DIDIER CAUCAL

A fast algorithm to decide on the equivalence of stateless DPDA

Informatique théorique et applications, tome 27, nº 1 (1993), p. 23-48
<http://www.numdam.org/item?id=ITA_1993__27_1_23_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 27, n° 1, 1993, p. 23 à 48)

A FAST ALGORITHM TO DECIDE ON THE EQUIVALENCE OF STATELESS DPDA (*)

by Didier CAUCAL ([1])

Communicated by J. E. PIN

Abstract. – *We give an algorithm to decide the equivalence of stateless dpda with acceptance on stack letters. This algorithm is polynomial in time and space in the valuation and the length of description of the compared automata, and exponential in the length of description, instead of the double exponential complexity of Oyamaguchi and Honda's algorithm.*

Résumé. – *On présente un algorithme pour décider de l'équivalence des automates à pile déterministes sans état (ou un seul état) et avec acceptation sur des lettres de pile. La complexité en temps et en espace de cet algorithme est polynomiale selon la valuation et la longueur des automates comparés, et exponentielle selon la longueur de description, au lieu de la complexité double exponentielle de l'algorithme de Oyamaguchi et Honda.*

INTRODUCTION

This paper is devoted to the equivalence of stateless dpda accepting by specific letters occuring on the top of stack. The problem is to decide whether two such automata recognize the same language. Oyamaguchi and Honda [9] solve it by an algorithm having double exponential complexity in time and space. Their algorithm uses Valiant's method [11, 12], *i.e.* given two automata, it builds a third one simulating their product, which recognizes the empty language if and only if they are equivalent.

To solve this problem efficiently, we begin (in Section 1) by reducing it linearly to the same one for two stateless dpda, having exactly one ε-transition of the form $E \xrightarrow{\varepsilon} \varepsilon$, and whose acceptance test is the presence of E on the top of the stack.

(*) Submitted November 1990, final version June 1992.
([1]) I.R.I.S.A., Campus de Beaulieu, 35042 Rennes, France, E-mail: caucal@irisa.fr

We then give (in Section 2) an efficient algorithm to decide the equivalence of two even more restricted automata: they are deterministic, stateless, without ε-transition and their acceptance test is the empty stack; they correspond to simple grammars [7]. This algorithm (already given in [2]) uses a branching method [7, 8, 6, 10], *i.e.* it builds a finite tree the nodes of which are labelled by two non-terminal words, and the root by the two axioms. We show that the complexity of this algorithm is $O(n^3 . v)$ where n is the size of the compared grammars, and v is the greatest valuation of the non-terminals (the valuation of a non-terminal is the shortest length of the generated words). This valuation is bounded above by an exponential function in the size of the grammars.

Finally (in Section 3) we solve the initial problem by an algorithm using the former one, and building also a finite decision tree. Its complexity is $O(n^8 . v^2)$ where n is the size of the compared automata, and v is the greatest finite valuation of the stack letters.

1. A REDUCTION OF STATELESS DPDA EQUIVALENCE

In this section, we recall the notion of stateless dpda and the associated equivalence problem; then we show that this problem can be restricted to a subset of the stateless dpda.

If a pda has only one state, then this state can be omitted; we say it is a stateless pda. With every stateless automaton is associated a subset of stack letters, called accepting letters, so that a word is accepted by the automaton, if after reading it, the latter on top of stack is of accepting. Two stateless pda are equivalent if they accept the same language. In this section, we translate the equivalence decision for the class C of stateless dpda into the equivalence decision for the class $C_0 \subset C$ of stateless dpda with only one ε-transition $E \xrightarrow{\varepsilon} \varepsilon$ where E is the only accepting letter. To express formally this result, we recall the following definitions.

The class C of *stateless dpda* on the alphabet Σ is the set of quadruples (X, Δ, A_0, X_0) where

(*a*) X is the stack alphabet, disjoint of Σ

(*b*) Δ is the transition function of $X \times (\Sigma \cup \{\varepsilon\}) \rightarrow X^*$ such that

$$(A, a) \in \mathrm{Dom}(\Delta) \wedge a \in \Sigma \quad \Rightarrow \quad (A, \varepsilon) \notin \mathrm{Dom}(\Delta)$$

(*c*) $A_0 \in X$ is the bottom stack letter

(*d*) $X_0 \subseteq X$ is a subset of stack letters, called accepting letters.

To every automaton $M=(X, \Delta, A_0, X_0)$ of C, we associate the context-free grammar

$$G_M=\{(A, a\alpha) \mid (A, a, \alpha)\in\Delta\}$$

of all transitions of M, with axiom A_0, the set X of non-terminals and the set Σ of terminals. The language $L(M)$ accepted by M is defined as follows:

$$L(M)=\{u\in\Sigma^* \mid \exists\alpha, A_0 \xrightarrow[G_M]{*} u\alpha \wedge \alpha(1)\in X_0\}.$$

Two automata M and N of C are called equivalent if $L(M)=L(N)$. The equivalence problem in a class $D\subseteqq C$ is the decidability of the equivalence of two automata in D. To solve this problem in C, we can restrict to the subset C_0 of the automata $(X, \Delta, A_0, \{E\})$ with initial stack word A_0, with only one accepting letter E, and one ε-transition $(E, \varepsilon, \varepsilon)$.

PROPOSITION 1.1: *We can transform in an effective and linear way, every automaton in C into an equivalent automaton in* C_0.

Proof: Let $M=(X, \Delta, \alpha_0, X_0)$ be a dpda on Σ and E a symbol not in $\Sigma\cup X$. We note $u[v/A]$ the word constructed from the word u by replacing each letter A by the word v. The construction of an automaton N in C_0 equivalent to M is carried out in the four following steps:

(i) Let N_1 be the set of stack letters A such that ε is accepted by the automaton (X, Δ, A, X_0), *i.e.*

$$N_1=\{A\in X \mid \exists\alpha, A \xrightarrow[G_M]{*} \alpha \wedge \alpha(1)\in X_0\}.$$

Then $X_0\subseteqq N_1\subseteqq X_0\cup\{A\in X \mid (A, \varepsilon)\in\mathrm{Dom}(\Delta)\}$ and N_1 is linearly constructible in the number $\#\Delta$ of transitions. Let

$$\Delta_1=\{(A, a, \alpha[EB/B]_{B\in N_1} \mid (A, a, \alpha)\in\Delta\}$$

be the set of transitions obtained from each transition (A, a, α) of Δ, by writing in α the letter E before every letter in N_1. In the same way, we put down $\alpha_1=\alpha_0[EB/B]_{B\in N_1}$. The automaton

$$M_1=(X, \Delta_1\cup\{(E, \varepsilon, \varepsilon)\}, \alpha_1, \{E\})$$

is equivalent to M.

(ii) Let N_2 be the set of stack letters A such that the automaton (X, Δ, A) empties its stack, *i.e.*

$$N_2=\{A\in X \mid A \overset{*}{\underset{G_M}{\rightarrow}} \varepsilon\}.$$

So $N_2 \subseteq \{A\in X \mid (A, \varepsilon)\in \mathrm{Dom}(\Delta)\}$ is linearly constructible in the size (length of description) of Δ. Let

$$\Delta_2=\{(A, a, \alpha[\varepsilon/B]_{B\in N_2}) \mid (A, a, \alpha)\in\Delta_1 \wedge A\notin N_2\}$$

be the set of transitions obtained from each transition (A, a, α) of Δ_1 for A not in N_2, by erasing in α all letters from N_2. Also, we put $\alpha_2=\alpha_1[\varepsilon/B]_{B\in N_2}$. The automaton $M_2=(X, \Delta_2\cup\{(E, \varepsilon, \varepsilon)\}, \alpha_2, \{E\})$ is equivalent to M_1.

(iii) Let N_3 be the set of stack letters in $X-N_2$ for which only sequences of ε-moves can be performed, *i.e.*

$$N_3=(X-N_2)-\{A\in X \mid \exists\, a\in\Sigma, \exists\,\alpha, A \overset{*}{\underset{G_M}{\rightarrow}} a\alpha\}.$$

So $N_3 \subseteq \{A\in X \mid (A, \varepsilon)\in\mathrm{Dom}(\Delta)\}-N_2$ is constructible from Δ_2 in $O(\#\Delta_2)$. For each word α, we write $[\alpha]$ the greatest prefix of α in $(X\cup\{E\}-N_3)^*$, and we put down

$$\Delta_3=\{(A, a, [\alpha]) \mid (A, a, \alpha)\in\Delta_2 \wedge A\notin N_3\} \qquad \text{(here a can be equal to } \varepsilon)$$

and $\alpha_3=[\alpha_2]$. The automaton $M_3=(X, \Delta_3\cup\{(E, \varepsilon, \varepsilon)\}, \alpha_3, \{E\})$ is equivalent to M_2.

(iv) Let Δ_4 be the set of transitions of Δ_3 in Greibach normal form, *i.e.*

$$\Delta_4=\{(A, a, \alpha) \mid A\in N-(N_2\cup N_3),$$

$$a\in\Sigma, \alpha\in N^*, \exists\,\beta\in N^*, A \overset{*}{\underset{G_{M_3}}{\rightarrow}}{}_{\mathrm{le}}\, \beta \overset{*}{\underset{G_{M_3}}{\rightarrow}}{}_{\mathrm{le}}\, a\alpha\},$$

where $\overset{*}{\underset{G_{M_3}}{\rightarrow}}{}_{\mathrm{le}}$ is a step of left rewriting according to the grammar G_{M_3}.

Note that Δ_4 is constructible from Δ_3 in $O(\#\Delta_3)$ but subject to a suitable representation (any right hand side α is an address sequence of memorized factors). Furthermore, the automaton $N=(X, \Delta_4\cup\{(E, \varepsilon, \varepsilon)\}, \alpha_3, \{E\})\in C_0$ and is equivalent to M_3, therefore to M. ■

To decide equivalence in C_0, we begin by solving it in the subset S of all automata $(X, \Delta, A_0, \{E\})$ in C_0 such that $A_0 \neq E$ and for every transition (A, a, α) of Δ, the axiom A_0 does not occur in α, and E cannot appear in α except in its last position, and only if $A = A_0$, *i.e.*

$$(A, a, \alpha) \in \Delta \wedge i \in \{1, \ldots, |\alpha|\}$$
$$\Rightarrow \quad \alpha(i) \neq A_0 \wedge (\alpha(i) = E \quad \Rightarrow \quad A = A_0 \wedge i = |\alpha|).$$

To every automaton $M = (X, \Delta, A_0, \{E\})$ in S, we associate in a bi-univoque way, the real-time stateless dpda $f(M) = (X, f(\Delta), A_0)$ accepting $L(M)$ by empty stack, with

$$f(\Delta) = \{(A, a, \alpha) \in \Delta \mid A \neq E \wedge A \neq A_0\} \cup \{(A_0, a, \alpha) \mid (A_0, a, \alpha E) \in \Delta\}.$$

The grammars associated with these automata are the simple grammars (they are redefined in the next section). In other term, the equivalence problem in S is nothing else than the equivalence problem for simple grammars. We solve it efficiently in the next section.

2. THE EQUIVALENCE OF SIMPLE GRAMMARS

In this section, we recall the notion of a simple grammar and the associated equivalence problem. Then we solve this problem efficiently.

A simple grammar is a grammar in Greibach normal form and LL (1). Korenjak and Hopcroft [7], Harrison [5] (among others), have given algorithms to decide the equivalence of simple grammars. Their complexities are at least $O(n^v)$ where n is the global size of the compared grammars, and v is the greatest valuation of the non-terminals. Here, we decide the equivalence of simple grammars by an algorithm (given in [2]) of complexity $O(n^3 v)$.

We consider here a *context-free grammar* as a finite relation $G \subseteq X \times X^*$ where X is an alphabet. The set $N_G = \{A \mid \exists \alpha, A\, G\, \alpha\}$ of left members of G is the alphabet of *non-terminals* of G; they will be denoted by upper-case letters. The set $T_G = \{\alpha(i) \in X - N_G \mid \exists A, A\, G\, \alpha \wedge 1 \leq i \leq |\alpha|\}$ of letters of $X - N_G$ appearing in G is the alphabet of *terminals* of G; they will be denoted by lower-case letters. A rewriting step according to G is denoted by $\underset{G}{\rightarrow}$ or $\rightarrow$. For instance, every rule $(A, \alpha) \in G$ can be written $A \rightarrow \alpha$, which will be our notation henceforth. The language $L(G, \alpha)$ of terminal words generated by G from α is defined by

$$L(G, \alpha)=\left\{u \in T_G^* \mid \alpha \underset{G}{\overset{*}{\rightarrow}} u\right\}.$$

The *valuation* $v_G(\alpha)$ of a word α according to G is the shortest length of the words in $L(G, \alpha)$, *i.e.*

$$v_G(\alpha)=\min(\{\infty\} \cup \{|u| \mid u \in L(G, \alpha)\}).$$

We say that G has a *finite valuation* if every non-terminal A has a finite valuation, *i.e.* $L(G, A)$ is non-empty.

The *equivalence problem* in a class C of context-free grammars is to decide the equality $L(G, A)=L(H, B)$ for all grammars G and H in C and all non-terminals A and B in G and H respectively. Given a context-free grammar G of size n (length of description), we can construct in $O(n)$ the set $\{A \in N_G \mid L(G, A)=\varnothing\}$ of non-terminals with infinite valation. Then, the equivalence problem for every class is linearly reducible to the equivalence problem for the subclass of grammars of finite valuation.

To every grammar G, we associate the equivalence $\equiv_G$ on N_G^* such that $\alpha \equiv_G \beta$ *if* $L(G, \alpha)=L(G, \beta)$. A context-free grammar G is called *simple* if

(i) G is in Greibach normal form: all rules have the form

$$A \rightarrow a\alpha \text{ where } a \in T \text{ and } \alpha \in N^*$$

(ii) G is $LL(1)$: $A \rightarrow a\alpha \wedge A \rightarrow a\beta \quad \Rightarrow \quad \alpha=\beta$.

The equivalence problem for the simple grammars of finite valuation reduces to deciding the equivalence of any two non-terminal words under the equivalence $\equiv_G$ where G is an arbitrary simple grammar of finite valuation. Indeed, given two simple grammars G and H of finite valuation, and two non-terminals A of G and B of H, we suppose by renaming that the set N_G of non-terminals of G is disjoint from N_H; the grammar $K=G \cup H$ is then simple, has a finite valuation, and $L(G, A)=L(H, B)$ if and only if $A \equiv_K B$.

From now on, G is a simple grammar of finite valuation, and all assertions and notations will be relative to G unless stated otherwise. To decide if $\alpha \equiv \beta$, we define a branching algorithm, that is to say we come down to decide (recursively) if a finite number of equivalences $\gamma_i \equiv \delta_i$ are all true. The latter ones are deduced from $\alpha \equiv \beta$ by two transformations T_A and T_B defined below. The operation T_A, called the *left parallel derivation* and introduced by Harrison [5], is a mapping of $N^* \times N^*$ into its power set, and defined by:

$$T_A(\alpha, \beta) = \{(\varepsilon, \varepsilon)\} \quad \text{if} \quad \alpha = \beta = \varepsilon$$
$$T_A(\alpha, \beta) = \varnothing \quad \text{if} \quad \neg (\forall a \in T, (\exists \gamma, \alpha \to a\gamma) \Leftrightarrow (\exists \delta, a\delta))$$
$$T_A(\alpha, \beta) = \{(\gamma, \delta) \mid \exists a \in T, \alpha \to a\gamma \wedge \beta \to a\delta\} \quad \text{otherwise.}$$

This transformation is applied if α or β is reduced to one letter; else we apply the transformation T_B below. To every non-terminal A, we associate a word $\mathrm{Val}(A)$ in $L(G, A)$ of minimal length, *i. e.* $\mathrm{Val}(A) \in L(G, A)$ and $|\mathrm{Val}(A)| = v(A)$. The T_B transformation, called the *cutting* transformation, is a mapping of $N^+ \times N^+$ into the powerset of $N^* \times N^*$ and defined by

$$T_B(A\alpha, B\beta) = \{(\delta, \gamma) \mid (\gamma, \delta) \in T_B(B\beta, A\alpha)\} \quad \text{if} \quad v(A) < v(B)$$

$$T_B(A\alpha, B\beta) = \{(A, B\gamma), (\gamma\alpha, \beta)\}$$
$$\text{if} \quad (v(B) \leqq v(A)) \wedge (A \overset{*}{\to} \mathrm{Val}(B)\gamma) \wedge (\gamma \in N^*)$$

$$T_B(A\alpha, B\beta) = \varnothing \quad \text{otherwise.}$$

The set of the so-obtained equivalences is organized as a tree with root (α, β), where every node labelled by (γ, δ) has its successors labelled by the equivalences obtained from one of the two transformations above. The tree is expanded recursively in preorder (it is the lexicographic order on the nodes). The base cases on (γ, δ) are the following:

1) $\gamma = \delta$: the equivalence is true
2) $T_A(\gamma, \delta) = \varnothing$ or $T_B(\gamma, \delta) = \varnothing$: the equivalence is false
3) $v(\gamma) \neq v(\delta)$: the equivalence is false.

The algorithm is formally described below. Considering that all halting cases must succeed for the equivalence to be true, we stop the execution as soon as we meet a failure. Before developping a pair, we reduce it according to a canonical (each word has a unique irreducible form) relation R computed during the building of the tree.

procedure Decide (α, β) $\{R$ is a global variable initially empty$\}$

(*a*) Two words having different valuations cannot be equivalent.

if $v(\alpha) \neq v(\beta)$ **then** Halt(failure) **endif**

(*b*) We compute normal forms of α and β according to R, then we remove the greatest common prefix.

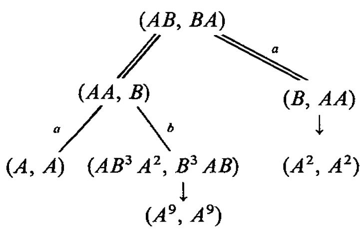
if $\alpha \neq \beta$ **then**
  $\alpha \leftarrow$ the irreducible word reduced from α according to R
  $\beta \leftarrow$ the irreducible word reduced from β according to R
  if $\alpha \neq \beta$ **then** $(\lambda\gamma, \lambda\delta) \leftarrow (\alpha, \beta)$ with $|\lambda|$ max.; $(\alpha, \beta) \leftarrow (\gamma, \beta)$ **endif**
endif

(*c*) If α or β is a non-terminal then we add (α, β) or (β, α) to R and we apply T_A, else we simply apply T_B. If the application fails then the execution stops.

if $\alpha \neq \beta$ **then**
  if $\min(|\alpha|, |\beta|) > 1$ **then** $Q \leftarrow T_B(\alpha, \beta)$ **else**
    $Q \leftarrow T_A(\alpha, \beta)$
    if $|\alpha| > 1$ **then** $(\alpha, \beta) \leftarrow (\beta, \alpha)$ **endif**
    $R \leftarrow \{(A, \gamma \downarrow \{(\alpha, \beta)\}) \mid AR\gamma\} \cup \{(\alpha, \beta)\}$
  endif
  if $Q = \varnothing$ **then** Halt(failure) **else**
    for every $(\gamma, \delta) \in Q$ **do** Decide (γ, δ) **endfor**
  endif
endif
endprocedure

Figures A and B describe the execution trees in which the nodes are labelled by the calling parameters. Furthermore, for clarity, if the reduction step modifies the pair (α, β) then the reduced pair is added to the tree. The operations T_A, T_B and the reduction are represented respectively by one line, two lines and an arrow.

Let the following simple grammar: $G = \{(A, a), (A, bABBBA), (B, aA), (B, bBBBAB)\}$.
The algorithm applied to (AB, BA) builds the following tree:



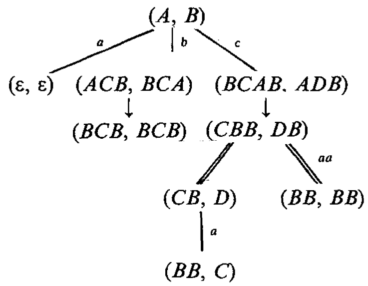
Therefore $AB \equiv BA$ and $R = \{(B, AA)\}$.

Figure A. — An equivalence case.

Let thè following simple grammar:

$$G=\{(A, a), (A, bACB), (A, cBCAB), (B, a), (B, bBCA), (B, cADB), (C, aB), (D, aC)\}.$$

The algorithm applied to (A, B) builds the following tree:



Then A is not equivalent to B and $R=\{(A, B), (D, CD)\}$.

Figure B. — A non-equivalence case.

Let us show that this algorithm decide the equivalence $\equiv$.

PROPOSITION 2.1: *The algorithm* Decide (α, β) *is well defined, always stops, and returns failure if and if* α *is not equivalent to* β.

To prove Proposition 2.1, we need some intermediate results. We begin to establish some basic properties of $\equiv$ in relation to transformations. First, the mapping T_A is *valid* [5] in the following way:

$$\alpha\equiv\beta \quad\Leftrightarrow\quad \emptyset\neq T_A(\alpha, \beta)\subset \equiv.$$

To iterate the mapping T_A, we extend it to every subset Q of $N^*\times N^*$ as follows:

$$T_A(Q)=\emptyset \quad \text{if there exists } (\alpha, \beta)\in Q,\ T_A(\alpha, \beta)=\emptyset$$
$$T_A(Q)=\{(\lambda, \mu)\mid \exists(\alpha, \beta)\in Q,\ (\lambda, \mu)\in T_A(\alpha, \beta)\} \quad \text{in the other case.}$$

The study of the equivalence of a couple by iterating T_A is expressed by the lemma below.

LEMMA 2.2: $\alpha\equiv\beta \quad\Leftrightarrow\quad \forall n,\ T_A^n(\alpha, \beta)\neq\emptyset$.

Proof: $\Rightarrow$: By induction and the validity of T_A.

$\Leftarrow$: If α is not equivalent to β then there exists a word u of minimal length belonging to only one the languages $L(G, \alpha)$ and $L(G, \beta)$. By symmetry of α and β, we can suppose $u\in L(G, \alpha)-L(G, \beta)$. Let v be the greatest prefix of u such that there exists $\delta\in N^*$ with $\beta \overset{*}{\rightarrow}_g v\delta$. By definition of u, there exists

$(\gamma, \delta) \in T_A^{|v|}(\alpha, \beta)$ with $\alpha \overset{*}{\rightarrow}_g v\gamma$. By definition of v, $T_A(\gamma, \delta) = \varnothing$ hence $T_A^{|v|+1}(\alpha, \beta) = \varnothing$. ■

Lemma 2.2 gives a semi-decision procedure for the non equivalence.

We say that a binary relation R on N^* is *closed by* T_A if $\varnothing \neq T_A(R) \subseteq R$.

COROLLARY 2.3: *Every relation closed by T_A transformation is included in* $\equiv$.

Proof: If $\varnothing \neq T_A(R) \subseteq R$ then by induction on n, $\varnothing \neq T_A^n(R) \subseteq R$ and by Lemma 2.2, $R \subseteq \equiv$. ■

A more general condition than the closure by T_A was given by Courcelle [3]. A set R of couples of non-terminal words is *self-proving* if the set $T_A(R)$ of the couples obtained by T_A transformation is non empty, and is included in the smallest conguence contaning R, *i.e.*

$$\varnothing \neq T_A(R) \subseteq \underset{R}{\overset{*}{\leftrightarrow}}.$$

Before extending Corollary 2.3 to self-proving relations, we establish that every element of T_A applied to the derivation according to R is obtained by derivation according to $R \cup T_A(R)$.

LEMMA 2.4: *Given a relation R such that $T_A(R) \neq \varnothing$, we have*

$$\varnothing \neq T_A\left(\underset{R}{\overset{*}{\rightarrow}}\right) \subseteq \underset{S}{\overset{*}{\rightarrow}} \qquad \textit{where} \quad S = R \cup T_A(R).$$

Proof: For $T_A(R) \neq \varnothing$ and $S = R \cup T_A(R)$, we verify by induction on n that

$$\varnothing \neq T_A\left(\underset{R}{\overset{n}{\rightarrow}}\right) \subseteq \underset{S}{\overset{*}{\rightarrow}}. \quad ■$$

It follows that the self-provability of a relation R corresponds to the closure by T_A of the smallest congruence containing R.

PROPOSITION 2.5: *A relation R is self-proving if and only if $\underset{R}{\overset{*}{\leftrightarrow}}$ is closed by transformation T_A.*

Proof: $\Rightarrow$: Let R be a self-proving relation, i.e. $\varnothing \neq T_A(R) \subseteq \underset{R}{\overset{*}{\leftrightarrow}}$.

As $T_A(R^{-1})=(T_A(R))^{-1}$ and by Lemma 2.4, we have

$$\varnothing \neq T_A\left(\overset{*}{\underset{R}{\leftrightarrow}}\right) \subseteq \overset{*}{\underset{S}{\leftrightarrow}} \qquad \text{where} \quad S=R \cup T_A(R).$$

So $\overset{*}{\underset{S}{\leftrightarrow}}=\overset{*}{\underset{R}{\leftrightarrow}}$ therefore $\overset{*}{\underset{R}{\leftrightarrow}}$ is closed by T_A.

$\Leftarrow$: Immediate. ■

From Corollary 2.3 and Proposition 2.5 follows the forthcoming corollary.

COROLLARY 2.6: *Every self-proving relation is included in* $\equiv$.

As transformation T_A, the mapping T_B is *valid* [5], that is to say for every non empty non-terminal words α and β, we have

$$\alpha \equiv \beta \quad \Leftrightarrow \quad \varnothing \neq T_B(\alpha, \beta) \subset \equiv .$$

The decision algorithm, constructs a *fundamental* relation R, that is to say a binary relation on N^* verifying the following conditions:

(*a*) $\mathrm{Dom}(R) \subseteq N$ and $\mathrm{Im}(R) \subseteq (N\text{-}\mathrm{Dom}(R))^*$

(*b*) R is functional: if $AR\alpha$ and $AR\beta$ then $\alpha=\beta$.

LEMMA 2.7: *Given a fundamental relation* R, *we have*

$$\# R \leqq \# N \quad \textit{and} \quad \underset{R}{\rightarrow} \textit{ is canonical.}$$

Proof: Let R be a fundamental relation. From (*b*) and (*a*), $\# R \leqq \# \mathrm{Dom}(R) \leqq \# N$. By (*a*), every derivation according to R from $\alpha \in N^*$ is of length at most $|\alpha|$, so that $\underset{R}{\rightarrow}$ is noetherian (of finite termination). As $\mathrm{Dom}(R) \subseteq N$ and R is functional, the relation $\underset{R}{\rightarrow}$ is confluent. Finally $\underset{R}{\rightarrow}$ is canonical. ■

Now, we are able to establish Proposition 2.1.

Proof of Proposition 2.1: Let us consider the sequence $(\alpha_i, \beta_i, R_i)_{i \geqq 0}$ of successive calling parameters of Decide with

$$(\alpha_0, \beta_0)=(\alpha, \beta) \quad \text{and} \quad R_0=\varnothing,$$

and such that if the step (*b*) (of reduction) of the algorithm applied to (α_i, β_i) gives a couple (λ, μ) distinct of (α_i, β_i), then $(\alpha_{i+1}, \beta_{i+1}, R_{i+1})=(\lambda, \mu, R_i)$.

(i) One verifies by induction on i that the relation R_i is fundamental. By Lemma 2.7 and for every i, $\# R_i \leqq \# N$. So, the total number of nodes whose labels have been developped by T_A is finite, and it follows that the sequence $(\alpha_i, \beta_i, R_i)_{i \geqq 0}$ is finite. Hence, the algorithm is well defined and always stops.

(ii) If $\alpha \equiv \beta$ then by validity of T_A and T_B, we show by induction on $i \geqq 0$ that $\alpha_i \equiv \beta_i$. So the algorithm does not return a failure.

(iii) If the algorithm does not return a failure, we must prove that $\alpha \equiv \beta$. Let R be the set of (α_i, β_i) which has been expended by T_A. By induction on $i \geqq 0$, we have $R_i \subseteq \overset{*}{\underset{R}{\leftrightarrow}}$. Let p be the last index of the sequence (α_i, β_i, R_i). As the algorithm does not return a failure, $\alpha_p = \beta_p$, and by inverse induction on $i \leqq p$, we have $\alpha_i \overset{*}{\underset{R}{\leftrightarrow}} \beta_i$. In particular $\alpha \overset{*}{\underset{R}{\leftrightarrow}} \beta$ and $\varnothing \neq T_A(R) \subseteq \overset{*}{\underset{R}{\leftrightarrow}}$, *i.e.* R is self-proving. By Corollary 2.6, $R \subseteq\ \equiv$ then $\overset{*}{\underset{R}{\leftrightarrow}} \subseteq\ \equiv$, hence $\alpha \equiv \beta$. ∎

Let us compute the complexity of the algorithm applied to a pair of non-terminals. Let n be the size of G, let $v = \max\{v(A) \mid A \in N\}$ be the valuation of G, and $\|G\| = \max\{|\gamma| \mid \exists A, A \to \gamma\}$ the maximal length of the right hand sides of G. Let us not that the maximal valuation of the calling parameters is in $O(\|G\|.v)$.

The cost of transformation T_A is $O(\# T(\|G\| + v))$ and the number of pairs developped by T_A is at most $\# N$. Hence the cost of all T_A transformations is $O(\# N.\# T.\|G\| + \# N.\# T.v)$. Similarly, the cost of transformation T_B is $O(\|G\|.v)$ and the number of pairs developped by T_B is at most $\# N$, hence the cost of all T_B transformations is $O(\# N.\|G\|.v)$. The cost of a reduction is $O(\|G\|.v)$ and the total number of calls is $O(\# N.\# T)$. Hence the total cost of the reductions is in $O(\# N.\# T.\|G\|.v)$. The construction of relation R is $O(\# N^2.v)$. Finally, the complexity of the algorithm when applied to non-terminals, is $O(\# N.\# T.\|G\|.v)$ or $O(n^3.v)$. Since the valuation v is $O(\|G\|^{\# N})$, hence in $O(n^n)$, we get the result.

THEOREM 2.8: *The equivalence problem of simple grammars is decidable by an algorithm of complexity $O(n^3 v)$ or $O(n^n)$ where n is the size of the compared grammars, and v is the greatest finite valuation of the non-terminals.*

This theorem is basic for building an efficient algorithm to decide on the equivalence of stateless dpda, described in the next section.

3. THE EQUIVALENCE OF STATELESS DPDA

In this section, we solve the equivalence problem of stateless dpda, by means of a branching algorithm using the former one. The complexity of the algorithm is polynomial in the size of the automata and in the greatest finite valuation of the stack letters.

In section 1, we have reduced the equivalence problem in the class of the stateless dpda to the one in the class C_0 of the stateless dpda, with only one letter E of acceptance, and the only ε-transition $E \overset{\varepsilon}{\to} \varepsilon$. To every automaton M in C_0, we associate a grammar G_M satisfying:

(*a*) $\exists E, E \to \varepsilon$

(*b*) $G-\{(E, \varepsilon)\}$ is a simple grammar, *i.e.*

$$G-\{(E, \varepsilon)\} \quad \subset \quad (N-\{E\}) \times T.N^*$$
$$(A \to a\alpha \wedge A \to a\beta \wedge a \in T) \quad \Rightarrow \quad (\alpha=\beta).$$

Such a grammar G will be called a *simple extended grammar*. We define

$$T(G, \alpha)=\{u \in T_G^* \mid \exists \beta, \alpha \overset{*}{\underset{G}{\to}} u E \beta\}$$

the language of the terminal words u such that uE is a left factor of a word generated by G from α. Hence, the language $L(M)$ accepted by an automaton $M=(X, \Delta, A_0, \{E\})$ in C_0 is equal to $T(G_M, A_0)$. The equivalence problem for C_0, then for C, is directly reducible to the decidability of the equivalence $\sim_G$ on N^* for every simple extended grammar G, with $\alpha \sim_G \beta$ iff $T(G, \alpha)=T(G, \beta)$. We must be careful to distinguish the equivalence $\sim_G$ from the equivalence $\equiv_G$ of the generated languages, defined in the above section. Furthermore, the previous algorithm can be used for deciding $\alpha \equiv_G \beta$ for every simple extended grammar G, because $\alpha \equiv_G \beta$ iff $\alpha[\varepsilon/E] \equiv_{G_0} \beta[\varepsilon/E]$ where $\alpha[\varepsilon/E]$ is the result of substituting ε for E in α, and $G_0=\{(A, \alpha[\varepsilon/E]) \mid AG\alpha \wedge A \neq E\}$ is a simple grammar.

In the sequel, G is a simple extended grammar, and E is the non-terminal of G such that $E \to \varepsilon$. Before defining a decision procedure for $\alpha \sim_G \beta$, we need an operation of simplification on non-terminal words. We partition N:

$N_\infty=\{A \in N \mid L(G, A)=\varnothing\}$ the set of non-terminals of infinite valuation,

$N_f=N-N_\infty$ the set of non-terminals of finite valuation, and define

$$N_{\varnothing}=\{A\in N \mid T(G, A)=\varnothing\}.$$

We simplify every non-terminal word α in the non-terminal word $[\alpha]$ in three steps: take the greatest prefix of α belonging to $N_f^*.(N_\infty \cup \{\varepsilon\})$, then suppress the greatest suffix in $N_{\varnothing}^*$, and finally replace the maximal factors of $E^2 E^*$ by E. Then $\alpha \sim [\alpha]$ and we denote by $[N^*]=\{[\alpha] \mid \alpha \in N^*\}$ the set of simplified non-terminal words.

To decide whether $\alpha \sim \beta$, we define a branching algorithm as in Section 2, which develops a tree, with a root labelled by (α, β), by means of three transformations T_A, T_B and T_C. The operation T_A of *left parallel derivation* is the mapping of $N^* \times N^*$ in the power set of $N^* \times N^*$ defined by

$$T_A(\alpha, \beta)=T_A([\alpha], [\beta])$$

and for every α and β in $[N^*]$ by

$$\begin{aligned}
&T_A(\alpha, \beta)=\{(\varepsilon, \varepsilon)\} && \text{if } \alpha=\beta=\varepsilon\\
&T_A(\alpha, \beta)=T_A(\gamma, \delta) && \text{if } \alpha=\mathrm{E}\gamma \text{ and } \beta=E\delta\\
&T_A(\alpha, \beta)=\varnothing && \text{if } \neg\,((\alpha(1)=E \Leftrightarrow \beta(1)=E)\\
& && \wedge\ \forall a\in T, (\exists \gamma, \alpha \to a\gamma \wedge [\gamma]\neq\varepsilon) \Leftrightarrow (\exists \delta, \beta \to a\delta \wedge [\delta]\neq\varepsilon))\\
&T_A(\alpha, \beta)=\{(\gamma, \delta) \mid \exists a\in T, \alpha \to a\gamma \wedge \beta \to a\delta\} && \text{otherwise.}
\end{aligned}$$

Let us define T_B. To every non-terminal A in N_f, we associate a word $\mathrm{Val}(A)$ in $L(G, A)$ of minimal length. To every pair (A, B) of non-terminals in $N_f - \{E\}$, we associate the following set:

$$\mathrm{Dif}(A, B)=\{(\gamma, \varepsilon) \mid \gamma\in N_f^* \wedge A \underset{G}{\overset{*}{\to}}_{le} \mathrm{Val}(B)\gamma \wedge A\equiv B\gamma\}$$

$$\cup\{(\varepsilon, \gamma) \mid \gamma\in N_f^* \wedge B \underset{G}{\overset{*}{\to}}_{le} \mathrm{Val}(A)\gamma \wedge B\equiv A\gamma\},$$

where $\underset{G}{\to}_{le}$ is the *leftmost rewriting step* according to G, i.e.

$$uA\beta \underset{G}{\to}_{le} u\alpha\beta \quad \text{for every } u\in T_G^*, (A\to\alpha)\in G \text{ and } \beta\in(T_G \cup N_G)^*.$$

Given a non-terminal word α, we write $\langle\alpha\rangle$ for the greatest suffix of α whose first letter is not E, and set $E_\alpha=E$ if the first letter of α is E, else $E_\alpha=\varepsilon$. Then $\alpha \sim E_\alpha\langle\alpha\rangle$ and for every (γ, ε), $(\delta, \varepsilon)\in\mathrm{Dif}(A, B)$, we have

$\langle \gamma \rangle = \langle \delta \rangle$. The *cutting* operation T_B is defined if $\mathrm{Dif}(A, B) \neq \varnothing$ by

$$T_B(A\alpha, B\beta) = \{(1, (AE_\alpha, BE_\beta < \gamma E_\alpha^>)),$$
$$(2, (\langle \gamma\alpha \rangle, \langle \beta \rangle))\} \text{ if there exists } (\gamma, \varepsilon) \in \mathrm{Dif}(A, B)$$
$$T_B(A\alpha, B\beta) = T_B(B\beta, A\alpha) \text{ otherwise.}$$

The operation T_C is another cutting operation, complementary to T_B. It is defined directly in the algorithm and depends on a relation S computed during the building of the tree. We apply T_B, T_C, T_A in this order, except for the first pair obtained by T_B which is developed by T_A. The tree is again expanded in preorder (by lexicographic order on the nodes). The base cases of the recursion on $\gamma \sim \delta$ are the following:

1) $\gamma = \delta$: the equivalence is true
2) $T_A(\gamma, \delta) = \varnothing$: the equivalence is false
3) $[\gamma](1) \neq [\delta](1) \wedge ([\gamma] = \varepsilon \vee [\delta] = \varepsilon \vee \gamma(1) = E \vee \delta(1) = E)$: the equivalence is false.

The algorithm is formally described below. Considering that all halting cases must succeed for the equivalence to be true, we stop the execution as soon as we meet a failing case. Before developing a pair, we reduce it according to another relation R computed during the building of the tree.

procedure Decide (α, β) $\{R$ and S are global variables initialized to the empty set$\}$

(*a*) We compute an irreducible pair of (α, β) according to R then we suppress the greatest possible left common factor.

if $\alpha \neq \beta$ **then**
  $\alpha \leftarrow$ an irreducible word reduced from α according to R
  $\beta \leftarrow$ an irreducible word reduced from β according to R
  $(\alpha, \beta) \leftarrow ([\alpha], [\beta])$
  if $\alpha \neq \beta \wedge \alpha(1) = \beta(1)$ **then**
    $(\lambda\gamma, \lambda\delta) \leftarrow (\alpha, \beta)$ with $|\lambda|$ max. such that $\gamma(1) \neq E$ and $\delta(1) \neq E$
    $(\alpha, \beta) \leftarrow (\gamma, \delta)$
  endif
endif

(*b*) We test if (α, β) is trivially non equivalent

if $\alpha(1) \neq \beta(1) \wedge \{\alpha(1), \beta(1)\} \cap \{\varepsilon, E\} \neq \varnothing$ **then** Halt (failure) **endif**

(*c*) Transformation of the current node.

if $\alpha \neq \beta$ **then**
  $(A\rho, B\eta) \leftarrow (\alpha, \beta)$ with letters A and B
  if $A, B \in N_f$ and $\mathrm{Dif}(A, B) \neq \varnothing$ **then**

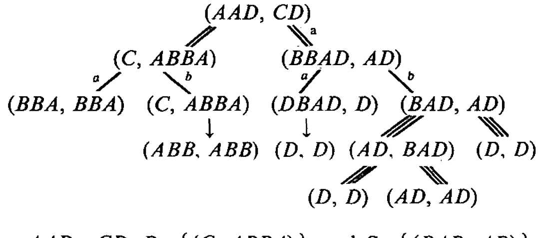
{we develop by T_B: the first pair obtained is stored in R and then developped by T_A}
$Q \leftarrow T_B(\alpha, \beta)$
$(\alpha, \beta) \leftarrow (\gamma, \delta)$ for $(1, (\gamma, \delta)) \in Q$
$Q' \leftarrow T_A(\alpha, \beta)$
if $\alpha(1) = \beta(1)$ **then** $\alpha, \beta) \leftarrow (\alpha(1)\mathrm{E}, \alpha(1))$ **endif**
$R \leftarrow R \cup \{(\alpha, \beta)\}$
Each right hand side α in R is replaced by one of its normal forms $\alpha \downarrow R$
if $Q' = \varnothing$ **then** Halt (failure) **else**
for every $(\gamma, \delta) \in Q'$ **do** Decide (γ, δ) **endfor**
endif
Decide (γ, δ) with $(2, (\gamma, \delta)) \in Q$
else
if there exists $(A\gamma, B\delta) \in S \cup S^{-1} \wedge E_\gamma = E_\rho \wedge E_\delta = E_\eta$ **then**
{we update S then we develop by T_C}
if $|\gamma| < |\rho|$ **then** $\lambda \leftarrow \gamma$ **else** $\lambda \leftarrow \rho$ **endif**
if $|\delta| < |\eta|$ **then** $\mu \leftarrow \delta$ **else** $\mu \leftarrow \eta$ **endif**
$S \leftarrow S - \{(A\gamma, B\delta), (B\delta, A\gamma)\}) \cup \{(A\lambda, B\mu)\}$
Decide (ρ, γ)
Decide (η, δ)
else
{the current label is stored in S then developed by T_A}
$S \leftarrow S \cup \{(\alpha, \beta)\}$
$Q \leftarrow T_A(\alpha, \beta)$
if $Q = \varnothing$ **then** Halt (failure) **else**
for every $(\gamma, \delta) \in Q$ **do** Decide (γ, δ) **endfor**
endif
endif
endif
endif
endprocedure

Figures C and D describe the execution trees of the algorithm where the nodes are labelled by the calling parameters of the procedure Decide.

Let us consider the following grammar:

$$G = \{(A, a), (A, bA), (B, aD), (B, b), (C, aBBA), (C, bC), (D, aED), (D, bED), (E, \varepsilon)\}.$$

We have $N_f = \{A, B, C, E\}$; $N_\infty = \{D\}$; $N_\varnothing = \{A\}$.
The algorithm applied to (AAD, CD) builds the following tree:



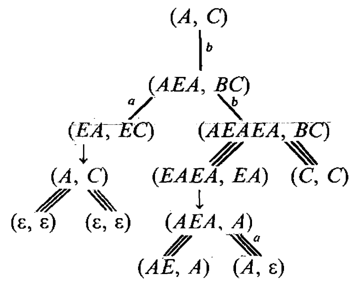
Then $AAD \sim CD$, $R = \{(C, ABBA)\}$ and $S = \{(BAD, AD)\}$.

Figure C. — An equivalence case.

Let the following simple grammar:

$$G=\{(A, a), (A, bAEA), (B, aE), (B, bB), (C, bBC), (E, \varepsilon)\}.$$

We have $N_f=\{A, B, E\}$; $N_\infty=\{C\}$; $N_\varnothing=\varnothing$.
The algorithm applied to (A, C) builds the following tree:



Then A is not equivalent to C, $R=\{(AE, A)\}$ and $S=\{(A, C), (AEA, BC)\}$.

Figure D. — A non-equivalence case.

Furthermore, for clarity, the first pair obtained by a transformation T_B, which is not a calling parameter, is added to the tree. Finally, if the reduction step changes the pair (α, β) then the reduced pair is added to the tree. Operations T_A, T_B, T_C and the reduction are represented respectively by one, two, three lines, and an arrow.

Let us show that this algorithm decide the equivalence $\sim$.

PROPOSITION 3.1: *The algorithm* Decide(α, β) *is well defined, always stops, and returns failure if and only if we do not have* $\alpha \sim \beta$.

To prove Proposition 3.1, we will establish basic properties of $\sim$ in relation to transformations. First, let us notice that the mapping T_A is valid in the following sense:

$$\alpha \sim \beta \quad \Leftrightarrow \quad \varnothing \neq T_A(\alpha, \beta) \subset \sim .$$

To iterate mapping T_A, we extend it to each subset Q of $N^* \times N^*$ as follows:

$$T_A(Q)=\varnothing \quad \text{if it exists } (\alpha, \beta)\in Q,\ T_A(\alpha, \beta)=\varnothing$$
$$T_A(Q)=\{(\lambda, \mu) \mid \exists (\alpha, \beta)\in Q,\ (\lambda, \mu)\in T_A(\alpha, \beta)\} \quad \text{in the other case.}$$

The study of the equivalence of a couple by iteration of T_A is expressed by the lemma below.

LEMMA 3.2: $\alpha \sim \beta \Leftrightarrow \forall n,\ T_A^n(\alpha, \beta) \neq \emptyset$.

Proof: $\Rightarrow$: By induction on n and validity of T_A.

$\Leftarrow$: If α is not equivalent to β then there exists a word u of minimal length belonging to only one of the languages $T(G, \alpha)$ and $T(G, \beta)$. By symmetry of α and β, we can suppose that $u \in T(G, \alpha) - T(G, \beta)$. Let v be the greatest prefix of u such that there exists $\delta \in N^*$ with $\beta \xrightarrow{*}_g v\delta$. If $v = u$ then by definition of u, $T_A^{|u|+1}(\alpha, \beta) = \emptyset$ else $|v| < |u|$ and $T_A^{|v|+1}(\alpha, \beta) = \emptyset$. ■

Lemma 3.2 gives a semi-decision procedure for the non equivalence.

We say that a binary relation R on N^* is *closed by* T_A if $\emptyset \neq T_A(R) \subseteq R$.

COROLLARY 3.3: *Every relation closed by T_A transformation, is included in $\sim$.*

Proof: If $\emptyset \neq T_A(R) \subseteq R$ then by induction on n, $\emptyset \neq T_A^n(R) \subseteq R$ and by Lemma 3.2, $R \subseteq \sim$. ■

Compared to the relation $\equiv$, the difficulty in studying $\sim$ is that $\sim$ is not a congruence and is not simplifiable (for the concatenation). For instance, with $G = \{(A, aE), (B, aEC), (C, a), (E, \varepsilon)\}$, we have $A \sim B$ but not $AA \sim BA$. Nevertheless and taking $\equiv$ into account, Lemma 3.4 gives for $\sim$ some closure conditions and right simplification.

LEMMA 3.4: *Given non-terminal words α, β, γ, the following properties hold:*

(i) *if* $\alpha \sim \beta$ *then* $\gamma\alpha \sim \gamma\beta$

(ii) *if* $\gamma\alpha \sim \gamma\beta$ *and* $\gamma \in N_f^*$ *then* $\langle \alpha \rangle \sim \langle \beta \rangle$

(iii) *if* $\alpha \sim \beta$ *and* $\alpha \equiv \beta$ *then* $\alpha\gamma \sim \beta\gamma$

(iv) *if* $\alpha\gamma \sim \beta\gamma$ *and* $\alpha \equiv \beta$ *and* $\gamma(1) \neq E$ *then* $\alpha \sim \beta$.

Proof: Let us show (iii). Let $\alpha \sim \beta$ such that $\alpha \equiv \beta$, and let us consider u in $T(G, \alpha\gamma)$. We distinguish the two following cases:

Case 1: $u \in T(G, \alpha)$. As $T(G, \alpha) = T(G, \beta) \subseteq T(G, \beta\gamma)$, we have $u \in T(G, \beta\gamma)$.

Case 2: $u \notin T(G, \alpha)$. So it exists $u' \in L(G, \alpha)$ and $u'' \in T(G, \gamma)$ such that $u' u'' = u$. Consequently $u' \in L(G, \beta)$, then $u = u' u'' \in T(G, \beta\gamma)$.

So $T(G, \alpha\gamma) \subseteq T(G, \beta\gamma)$ and in a symmetric way, we have $\alpha\gamma \sim \beta\gamma$.

The other proofs follow the same path. ■

So we restrict the rewriting according to a binary relation R on N^* to the relation $\underset{R}{\Rightarrow}$ defined for every non terminal words α and β by:

$\alpha \underset{R}{\Rightarrow} \beta$ if and only if there exist λ, $\mu \in N^*$ and $(\gamma, \delta) \in R$ such that $\alpha = \lambda\gamma\mu$ and $\beta = \lambda\delta\mu$ and (if $\mu \neq \varepsilon$ then $\gamma \equiv \delta$).

We write $\underset{R}{\Leftrightarrow}$ the symmetric closure of $\underset{R}{\Rightarrow}$, and $\overset{*}{\underset{R}{\Leftrightarrow}}$ the reflexive and transitive closure of $\underset{R}{\Leftrightarrow}$. The equivalence $\overset{*}{\underset{R}{\Leftrightarrow}}$ is not closed by right concatenation, and therefore is not a congruence. Nevertheless, we can retake the notion of self-proving relation defined by Courcelle [3]: a binary relation R on $[N^*]$ is *self-proving* if $\varnothing \neq T_A(R) \subseteqq \overset{*}{\underset{R}{\Leftrightarrow}}$.

In the same way as Lemma 2.4, Proposition 2.5, and Corollary 2.6, we have the results below.

LEMMA 3.5: *Given a binary relation R on $[N^*]$ such that $T_A(R) \neq \varnothing$, we have*

$$\varnothing \neq T_A(\overset{*}{\underset{R}{\Rightarrow}}) \subseteqq \overset{*}{\underset{S}{\Rightarrow}} \qquad \textit{with} \quad S = R \cup T_A(R).$$

PROPOSITION 3.6: *A relation R is self-proving if and only if $\overset{*}{\underset{R}{\Leftrightarrow}}$ is closed by transformation T_A.*

COROLLARY 3.7: *Every self-proving relation is included in $\sim$.*

As transformation T_A, the mapping T_B restricted to the non-terminal words, is valid.

PROPOSITION 3.8: *For all non-terminal and non empty words α and β such that* $\mathrm{Dif}(\alpha(1), \beta(1)) \neq \varnothing$, $\alpha \sim \beta$ *if and only if* $T_B(\alpha, \beta) \subset \sim$.

Proof: Let us consider the non terminal words $A\alpha$ and $B\beta$ such that $(\gamma, \varepsilon) \in \mathrm{Dif}(A, B)$. Let us show that

$$T_B(A\alpha, B\beta) = \{(AE_\alpha, BE_\beta\langle\gamma E_\alpha\rangle), (\langle\gamma\alpha\rangle, \langle\beta\rangle)\}$$

is included in $\sim$ if and only if $A\alpha \sim B\beta$.

As $\alpha \sim E_\alpha\langle\alpha\rangle$, we have by Lemma 3.4 (i) $\gamma\alpha \sim \gamma E_\alpha\langle\alpha\rangle$.

Furthermore $\langle \gamma E_\alpha \langle \alpha \rangle\rangle = \langle \gamma E_\alpha \rangle \langle \alpha \rangle$, so we get the following property (1):

$$\langle \gamma\alpha \rangle \sim \langle \gamma E_\alpha \rangle \langle \alpha \rangle. \tag{1}$$

(i) Suppose that $A\alpha \sim B\beta$. By Lemma 3.2, we get $\langle \gamma\alpha \rangle \sim \langle \beta \rangle$. By Lemma 3.4, we have $BE_\beta \langle \gamma\alpha \rangle \sim BE_\beta \langle \beta \rangle \sim B\beta \sim A\alpha \sim AE_\alpha \langle \alpha \rangle$ and with (1), we get $BE_\beta \langle \gamma E_\alpha \rangle \langle \alpha \rangle \sim \mathrm{AE}_\alpha \langle \alpha \rangle$. As $A \equiv B\gamma$, we get by Lemma 3.4 $BE_\beta \langle \gamma E_\alpha \rangle \sim \mathrm{AE}_\alpha$. Finally $T_B(A\alpha, B\beta) \subset \sim$.

(ii) Suppose that $T_B(A\alpha, B\beta) \subset \sim$. So with (1) and Lemma 3.4, we get

$$A\alpha \sim AE_\alpha \langle \alpha \rangle \sim BE_\beta \langle \gamma E_\alpha \rangle \langle \alpha \rangle \sim \mathrm{BE}_\beta \langle \gamma\alpha \rangle \sim BE_\beta \langle \beta \rangle \sim B\beta. \quad \blacksquare$$

We are left with the study of the transformation T_C. For this, we need the following lemma.

LEMMA 3.9: *Given non terminal words* α *and* β *of* N_f^* *such that* $\neg\ (\alpha \equiv \beta)$,

$$\textit{if } \alpha\gamma \sim \beta\gamma \textit{ and } \alpha\delta \sim \beta\delta \textit{ and } E_\gamma = E_\delta \textit{ then } \gamma \sim \delta.$$

Proof: (i) Suppose that $\alpha \sim \gamma\alpha$, $\beta \sim \gamma\beta$ and $\langle \gamma \rangle \neq \varepsilon$, we show that $\alpha \sim \beta$. From Lemma 3.4, the relation $\sim$ is closed by left concatenation, then $\alpha \sim \gamma^i\alpha$ and $\beta \sim \gamma^i\beta$ for every integer i. Let $u \in T(G, \alpha)$. As $T(G, \alpha) = T(G, \gamma^{|u|+1}\alpha)$ and $\langle \gamma \rangle \neq \varepsilon$, we have $u \in T(G, \gamma^{|u|+1}) \subseteq T(G, \gamma^{|u|+1}\beta) = T(G, \beta)$. By symmetry of α and β, it follows that $\alpha \sim \beta$.

(ii) Suppose that $\alpha\gamma \sim \beta\gamma$, $\alpha\delta \sim \beta\delta$ and $E_\gamma = E_\delta$ with $\alpha, \beta \in N_f^*$ such that $L(G, \alpha) \neq L(G, \beta)$. We show that $\gamma \sim \delta$. There exists a minimal word u belonging to only one of the languages $L(G, \alpha)$ and $L(G, \beta)$. Without loss of generality, we can suppose $u \in L(G, \alpha) - L(G, \beta)$.

Either there exists $\lambda \in N^*$ such that $\beta \overset{*}{\rightarrow}_g u\lambda$ and $\langle \lambda \rangle \neq \varepsilon$. By hypothesis and by Lemma 3.2, we get $\langle \gamma \rangle \sim \langle \lambda\gamma \rangle$ and $\langle \delta \rangle \sim \langle \lambda\delta \rangle$. From property (1) in the proof of Proposition 3.8, it follows that $\langle \gamma \rangle \sim \langle \lambda E_\gamma \rangle \langle \gamma \rangle$ and $\langle \delta \rangle \sim \langle \lambda E_\delta \rangle \langle \delta \rangle$. As $E_\gamma = E_\delta$ and by (i), $\langle \gamma \rangle \sim \langle \delta \rangle$ then $\gamma \sim \delta$.

Or $\gamma, \delta \in N_\varnothing^*$ hence $\gamma \sim \delta$. $\blacksquare$

Let us show that the equivalence $\sim$ is closed by T_C.

PROPOSITION 3.10: *Given* $A\alpha$, $B\beta$, $A\gamma$, $B\delta \in [N^*]$ *such that* $A, B \in N - \{E\}$,

$$\mathrm{Diff}(A, B) = \varnothing, \quad E_\alpha = E_\gamma \quad \textit{and} \quad E_\beta = E_\delta,$$

if $A\alpha \sim B\beta$ *and* $A\gamma \sim B\delta$ *then* $\alpha \sim \gamma$ *and* $\beta \sim \delta$.

Proof: (i) If $A \in N_\infty$ or $B \in N_\infty$ then by symmetry, we can suppose that $A \in N_\infty$. As $A\alpha$, $A\gamma \in [N^*]$, we have $\alpha = \gamma = \varepsilon$, therefore $B\beta \sim A \sim B\delta$. If $B \in N_\infty$ then $\beta = \delta = \varepsilon$ else by Lemma 3.4, we have $\langle \beta \rangle \sim \langle \delta \rangle$, and $E_\beta = E_\delta$, we get $\beta \sim \delta$.

(ii) If $A \in N_f$ and $B \in N_f$ then we consider the two following cases:

Case 1: there exists $\lambda \in N^*$ such that $A \overset{*}{\rightarrow}_g \mathrm{Val}(B)\lambda$ or $B \overset{*}{\rightarrow}_g \mathrm{Val}(A)\lambda$. By symmetry of A and B, we can suppose that $A \overset{*}{\rightarrow}_g \mathrm{Val}(B)\lambda$. As $\mathrm{Dif}(A, B) = \varnothing$, we have $\neg (A \equiv B\lambda)$. From Lemma 3.2, we get $\langle \lambda\alpha \rangle \sim \langle \beta \rangle$ and $\langle \lambda\gamma \rangle \sim \langle \delta \rangle$. We have the two following subcases:

Either $\lambda \notin N_f^*$ then $\langle \beta \rangle \sim \langle \delta \rangle$, hence $\beta \sim \delta$, so $A\alpha \sim A\gamma$ and by Lemma 3.4, $\alpha \sim \gamma$.

Or $\lambda \in N_f^*$ then by property (1) in the proof of Proposition 3.8, we get

$$\langle AE_\alpha \rangle \langle \alpha \rangle \sim \langle BE_\beta \rangle \langle \lambda E_\alpha \rangle \langle \alpha \rangle$$

and

$$\langle AE_\gamma \rangle \langle \gamma \rangle \sim \langle BE_\beta \rangle \langle \lambda E_\gamma \rangle \langle \gamma \rangle.$$

So by Lemma 3.9, we have $\langle \alpha \rangle \sim \langle \gamma \rangle$, hence $\alpha \sim \gamma$, so $B\beta \sim B\delta$ and by Lemma 3.4, $\beta \sim \delta$.

Case 2: on the contrary of Case 1, we have $\alpha, \beta, \gamma, \delta \in N_\varnothing^*$ hence $\alpha = \beta = \gamma = \delta = \varepsilon$. In particular $\alpha \sim \gamma$ and $\beta \sim \delta$. ■

The decision algorithm constructs a *fundamental* relation R, that is to say a binary relation on N^* verifying the following conditions:

(*a*) $\mathrm{Dom}(R) \subseteq N \cup N.\{E\}$ and $\mathrm{Im}(R) \subseteq (N - \{E\}).N^*$

(*b*) R is irreducible: $\mathrm{Im}(R) \cap N^*.\mathrm{Dom}(R).N^* = \varnothing$

(*c*) R is functional: if $\alpha R \beta$ and $\alpha R \gamma$ then $\beta = \gamma$.

LEMMA 3.11: *Given a fundamental relation* R,

$$\# R \leqq 2.\# N \text{ and } \underset{R}{\rightarrow} \textit{ is of finite termination.}$$

Proof: Let R be a fundamental relation. From conditions (*a*) and (*b*) of the definition, every derivation according to R from a non-terminal word α is of length at most $|\alpha|$, so $\underset{R}{\rightarrow}$ is of finite termination. Moreover by (*a*) and (*c*), $\# R = \# \mathrm{Dom}(R) \leqq 2.\# N$. ■

Now, we can establish Proposition 3.1.

Proof of Proposition 3.1: Let us consider the sequence $(\alpha_i, \beta_i, R_i, S_i)_{i \geqq 0}$ of successive calling parameters of the procedure Decide applied to (α, β) where $(\alpha_0, \beta_0)=(\alpha, \beta)$ and $R_0=S_0=\varnothing$, and such that if step (a) (of reduction) of the algorithm applied to (α_i, β_i) gives a couple (λ, μ) distinct to (α_i, β_i), then $(\alpha_{i+1}, \beta_{i+1}, R_{i+1}, S_{i+1})=(\lambda, \mu, R_i, S_i)$.

(i) By induction on i, we verify that R_i is fundamental, and that S_i is a binary relation on $(N-\{E\}).N^*$ such that

$$\text{if } A\lambda\ S_i\ B\mu \text{ and } A\rho\ S_i\ B\eta \text{ and } E_\lambda=E_\rho \text{ and } E_\mu=E_\eta \text{ then } \lambda=\rho \text{ and } \mu=\eta.$$

So there is only a finite number of nodes (α_i, β_i) expanded by T_A. So much holds for T_B. Let i_0 be the greatest integer i such that (α_i, β_i) has been expanded by T_A. For every $i>i_0$ such that (α_i, β_i) has been expanded by T_C, we have one of the two following cases:

$$«S_{i+1}» < «S_i» \text{ where } «R» \text{ is the sum of the } |\lambda|+|\mu| \text{ for } (\lambda, \mu)\in R$$

or

$$S_{i+1}=S_i \text{ and for every } (\lambda, \mu)\in T_C(\alpha_i, \beta_i),\ \max(|\lambda|.\ |\mu|)<\max(|\alpha_i|, |\beta_i|).$$

So the total number of nodes developed by T_C is finite, Finally, the sequence $(\alpha_i, \beta_i, R_i, S_i)_{i\geqq 0}$ is finite. Hence, the algorithm Decide is well defined and always stops.

(ii) if $\alpha \sim \beta$ then, using Lemma 3.2 and Propositions 3.8 and 3.10, we show by induction on $i\geqq 0$ that $\alpha_i \sim \beta_i$. Then the algorithm does not return a failure.

(iii) Let us suppose that the algorithm does not return a failure and we show that $\alpha\sim\beta$. So $\alpha_p=\beta_p$ where p is the last index of the sequence $(\alpha_i, \beta_i, R_i\ S_i)$. We add to $N_\varnothing$ a new symbol \$, and we consider the canonical relation

$$S=\{(EE, E)\}\cup(AB, A)\,|\,A\in N_\infty \wedge B\in N\}\cup\{(A\$, \$)\,|\,A\in N_\varnothing\}.$$

So, for every non terminal word α, $[\alpha]\$$ is the canonical form of $\alpha\$$ according to S. Given a binary relation T on N^*, we write $T\$=\{\gamma\$, \delta\$)\,|\,\gamma T\delta\}$. We want to show that the relation $R=S\cup R_p\cup S_p\$$ is self-proving. By induction on $i\leqq p$, we establich the following inclusion (1):

$$R_i\cup S_i\$\cup\{(\alpha_i\$, \beta_i\$)\}\subseteq \overset{*}{\underset{R}{\Leftrightarrow}}. \tag{1}$$

Let Q be the set of pairs expanded by T_A obtained as the first pair of a T_B transformation. Then $R_p \subseteq \overset{*}{\underset{R_p}{\leftrightarrow}} = \overset{*}{\underset{Q}{\Leftrightarrow}}$. As the algorithm does not return a failure, $T_A(Q) \neq \varnothing$ and from (1), $P = Q \cup T_A(Q) \subseteq \overset{*}{\underset{R}{\Leftrightarrow}}$. From Lemma 3.5, $\varnothing \neq T_A(R_p) \subseteq \overset{*}{\underset{P}{\Leftrightarrow}} \subseteq \overset{*}{\underset{R}{\Leftrightarrow}}$.

Let us show that $\varnothing \neq T_A(S_p \$) \subset \overset{*}{\underset{R}{\Leftrightarrow}}$. Let $A\lambda\ S_p\ B\mu$. Let us consider the following set I:

$$I = \{ i \geqq 0 \mid \exists\, \eta,\ \rho,\ (A\eta,\ B\rho) \in (S_{i+1} \cup S_{i+1}^{-1}) - (S_i \cup S_i^{-1}) \}.$$

By inverse induction on $i \in I$ and for $(A\lambda_i,\ B\mu_i) = (\alpha_i,\ \beta_i)$ or $(A\lambda_i,\ B\mu_i) = (\beta_i,\ \alpha_i)$, we have the following property (2):

$$\lambda\$ \overset{*}{\underset{R}{\Leftrightarrow}} \lambda_i \$ \quad \text{and} \quad \mu\$ \overset{*}{\underset{R}{\Leftrightarrow}} \mu_i \$. \tag{2}$$

Let i_0 be the smallest integer in I. So $(\alpha_{i_0},\ \beta_{i_0})$ has been expanded by T_A, then

$$T_A(A\lambda,\ B\mu) \neq \varnothing.$$

Let $(\gamma\lambda,\ \delta\mu) \in T_A(A\lambda,\ B\mu)$. Then $(\gamma\lambda_{i_0},\ \delta\mu_{i_0}) \in T_A(A\lambda_{i_0},\ B\mu_{i_0})$ and with (1), $\gamma\lambda_{i_0}\$ \overset{*}{\underset{R}{\Leftrightarrow}} \delta\mu_{i_0}\$$. It follows with (2) that $\gamma\lambda\$ \overset{*}{\underset{R}{\Leftrightarrow}} \delta\mu\$$. So

$$\varnothing \neq T_A(A\lambda\$,\ \mathrm{B}\mu\$) \subset \overset{*}{\underset{R}{\Leftrightarrow}},$$

hence $\varnothing \neq T_A(S_p \$) \subset \overset{*}{\underset{R}{\Leftrightarrow}}$.

Furthermore

$$T_A(S) = \{ (\varepsilon,\ \varepsilon) \} \cup T_A(\{ A,\ A) \mid A \in N_\infty \}),$$

hence $\varnothing \neq T_A(S) \subset \overset{*}{\underset{R}{\Leftrightarrow}}$.

Finally $\varnothing \neq T_A(R) \subset \overset{*}{\underset{R}{\Leftrightarrow}}$, *i.e.* R is self-proving, and from Corollary 3.7, $\overset{*}{\underset{R}{\Leftrightarrow}} \subseteq \sim$.

From this inclusion and Property (1), we infer in particular $\alpha_0 \$ \sim \beta_0 \$$, hence $\alpha \sim \beta$. ■

Let us evaluare the complexity of the algorithm applied to a pair of non-terminals. Denote by n the size of G, let $v = \max\{v(A) \mid A \in N_f\}$ be the finite valuation of G, and $\|G\| = \max\{|\gamma| \mid \exists A, A \to \gamma\}$ the maximum length of the right hand sides of G. The total numbers of pairs developed by T_A is at most $2(\# N)^2$, and the same holds for the number of pairs developed by T_B. So the maximum length of the calling parameters of the algorithm is in $O(m)$ where $m = (\# N)^2 . \|G\| . v$. Hence the total number of pairs developped by T_C is $O(\# N^2 . m)$, and the same bound holds for the number of pairs in the tree. The cost of transformation T_A is $O(\# T . m)$. From the complexity of the former algorithm, the cost of transformation T_B is $O(m + \# T . \# N . \|G\| . v)$. The cost of transformation T_C is $O(m)$. The construction of relation S is $O(\# N^2 . m^2)$. The construction of relation R is $O(\# N^2 . v)$. The cost of a reduction is $O(m)$, hence the total cost of a reduction is $O(\# N^2 . m^2)$ or $O(n^8 . v^2)$. Finally, the complexity of the algorithm applied to non-terminals, is $O(n^8 . v^2)$. As the valuation v is $O(\|G\|^{\# N})$ or $O(n^n)$, we finally get the following theorem.

THEOREM 3.12: *The equivalence problem of stateless dpda is decidable by an algorithm of complexity* $O(n^8 . v^2)$ *or* $O(n^n)$ *where n is the size of the compared automata, and v is the greatest finite valuation of the stack letters.*

Probably, the complexity $O(n^8 . v^2)$ may be improved. But contrary to the way of thinking [3], the aim of this paper was to get a polynomial complexity in the size n and the finite valuation v to decide on the equivalence problem for stateless dpda.

APPLICATION

The algorithm in Section 3 allows also to decide efficiently the equivalence of monadic recursive program schemes. Recall that a *recursive program scheme* S, or simply a scheme, on a graded alphabet F and an enumerable set $V = \{v_1, \ldots, v_n, \ldots\}$ of variables is a finite set of rules $f(v_1, \ldots, v_n) \to t$, where f is a member of F of arity n and t is a term on $F \cup \{v_1, \ldots, v_n\}$, satisfying the following conditions:

(i) S is functional: $f(v_1, \ldots, v_n) \to t$ and $f(v_1, \ldots, v_n) \to t'$ imply $t = t'$

(ii) S is in *Greibach form*: $f(v_1, \ldots, v_n) \to t$ imply $t = g(t_1, \ldots, t_m)$ and $g(v_1, \ldots, v_m)$ is not a left member of S.

Denote by $N(S)=\{f \mid \exists n \exists t, f(v_1, \ldots, v_n) \rightarrow t\}$ the set of *defined functions* of S, and $T(S)$ the subset of $F-N(S)$ of *base functions* used by S. The *solution* of a scheme in a term t on $F \cup V$ is the unfolded tree $S^\infty(t)$ defined recursively as follows:

$$S^\infty(t)=t \quad \text{if} \quad t \in V \quad \text{or} \quad t \in F \text{ (with arity zero)}$$
$$S^\infty(t)=f(S^\infty(t_1), \ldots, S^\infty(t_n)) \quad \text{if} \quad t=f(t_1, \ldots, t_n) \quad \text{and} \quad f \notin N(S)$$
$$S^\infty(t)=S^\infty(t'[v_1 \leftarrow t_1, \ldots, v_n \leftarrow t_n])$$
$$\text{if } t=f(t_1, \ldots, t_n) \quad \text{and} \quad f(v_1, \ldots, v_n) \rightarrow t'.$$

We say that two terms t and t' are *equivalent* according to a scheme S if they have the same solution, *i.e.* $S^\infty(t)=S^\infty(t')$. A scheme S is called *monadic* if it uses a unique variable v, *i.e.* all rules are of the form $f(v) \rightarrow t$. A scheme S is *reduced* if the solution $S^\infty(f(v_1, \ldots, v_n))$ of every defined function f, has a finite branch.

The equivalence problem for the monadic reduced schemes without constant base function (of arity zero), is linearly reducible [4] to the equivalence problem for the simple grammars, which can be decide efficiently by the algorithm of Section 2. Similarly, we will reduce linearly the decidability of the equivalence for monadic schemes to the equivalence problem for stateless dpda.

We take two terms s and t with v as unique variable, but not equal to v. They are equivalent according to a monadic scheme S if and only if A and B are equivalent in the new system $S'=S \cup \{A(v) \rightarrow s, B(v) \rightarrow t\}$ where A and B are two new symbols. The new system S' is monadic and functional. Even if it entails the rewriting of s and t according to S, we can suppose that S' is also in Greibach form. Then it is a monadic scheme. We want to put S' in *Greibach normal form, i.e.* if $f(v) \rightarrow g(t_1, \ldots, t_m)$ then the t_i's are terms on $N(S') \cup \{v\}$, and such that $A(v)$ and $B(v)$ are equivalent according to S' if and only if they are equivalent according to S. We replace each constant a by a filiform infinite tree $(a')^\infty$ by substituting $a''(v)$ to a in all the rules of S, and adding a rule $a''(v) \rightarrow a'(a''(v))$. Then we rename some subterms and add new rules to transform the scheme into a scheme S'' monadic and in Greibach normal form, such that $S''^\infty(A(v))=S''^\infty(B(v))$ if and only if $S^\infty(A(v))=S^\infty(B(v))$.

Finally, to every monadic scheme S in Greibach normal form, and to every function A defined by S, we associate the stateless dpda defined below:

(*a*) the input alphabet is $\{(g, i) \mid g \in T(S) \wedge 1 \leq i \leq \text{arity}(g)\}$,

(*b*) the stack alphabet is $N(S) \cup \{E\}$ where E is the bottom stack letter, (*c*) the transitions are all the rules of the form $f \overset{(g\,.\,i)}{\vdash} t_i$ $(1 \leqq i \leqq m)$ when

$$f(v) \to g(t_1 v, \ldots, t_m v),$$

(*d*) the axiom is A

(*e*) the acceptance test is the presence of any letter on the top of the stack.

This automaton recognizes all partial branches of the unfolded tree $S^\infty(A)$. Now such a tree is characterized without ambiguity by the set of its partial branches. To compare $S^\infty(A)$ with $S^\infty(B)$, we are brought back to test the equivalence of two stateless dpda. As a result, we can decide the equivalence of monadic schemes by the help of an algorithm of polynomial complexity in the length of description and the finite valuation, where the finite valuation of a scheme S is the greatest finite valuation of the defined functions (the valuation of a defined function is the shortest length of the branches of its solution tree).

REFERENCES

1. D. CAUCAL, Décidabilité de l'égalité des langages algébriques infinitaires simples, *L.N.C.S.*, Vol. *210*, 1986, pp. 37-48.
2. D. CAUCAL, A Fast Algorithm to Decide on Simple Grammars Equivalence, *L.N.C.S.*, Vol. *401*, 1989, pp. 66-85.
3. B. COURCELLE, An Axiomatic Approach to the KH Algorithms, *Math. Systems Theory*, Vol. *16*, 1983, pp. 191-231.
4. B. COURCELLE and J. VUILLEMIN, Completeness Result for the Equivalence of Recursive Schemes, *J.C.S.S.*, Vol. 12, 1976, pp. 179-197.
5. M. HARRISON, *Introduction to Formal Language Theory*, Addison-Wesley, 1978.
6. M. HARRISON, I. HAVEL and A. YEDUHAÏ, On Equivalence of Grammars Through Transformation Trees, *T.C.S.*, Vol. 9, 1979, pp. 191-231.
7. A. KORENJAK and J. HOPCROFT, Simple Deterministic Languages, *Seventh annual I.E.E.E. switching and automata theory conference*, 1966, pp. 36-46.
8. T. OLSHANSKY and A. PNUELI, A Direct Algorithm for Checking Equivalence of LL (k) Grammars, *T.C.S.*, Vol. *4*, 1977, pp. 321-349.
9. M. OYAMAGUCHI and N. HONDA, The Decidability of Equivalence for Deterministic Stateless Pushdown Automata, *Information and Control*, Vol. *38*, 1978, pp. 367-376.
10. E. TOMITA, An Extended Direct Branching Algorithm for Checking Equivalence of Deterministic Pushdown Automata, *T.C.S.*, Vol. *32*, 1984, pp. 87-120.
11. L. VALIANT, The Equivalence Problem for Deterministic Finite-Turn Pushdown Automata, *Information and Control*, Vol. *25*, 1974, pp. 123-153.
12. L. VALIANT and M. PATERSON, Deterministic One-Counter Automata, *J.C.S.S.*, Vol. *10*, 1975, pp. 340-350.