow{w_i} q_i\gamma_i$ and $\gamma_i \neq \epsilon$, and as $\beta \in \Gamma^+$ this means that $\gamma_i \in \Gamma^+$. Also because $|w_i| < M$ (and $|\beta| \leq M$) it follows that $|\gamma_i| \leq MG$. Let $U \stackrel{\text{def}}{=} (q_1\gamma_1, \ldots, q_k\gamma_k)$ where for each $i \in D(pX)$ $q_i\gamma_i$ is determined as above and for each $i \notin D(pX)$, $q_i\gamma_i = pX$. By definition U is a simple constant, and by construction 2 holds. By induction on n it follows that for any $p'\alpha'$ such that $\alpha' \in \Gamma^+$ and $D(p'\alpha') \subseteq D(pX)$, $p'\alpha'U\delta \equiv_n p'\alpha'\alpha$. Therefore $pXU\delta \equiv pX\alpha$, which implies 1. □

The other constants in W are *recursive*. Each recursive constant has a definition $V \stackrel{\text{def}}{=} (q_1\lambda_1, \ldots, q_k\lambda_k)$, where each λ_i is either the empty sequence or of the form $\lambda_i'V$ where V is the defining constant and λ_i' is a non-empty sequence of stack symbols which may contain simple (but not recursive) constants. Their first use is to relate bisimilarity and a-bisimilarity. Let $I \stackrel{\text{def}}{=} (p_1\epsilon, \ldots, p_1\epsilon)$ be an initial degenerate recursive constant. Clearly the following holds.

Proposition 8 $pX\alpha \sim qY\beta$ *iff* $pX\alpha I \equiv qY\beta I$.

The second and more substantive use of recursive constants is to provide a very delicate form of cancellation which turns out to be sufficiently strong for showing semi-decidability of a-bisimilarity. Simple cancellation, if $p\alpha\delta \equiv q\beta\delta$ then $p\alpha \equiv q\beta$, fails as does the more refined notion, if $p\alpha\delta \equiv q\beta\delta$ for infinitely many different δ then $p\alpha \equiv q\beta$. What could be even weaker? The idea is, roughly speaking, if $p\alpha\delta \equiv q\beta\delta$ for infinitely many δ then there are just finitely many "regular shapes" such that each δ has one of these shapes, where the notion of shape is captured using recursive constants. More precisely, we state the exact result.

Lemma 2 *Fix* pX *and* $q\beta$ *assuming that* $\beta \in \Gamma^+$ *and* $|\beta| \leq M$ and $\max(pX) < \min(q\beta)$. *There is a finite set of recursive constants* $\mathcal{V}$ *such that for any* α *which does not contain recursive constants and for any* δ *if* $pX\alpha\delta \equiv q\beta\delta$ *then there is a* $V \stackrel{\text{def}}{=} (q_1\lambda_1, \ldots, q_k\lambda_k)$ *in* $\mathcal{V}$ *such that*

1. $pX\alpha V \equiv q\beta V$, *and*
2. *for all* $j \in D(pX\alpha) \cup D(q\beta)$. *if* $\lambda_j = \epsilon$ *then* $[p_j\delta] \equiv [q_j\delta]$, *and*
3. *for all* j. *if* $\lambda_j = \lambda_j'V$ *then* $[p_j\delta] \equiv q_j\lambda_j'\delta$.

We shall prove this result in two stages.

The first stage is central, and shows how a family of finite shapes can be iteratively constructed.

Lemma 3 *Fix* $pX\alpha$ *and* $qY\beta$ *assuming that* α *and* β *do not contain recursive constants. There exists a finite family* $\mathcal{V}$ *of recursive constants such that for any* δ *if* $pX\alpha\delta \equiv qY\beta\delta$ *then there is a* $V \stackrel{\text{def}}{=} (q_1\lambda_1, \dots, q_k\lambda_k)$ *in* $\mathcal{V}$ *such that*

1. $pX\alpha V \equiv qY\beta V$, *and*
2. *for all* $i \in D(pX\alpha) \cup D(qY\beta)$. *if* $\lambda_i = \epsilon$ *then* $[p_i\delta] \equiv [q_i\delta]$, *and*
3. *for all* i. *if* $\lambda_i = \lambda_i' V$ *then* $[p_i\delta] \equiv q_i\lambda_i'\delta$.

Proof: Let $\Delta = \{\delta : pX\alpha\delta \equiv qY\beta\delta\}$. If Δ is empty then let $\mathcal{V}$ be empty, and the result holds. Otherwise assume a total ordering on the state set $\{p_1, \dots, p_k\}$, so that $p_i < p_j$ whenever $i < j$. We say that the recursive constant V is definitionally equivalent to V' if their definitions are the same except for their occurrences of V and V': in which case, for all $n \geq 0$, $pX\alpha V \equiv_n pX\alpha V'$ for any p, X and α.

For each $\delta \in \Delta$ we define the family $\{V_i^\delta : 0 \leq i \leq k^2\}$ iteratively, so that for each i properties 2 and 3 hold for V_i^δ and property 1 holds when $i = k^2$. Furthermore, for each i the set $\{V_i^\delta : \delta \in \Delta\}$ when quotiented by definitional equivalence is finite. From this the result follows by taking $\mathcal{V}$ to be the family $\{V_{k^2}^\delta : \delta \in \Delta\}$ after quotienting by definitional equivalence.

The element $V_0^\delta \stackrel{\text{def}}{=} (p_1\epsilon, \dots p_k\epsilon)$. Clearly, both 2 and 3 hold, and the set $\{V_0^\delta : \delta \in \Delta\}$ is a singleton up to definitional equivalence. Assume V_i^δ for $0 \leq i < k^2$ has been defined, and that both 2 and 3 hold for it, and that the set $\{V_i^\delta : \delta \in \Delta\}$ is finite up to definitional equivalence. If property 1 is also true then let V_j^δ, for all $j : i \leq j \leq k^2$ be V_i^δ. Otherwise V_{i+1}^δ is constructed as a refinement of V_i^δ. As 1 fails there is a least $n \geq 0$ such that $pX\alpha V_i^\delta \not\equiv_n qY\beta V_i^\delta$, and we know that $n > 0$. However, we also know that $pX\alpha\delta \equiv qY\beta\delta$. Hence $pX\alpha \xrightarrow{a} [p_1'\alpha_1]$ and $qY\beta \xrightarrow{a} [q_1\beta_1]$ for some a and $p_1'\alpha_1$ and $q_1\beta_1$ such that $[p_1'\alpha_1\delta] \equiv [q_1\beta_1\delta]$ and $[p_1'\alpha_1 V_i^\delta] \not\equiv_{n-1} [q_1\beta_1 V_i^\delta]$. The sequences α_1 and β_1 do not contain recursive constants. If both α_1 and β_1 are non-empty then we can obtain a subsequent pair $p_2'\alpha_2$ and $q_2\beta_2$ such that $[p_2'\alpha_2\delta] \equiv [q_2\beta_2\delta]$ and $[p_2'\alpha_2 V_i^\delta] \not\equiv_{n-2} [q_2\beta_2 V_i^\delta]$, and so on. Therefore as n is finite we must reach a pair $p_j'\alpha_j$ and $q_j\beta_j$ such that $[p_j'\alpha_j\delta] \equiv [q_j\beta_j\delta]$ and $[p_j'\alpha_j V_i^\delta] \not\equiv_{n-j} [q_j\beta_j V_i^\delta]$, where α_j or β_j is empty (and neither contain recursive constants).

Without loss of generality assume that $\alpha_j = \epsilon$. Let $r\lambda$ be the entry for p_j' in V_i^δ. There are two cases.

Case 1 $\lambda = \epsilon$. Now consider β_j. First assume that $\beta_j \neq \epsilon$. Let V_{i+1}^δ be V_i^δ except that $q_j\beta_j V_{i+1}^\delta$ is associated with the state p_j' instead of $r\epsilon$ (and throughout the other entries V^δ_{i+1} replaces V^δ_i). Clearly properties 2 and 3 hold for V^δ_{i+1} given that they hold for V^δ_i. Next assume instead that $\beta_j = \epsilon$. Assume that $s\gamma$ is q_j's entry in V^δ_i. There are two subcases. First, $\gamma \neq \epsilon$. Then V^δ_{i+1} is V^δ_i except that $s\gamma$ is associated with p'_j (and throughout all entries V^δ_{i+1} replaces V^δ_i). Second, $\gamma = \epsilon$. If $p'_j < q_j$ in the total order on states V^δ_{i+1} is V^δ_i except that $r\epsilon$ is associated with q_j instead of $s\epsilon$, and if $q_j < p'_j$ then V^δ_{i+1} is V^δ_i except that $s\epsilon$ is associated with p'_j (and in both cases throughout the other entries V^δ_{i+1} replaces V^δ_i). Again clearly properties 2 and 3 both hold for V^δ_{i+1}.

Case 2 $\lambda \neq \epsilon$, and so $\lambda = \lambda' V^\delta_i$. Again consider β_j. If $\beta_j \neq \epsilon$ then as $[p'_j\delta] \equiv q_j\beta_j\delta$ by 3 we know that $[p'_j\delta] \equiv r\lambda'\delta$, and so $r\lambda'\delta \equiv q_j\beta_j\delta$. However $r\lambda' V^\delta_i \not\equiv_{n-j} q_j\beta_j V^\delta_i$ (as $[p'_j\alpha_j V^\delta_i] = r\lambda' V^\delta_i$). Therefore the proof proceeds as before by defining further pairs $r_m\lambda'_m$ and $q_{j+m}\beta_{j+m}$ such that $[r_m\lambda'_m\delta] \equiv [q_{j+m}\beta_{j+m}\delta]$ and $[r_m\lambda'_m V^\delta_i] \not\equiv_{n-(j+m)} [q_{j+m}\beta_{j+m}V^\delta_i]$. Otherwise $\beta_j = \epsilon$. Let $s\gamma$ be the entry for q_j in V^δ_i. If $\gamma = \epsilon$ then we proceed as in case 1 above (with the α and β roles reversed). If $\gamma \neq \epsilon$ then $\gamma = \gamma' V^\delta_i$ where γ' does not contain recursive constants. By 3 for V^δ_i $[q_j\delta] \equiv s\gamma'\delta$ and $[p'_j\delta] \equiv r\lambda'\delta$ and therefore $r\lambda'\delta \equiv s\gamma'\delta$, and also $r\lambda' V^\delta_i \not\equiv_{n-j} s\gamma' V^\delta_i$. Therefore again we define further pairs $r_m\lambda'_m$ and $s_m\gamma'_m$ such that $[r_m\lambda'_m\delta] \equiv [s_m\gamma'_m\delta]$, and $[r_m\lambda'_m V^\delta_i] \not\equiv_{n-(j+m)} [s_m\gamma'_m V^\delta_i]$. As n is finite there can only be finitely many invocations of 3 before V^δ_i is updated according to the first case.

The construction of V^δ_{i+1} updates exactly one entry in V^δ_i which must be of the form $r\epsilon$. The update becomes either non-empty or is replaced by an empty entry from an earlier state in the total order. For each V^δ_i the number of possible different recursive constants V^δ_{i+1} by this construction given that $pX\alpha V^\delta_i \not\equiv_n qY\beta V^\delta_i$ (with n least) is finite. Consequently the set $\{V^\delta_{i+1} : \delta \in \Delta\}$ is finite up to definitional equivalence. Moreover given the regime for updating, there can be at most k^2 updates, and therefore $pX\alpha V^\delta_{k^2} \equiv qY\beta V^\delta_{k^2}$. □

This takes us to the proof of Lemma 2.

Proof of Lemma 2: Assume pX and $q\beta$ where $\beta \in \Gamma^+$ and $|\beta| \leq M$ and $\max(pX) < \min(q\beta)$. We show the following:

There exists a finite family $\mathcal{U}$ of simple constants and a finite family $\mathcal{V}$ of recursive constants such that for any α not containing recursive constants and for any δ if $pX\alpha\delta \equiv q\beta\delta$ then there is a $U \stackrel{\text{def}}{=} (r_1\gamma_1, \ldots, r_k\gamma_k)$ in $\mathcal{U}$ and a $V \stackrel{\text{def}}{=} (q_1\lambda_1, \ldots, q_k\lambda_k)$ in $\mathcal{V}$ such that

1. $pXUV \equiv q\beta V$, and
2. for all $j \in D(pX)$. $[p_j\alpha V] \equiv r_j\gamma_j V$, and
3. for all $j \in D(pX\alpha) \cup D(q\beta)$. if $\lambda_j = \epsilon$ then $[p_j\delta] \equiv [q_j\delta]$, and

4. for all j. if $\lambda_j = \lambda'_j V$ then $[p_j\delta] \equiv q_j\lambda'_j\delta$.

Lemma 2 now follows because 1 and 2 imply that $pX\alpha V \equiv q\beta V$ (using Propositions 5 and 7). Let $\Delta = \{(\alpha,\delta) : pX\alpha\delta \equiv q\beta\delta\}$. If Δ is empty let $\mathcal{U}$ and $\mathcal{V}$ be empty, and the result follows. Otherwise Δ is non-empty. By Lemma 1 for each $(\alpha,\delta) \in \Delta$ there is a simple $U \stackrel{\text{def}}{=} (r_1\gamma_1,\ldots,r_k\gamma_k)$ such that $pXU\delta \equiv q\beta\delta$ and for all $j \in D(pX)$. $[p_j\alpha\delta] \equiv r_j\gamma_j\delta$. By the construction in Lemma 1 we know that there are only finitely many such U (as each component of their definition is drawn from the set $\{q'\gamma' : q\beta \xrightarrow{w} q'\gamma' \text{ for } |w| \leq \max(pX)\}$). Let $U_1,\ldots,U_l$ be the different simple constants. We partition Δ into sets $\Delta_1,\ldots,\Delta_l$ such that for each $(\alpha,\delta) \in \Delta_t$, with $U_t \stackrel{\text{def}}{=} (r_1\gamma_1,\ldots,r_k\gamma_k)$, $pXU_t\delta \equiv q\beta\delta$ and for all $j \in D(pX)$. $[p_j\alpha_i\delta] \equiv r_j\gamma_j\delta$. By Lemma 3 for each Δ_t there are finitely many recursive V such that $pXU_tV \equiv q\beta V$ and for each $(\alpha,\delta) \in \Delta_t$ one of these recursive constants $V \stackrel{\text{def}}{=} (q_1\lambda_1,\ldots,q_k\lambda_k)$ has the property that $[p_j\delta] \equiv [q_j\lambda'_j\delta]$ for all $j \in D(q\beta) \cup D(pXU_t)$ (where if $\lambda_j = \lambda' V$ then $\lambda'_j = \lambda'$ and if $\lambda_j = \epsilon$ then $\lambda'_j = \epsilon$). Let $\mathcal{V}_t$ be this finite family of recursive constants. The proof is finished if for each $(\alpha,\delta) \in \Delta_t$ there is a $V \in \mathcal{V}_t$ such that for all $j \in D(pX)$. $[p_j\alpha V] \equiv q_j\gamma_j V$. Using a similar proof method to that in Lemma 3 we refine this set of recursive constants so that it becomes true. Let $\mathcal{V}_{t0}$ be $\mathcal{V}_t$. Take the least n such that there is an $(\alpha,\delta) \in \Delta_t$ and $V \stackrel{\text{def}}{=} (q_1\lambda_1,\ldots,q_k\lambda_k) \in \mathcal{V}_{t0}$ with $[p_j\delta] \equiv [q_j\lambda'_j\delta]$ for all $j \in D(q\beta) \cup D(pX\alpha)$ such that $[p_j\alpha V] \not\equiv_n q_j\gamma_j V$. However we know that $[p_j\alpha\delta] \equiv q_j\gamma_j\delta$. Therefore, by the technique in the proof of Lemma 3 this means that for some j, $[p_jV] \not\equiv_0 [q'\lambda V]$ and $[p_j\delta] \equiv [q'\lambda\delta]$ and the jth component of V, $q_j\lambda_j$, has the form $r\epsilon$. We now update V. There are two cases. First if $\lambda \neq \epsilon$ then each entry $r\epsilon$ in V is replaced with $q'\lambda V$. If $\lambda = \epsilon$ then replace each entry $r\epsilon$ with the entry for q' (which cannot be $r\epsilon$). All other entries in V remain untouched. Call the resulting constant V'. A small exercise shows that for any α and β if $[p\alpha V] \equiv [q\beta V]$ then $[p\alpha V'] \equiv [q\beta V']$ (and so, in particular, $pXU_tV' \equiv q\beta V'$): this is the reason why all entries $r\epsilon$ are updated at once. Let $\mathcal{V}_{t1}$ be $\mathcal{V}_{t0}$ with V replaced by V'. Now keep repeating the procedure. As with the argument in Lemma 3 the number of possible updates must be finite (here less than rk^2 where r is the number of constants in $\mathcal{V}_{t0}$). □

4 Tableaux

Given a pushdown description and two normed processes $p\alpha$ and $q\beta$, the aim is to show semi-decidability of $p\alpha \sim q\beta$. If $\alpha = \epsilon$ or $\beta = \epsilon$ then checking for decidability is clear. Otherwise the problem reduces to decidability of $p\alpha I \equiv q\beta I$ where I is the initial recursive constant. Therefore, we need to extend the pushdown description with a finite family of stack constants

$\mathcal{W}$. As we saw in the previous section we can include all potential simple constants. For the recursive constants we have no such upper bound (and hence the reason for semi-decidability). However, besides the initial constant I, we only need to introduce a finite family (see Lemma 2 of the previous section) for pairs pX and $q\beta$, $\beta \in \Gamma^+$, and $|\beta| \leq M$, and for which there is an α and a δ such that $pX\alpha\delta \equiv q\beta\delta$.

We complete the decidability result by presenting a tableau proof system for a-bisimilarity. The proof system is goal directed, and consists of a finite set of rules each of the form

$$\frac{\text{Goal}}{\text{Subgoal}_1 \ \ldots \ \text{Subgoal}_n}\ \mathcal{C}$$

where Goal is what currently is to be proved and the subgoals are what it reduces to, and $\mathcal{C}$ is a possible side condition on the rule application. Each goal and subgoal has the form $p\alpha = q\beta$ (the proof analogue of $p\alpha \equiv q\beta$) where the constituents are extended normed pushdown processes. Each rule is backwards sound: if all the subgoals are true then so is the goal. As we are dealing with infinite state systems there is also the important notion of when a current goal counts as terminal, for the rules only apply to nonterminals. Terminal goals are classified as either successful or unsuccessful. A *tableau proof* for Goal is a finite proof tree, whose root is Goal and all of whose leaves are successful terminals, and all of whose inner subgoals are the result of an application of one of the rules to the goal immediately above them. If the successful terminals are true it follows that the root goal is also true. We show that $pX\alpha \sim qY\beta$ iff there is a tableau proof for $pX\alpha I = qY\beta I$.

The tableau proof rules are presented in Figure 1. There is a cancellation rule CAN which introduces recursive constants and a reduction rule RED which introduces simple constants: their formulation directly reflects the important Lemmas of the previous section. The third rule is unfold UNF: for each transition from $pX\alpha$ there is a corresponding transition from $qY\beta$, and the resulting pairs become subgoals (and vice-versa). A tableau is built from proof steps, possibly interspersed with applications of UNF. A proof step has the form

$$\cfrac{pX\alpha\delta = q\beta\delta}{\ldots[q_i\lambda_i'\delta] = [p_i\delta]\ldots \qquad \cfrac{pX\alpha V = q\beta V}{\ldots[p_j\alpha V] = q_j\gamma_j V\ldots \quad \cfrac{pXUV = q\beta V}{\ldots}\ \text{UNF}}\ \text{RED}}\ \text{CAN}$$

or when CAN is not applicable, it has the form

Cancellation (CAN)

$$\frac{pX\alpha\delta = q\beta\delta}{[q_{i1}\lambda'_{i1}\delta] = [p_{i1}\delta]\ldots[q_{il}\lambda'_{il}\delta] = [p_{il}\delta] \quad pX\alpha V = q\beta V}\ C1$$

Condition C1:

1. $\beta \in \Gamma^+$ and $|\beta| \leq M$ and $\max(pX) < \min(q\beta)$ and $|\delta| > 1$, and
2. $V \stackrel{\text{def}}{=} (q_1\lambda_1, \ldots q_k\lambda_k)$ is recursive, and
3. $D(pX\alpha) \cup D(q\beta) = \{i1, \ldots, il\}$, and
4. if $\lambda_{ij} = \epsilon$ then $\lambda'_{ij} = \epsilon$, and if $\lambda_{ij} = \lambda''V$ then $\lambda'_{ij} = \lambda''$.

Reduction (RED)

$$\frac{pX\alpha = q\beta\delta}{[p_{i1}\alpha] = q_{i1}\gamma_{i1}\delta\ldots\ [p_{il}\alpha] = q_{il}\gamma_{il}\delta \quad pXU\delta = q\beta\delta}\ C2$$

Condition C2:

1. $\beta \in \Gamma^+$ and $|\beta| \leq M$ and $\max(pX) < \min(q\beta)$ and $|\delta| > 1$, and
2. $U \stackrel{\text{def}}{=} (q_1\gamma_1, \ldots, q_k\gamma_k)$ is simple, and $D(pX) = \{i1, \ldots, il\}$.

Unfold (UNF)

$$\frac{pX\alpha = qY\beta}{p_1\alpha_1 = q_1\beta_1 \quad \ldots \quad p_l\alpha_l = q_l\beta_l}\ C3$$

Condition C3: For any a

1. if $pX\alpha \xrightarrow{a} p'\alpha'$ then $\exists i : 1 \leq i \leq l.\ p'\alpha' = p_i\alpha_i$ and $qY\beta \xrightarrow{a} q_i\beta_i$,
2. if $qY\beta \xrightarrow{a} q'\beta'$ then $\exists i : 1 \leq i \leq l.\ q'\beta' = q_i\beta_i$ and $pX\alpha \xrightarrow{a} p_i\alpha_i$.

Figure 1: Tableau rules

Successful terminals

1. $p\alpha = p\alpha$
2. $p\alpha = q\beta$ and in the proof tree the same equation $p\alpha = q\beta$ occurs above on the path to the root.

Unsuccessful terminals

1. $p\epsilon = q\epsilon$ and $p \neq q$
2. $p\alpha = q\beta$ and $\min(p\alpha) \neq \min(q\beta)$
3. $p\alpha = q\beta$ and $p\alpha \xrightarrow{a}$ but not$(q\beta \xrightarrow{a})$
4. $p\alpha = q\beta$ and $q\beta \xrightarrow{a}$ but not$(p\alpha \xrightarrow{a})$

Figure 2: Terminal nodes

$$\dfrac{pX\alpha = q\beta\delta}{\ldots[p_i\alpha] = q_i\gamma_i\delta\ldots \qquad \dfrac{pXU\delta = q\beta\delta}{\ldots[q_j\lambda'_j\delta] = [p_j\delta]\ldots \quad \dfrac{pXUV = q\beta V}{\ldots}\;\mathrm{UNF}}\;\mathrm{CAN}}\;\mathrm{RED}$$

The idea is to repeat proof steps whose roots are themselves leaves of a proof step except when the side conditions for CAN and RED do not apply in which case the UNF rule is applied instead. The proof step starting with CAN takes priority over the one starting with RED, which takes priority over the use of UNF. Note that a recursive constant may only appear as a final stack symbol, and that simple constants are only explicitly introduced into processes on the left side of $=$: they can appear on the right hand side of $=$ through the presence of recursive constants, when a process becomes $[q_iV]$.

The conditions for being a terminal node are described in Figure 2. A node labelled $p\alpha = q\beta$ in a proof tree is a successful terminal if it is an identity ($q\beta$ is $p\alpha$) or a repeat: that is, there is a node above it on the path to the root also labelled $p\alpha = q\beta$. In the conditions for being an unsuccessful terminal we use the standard notation $r\lambda \xrightarrow{a}$ as an abbreviation for $\exists r'\lambda'.\, r\lambda \xrightarrow{a} r'\lambda'$. Clearly, if a node labelled $p\alpha = q\beta$ is an unsuccessful terminal then $p\alpha \not\equiv q\beta$.

The tableau proof rules are backwards sound with respect to the approximants $\equiv_n$, and this fact is used in the following soundness result.

Lemma 1 *If there is a tableau proof for* $p\alpha = q\beta$ *then* $p\alpha \equiv q\beta$.

Proof: It is straightforward to show that for any $n \geq 0$ if all the leaves $p'_i\alpha'_i = q'_i\beta'_i$ of a proof step have the property that $p'_i\alpha'_i \equiv_n q'_i\beta'_i$ then the root of the proof step $p'\alpha' = q'\beta'$ has the property $p'\alpha' \equiv_{n+1} q'\beta'$. This is clearly the case for the UNF rule. For the other two proof steps it follows by a similar kind of backwards soundness for the rules CAN and RED: in the case of CAN if each leaf $[q_{ij}\lambda'_{ij}\delta] = [p_{ij}\delta]$ has the property $[q_{ij}\lambda'_{ij}\delta] \equiv_n [p_{ij}\delta]$ and the leaf $pX\alpha V = q\beta V$ has the property $pX\alpha \equiv_{n+1} q\beta V$ then the root $pX\alpha\delta = q\beta\delta$ obeys $pX\alpha\delta \equiv_{n+1} q\beta\delta$, and in the case of RED if each leaf $[p_{ij}\alpha] = q_{ij}\gamma_{ij}\delta$ has the property $[p_{ij}\alpha] \equiv_n q_{ij}\gamma_{ij}\delta$ and the leaf $pXU\delta = q\beta\delta$ has the feature $pXU\delta \equiv_{n+1} q\beta\delta$ then the root $pX\alpha = q\beta\delta$ is such that $pX\alpha \equiv_{n+1} q\beta\delta$.

Assume that there is a successful tableau built using proof steps with root $p\alpha = q\beta$, and suppose that $p\alpha \not\equiv q\beta$. Clearly it can not be the case that $p\alpha \not\equiv_0 q\beta$ (for then $p\alpha = q\beta$ would be an unsuccessful terminal). Hence there is a least n such that $p\alpha \equiv_n q\beta$ and $p\alpha \not\equiv_{n+1} q\beta$. Consequently there is a leaf $p'\alpha' = q'\beta'$ of the first proof step such that $p'\alpha' \not\equiv_n q'\beta'$ by backwards soundness described above. Take the leaf $p_{i1}\alpha_{i1} = q_{i1}\beta_{i1}$ for which there is a least $n1$ such that $p_{i1}\alpha_{i1} \equiv_{n1} q_{i1}\beta_{i1}$ and $p_{i1}\alpha_{i1} \not\equiv_{n1+1} q_{i1}\beta_{i1}$ (and so $n1 < n$). Repeatedly apply this argument. At each stage we have a root of a proof step $p_{ij}\alpha_{ij} = q_{ij}\beta_{ij}$ such that $p_{ij}\alpha_{ij} \equiv_{nj} q_{ij}\beta_{ij}$ and $p_{ij}\alpha_{ij} \not\equiv_{nj+1} q_{ij}\beta_{ij}$, and $nj < nl$ whenever $j > l$. As the tableau is finite, this sequence picks out a finite path through the tableau (when the intermediate subgoals between the root of a proof step and its chosen leaf are added). However the tableau is successful and so the final goal is a successful terminal. Clearly it can not be an identity $p'\alpha' = p'\alpha'$ since $p'\alpha' \equiv_n p'\alpha'$ for all $n \geq 0$. Consequently the final goal $p'\alpha' = q'\beta'$ also occurs earlier in the path. But now we have a contradiction as there can not be nj and nl (with $nj \neq nl$) where $p'\alpha' \equiv_{nj} q'\beta'$ and $p'\alpha' \not\equiv_{nj+1} q'\beta'$ and $p'\alpha' \equiv_{nl} q'\beta'$ and $p'\alpha' \not\equiv_{nl+1} q'\beta'$. □

Now the main result which shows completeness of the tableau method.

Theorem 1 $pX\alpha \sim qY\beta$ *iff there is a tableau proof for* $pX\alpha I = qY\beta I$.

Proof: One half follows by Lemma 1 above and Proposition 8 of the previous section: if there is a tableau proof for $pX\alpha I = qY\beta I$ then $pX\alpha \sim qY\beta$. For the other half suppose $pX\alpha \sim qY\beta$. By Proposition 8 $pX\alpha I \equiv qY\beta I$. We now construct a tableau proof for $pX\alpha I = qY\beta I$. First we introduce all the appropriate constants. For each pX and $q\beta$, $\beta \in \Gamma^+$, $|\beta| \leq M$ and where $\max(pX) < \min(q\beta)$, we define finite sets of constants as determined by Lemmas 1 and 2 of the previous section. Now we repeatedly build proof steps that preserve truth until we reach terminals. Lemma 2 of the previous section and the introduction of constants guarantee that if $pX\alpha\delta = q\beta\delta$

is a subgoal with $\beta \in \Gamma^+$ and $|\beta| \leq M$ and $\max(pX) < \min(q\beta)$ and $|\delta| > 1$ then the proof step whose initial rule is CAN is applicable. If CAN is not applicable to $pX\alpha = q\beta\delta$ and $\beta \in \Gamma^+$ and $|\beta| \leq M$ and $\max(pX) < \min(q\beta)$ and $|\delta| > 1$ then the other proof step applies. Otherwise the rule UNF is always applicable to a nonterminal subgoal (as any process in a tableau has the form $pX\alpha$ or $p\epsilon$). Clearly, as truth is preserved it is not possible to reach an unsuccessful terminal goal.

The only impediment is the possibility that the proof construction never ends, that we build a proof tree with an infinite path. The rest of the proof shows that this is impossible. Note that because of normedness, for each $k \geq 1$ the set $\{r\delta \,:\,$ for some $p\alpha$ such that $|\alpha| < k$, $r\delta \equiv p\alpha$ and δ and α contain at most one recursive constant$\}$ is finite.

Let S be the following measure

$$\max(\{|\lambda_i| + M \,:\, V \stackrel{\text{def}}{=} (q_1\lambda_1, \ldots, q_k\lambda_k) \text{ is recursive}\} \cup \{MG + 2\} \cup \{4\})$$

In a goal $p\alpha = q\beta$ we say that $p\alpha$ is a left process and $q\beta$ is a right process. Note the following observations:

1. Any occurrence of a left or right process has at most one recursive constant which can only appear as the final stack symbol.

2. The only way that simple constants can be introduced into a right process is through the UNF rule, in the circumstance that $pX\alpha = qYV$ and $qY \xrightarrow{a} r\epsilon$: this means that in a right process $q\beta U\delta$ where U is simple, $|U\delta| < S$.

3. The only circumstance that both CAN and RED are not applicable to a nonterminal goal $pX\alpha = q\beta\delta$, $|\beta| \in \Gamma^+$, is when $\max(pX) \not< \min(q\beta)$ and $\delta = \epsilon$ or $\delta = V$ or $\delta = U\delta_1$ and U is simple. In which case $|\beta\delta| < S$.

Suppose $p_1\alpha_1 = q_1\beta_1, \ldots, p_n\alpha_n = q_n\beta_n, \ldots$ is an infinite path of distinct goals in the tableau (with $p_1\alpha_1 = pX\alpha I$ and $q_1\beta_1 = qY\beta I$). Let g_i be the ith goal in this sequence. We show that there must be a repeat goal in this sequence: for some i goal g_i is a terminal. A little notation. Consider the rule that takes g_i to g_{i+1}: if the rule is UNF we say g_{i+1} is an UNF successor of g_i, if the rule is CAN then we say it is an l-successor if it is of the form $[q_i\lambda_i'\delta] = [p_i\delta]$ and an r-successor if it has the form $pX\alpha V = q\beta V$. Similarly, if the rule is RED then we say it is an l-successor if it is of the form $[p_i\alpha] = q_i\gamma\delta$ and an r-successor if it has the form $pXU\delta = q\beta\delta$.

The following are true for the sequence $g_1, \ldots, g_n, \ldots$

1. There is an i such that for all $j \geq i$ the goal g_j is not an r-successor of CAN because the right process of such a goal has the form $q\beta V$ where $|\beta| < M$ (and so $|\beta V| < S$).

2. There is an i such that for all $j \geq i$ the goal g_j is not an UNF successor. First by 1 above there is an i such that no later goal is an UNF successor via the two kinds of proof steps (as it would then be preceded by a right successor of CAN). Secondly, by observation 3 earlier CAN and RED are not applicable to a goal only when its right process is small.

Hence there is a suffix $g_i, \dots, g_n, \dots$ of goals such that each goal is a successor of RED or an l-successor of CAN. If in this suffix the goal g_j is an r-successor of RED then g_{j+1} is an l-successor of CAN. Clearly, there cannot be $j \geq i$ such that for all $k \geq j$ the goal g_k is an l-successor of CAN as each application reduces the right process by at least one stack symbol. Similarly, there cannot be $j \geq i$ such that for all $k \geq j$ the goal g_k is an l-successor of RED as each application reduces the left process by one stack symbol, except when a simple constant or a recursive constant is encountered: in the first case the stack may temporarily expand but the number of simple constants is reduced by one and in the second case, whenever a recursive constant is exposed the left process has size less than S. Consequently infinitely often there must be a change over from l-successors of CAN to a successor of RED. Consider any subsequence $g_i, \dots, g_{i+k}$ of goals which are l-successors of CAN and such that CAN is not applicable to g_{i+k}. So g_{i+k-1} has the form $pX\alpha\delta = q\beta\delta$ and g_{i+k} has the form $[q_i\lambda_i'\delta] = [p_i\delta]$: either δ is small (of length less than $M + 2$) or δ has the form $\beta_1 U \delta_1$ where U is simple (and $|\beta_1| \leq M$). Hence either $|\beta\delta| < S$ or the right process of g_{i+k} has size less than S. Hence there can not be such an infinite sequence of goals without a repeat. □

5 Conclusion and open questions

We have shown that bisimulation equivalence is decidable for normed pushdown processes. However, no complexity measure is available as the decision procedure essentially relies on Lemma 2 of Section 3. Moreover, it is not clear if the proof can be generalised to include unnormed pushdown processes: the main problem is that we are then unable to show completeness of the tableau proof system.

Normed BPA processes and normed pushdown processes are two means for generating the context-free languages (without the empty string). These languages are also generable using richer descriptions. Let Γ be a finite family of nonterminals and $\mathcal{A}$ a finite family of actions. A basic transition has the form $\alpha \xrightarrow{a} \beta$ where $\alpha, \beta \in \Gamma^*$ and $a \in \mathcal{A}$. We can then define a process as a sequence of nonterminals whose behaviour is determined by the basic transitions together with the prefix rule:

$$\text{PRE} \quad \text{if } \alpha \xrightarrow{a} \beta \text{ then } \alpha\delta \xrightarrow{a} \beta\delta$$

The notion of normed process is then appropriately definable (for instance by utilising a finite set of final states which have no transitions, as in [24]).

In the table below is a Caucal hierarchy of process descriptions according to how the family of basic transitions is specified. In each case we assume a finite family of rules. Type 3 captures finite-state processes, Type 2 captures BPA processes in Greibach normal form, and Type $1\frac{1}{2}$, in fact, captures pushdown processes. For Type 0 and below this means that in each case there are finitely many basic transitions. In the other cases R_1 and R_2 are regular expressions over the nonterminals. The idea is that each rule $R_1 \xrightarrow{a} \beta$ stands for the possibly infinite family of basic transitions $\{\alpha \xrightarrow{a} \beta : \alpha \in R_1\}$ and $R_1 \xrightarrow{a} R_2$ stands for the family $\{\alpha \xrightarrow{a} \beta : \alpha \in R_1 \text{ and } \beta \in R_2\}$. For instance a Type -1 rule of the form $X^*Y \xrightarrow{a} Y$ includes for each n the basic transition $X^nY \xrightarrow{a} Y$.

	Basic Transitions
Type -2	$R_1 \xrightarrow{a} R_2$
Type -1	$R_1 \xrightarrow{a} \beta$
Type 0	$\alpha \xrightarrow{a} \beta$
Type $1\frac{1}{2}$	$\alpha \xrightarrow{a} \beta$ where $\lvert\alpha\rvert = 2$ and $\lvert\beta\rvert > 0$
Type 2	$X \xrightarrow{a} \beta$
Type 3	$X \xrightarrow{a} Y$ or $X \xrightarrow{a} \epsilon$

This Caucal hierarchy is implicit in Caucal's work on understanding context-free graphs, and understanding when the monadic second-order theory of graphs is decidable [8, 7, 9, 6, 10]. With respect to language equivalence, there is no distinction between Type -2 and Type 2. With respect to bisimulation equivalence this is not the case. However Caucal showed in [8] that Type 0 processes coincide (up to isomorphism of their transition graphs) with pushdown processes (and hence Type $1\frac{1}{2}$ processes). Consequently a corollary of the main result in this paper is that bisimulation equivalence is decidable for normed Type 0 processes. This leaves as open questions whether it is also decidable for normed processes in Type -1 and Type -2. These could turn out to be very difficult open questions, as we note below. One other reason for interest in these problems is that decidability questions about weak bisimulation can be coded as decidability questions about strong bisimulation at a higher level in the hierarchy: for instance, weak bisimulation of totally normed BPA processes can be coded into strong bisimulation of normed pushdown processes [18].

Another area for further work is the long standing issue of decidability of language equivalence for deterministic pushdown automata, DPDA. These are normed deterministic pushdown processes which also have ϵ moves. When processes are deterministic language equivalence coincides with bisimulation equivalence. Hence the result proved in this paper generalises [26] which proves decidability of language equivalence for real-time DPDA accepting by the empty stack: a real-time DPDA has no ϵ moves. However the proof method is very different: [26] uses Valiant's parallel stacking technique whereas the method here relies on congruence and cancellation. Further work is needed to establish whether we can offer new insight into this difficult open problem. However this problem is related to the open questions above, as a DPDA with ϵ transitions can be coded as Type -1 process without ϵ transitions. For a different perspective on these issues see [9].

Acknowledgement: The author would like to thank Mojmir Kretinsky whose visit to Edinburgh rekindled the work reported here, and Olaf Burkart and Didier Caucal for comments on an earlier draft.

References

[1] Baeten, J. and Bergstra, J. (1991). Recursive process definitions with the state operator. *Theoretical Computer Science*, **82**, 285-302.

[2] Baeten, J., Bergstra, J., and Klop, J. (1987). Decidability of bisimulation equivalence for processes generating context-free languages. *Lecture Notes in Computer Science*, **259**, 94-113.

[3] Baeten, J., Bergstra, J., and Klop, J. (1993). Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of Association of Computing Machinery*, **40**, 653-682.

[4] Burkart, O., and Steffen, B.(1995). Composition, decomposition, and model checking of pushdown processes. *Nordic Journal of Computing*, **2**, 89-125.

[5] Burkart, O., Caucal, D., and Steffen, B. (1994). An elementary decision procedure for arbitrary context-free processes. *Tech. Report* **94-28**, RWTH Aachen (extended version of *Lecture Notes in Computer Science*, **969**, 423-433).

[6] Burkart, O., Caucal, D., and Steffen, B. (1996). Bisimulation collapse and the process taxonomy. *Lecture Notes in Computer Science*, **1119**, 247-262.

[7] Caucal, D. (1990). Graphes canoniques de graphes algébriques. *Informatique Théorique et Applications (RAIRO)*, **24**,339-352.

[8] Caucal, D. (1992). On the regular structure of prefix rewriting. *Theoretical Computer Science*, **106**, 61-86.

[9] Caucal, D. (1995). Bisimulation of context-free grammars and of pushdown automata. *CSLI Lecture Notes*, **53**, 85-106.

[10] Caucal, D. (1996). On infinite transition graphs having a decidable monadic theory. *Lecture Notes in Computer Science*, **1099**, 194-205.

[11] Caucal, D., and Monfort, R. (1990). On the transition graphs of automata and grammars. *Lecture Notes in Computer Science*, **484**, 311-337.

[12] Christensen, S., Hirshfeld, Y., and Moller, F. (1993). Decomposability, decidability and axiomatisability for bisimulation equivalence on basic parallel processes. *Proceedings 8th Annual Symposium on Logic in Computer Science*, IEEE Computer Science Press.

[13] Christensen, S., Hirshfeld, Y., and Moller, F. (1993). Bisimulation is decidable for basic parallel processes. *Lecture Notes in Computer Science*, **715**, 143-157.

[14] Christensen, S., Hüttel, H., and Stirling, C. (1995). Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, **121**, 143-148.

[15] Groote, J. (1992). A short proof of the decidability of bisimulation for normed BPA processes. *Information Processing Letters*, **42**, 167-171.

[16] Groote, J., and Hüttel, H. (1994). Undecidable equivalences for basic process algebra. *Information and Computation*, **115**, 354-371.

[17] Hirshfeld, Y. (1994). Deciding equivalences in simple process algebras. *Tech. Report* ECS-LFCS-94-294, Edinburgh University.

[18] Hirshfeld, Y. (1996). Bisimulation trees and the decidability of weak bisimulations. *To appear.*

[19] Hirshfeld, Y., Jerrum, M., and Moller, F. (1994). A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Procs. IEEE 35th Annual Symposium on Foundations of Computer Science*, 623-631.

[20] Hopcroft, J., and Ullman, J. (1979). *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley.

[21] Hüttel, H. (1994). Undecidable equivalences for basic parallel processes. *Lecture Notes in Computer Science*, **789**.

[22] Hüttel, H., and Stirling, C. (1991). Actions speak louder than words: proving bisimilarity for context free processes. *Proceedings 6th Annual Symposium on Logic in Computer Science*, IEEE Computer Science Press, 376-386.

[23] Jančar, P. (1994). Decidability questions for bisimilarity of Petri nets and some related problems. *Lecture Notes in Computer Science*, **775**, 581-594.

[24] Moller, F. (1996). Infinite results. *Lecture Notes in Computer Science*, **1119**, 195-216.

[25] Muller, D., and Schupp, P. (1985). The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, **37**, 51-75.

[26] Oyamuguchi, M., Honda, N., and Inagaki, Y. (1980). The equivalence problem for real-time strict deterministic languages. *Information and Control*, **45**, 90-115.