An Introduction to Decidability of DPDA Equivalence

Colin Stirling

Division of Informatics
University of Edinburgh
email: cps@dcs.ed.ac.uk

1 Introduction

The DPDA equivalence problem was posed in 1966 [4]: is there an effective procedure for deciding whether two configurations of a deterministic pushdown automaton (a DPDA) accept the same language? The problem is whether language equivalence is decidable for deterministic context-free languages. Despite intensive work throughout the late 1960s and 1970s, the problem remained unsolved until 1997 when Sénizergues announced a positive solution [11]. It seems that the notation of pushdown configurations, although simple, is not rich enough to sustain a proof. Deeper algebraic structure needs to be exposed. The full proof by Sénizergues, in journal form, appeared earlier this year [12]. It exposes structure within a DPDA by representing configurations as boolean rational series, and he develops an algebraic theory of their linear combinations. Equivalence between configurations is captured within a deduction system. The equations within the proof system have associated weights. Higher level strategies (transformations) are defined which guide proof. A novel feature is that these strategies depend upon differences between weights of their associated equations. Decidability is achieved by showing that two configurations are equivalent if, and only if, there is a finite proof of this fact.

I produced a different proof of decidability that is essentially a simplification of Sénizergues's proof [14]. It is based on a mixture of techniques developed in concurrency theory and language theory. The first step is to view the DPDA problem as a bisimulation equivalence problem for a process calculus whose expressions generate infinite state transition systems. The process calculus is built from determinising strict grammars: strict grammars were introduced by Harrison and Havel [5] because they are equivalent to DPDA. Tableaux proof systems have been used to show decidability of bisimulation equivalence between infinite state processes, see, for instance, [8, 2]. I use this method for the DPDA problem. However, the tableau proof system uses conditional proof rules that involve distances between premises. Essentially this is Sénizergues's use of weights, and the idea was developed from trying to understand his proof.

The proof of decidability is unsatisfactory. It is very complex because the proof of termination uses a mechanism for "decomposition" that in [14] is based on unifiers and auxiliary recursive nonterminals (from [3, 13]). Sénizuergues uses

a more intricate mechanism. This means that the syntax of the starting process calculus has to be extended in [14] with auxiliary symbols. It also introduces nondeterminism into tableaux with the consequence that the decision procedure (in both [12, 14]) is two semidecision procedures. The result is that there is no known upper bound on complexity.

In this paper I describe a simpler decision procedure that should lead to an elementary complexity upper bound. It is a deterministic procedure that avoids the decomposition mechanism for termination (the rule CUT in [14] and the transformation T_C in [12]). Instead, there is a new and much simpler analysis of termination. It also means that the syntax of the starting process calculus is not extended. The paper is entirely introductory, and contains no proofs. Section 2 introduces the DPDA problem as a bisimulation equivalence problem. Section 3 describes some features of the process calculus in more detail. Finally, Section 4 introduces the deterministic tableau proof decision procedure. The aim of the paper is to provide the reader with a clear indication of the decision procedure: more details, and full proofs, can be found at the author's home page.

2 DPDA and strict grammars

A deterministic pushdown automaton, a DPDA, consists of finite sets of states P, stack symbols S, alphabet A and basic transitions T. A basic transition is $pS \stackrel{a}{\longrightarrow} q\alpha$ where p, q are states in P, $a \in \mathsf{A} \cup \{\varepsilon\}$, S is a stack symbol in S and α is a sequence of stack symbols in S^*. Basic transitions are restricted.

if $pS \stackrel{a}{\longrightarrow} q\alpha \in \mathsf{T}$ and $pS \stackrel{a}{\longrightarrow} r\beta \in \mathsf{T}$ and $a \in \mathsf{A} \cup \{\varepsilon\}$, then $q = r$ and $\alpha = \beta$
if $pS \stackrel{\epsilon}{\longrightarrow} q\alpha \in \mathsf{T}$ and $pS \stackrel{a}{\longrightarrow} r\lambda \in \mathsf{T}$ then $a = \epsilon$

A configuration of a DPDA has the form $p\delta$ where $p \in \mathsf{P}$ is a state and $\delta \in \mathsf{S}^*$ is a sequence of stack symbols. The transitions of a configuration are determined by the following prefix rule, assuming that $\beta \in \mathsf{S}^*$: if $pS \stackrel{a}{\longrightarrow} q\alpha \in \mathsf{T}$ then $pS\beta \stackrel{a}{\longrightarrow} q\alpha\beta$.

The transition relation $\stackrel{a}{\longrightarrow}$, $a \in \mathsf{A} \cup \{\varepsilon\}$, between configurations is extended to words $\stackrel{w}{\longrightarrow}$, $w \in \mathsf{A}^*$. First, $p\alpha \stackrel{\epsilon}{\longrightarrow} p_n\alpha_n$, if $p_n = p$ and $\alpha_n = \alpha$ or there is a sequence of basic transitions $p\alpha \stackrel{\epsilon}{\longrightarrow} p_1\alpha_1 \stackrel{\epsilon}{\longrightarrow} \ldots \stackrel{\epsilon}{\longrightarrow} p_n\alpha_n$. If $w = av \in \mathsf{A}^+$, then $p\alpha \stackrel{w}{\longrightarrow} q\beta$ if $p\alpha \stackrel{\epsilon}{\longrightarrow} p'\alpha' \stackrel{a}{\longrightarrow} q'\beta' \stackrel{v}{\longrightarrow} q\beta$. A configuration $pS\alpha$ is either "stable" and has no ε-transitions or it is "unstable" and only has a single ε-transition. We assume that a final configuration $p\varepsilon$ with the empty stack is also stable. Clearly, if $p\alpha \stackrel{w}{\longrightarrow} q\beta$ and $p\alpha \stackrel{w}{\longrightarrow} r\delta$ and $q\beta$ and $r\delta$ are stable, then $q = r$ and $\beta = \delta$. The language accepted, or generated, by a configuration $p\delta$, written $\mathsf{L}(p\delta)$, is the set of words $\{w \in \mathsf{A}^* \ : \ \exists q \in \mathsf{P}.\, p\delta \stackrel{w}{\longrightarrow} q\epsilon\}$. That is, acceptance is by empty stack and not by final state. The motivation for providing a positive solution to the decision problem is to establish decidability of language equivalence between deterministic context-free languages. However, DPDAs which have empty stack acceptance can only recognise the subset of deterministic context-free languages that are prefix-free: a language L is prefix-free if $w \in L$, then no proper prefix of w is also in L. However, DPDAs whose

acceptance is by final state do recognise all deterministic context-free languages. A DPDA with acceptance by final state has an extra component $F \subseteq \mathsf{P}$ that is the subset of accepting states: in which case, $\mathsf{L}(p\alpha)$ is the set of words $\{w : p\alpha \xrightarrow{w} q\beta \text{ and } q \in F\}$. For any determinisitic context-free language L, there is a stable configuration of a DPDA with empty stack acceptance that accepts the language $\{w\$: w \in L\}$ where \$ is a new alphabet symbol, an end marker.

The DPDA problem is whether $\mathsf{L}(p\alpha) = \mathsf{L}(q\beta)$. Clearly, it is sufficient to restrict the problem to stable configurations. Moreover, one can assume that the DPDA is in normal form: if $pS \xrightarrow{a} q\alpha \in \mathsf{T}$, then $|\alpha| \leq 2$; if $pS \xrightarrow{\varepsilon} q\alpha \in \mathsf{T}$ then $\alpha = \varepsilon$; and there are no redundant transitions in T (a transition $pS \xrightarrow{a} q\alpha$ is redundant if $\mathsf{L}(q\alpha) = \emptyset$).

Example 1 Let $\mathsf{P} = \{p, r\}$, $\mathsf{S} = \{X, Y\}$ and $\mathsf{A} = \{a, b, c\}$. The basic transitions T are: $pX \xrightarrow{a} pX$, $pX \xrightarrow{b} p\epsilon$, $pX \xrightarrow{c} pX$, $rX \xrightarrow{\epsilon} p\epsilon$, $pY \xrightarrow{a} p\epsilon$, $pY \xrightarrow{b} r\epsilon$, $pY \xrightarrow{c} pYY$ and $rY \xrightarrow{\epsilon} r\epsilon$. This example of a DPDA is in normal form. □

$\mathsf{G}(p\alpha)$ is the possibly infinite state determinisitic transition graph generated by a stable configuration $p\alpha$ that abstracts from the basic ϵ-transitions. Transitions in $\mathsf{G}(p\alpha)$ are only labelled by elements of A and only relate stable configurations: $q\beta \xrightarrow{a} r\delta$ is a transition if both configurations are stable and $q\beta \xrightarrow{a} q'\beta' \xrightarrow{\varepsilon} r\delta$. The graph $\mathsf{G}(pYX)$ is pictured in Figure 1, where pYX is a configuration from Example 1. There is a path $p\alpha \xrightarrow{a_1} p_1\alpha_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} p_n\varepsilon$

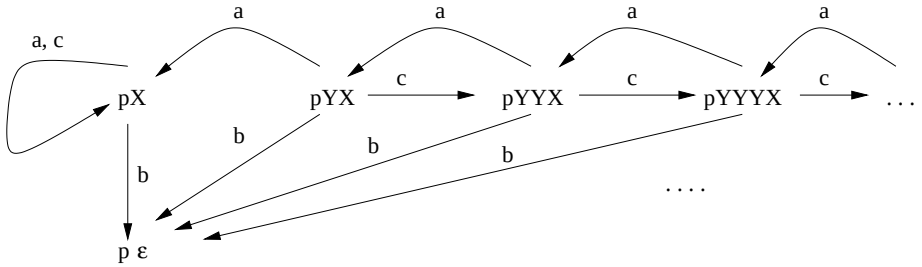

Fig. 1. The graph $\mathsf{G}(pYX)$

in $\mathsf{G}(p\alpha)$ if, and only if, $a_1 \ldots a_n \in \mathsf{L}(p\alpha)$. Because these graphs are deterministic and there are no redundant transitions, language equivalence coincides with bisimulation equivalence.

There is not an obvious relation between the lengths of stacks of equivalent configurations[1]: in the case of Example 1, $\mathsf{L}(pY^nX) = \mathsf{L}(pY^mX)$ for every m and n. Techniques for proving decidability of bisimulation equivalence, as developed

[1] A main attack on the decision problem in the 1970s examined differences between stack lengths and potentially equivalent configurations that eventually resulted in a proof of decidability for real-time DPDAs, that have no ε-transitions, [10].

in the 1990s [1], use decomposition and congruence that allows substitutivity of subexpressions in configurations. However, $\mathsf{L}(pY^n) = \mathsf{L}(pY^m)$ only if $n = m$. Moreover, the operation of stack extension is not a congruence. An important step is to provide a syntactic representation of stable DPDA configurations that dispenses with ε-transitions and that supports congruence, a process calculus that can directly generate transition graphs such as $\mathsf{G}(pYX)$. The key is *nondeterministic* pushdown automata with a single state and without ϵ-transitions, introduced by Harrison and Havel [6] as grammars, and further studied in [5, 7].

Because the state is redundant, a configuration of a pushdown automaton with a single state is a sequence of stack symbols. Ingredients of such an automaton without ε-transitions, an SDA, are finite sets of stack symbols S, alphabet A and basic transitions T. Each basic transition has the form $S \xrightarrow{a} \alpha$ where $a \in \mathsf{A}$, S is a stack symbol and α is a sequence of stack symbols. A configuration of an SDA is a sequence of stack symbols whose transitions are determined by the prefix rule, assuming $\beta \in \mathsf{S}^*$: if $S \xrightarrow{a} \alpha \in \mathsf{T}$, then $S\beta \xrightarrow{a} \alpha\beta$. The language $\mathsf{L}(\alpha)$ accepted, or generated, by a configuration α is the set $\{w \in \mathsf{A}^* : \alpha \xrightarrow{w} \epsilon\}$, so acceptance is again by empty stack. Unlike pushdown automata with multiple states, language (and bisimulation) equivalence is a congruence with respect to stacking: if $\mathsf{L}(\alpha) = \mathsf{L}(\beta)$, then $\mathsf{L}(\alpha\delta) = \mathsf{L}(\beta\delta)$. An SDA can be transformed into normal form: if $S \xrightarrow{a} \alpha \in \mathsf{T}$, then $|\alpha| \leq 2$ and $\mathsf{L}(\alpha) \neq \emptyset$.

Any context-free language that does not contain the empty word ε is generable by an SDA, and so the language equivalence problem is undecidable. However, if the SDA is deterministic, then the decision problem is decidable. A deterministic SDA, more commonly known as a "simple grammar", has restricted basic transitions: if $S \xrightarrow{a} \alpha \in \mathsf{T}$ and $S \xrightarrow{a} \beta \in \mathsf{T}$, then $\alpha = \beta$. Decidability of language equivalence between configurations of an SDA was proved by Korenjak and Hopcroft in 1966 [9]. However, the languages generable by deterministic SDA are strictly contained in the languages generable by DPDA: for instance, $\{a^n b^{n+1} : n > 0\} \cup \{a^n c : n > 0\}$ is not generable by a deterministic SDA.

Instead of assuming determinism, Harrison and Havel included an extra component, $\equiv$, an equivalence relation on the stack symbols S, in the definition of an SDA that partitions S into disjoint subsets.

Example 2 The following SDA has alphabet $\mathsf{A} = \{a, b\}$ and stack symbols $\mathsf{S} = \{A, C, X, Y\}$. The partition of S is $\{\{A\}, \{C\}, \{X\}, \{Y\}\}$. The basic transitions T are $X \xrightarrow{a} YX$, $X \xrightarrow{b} \epsilon$, $Y \xrightarrow{b} X$, $A \xrightarrow{a} C$, $A \xrightarrow{b} \epsilon$ and $C \xrightarrow{b} AA$. This SDA is deterministic. □

Example 3 The set $\mathsf{S} = \{X, Y, Z\}$, $\mathsf{A} = \{a, b, c\}$ and T contains $X \xrightarrow{a} X$, $X \xrightarrow{b} \epsilon$, $X \xrightarrow{c} X$, $Y \xrightarrow{a} \epsilon$, $Y \xrightarrow{c} YY$, $Z \xrightarrow{b} \epsilon$, $Z \xrightarrow{c} Z$ and $Z \xrightarrow{c} YZ$. The partition of S is $\{\{X\}, \{Y, Z\}\}$ which means that, for example, $Y \equiv Z$. The graphs of X and Z are illustrated in Figure 2. In particular, $\mathsf{G}(Z)$ is nondeterministic because there are two c-transitions from Z. □

The relation, $\equiv$, on S is extended to an equivalence relation between sequences of stack symbols, and the same relation, $\equiv$, is used for the extension:

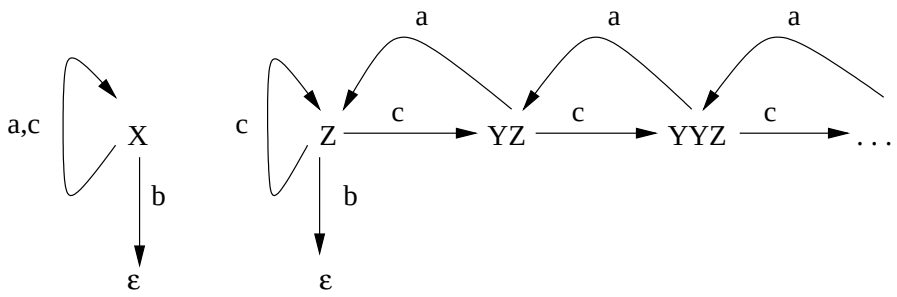

Fig. 2. The graph $\mathsf{G}(X)$ and $\mathsf{G}(Z)$

$\alpha \equiv \beta$ if, either $\alpha = \beta$, or $\alpha = \delta X \alpha'$ and $\beta = \delta Y \beta'$ and $X \equiv Y$ and $X \neq Y$. An instance of equivalent sequences, from Example 2, above, is $XXYY \equiv XXZ$ because $Y \equiv Z$. Some simple properties of $\equiv$ are: $\alpha\beta \equiv \alpha$ if, and only if, $\beta = \epsilon$; $\alpha \equiv \beta$ if, and only if, $\delta\alpha \equiv \delta\beta$; if $\alpha \equiv \beta$ and $\gamma \equiv \delta$, then $\alpha\gamma \equiv \beta\delta$; if $\alpha \equiv \beta$ and $\alpha \neq \beta$, then $\alpha\gamma \equiv \beta\delta$; if $\alpha\gamma \equiv \beta\delta$ and $|\alpha| = |\beta|$, then $\alpha \equiv \beta$.

Definition 1 The relation $\equiv$ on S is strict when the following two conditions hold. (1) If $X \equiv Y$ and $X \xrightarrow{a} \alpha$ and $Y \xrightarrow{a} \beta$, then $\alpha \equiv \beta$. (2) If $X \equiv Y$ and $X \xrightarrow{a} \alpha$ and $Y \xrightarrow{a} \alpha$, then $X = Y$.

An SDA with partition $\equiv$ is strict deterministic (or, just strict) if the relation $\equiv$ on S is strict[2]. Examples 1 and 2, above, are strict. In the case of Example 1, each partition is a singleton set and hence by (1) of Definition 1 this implies determinism. In this extreme case, it follows that $\alpha \equiv \beta$ if, and only if, $\alpha = \beta$ and, therefore, an SDA is then a simple grammar. If the partition involves larger sets, as is the case in Example 2, then constrained nondeterminism is allowed. There are transitions $Z \xrightarrow{c} Z$ and $Z \xrightarrow{c} YZ$. However, $Z \equiv YZ$ because $Y \equiv Z$.

Proposition 1 (1) *If* $\alpha \xrightarrow{w} \alpha'$ *and* $\beta \xrightarrow{w} \beta'$ *and* $\alpha \equiv \beta$ *then* $\alpha' \equiv \beta'$. (2) *If* $\alpha \xrightarrow{w} \alpha'$ *and* $\beta \xrightarrow{w} \alpha'$ *and* $\alpha \equiv \beta$ *then* $\alpha = \beta$. (3) *If* $\alpha \equiv \beta$ *and* $w \in \mathsf{L}(\alpha)$, *then for all words* v, *and* $a \in \mathsf{A}$, $wav \notin \mathsf{L}(\beta)$. (4) *If* $\alpha \equiv \beta$ *and* $\alpha \neq \beta$, *then* $\mathsf{L}(\alpha) \cap \mathsf{L}(\beta) = \emptyset$.

The definition of a configuration of an SDA is extended to sets of sequences of stack symbols, $\{\alpha_1, \ldots, \alpha_n\}$, written in sum form $\alpha_1 + \ldots + \alpha_n$. Two sum configurations are equal, written using $=$, if they are the same set. A degenerate case is the empty sum, written $\emptyset$. The language of a sum configuration is defined using union: $\mathsf{L}(\alpha_1 + \ldots + \alpha_n) = \bigcup \{\mathsf{L}(\alpha_i) : 1 \leq i \leq n\}$.

Definition 2 A sum configuration $\beta_1 + \ldots + \beta_n$ is *admissible*, if $\beta_i \equiv \beta_j$ for each pair of components, and $\beta_i \neq \beta_j$ when $i \neq j$.

[2] More generally, an SDA without a partition is strict deterministic if there exists a strict partition of its stack symbols. Harrison and Havel show that it is decidable whether an SDA is strict deterministic [6].

The empty sum, $\emptyset$, is therefore admissible. In [7] admissible configurations are called "associates". Some admissible configurations of Example 2, above, are XX, $ZZZ + ZZY$, $YX + Z$, $Z + YZ$ and $Z + YZ + YYZ$. An example of a configuration that is not admissible is $X + Y$ because $X \not\equiv Y$. A simple corollary of Proposition 1 is that admissibility is preserved by word transitions: if $\{\beta_1, \ldots, \beta_n\}$ is admissible, then $\{\beta' : \beta_i \xrightarrow{w} \beta',\ 1 \leq i \leq n\}$ is admissible.

A strict SDA can be determinised, by determinising the basic transitions T to T^{d}: for each stack symbol X and $a \in \mathsf{A}$, the transitions $X \xrightarrow{a} \alpha_1$, ..., $X \xrightarrow{a} \alpha_n$ in T are replaced by the single transition $X \xrightarrow{a} \alpha_1 + \ldots + \alpha_n$ in T^{d}. The sum configuration $\alpha_1 + \ldots + \alpha_n$ is admissible. Therefore, for each stack symbol X and $a \in \mathsf{A}$ there is a unique transition $X \xrightarrow{a} \sum \alpha_i \in \mathsf{T}^{\mathsf{d}}$. The prefix rule for generating transitions is also extended to admissible configurations: if $X_1\beta_1 + \ldots + X_m\beta_m$ is admissible and $X_i \xrightarrow{a} \sum \alpha_{ij} \in \mathsf{T}^{\mathsf{d}}$ for each i, then $X_1\beta_1 + \ldots + X_m\beta_m \xrightarrow{a} \sum \alpha_{1j}\beta_1 + \ldots + \sum \alpha_{mj}\beta_m$. The resulting configuration is admissible. Given a determinised strict SDA and an admissible configuration E, the graph $\mathsf{G}^{\mathsf{d}}(E)$ is the transition graph generated by E, except that redundant transitions, $E' \xrightarrow{a} \emptyset$, are omitted.

Example 4 In the case of Example 3, above, T^{d} contains: $X \xrightarrow{a} X$, $Y \xrightarrow{a} \epsilon$, $Z \xrightarrow{a} \emptyset$, $X \xrightarrow{b} \epsilon$, $Y \xrightarrow{b} \emptyset$, $Z \xrightarrow{b} \epsilon$, $X \xrightarrow{c} X$, $Y \xrightarrow{c} YY$ and $Z \xrightarrow{c} YZ + Z$. The graph $\mathsf{G}^{\mathsf{d}}(YX + Z)$ is pictured in Figure 3. □

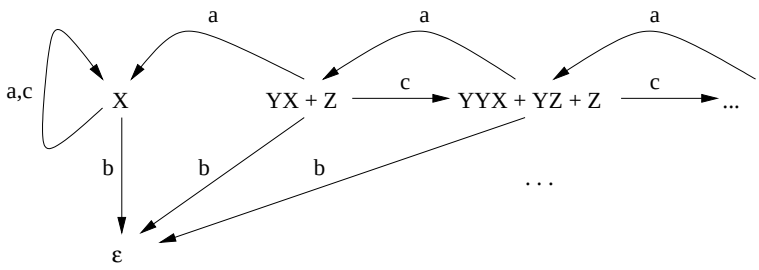

Fig. 3. The graph $\mathsf{G}^{\mathsf{d}}(YX + Z)$

Admissible configurations of strict SDAs generate exactly the same languages as configurations of DPDA with empty stack acceptance [6]. There is a straightforward transformation of a DPDA into a strict SDA, and DPDA configurations into admissible SDA configurations that preserve language equivalence. Assume a DPDA in normal form with sets P, S, A and T. An SDA is constructed, in stages. A) For $p, q \in \mathsf{P}$ and $X \in \mathsf{S}$, introduce an SDA stack symbol $[pXq]$. B) For transitions, the initial step is to define the following for $a \in \mathsf{A}$: if $pX \xrightarrow{a} q\epsilon \in \mathsf{T}$, then $[pXq] \xrightarrow{a} \epsilon$; if $pX \xrightarrow{a} qY \in \mathsf{T}$, then $[pXr] \xrightarrow{a} [qYr]$ for each $r \in \mathsf{P}$; if $pX \xrightarrow{a} qYZ \in \mathsf{T}$, then $[pXr] \xrightarrow{a} [qYp'][p'Zr]$ for each r and p' in P. C) $[pSq]$ is an ε-symbol, if $pX \xrightarrow{\epsilon} q\epsilon \in \mathsf{T}$. All ε-symbols are

erased from the right hand side of any transition given in B. D) Finally, the SDA is normalised. Clearly, the relation $[pSq] \equiv [pSr]$ is strict. Although the transformation does not preserve determinism, this is overcome by determinising the SDA. Any configuration $pX_1X_2 \ldots X_n$ of the DPDA is transformed into $\mathrm{sum}(p\alpha) = \sum [pX_1p_1][p_1X_2p_2] \ldots [p_{n-1}X_np_n]$ where the summation is over all $p_i \in \mathsf{P}$, after all ϵ-symbols are erased, and all components involving redundant stack symbols are removed. And $\mathsf{L}(p\alpha) = \mathsf{L}(\mathrm{sum}(p\alpha))$. An example is the conversion of the DPDA of Example 1. The resulting strict SDA is Example 4, above, when $X = [pXp]$, $Y = [pYp]$ and $Z = [pYr]$. The configuration $pYYX$ transforms to $[pYp][pYp][pXp] + [pYp][pYr] + [pYr]$. Harrison and Havel also prove the converse, that any strict SDA can be transformed into a DPDA [6]. The DPDA problem is equivalent to language equivalence between admissible configurations of a determinised strict SDA (which is the same as the bisimulation equivalence problem).

3 Heads and tails

Assume a fixed determinised strict SDA in normal form with ingredients S, A, $\mathsf{T_d}$ and $\equiv$. We assume a total ordering on A, and we say that word u is shorter than v if $|u| < |v|$ or $|u| = |v|$ and u is lexicographically smaller than v. Let $E, F, G, \ldots$ range over admissible configurations, and $E = F$ if they are the same set of sequences. A useful notation is "the configuration E after the word u", written $E \cdot u$, that is the unique admissible configuration F such that $E \xrightarrow{u} F$, which can be $\emptyset$. The language accepted by configuration E, $\mathsf{L}(E)$, is $\{u : (E \cdot u) = \epsilon\}$. Two configurations E and F are language equivalent, written $E \sim F$, if they accept the same language, $\mathsf{L}(E) = \mathsf{L}(F)$. Language equivalence can also be approximated. If $n \geq 0$, then E and F are n-equivalent, written $E \sim_n F$, provided that they accept the same words whose length is at most n: for all words w such that $|w| \leq n$, $(E \cdot w) = \varepsilon$ if, and only if, $(F \cdot w) = \varepsilon$.

Proposition 1 (1) $E \sim F$ *if, and only if, for all* $n \geq 0$, $E \sim_n F$. (2) *If* $E \not\sim F$, *then there is an* $n \geq 0$ *such that* $E \sim_n F$ *and* $E \not\sim_{n+1} F$. (3) $E \sim F$ *if, and only if, for all* $u \in \mathsf{A}^*$, $(E \cdot u) \sim (F \cdot u)$. (4) $E \sim_n F$ *if, and only if, for all* $u \in \mathsf{A}^*$ *where* $|u| \leq n$, $(E \cdot u) \sim_{n-|u|} (F \cdot u)$. (5) *If* $E \sim_n F$ and $0 \leq m < n$, *then* $E \sim_m F$. (6) *If* $E \sim_n F$ *and* $F \not\sim_n G$, *then* $E \not\sim_n G$.

Definition 1 For each stack symbol X, the word $w(X)$ is the shortest word in the set $\{u : (X \cdot u) = \epsilon\}$.

A feature of the decision procedure is repeating patterns within admissible configurations. An admissible configuration is written in sum form $\beta_1 + \ldots + \beta_n$ where each β_i is distinct. The operation $+$ can be extended: if E and F are admissible and $E \cup F$ is admissible and E, F are disjoint, $E \cap F = \emptyset$, then $E + F$ is the admissible configuration $E \cup F$. The operation $+$ is partial. Sequential composition, written as juxtaposition, is also used: if E and F are admissible, then EF is the configuration $\{\beta\gamma : \beta \in E \text{ and } \gamma \in F\}$, that is admissible. Some properties are: if $E + F$ is admissible and $u \in \mathsf{L}(E)$, then $uv \notin \mathsf{L}(F)$; if $E + F$ is

admissible, then $\mathsf{L}(E) \cap \mathsf{L}(F) = \emptyset$; $\mathsf{L}(EF) = \{uv : u \in \mathsf{L}(E) \text{ and } v \in \mathsf{L}(F)\}$. Also, the following identities hold: $E + \emptyset = E = \emptyset + E$, $E\emptyset = \emptyset = \emptyset E$, $E\varepsilon = E = \varepsilon E$, $(E + F)G = EG + FG$ and $G(E + F) = GE + GF$. Admissible configurations can have different "shapes", using + and sequential composition.

Definition 2 $E = E_1G_1 + \ldots + E_nG_n$ is in head/tail form, if the head $E_1 + \ldots + E_n$ is admissible and at least one $E_i \neq \emptyset$, and each tail $G_i \neq \emptyset$.

Example 1 $E = YYYX + YYZ + YZ + Z$ is an admissible configuration of Example 4 of the previous section. The partition of the stack symbols is $\{\{X\}, \{Y, Z\}\}$. E has head/tail form, $YG_1 + ZG_2$, where $G_1 = YYX + YZ + Z$ and $G_2 = \varepsilon$. Also, E has head/tail form, $YYH_1 + YZH_2 + ZH_3$, where $H_1 = YX + Z$ and $H_2 = H_3 = \varepsilon$. $(E \cdot c)$ is $(YY \cdot c)H_1 + (YZ \cdot c)H_2 + (Z \cdot c)H_3 = YYYH_1 + YYZH_2 + (YZ + Z)H_3$. E cannot be presented as $YYG'_1 + YYG'_2 + YG'_3 + ZG'_4$: this is not a valid head/tail form because the head $YY + YY + Y + Z$ is not admissible ($YY \not\equiv Y$) and it is not disjoint ($YY + YY$ is not a proper sum). □

In the following, if a configuration E is presented as $E_1G_1 + \ldots + E_nG_n$, then assume that it fulfills the conditions of Definition 2 of a head/tail form. The following result lists some properties of head/tail forms. Language equivalence and its approximants are congruences with respect to + and sum. Consequently, head/tail forms allow substitutivity of equivalent subexpressions into tails (because admissibility is preserved).

Proposition 2 *Assume* $E = E_1G_1 + \ldots + E_nG_n$. (1) *If* $(E_i \cdot u) = \epsilon$, *then for all* $j \neq i$, $(E_j \cdot u) = \emptyset$ *and* $(E \cdot u) = G_i$. (2) *If* $(E_i \cdot u) \neq \emptyset$, *then* $(E \cdot u) = (E_1 \cdot u)G_1 + \ldots + (E_n \cdot u)G_n$. (3) *If* $H_i \neq \emptyset$, $1 \leq i \leq n$, *then* $E_1H_1 + \ldots + E_nH_n$ *is a head/tail form.* (4) *If each* $H_i \neq \emptyset$ *and each* $E_i \neq \varepsilon$ *and for each* j *such that* $E_j \neq \emptyset$ $H_j \sim_m G_j$, *then* $E \sim_{m+1} E_1H_1 + \ldots + E_nH_n$. (5) *If* $H_i \sim G_i$, $1 \leq i \leq n$, *then* $E \sim E_1H_1 + \ldots + E_nH_n$.

Two configurations may have the same heads and differ in their tails, or may have the same tails and differ in their heads. If E has the head/tail form $E_1G_1 + \ldots + E_nG_n$ and F has a similar head/tail form $F_1G_1 + \ldots + F_nG_n$ involving the same tails[3], then the imbalance between E and F, relative to this presentation, is $\max\{|E_i|, |F_i| : 1 \leq i \leq n\}$. If the imbalance is 0, then they are the same configurations.

Definition 3 If $E = E_1G_1 + \ldots + E_nG_n$ and $F = F_1H_1 + \ldots + F_mH_m$, then F in its head/tail form is a tail extension of E in its head/tail form provided that each $H_i = K_1^iG_1 + \ldots + K_n^iG_n$, $1 \leq i \leq m$. When F is a tail extension of E, the associated extension e is the m-tuple $(K_1^1 + \ldots + K_n^1, \ldots, K_1^m + \ldots + K_n^m)$ without the G_is, and F is said to extend E by e.

Extensions are matrices. If E'' extends E' by e and E' extends E by f, then E'' extends E by ef ("matrix multiplication"). A special instance of an

[3] Any pair of configurations E and F have a head/tail form involving the same tails: $E = EG$ and $F = FG$ when $G = \varepsilon$.

extension occurs when the tails are the same. If $E = E_1G_1 + \ldots + E_nG_n$ and $F = F_1G_1 + \ldots + F_nG_n$, then F extends E by $e = (\varepsilon + \emptyset + \ldots + \emptyset, \ldots, \emptyset + \emptyset + \ldots + \varepsilon)$. The extension e is abbreviated to the identity (ε).

Example 2 The following uses Example 4 of the previous section. $E = YG_1 + ZG_2$ where $G_1 = X$ and $G_2 = \varepsilon$. $E' = YG'_1 + ZG'_2$ where $G'_1 = YX + Z$ and $G'_2 = \varepsilon$. $E'' = YG''_1 + ZG''_2$ where $G''_1 = YYX + YZ + Z$ and $G''_2 = \varepsilon$. E' extends E by $e = (Y + Z, \emptyset + \varepsilon)$ and E'' extends E' by $f = e = (Y + Z, \emptyset + \varepsilon)$. Therefore, E'' extends E by $ef = (YY + (YZ + Z), \emptyset + \varepsilon)$. □

4 The decision procedure

The procedure for deciding $E \sim F$ is to build a goal directed proof tree, a tableau, with initial goal $E \doteq F$, "is $E \sim F$?", using proof rules that reduce goals to subgoals. There are just three rules, presented in Figure 4. UNF, for

UNF

$$\frac{E \doteq F}{(E \cdot a_1) \doteq (F \cdot a_1) \quad \ldots \quad (E \cdot a_k) \doteq (F \cdot a_k)} \; \mathsf{A} = \{a_1, \ldots, a_k\}$$

BAL(R)

$$\frac{\begin{array}{l} F \doteq X_1H_1 + \ldots + X_kH_k \\ \quad \vdots \\ F' \doteq E_1H_1 + \ldots + E_kH_k \end{array}}{F' \doteq E_1(F \cdot w(X_1)) + \ldots + E_k(F \cdot w(X_k))} \; \mathrm{C}$$

BAL(L)

$$\frac{\begin{array}{r} X_1H_1 + \ldots + X_kH_k \doteq F \\ \vdots \\ E_1H_1 + \ldots + E_kH_k \doteq F' \end{array}}{E_1(F \cdot w(X_1)) + \ldots + E_k(F \cdot w(X_k)) \doteq F'} \; \mathrm{C}$$

where C is the condition

1. Each $E_i \neq \varepsilon$ and at least one $H_i \neq \varepsilon$.
2. There are precisely $\max\{\, |w(X_i)| \,:\, E_i \neq \emptyset \text{ for } 1 \leq i \leq k\}$ applications of UNF between the top goal and the bottom goal, and no application of any other rule.
3. If u is the word associated with the sequence of UNFs, then $E_i = (X_i \cdot u)$ for each $i : 1 \leq i \leq k$.

Fig. 4. The tableau proof rules

"unfold", reduces a goal $E \doteq F$ to subgoals $(E \cdot a) \doteq (F \cdot a)$ for each a. It is

complete and sound. If the goal is true, then so are all the subgoals. Soundness is the converse. A finer version, (2) of Fact 1, uses approximants: if the goal fails at level $m+1$, then at least one subgoal fails at level m.

Fact 1 (1) *If* $E \sim F$ *and* $a \in \mathsf{A}$*, then* $(E \cdot a) \sim (F \cdot a)$. (2) *If* $E \not\sim_{m+1} F$*, then for some* $a \in \mathsf{A}$*,* $(E \cdot a) \not\sim_m (F \cdot a)$.

Example 1 Below is an application of UNF where X, Y and Z are from Example 4 of Section 2.

$$\frac{YX + Z \doteq YYX + YZ + Z}{X \doteq YX + Z \quad \epsilon \doteq \epsilon \quad YYX + YZ + Z \doteq YYYX + YYZ + YZ + Z}$$

The three subgoals are the result after a, b and c. □

If $E' \doteq F'$ is a subgoal that is a result of m consecutive applications of UNF (and no other rule) to $E \doteq F$, then there is an associated word u such that $|u| = m$ and $E' = (E \cdot u)$ and $F' = (F \cdot u)$. The other two rules in Figure 4 are conditional that involve two premises: the second premise goal reduces to the subgoal beneath it provided that the first premise is above it (on the path back to the root goal). They are BAL rules, for "balance", involving substitution of subexpressions, that allow a goal to be reduced to a balanced subgoal where the imbalance between the configurations is bounded.

Example 2 An application of BAL(L) uses stack elements of Example 2 of Section 2.

$$\dfrac{\dfrac{XXXXXX \doteq AAAAAA}{YXXXXXX \doteq CAAAAA \quad \cdots}\,\text{UNF}}{YXAAAAA \doteq CAAAAA}\,\text{BAL(L)}$$

The second goal is the result of UNF when the label is a (and the other subgoal for b is omitted). $w(X) = b$, so $m = 1$. Therefore, BAL(L) applies to the second goal: $X_1 = X$, $H_1 = XXXXX$, $E_1 = YX$ and $F = AAAAAA$. So H_1 is replaced with $(F \cdot b) = AAAAA$. The imbalance between configurations of the last goal is 2. □

An application of BAL is said to use F, if F is the configuration in the initial goal of the rule, see Figure 4. The BAL rules are sound and complete. Completeness is straightforward. Soundness is more intricate. First, "global" soundness of the proof system is explained. If there is a successful tableau whose root is false, then there is a branch of the tableau within which each subgoal is false. The idea is refined using approximants. If the root is false then there is an offending branch (of false goals) in the tableau within which the approximant indices decrease whenever rule UNF has been applied. Soundness of an application of BAL is that if the two premise goals belong to an offending branch, then the subgoal preserves the level of falsity of the second premise goal.

Proposition 1 (1) *If* $X_1H_1+\ldots+X_kH_k \sim F$ *and* $E_1H_1+\ldots+E_kH_k \sim F'$, *then* $E_1(F\cdot w(X_1))+\ldots+E_k(F\cdot w(X_k)) \sim F'$. (2) *If* $X_1H_1+\ldots+X_kH_k \sim_{n+m} F$ *and* $E_1H_1+\ldots+E_kH_k \not\sim_{n+1} F'$ *and each* $E_i \neq \varepsilon$ *and* $m \geq \max\{|w(X_i)| : E_i \neq \emptyset\}$, *then* $E_1(F\cdot w(X_1))+\ldots+E_k(F\cdot w(X_k)) \not\sim_{n+1} F'$.

In Example 2, above, BAL(L) is applied after UNF. However, the other two rules UNF and BAL(R) also apply. It is intended that there be a unique tableau associated with any initial goal. So restrictions will be placed on which rule is to be applied when. First, the initial premise of a BAL is the one that is "closest" to the goal and, therefore, the one that involves the least number of applications of UNF. To resolve which rule should be applied, the following priority order is assumed: (1) if BAL(L) is permitted, then apply BAL(L), (2) if BAL(R) is permitted, then apply BAL(R), (3) otherwise, apply UNF. However, whether an application of BAL is permitted involves more than fulfillment of the side condition. It also depends on the previous application of a BAL.

Initially, either BAL is permitted provided that its side condition is true. If an application of BAL uses F, then the resulting goal contains the configuration $E_1(F\cdot w(X_1))+\ldots+E_k(F\cdot w(X_k))$. E_i is a "top" of the application of BAL and $(F\cdot w(X_i))$ is a "bottom". Assume an application of BAL(L). A subsequent application of BAL(L) is permitted provided the side condition of the rule is fulfilled. However, BAL(R) is not permitted until a bottom of the previous application of BAL(L) is exposed and the side condition of the rule is true. Between the application of BAL(L) and the goal $G_1 \doteq H_1$, below,

$$\begin{array}{rl} F & \\ \vdots & \text{BAL(L)} \\ E_1(F\cdot w(X_1))+\ldots+E_k(F\cdot w(X_k)) \doteq H & \\ \vdots \quad \vdots & \text{UNFs} \\ (F\cdot w(X_i)) = G_1 \doteq H_1 & \\ \vdots \quad \vdots & \\ G_k \doteq H_k & \end{array}$$

there are no other applications of BAL(L), and G_1 is a bottom, $(F\cdot w(X_i))$, of the previous application of BAL(L). BAL(R) is now permitted provided it uses configuration G_i, $i \geq 1$, and the side condition holds. BAL(R) is not permitted using a configuration from a goal above $G_1 \doteq H_1$, even when the side condition is true. The strategy is to apply a BAL rule whenever it is permitted, and if both BAL rules are permitted, then priority lies with BAL(L). If BAL(R) is applied, then the strategy is to repeatedly apply BAL(R), and to use UNF otherwise. BAL(L) is only permitted once a bottom of the previous application of BAL(R) becomes the right hand configuration of a goal and the side condition holds. The consequence is that when building a tableau proof tree, there is just one choice of which rule to apply next to any subgoal.

Example 3 An initial part of the tableau, continuing on from Example 2 above, is in Figure 5. At goal $(*)$, BAL(L) is applied. Either of the premises (1) and

$$\dfrac{\dfrac{\dfrac{\dfrac{XX^5 \doteq AA^5}{YXX^5 \doteq CA^5}\,\text{BAL(L)} \qquad X^5 \doteq A^5}{(1)\ YXA^5 \doteq CA^5}\,\text{UNF}}{\emptyset \doteq \emptyset \qquad (2)\ XXA^5 \doteq AA^6}\,\text{UNF} \qquad XA^5 \doteq A^6}{\dfrac{(*)\ YXXA^5 \doteq CA^6}{YXA^6 \doteq CA^6}\,\text{BAL(L)}}\,\text{UNF}$$

Fig. 5. Part of a tableau

(2) could be the initial premise for the application: however, by the discussion above it is the lower premise (2). □

Example 4 Below is the initial part of the tableau for Example 1, above.

$$\dfrac{\dfrac{(*)\ YX + Z \doteq YYX + YZ + Z}{(\mathbf{1}) \quad \epsilon \doteq \epsilon \qquad YYX + YZ + Z \doteq YYYX + YYZ + YZ + Z}\,\text{UNF}}{YYYX + YYZ + YZ + Z \doteq YYYX + YYZ + YZ + Z}\,\text{BAL(L)}$$

where (**1**) is the subtableau

$$\dfrac{\dfrac{\dfrac{(**)\ X \doteq YX + Z}{X \doteq X \quad \epsilon \doteq \epsilon \qquad X \doteq YYX + YZ + Z}\,\text{UNF}}{X \doteq YYX + YZ + Z}\,\text{BAL(R)} \qquad \dfrac{X \doteq YYYX + YYZ + YZ + Z}{X \doteq YYX + YZ + Z}\,\text{BAL(R)}}{X \doteq YX + Z \quad \epsilon \doteq \epsilon}\,\text{UNF}$$

The premise $(*)$ is the initial premise for the application of BAL(L), and $(**)$ is the initial premise for the first BAL(R). The leaf goals are either identities or repeats. In fact, it will turn out that this partial tableau is the completed successful tableau that establishes that $\mathsf{L}(pYX) = \mathsf{L}(pYYX)$ of Example 1 of Section 2. □

For the tableau construction to be a decision procedure, a notion of final goal is needed so that a tableau can be terminated. The tableau proof rules are locally complete, if a goal is true then so are subgoals. Consequently, if an obviously

false subgoal is reached, then the root goal is also false. So the criterion for being an unsuccessful final goal is that it is obviously false. This occurs when the goal has the form $\emptyset \doteq E$ or $E \doteq \emptyset$ and $E \neq \emptyset$. The tableau proof rules are also locally sound, if all the subgoals are true then so is the goal. Therefore, if an obviously true subgoal, $E \doteq E$, is reached then it should count as a successful final leaf. However, the tableau proof rules are sound in a finer version. In the case of UNF, if the goal is false at level $m+1$, $E \not\sim_{m+1} F$, then at least one subgoal fails at level m, $(E \cdot a) \not\sim_m (F \cdot a)$. And applications of BAL preserve the falsity index. Consequently, if a subgoal $E \doteq F$ is repeated in a branch, and there is at least one application of UNF between them, then the second occurrence of $E \doteq F$ can also count as a successful final goal. If the root of the tableau is false, then there is an offending path of false goals in the tableau within which the approximant indices decrease whenever UNF is applied. Consider the branch with $E \doteq F$ occurring twice: if this were an offending branch, then at the first occurrence, there is a least $n \geq 0$ such that $E \sim_n F$ and $E \not\sim_{n+1} F$. Therefore, at the second occurrence $E \not\sim_{(n+1)-k} F$ where k is the number of applications of UNF between the two; this is a contradiction when $k \geq 1$.

A repeat is an instance of a more general situation where goals may be growing in size, formally captured below by the "extension theorem". Roughly speaking, in a branch if there are goals where the rates of change of tails are repeating, then there is a successful final goal. A repeat is an instance when the rate of change is zero. In a long enough branch with multiple applications of BAL, there must be goals within the branch that have the same heads. The idea is to discern patterns of relations between their tails. Definition 3 of the previous section is lifted to goals. Assume $E = E_1H_1 + \ldots + E_nH_n$, $F = F_1H_1 + \ldots + F_nH_n$, $E' = E'_1G_1 + \ldots + E'_mG_m$ and $F' = F'_1G_1 + \ldots + F'_mG_m$ and goal h is $E \doteq F$ and goal g is $E' \doteq F'$. Goal h extends g by extension e, if E extends E' by e. A goal $E \doteq F$ is true at level m if $E \sim_m F$.

Theorem 1 [The extension theorem] *Assume there are two families of goals $g(i)$, $h(i)$, $1 \leq i \leq 2^n$, and each goal $g(i)$ has the form $E_1G_1^i + \ldots + E_nG_n^i \doteq F_1G_1^i + \ldots + F_nG_n^i$ and each goal $h(i)$ has the form $E_1H_1^i + \ldots + E_nH_n^i \doteq F_1H_1^i + \ldots + F_nH_n^i$. Assume extensions $e_1, \ldots, e_n$ such that for each e_j and $i \geq 0$, $g(2^j i + 2^{j-1} + 1)$ extends $g(2^j i + 2^{j-1})$ by e_j and $h(2^j i + 2^{j-1} + 1)$ extends $h(2^j i + 2^{j-1})$ by e_j. If each goal $g(i)$ is true at level m, $i : 1 \leq i \leq 2^n$, and each goal $h(j)$, $j : 1 \leq j < 2^n$, is true at level m, then $h(2^n)$ is true at level m.*

A simple instance is explained. Consider the proof tree of Example 3. There is a branch where the goals are expanding as follows: $YXA^5 \doteq CA^5$, ..., $YXA^6 \doteq CA^6$, ..., $YXA^7 \doteq CA^7$, ... And between these goals there is at least one application of UNF. To instantiate the extension theorem $n = 1$. The families of goals are as follows. $g(1) : YXG^1 \doteq CG^1$ where $G^1 = A^5$, $g(2) = h(1) : YXG^2 \doteq CG^2$ where $G^2 = A^6$, $h(2) : YXH^2 \doteq CH^2$ where $H^2 = A^7$. The extension is (A): $g(2)$ extends $g(1)$ by (A) and $h(2)$ extends $h(1)$ by (A). The theorem provides the following result: for any m, if $YXA^5 \sim_m CA^5$ and $YXA^6 \sim_m$

CA^6, then $YXA^7 \sim_m CA^7$. This justifies that the subgoal $YXA^7 \doteq CA^7$ is a successful final goal. The argument is the same as for a repeating goal, above.

Definition 1 Assume a branch of goals $d(0), \ldots, d(l)$. The goal $d(l)$ obeys the extension theorem if there are goals $g(i)$, $h(i)$, $1 \leq i \leq 2^n$ and extensions $e_1, \ldots, e(n)$ as described in Theorem 1, and the goals belong to $\{d(0), \ldots d(l)\}$, and $h(2^n)$ is $d(l)$ and there is at least one application of UNF between goal $h(2^n - 1)$ and $d(l)$.

The second occurrence of a repeating goal in a branch obeys the extension theorem if there is at least one application of UNF between the two occurrences. Assume it has the form $E_1 G_1^i + \ldots + E_n G_n^i \doteq F_1 G_1^i + \ldots + F_n G_n^i$. Except for $h(2^n)$, the goals $g(i)$ and $h(i)$ are the first occurrence of the repeating goal, and each extension is the identity, (ϵ).

Definition 2 Assume a branch of goals $g(0), \ldots, g(n)$ where $g(0)$ is the root goal. The goal $g(n)$ is a final goal in the following circumstances.

1. If $g(n)$ is an identity $E \doteq E$, then $g(n)$ is a successful final goal
2. If $g(n)$ obeys the extension theorem, then $g(n)$ is a successful final goal
3. If $g(n)$ has the form $E \doteq \emptyset$ or $\emptyset \doteq E$ and $E \neq \emptyset$, then $g(n)$ is an unsuccessful final goal.

Lemma 1 *In any infinite branch of goals $g(0), \ldots, g(n), \ldots$ where $g(0)$ is the root goal, there is an n such that $g(n)$ is a final goal.*

The deterministic procedure that decides whether $E \sim F$ is straightforward, and is defined iteratively.

1. Stage 0: start with the root goal $g(0)$, $E \doteq F$, that becomes a frontier node of the branch $g(0)$.
2. Stage $n + 1$: if a current frontier node $g(n)$ of branch $g(0), \ldots, g(n)$ is an unsuccessful final goal, then halt and return "unsuccessful tableau"; if each frontier node $g(n)$ of branch $g(0), \ldots, g(n)$ is a successful final goal, then return "successful tableau"; otherwise, for each frontier node $g(n)$ of branch $g(0), \ldots, g(n)$ that is not a final goal, apply the next rule to it, and the subgoals that result are the new frontier nodes of the extended branches.

Theorem 2 (1) *If $E \not\sim F$, then the decision procedure terminates with "unsuccessful tableau".* (2) *If $E \sim F$, then the decision procedure terminates with "successful tableau".*

Theorem 2 establishes decidability of language equivalence between DPDA configurations. Part 1 of Theorem 2 is straightforward. The other half, part 2, is more difficult and uses Lemma 1, above. However, a more refined analysis of the lemma should produce an elementary complexity upper bound. Currently, the proof of Lemma 1 abstracts from "oscillation" whereby goals can increase and decrease their sizes. A more involved proof would establish that a boundedly finite branch of goals contains a final goal.

The proof of Theorem 1, the extension theorem, follows from the following much simpler result.

Lemma 2 *If* $E = E_1G_1 + \ldots + E_nG_n$, $F = F_1G_1 + \ldots + F_nG_n$, $E' = E_1H_1 + \ldots + E_nH_n$ and $F' = F_1H_1 + \ldots + F_nH_n$ *and* $E \sim_m F$ *and* $E' \not\sim_m F'$, *then there is a word* u, $|u| \leq m$, *and an* i *such that either* $(E' \cdot u) = H_i$ *and* $(F' \cdot u) = (F_1 \cdot u)H_1 + \ldots + (F_n \cdot u)H_n$ *and* $(E' \cdot u) \not\sim_{m-|u|} (F' \cdot u)$, *or* $(F' \cdot u) = H_i$ *and* $(E' \cdot u) = (E_1 \cdot u)H_1 + \ldots + (E_n \cdot u)H_n$ *and* $(E' \cdot u) \not\sim_{m-|u|} (F' \cdot u)$.

References

1. Burkart, O., Caucal, D., Moller, F., and Steffen, B. (2001). Verification on infinite structures. In *Handbook of Process Algebra*, edited Bergstra, J., Ponse, A., and Smolka, S. 545-623, *North Holland*.
2. Christensen, S., Hirshfeld, Y. and Moller, F. (1993). Bisimulation equivalence is decidable for all basic parallel processes. *Lecture Notes in Computer Science*, **715**, 143-157.
3. Christensen, S., Hüttel, H., and Stirling, C. (1995). Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, **121**, 143-148.
4. Ginsberg, S., and Greibach, S. (1966). Deterministic context-free languages. *Information and Control*, 620-648.
5. Harrison, M. (1978). *Introduction to Formal Language Theory*, Addison-Wesley.
6. Harrison, M., and Havel, I. (1973). Strict deterministic grammars. *Journal of Computer and System Sciences*, **7**, 237-277.
7. Harrison, M., Havel, I., and Yehudai, A. (1979). On equivalence of grammars through transformation trees. *Theoretical Computer Science*, **9**, 173-205.
8. 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.
9. Korenjak, A and Hopcroft, J. (1966). Simple deterministic languages. *Procs. 7th Annual IEEE Symposium on Switching and Automata Theory*, 36-46.
10. Oyamaguchi, M., Honda, N., and Inagaki, Y. (1980). The equivalence problem for real-time strict deterministic languages. *Information and Control*, **45**, 90-115.
11. Sénizergues, G. (1997). The equivalence problem for deterministic pushdown automata is decidable. *Lecture Notes in Computer Science*, **1256**, 671-681.
12. Sénizergues, G. (2001). L(A) = L(B)? decidability results from complete formal systems. *Theoretical Computer Science*, **251**, 1-166.
13. Stirling, C. (1998). Decidability of bisimulation equivalence for normed pushdown processes. *Theoretical Computer Science*, **195**, 113-131.
14. Stirling, C. (2001). Decidability of DPDA equivalence. *Theoretical Computer Science*, **255**, 1-31.