RAIRO. INFORMATIQUE THÉORIQUE ET APPLICATIONS

BRUNO COURCELLE
JEAN H. GALLIER

Decidable subcases of the equivalence problem for recursive program schemes

RAIRO. Informatique théorique et applications, tome 21, n° 3 (1987), p. 245-286

<http://www.numdam.org/item?id=ITA_1987__21_3_245_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 21, n° 3, 1987, p. 245 à 286)

DECIDABLE SUBCASES OF THE EQUIVALENCE PROBLEM FOR RECURSIVE PROGRAM SCHEMES (*)

by Bruno COURCELLE (1) and Jean H GALLIER (2)

Communicated by A. ARNOLD

Abstract. – *The equivalence problems for polyadic recursive program schemes (interpreted over continuous algebras) and DPDA's are known to be interreducible. Several proofs have been given by Courcelle* [3, 4, 5] *and Gallier* [10]. *However, the decidability of either problem is still open. On the other hand, the equivalence problem is known to be decidable for various subclasses of the class of DPDA's* [20, 21, 22, 23, 24, 25].

In this paper, various classes of recursive program schemes which correspond to the subclasses of realtime strict DPDA's [15, 17] *and finite turn DPDA's* [11, 15] *via the constructions used in* [3, 4, 5, 10] *are defined and studied. The equivalence problem for some of these new classes of program schemes is shown to be decidable.*

Résumé. – *Le problème de l'équivalence des schémas de programmes récursifs polyadiques (interprétés dans des algèbres continues) et le problème de l'équivalence des DPDA sont interréductibles. Les preuves ont été données par Courcelle* [3, 4, 5] *et Gallier* [10]. *Mais la décidabilité reste un problème ouvert. Par contre le problème de l'équivalence est décidable pour différentes classes de DPDAs (Valiant, Oyamaguchi* [20-25]).

Cet article étudie les classes de schémas de programmes récursifs qui correspondent aux DPDA dits « realtime strict » et « finite-turm », par les constructions de Courcelle et Gallier [3, 4, 5, 10]. *Pour ces classes de schémas de programmes l'équivalence est décidable.*

0. INTRODUCTION

Equivalence problems for certain classes of context-free grammars, deterministic pushdown automata (for short, DPDA's), program schemes, string-and-tree-transducers, have been investigated quite extensively (from a vast

(*) Received July 1986, revised December 1986.

(1) Université de Bordeaux-I, U.E.R. de Mathématiques et Informatique, 351, cours de la Libération, 33405 Talence, France.

(2) Department of Computer and Information Sciences, Moore School of Electrical Engineering D2, University of Pennsylvania, Philadelphia, Pennsylvania 19104, U.S.A.

Note : Ce travail a été soutenu par le contrat du C.N.R.S. A.T.P.-4275 et par le projet de recherches coordonnées « Mathématiques et Informatique ».

literature, we only cite [1, 4, 6, 9, 10, 20 to 25]). In particular, the equivalence problem for DPDA's remains open, although a number of special subcases have been shown to be decidable (Valiant [24, 25], Oyamaguchi *et al.* [20 to 23]). The equivalence problem for recursive program schemes (abbreviated as RPS's) with respect to the class of all interpretations (all continuous algebras over a certain signature) has been shown to be interreducible with the equivalence problem for deterministic pushdown automata. This interreducibility has been established by means of several constructions (Courcelle [3, 4], Gallier [10]) associating a DPDA with a recursive program scheme and vice-versa.

The motivation for such a research is usually not the practical use of the results for at least three reasons. Firstly, a decidability result is often useless because it solves a problem whose scope is too restricted to be practically useful. For instance, this is the case for the decision procedure for the equivalence problem of $LL(k)$ grammars. Similarly, the strong equivalence of program schemes is usually considered much too restrictive to be of any practical use. If one wants to use program schemes for expressing program transformations, one must introduce classes of interpretations and consider the equivalence of program schemes modulo such classes (see [14]). Secondly, most interesting properties are undecidable (in particular for program schemes). Thirdly most known decision procedures for the above problems are of super-exponential complexity (in particular [1, 23, 25]), and thus, not practically usable. However, the motivations for such investigations exist and are twofold:

(1) To draw the boundary between decidable and undecidable problems;

(2) To increase our understanding of the deep properties of these objects (grammars, automata, program schemes, ...), since a decidability result *almost always rests upon some combinatorial property* of these objects.

For example, the decidability of realtime strict DPDA's is based on a finiteness property which is not valid for arbitrary DPDA's.

In the same spirit, interreduciblity results for open problems are interesting, because they usually establish a structural similarity between objects of various types. Such constructions remain interesting, even if these problems are found to be decidable or undecidable later.

This paper is devoted to a careful study of such constructions relating recursive program schemes and DPDA's. More precisely, the goals of this paper are:

1. to apply known decidability results together with the constructions of Courcelle and Gallier to determine classes of program schemes for which the equivalence problem is decidable,

2. to compare these constructions.

It is hoped that these investigations might yield alternative proofs for the known decidable cases. However, this topic is left for further research.

Gallier [10] has shown that the equivalence problem for monadic recursive program schemes reduces to the equivalence problem for stateless DPDA's, shown to be decidable by Oyamaguchi and Honda [20], and extending a result of Courcelle and Vuillemin [6].

In this paper, the class of realtime strict DPDA's [15 to 18] and finite turn DPDA's [11, 15] are considered and corresponding classes of program schemes are exhibited.

The following classes of recursive program schemes are defined:

$\mathscr{B}$ = class of balanced schemes,

$\mathscr{EB}$ = class of extended balanced schemes,

$\mathscr{B}'$ = class of e-limited schemes,

$\mathscr{EB}'$ = class of extended e-limited schemes,

$\mathscr{FTS}$ = class of finite-turn schemes,

$\mathscr{OR}$ = class of ordered schemes,

$\mathscr{UL}$ = class of ultralinear schemes.

Given a class $\mathscr{C}$ of DPDA's or RPS's, we denote by $EQ(\mathscr{C})$ the corresponding equivalence problem and the many-one reducibility relation is denoted by $\leqq$. If $\mathscr{R}_0$ stands for the class of realtime strict DPDA's, $\mathscr{R}$ for the class of realtime DPDA's accepting by final state and $\mathscr{FT}$ for the class of finite turn DPDA's, our results can be summarized as follows:

$$EQ(\mathscr{R}_0) \leqq EQ(\mathscr{B}) \leqq EQ(\mathscr{B}') \leqq EQ(\mathscr{R}_0),$$

$$EQ(\mathscr{R}) \leqq EQ(\mathscr{EB}) \leqq EQ(\mathscr{EB}') \leqq EQ(\mathscr{R}),$$

$$\mathscr{FTS} = \mathscr{OR},\ EQ(\mathscr{FTS}) \leqq EQ(\mathscr{FT}) \leqq EQ(\mathscr{FTS}),$$

$$EQ(\mathscr{UL}) \leqq EQ(\mathscr{FT}).$$

Hence, any "direct" proof of the decidability of $EQ(\mathscr{B})$ would yield another proof of $EQ(\mathscr{R}_0)$, hopefully simpler than that of Omayaguchi *et al.* [21] (such direct proofs exist in certain cases, *see* Courcelle and Vuillemin [6], Courcelle [7], or Caucal [2]). Similarly, a "direct" proof of the decidability of $EQ(\mathscr{FTS})$ would yield another proof of $EQ(\mathscr{FT})$, perhaps simpler than that of Valiant [25] and Beeri [1].

The paper is organized as follows. Definitions and results concerning DPDA's and strict deterministic grammars (Harrison and Havel [16, 17,

18]) are recalled in section 1. Those concerning RPS's together with the constructions of Courcelle [3, 4] and Gallier [10] are informally recalled in section 2. Section 3 deals with the class of program schemes associated with realtime strict DPDA's [15, 17] and section 4 with the class corresponding to finite-turn DPDA's (Ginsburg and Spanier [11], Harrison [15]).

1. DPDA's AND STRICT DETERMINISTIC GRAMMARS

This section gathers some basic definitions and results about DPDA's and strict deterministic grammars.

1.1. Deterministic Pushdown Automata

A *deterministic pushdown automaton*, for short a DPDA, is a 6-tuple

$$D=(K, X, \delta, q_0, F, Z_0),$$

where

K is a finite set of *states*,

X is a finite *alphabet*,

$q_0 \in K$ is a designated state called the *initial state*,

$F \subseteq K$ is a subset of *final states*,

δ is a partial function called the *transition function* with

$$\delta: \quad K \times (X \cup \{e\}) \times \Gamma \to K \times \Gamma^*$$

(with e denoting the empty string),

Γ is a finite alphabet of *pushdown store symbols*,

$Z_0 \in \Gamma$ is a designated symbol called the *bottom of stack marker* and δ satisfies the *determinism condition*

(D) : For every $(p, Z) \in K \times \Gamma$, if $\delta(p, e, Z)$ is defined then, for every $a \in X$, $\delta(p, a, Z)$ is not defined.

An *instantaneous description*, for short an *ID*, is a triple

$$(p, w, \beta) \in K \times X^* \times \Gamma^*.$$

An *initial ID* is any *ID* of the form (q_0, w, Z_0), where q_0 is the initial state, Z_0 the initial stack configuration and w is any input string. The *yield* or

compute relation $\vdash$ between ID's is defined as follows:

$$(p, aw, Z\alpha) \vdash (q, w, \beta\alpha),$$

where $a \in X \cup \{e\}$, $Z \in \Gamma$, $\alpha, \beta \in \Gamma^*$, $w \in X^*$, $p, q \in K$ if and only if

(1) either $a \neq e$ and $\delta(p, a, Z) = (q, \beta)$, or

(2) $a = e$ and $\delta(p, e, Z) = (q, \beta)$.

Moves of type (2) are called *e-moves*. As usual, $\vdash^*$ is the reflexive and transitive closure of $\vdash$ and $\vdash^+$ is its transitive closure.

For any two *ID*'s ID_1 and ID_2, if $ID_1 \vdash^+ ID_2$, we say that there is a *computation* from ID_1 to ID_2. Without loss of generality, we can assume that the symbol Z_0 stays at the bottom of the stack during a computation, except possibly at the end. Hence, the following conditions will be assumed: For all $p, q \in K$, $a \in X \cup \{e\}$,

(i) $\delta(p, a, Z_0) = (q, \beta Z_0)$ or (q, e), with $\beta \in (\Gamma - \{Z_0\})^*$.

(ii) For any $Z \neq Z_0$, $\delta(p, a, Z) = (q, \beta)$, with $\beta \in (\Gamma - \{Z_0\})^*$.

Given a DPDA D, the following three modes of *acceptance* are defined.

$$N(D) = \{w \in X^* \mid \exists\, q \in K,\ (q_0, w, Z_0) \vdash^* (q, e, e)\},$$

$$T(D) = \{w \in X^* \mid \exists\, f \in F,\ \exists\, \beta \in \Gamma^*,\ (q_0, w, Z_0) \vdash^* (f, e, \beta)\},$$

$$L(D) = \{w \in X^* \mid \exists\, f \in F,\ (q_0, w, Z_0) \vdash^* (f, e, e)\}.$$

$N(D)$ corresponds to *acceptance by empty store*, $T(D)$ to *acceptance by final state* and $L(D)$ to *acceptance by final state and empty store.*

A language L is a *deterministic context-free language* if $L = T(D)$ for some DPDA D. In this paper, we are mostly interested in the class of *prefix-free deterministic context-free languages* $PFDet = \{N(D) \mid D \text{ is a } DPDA\}$. Languages in *PFDet* are *prefix-free*, that is, for $u, v \in X^*$, if $u, uv \in L$ then $v = e$.

A DPDA is *realtime* if for all $(p, Z) \in K \times \Gamma$, $\delta(p, e, Z)$ is undefined. A DPDA is *k-limited* if, for all $p, q \in K$, $\alpha, \beta \in \Gamma^*$, $(p, e, \alpha) \vdash^n (q, e, \beta)$ implies $n \leqq k$. In other words, there is a uniform bound k on the number of consecutive e-moves. A DPDA is *e-limited* (or *quasi-realtime*) if it is k-limited for some $k \geqq 0$. It is well known that every e-limited DPDA is equivalent to a realtime DPDA (Harrison [15], Theorem 11.7.3).

Given an $ID(q, u, Z\beta)$, its *mode* is the pair (q, Z). An $ID(q, v, \beta)$ is *reachable* if $(q_0, uv, Z_0) \vdash^* (q, v, \beta)$ for some string u. A mode (q, Z) is *reachable* if some *ID* of the form $(q, v, Z\beta)$ is reachable. An $ID(p, v, \beta)$ is *live* if there exists $v' \in X^*$ (not necessarily equal to v), such that $(p, v', \beta) \vdash^*$

(q, e, e) for some $q \in K$. A DPDA is *faithful* if every reachable *ID* is live. The following lemma is shown in Courcelle [4].

LEMMA 1.1 :*Given a DPDA D, one can construct a faithful DPDA D′ such that* $N(D) = N(D')$. □

1.2. Jump DPDA's

Let $E: 2^X \times X^* \to X^*$ be the partial function such that, for $T \subseteq X$ and $w \in X^*$,

$$E(T, w) = \begin{cases} v \text{ if } w = uav \text{ for some } a \in T, u \in (X-T)^* \text{ and } v \in X^*, \\ \textit{undefined} \text{ otherwise, i. e. if } w \in (X-T)^*. \end{cases}$$

Intuitively, v is obtained by erasing w from left to right, up to and including the first occurence of some element in T. If $T = \{a\}$, $E(a, w)$ stands for $E(T, w)$.

A *Jump-DPDA* is a tuple $D = (K, X, \delta, q_0, F, Z_0)$, where K, X, δ, q_0, F, Z_0 are as for DPDA's but δ is a partial function

$$\delta: \quad K \times (X \cup \{e\}) \times \Gamma \to (K \times \Gamma^*) \cup (K \times \{E\} \times 2^{\Gamma})$$

satisfying the determinism condition (D) (E is a special symbol which means "erase").

The *compute* relation for jump-DPDA's is defined as follows: For $p, q \in K$, $u \in X^*$, $a \in X \cup \{e\}$, $\alpha, \beta \in \Gamma^*$, $Z \in \Gamma$,

$$(p, au, Z\alpha) \vdash (q, u, \gamma)$$

if and only if either

$$\delta(p, a, Z) = (q, \beta) \qquad \text{and} \qquad \gamma = \beta\alpha,$$

or

$$\delta(p, a, Z) = (q, E, T) \qquad \text{and} \qquad \gamma = E(T, Z\alpha).$$

In the latter case, a top segment of the pushdown store is erased in a "jump move".

The languages $N(D)$, $T(D)$ and $L(D)$ are defined as for DPDA's. A jump-DPDA is *realtime* if $\delta(p, e, Z)$ is undefined for all p and Z. A jump-DPDA has *simple jumps* if

$$\delta(p, a, Z) = (q, E, T) \quad \text{implies that} \quad Card(T) = 1.$$

The following Theorem proved in Courcelle [3] shows the importance of jump-DPDA's.

THEOREM 1.2: *A language L is in PFDet if and only if* $L=N(D)$ *for some one state, realtime jump-DPDA D with simple jumps.* □

1.3. DPDA's in atomic form

In subsequent sections, DPDA's having a special structure will be constructed and it is convenient to introduce the following special notation. Details can be found in Gallier [10]. For such DPDA's, $\Gamma \subseteq K \cup \{Z_0\}$ and there are four kinds of moves:

(1) **Read moves**

$$p \xrightarrow{read\ a} q, \qquad \text{for} \quad \delta(p, a, Z)=(q, Z), \quad Z \in \Gamma.$$

(2) **Push moves**

$$p \xrightarrow{push} q, \qquad \text{for} \quad \delta(p, e, Z)=(q, pZ), \quad Z \in \Gamma.$$

(3) **Pop moves**

$$p \xrightarrow{pop\ q} r, \qquad \text{for} \quad \delta(p, e, q)=(r, e).$$

(4) **Change state moves**

$$p \to q, \qquad \text{for} \quad \delta(p, e, Z)=(q, Z), \quad Z \in \Gamma.$$

Note that *push* moves, *pop* moves and *change state* moves are all *e*-moves, and that *read* moves, *push* moves and *change state* moves are independent of the top of stack symbol. We say that such DPDA's are in *atomic form*.

1.4. Strict and complete deterministic grammars

The concepts of a π-strict deterministic grammar and of a π-complete deterministic grammar were introduced in Courcelle [4] and are refinements of the strict deterministic grammars of Harrison and Havel [16, 17, 18].

Let $G=(N, X, P, S)$ be a context-free grammar with $V=X \cup N$ (N is the set of *nonterminals*, X the set of *terminals*). Let π be a partition of V. For any $Y, Z \in V$, $Y \equiv Z\ mod(\pi)$, for short $Y \equiv Z$, if and only if Y and Z are in a same block of π. We also say that Y and Z are (π)-*equivalent*.

A partition π of V is *strict* for G if:

P0: Each block of π is contained in X or N.

P1: For all $A, B \in N$, all $\alpha, \beta, \beta' \in V^*$, if $A \equiv B$, $A \to \alpha\beta$ and $B \to \alpha\beta'$ are in P, then either

$$\beta = \beta' = e \qquad \text{and} \qquad A = B,$$

or

$$\beta \neq e, \qquad \beta' \neq e \qquad \text{and} \qquad FIRST(\beta) \equiv FIRST(\beta').$$

[$FIRST$ is the function such that $FIRST(e) = e$ and $FIRST(au) = a$].

A strict partition π is *π-complete* if:

P2: For all $A \in N$, $Y, Z \in V$ and $\alpha, \beta \in V^*$, if $A \to \alpha Y \beta$ is in P and $Y \equiv Z$ then, there is some $A' \to \alpha Z \beta'$ in P such that $A \equiv A'$.

A grammar G with partition π is *π-strict deterministic*, for short π-SD, if π is strict and it is *π-complete deterministic*, for short π-CD, if π is strict and complete. A grammar is *strict deterministic* (resp. *complete deterministic*) if it is π-SD (resp. π-CD) for some partition π of V.

Note that Harrison and Havel [16, 17, 18] consider grammars only with strict partitions π such that X itself is a block of π. Let us finally recall the concepts of a π-strict and a π-complete language.

Let X be a finite alphabet and π a partition of X. A language $L \subseteq X^*$ is *π-strict* if:

(1) L is prefix-free

(2) For all $u, v, w \in X^*$, $a, b \in X$, if uav and $ubw \in L$, then $a \equiv b$.

L is *π-complete* if it is π-strict and:

(3) For all $u, v \in X^*$, $a, b \in X$ such that $uav \in L$ and $a \equiv b$, there exists some $w \in X^*$ such that $ubw \in L$.

The following Theorem whose proof can be found in Courcelle [4, Theorem 3.20] will also be needed (for a definition of the canonical grammar G_D, *see* Harrison [15]).

THEOREM 1.3: *Given a faithful DPDA D and a partition π such that $N(D)$ is π-complete, the reduced canonical grammar G_D associated with D is π'-complete deterministic for some partition π'.* □

2. RECURSIVE PROGRAM SCHEMES

The theory of infinite trees and recursive program schemes will not be reviewed in depth. The reader is advised to consult one of [4, 5, 7, 10, 12,

14] for details. The constructions in [3, 4, 5, 10] will be reviewed informally. The notation is essentially the one used in [7] and it is reviewed in this section.

2.1. Trees and recursive program schemes

Let F be a *ranked alphabet* of function symbols, where each symbol $f \in F$ has an *arity* $r(f) \geqq 0$. Let $V = \{v_1, v_2, \ldots\}$ be a countable set of nullary symbols, called *variables*, and, for each $k \geqq 0$, let $V_k = \{v_1, \ldots, v_k\}$. The set of *finite well-formed terms* (or finite trees) built from F and V is denoted by $T(F, V)$. The set of *infinite trees* built from F and V is denoted by $CT(F, V)$ [7, 10, 12]. In this paper, it is convenient to define infinite trees using the concept of a *tree domain* due to Gorn [13].

Tree domains

Let ω denote the set of nonnegative integers.

A *tree domain* D is a nonempty subset of $(\omega - \{0\})^*$ such that:

(1) For all $u, v \in (\omega - \{0\})^*$, if $uv \in D$, then u is also in D.

(2) For all $u \in (\omega - \{0\})^*$, all $i > 0$, if $ui \in D$, then $uj \in D$, for all j, $1 \leqq j \leqq i$.

An *F-tree*, for short a tree, is a function $t : D \to F$ from a tree domain D such that, for every $u \in D$, if $n = Card(\{i \mid ui \in D\})$ then $r(t(u)) = n$. The domain of a tree t is denoted by $dom(t)$ and the elements in $dom(t)$ are called *tree addresses* or *nodes*. We let $CT_\perp(F, V)$ denote $CT(F \cup \{\perp\}, V)$, where $\perp$ is a new constant symbol that is the least element of $CT_\perp(F, V)$.

Given two trees t_1, t_2 and a tree address $u \in dom(t_1)$, the result of replacing the subtree at u in t_1 with t_2 is denoted by $t_1[u \leftarrow t_2]$. Also, given a tree $t \in CT_\perp(F, V_k)$ and a k-tuple $(t_1, \ldots, t_k)$ of trees in $CT_\perp(F, V_n)$, the result of simultaneously substituting t_i for $v_i (1 \leqq i \leqq k)$ in t is denoted by $t[t_1/v_1, \ldots, t_k/v_k]$ (or by $t[t_1, \ldots, t_k]$ when the list $v_1, \ldots, v_k$ is known from the context).

A *recursive program scheme* (for short an RPS) is a pair $S = (\Sigma, \tau)$ consisting of:

(i) An *algebraic system* $\Sigma = \langle \varphi_i(v_1, \ldots, v_{n_i}) = \alpha_i;\ 1 \leqq i \leqq N \rangle$, where each α_i is a (finite) tree in $T(F \cup \Phi, V_{n_i})$ (with no occurrence of $\perp$).

(ii) A finite tree $\tau \in T(F \cup \Phi, V)$, where $\Phi = \{\varphi_1, \ldots, \varphi_N\}$ is a set of function variables (or nonterminals) and $n_i = r(\varphi_i)$, $1 \leqq i \leqq N$.

The system Σ has a least solution in $CT_\perp(F, V_{n_1}) \times \ldots \times CT_\perp(F, V_{n_N})$, which is denoted as $(\alpha_1^\nabla, \ldots, \alpha_N^\nabla)$. By substituting α_i^∇ for φ_i in τ, one obtains the (usually infinite) tree τ^∇ which represents all possible computations of S

in all interpretations. More precisely,

$$\tau^{\nabla} = \tau\{\alpha_1^{\nabla}/\varphi_1, \ldots, \alpha_N^{\nabla}/\varphi_N\},$$

with the notation of Courcelle [4, 5]. This tree is also denoted by S^{∇}.

Recall that two RPS's S and S' are (*strongly*) *equivalent*, that is, define the same function in all interpretations, if and only if $S^{\nabla} = S'^{\nabla}$. Hence, we shall be interested in deciding this equality.

Note that the class of schemes under investigation is not the class of monadic recursion schemes studied in Friedman [9], whose equivalence problem has also be shown interreducible with EQ(DPDA) in [9]. For this second class, the conditional operator *if then else* has a *fixed* (natural) interpretation, whereas in our class of schemes, it is treated as an ordinary ternary function symbol whose interpretation may be any continuous function. Hence the equivalence relation on these (syntactically similar) program schemes is different.

A system $\Sigma = \langle \varphi_i(v_1, \ldots, v_{n_i}) = \alpha_i;\ 1 \leqq i \leqq N \rangle$ is *trim* if the following conditions are satisfied for all i, $1 \leqq i \leqq N$:

(1) $\alpha_i^{\nabla} \in CT(F, V)$, that is, α_i^{∇} has no occurrences of $\perp$

(2) each variable v_j, $1 \leqq j \leqq n_i$, has some occurrence in α_i^{∇}

(3) α_i is either in $T(\Phi, V)$, or it is of the form $f(t_1, \ldots, t_k)$ for some f of arity k in F and some $t_1, \ldots, t_k \in T(\Phi, V)$.

A program scheme $S = (\Sigma, \tau)$ is *trim* if Σ is. It is shown in Courcelle [7] that for every program scheme $S = (\Sigma, \tau)$ such that τ^{∇} has no occurrence of $\perp$, one can construct an equivalent trim scheme S'. Actually, one can construct a program scheme S' which is trim and in *Greibach normal form*, that is, such that each α_i is rooted with a function symbol in F. However, this stronger result will not be needed. We also say that a system Σ is *weakly trim* if condition (3) is not required.

A system Σ as above can also be considered as a term rewriting system $\{\varphi_i(v_1, \ldots, v_{n_i}) \to \alpha_i \mid 1 \leqq i \leqq N\}$. In this case, we use the notations $\underset{\Sigma}{\Rightarrow}$ and $\underset{\Sigma}{\Rightarrow}^*$ for the associated binary relations on $T(F \cup \Phi, V)$.

The constructions of Courcelle and Gallier [4, 5, 10] are used to investigate infinite trees by means of languages. In this paper, only the *branch language* of [5] will be used and not the *address language* of [10]. We shall sometimes impose a further restriction on our systems, namely:

(2.1.1) $\qquad r(f), r(\varphi) \geqq 1, \qquad$ for all $f \in F$, $\varphi \in \Phi$.

It has been shown in [5, Prop. 5.11] that the equivalence problem for arbitrary RPS's reduces to the equivalence problem for RPS's satisfying (2.1.1).

With every ranked alphabet F (or Φ) satisfying (2.1.1), one associates the new alphabet

$$\bar{F} = \{ f_i \mid f \in F,\ 1 \leqq i \leqq r(f) \}.$$

The *branch language* $Brch(t)$ of a tree $t \in CT(F, V)$ is a subset of $\bar{F}^* V$ defined as follows. If $t \in CT(F, V)$, then

$$Brch(t) = \{ t(i_1)_{i_1} t(i_1 i_2)_{i_2} \ldots t(i_1 i_2 \ldots i_{n-1})_{i_{n-1}} t(i_1 i_2 \ldots i_n) \mid i_1 i_2 \ldots i_n \in dom(t),\ n \geqq 1,\ t(i_1 i_2 \ldots i_n) \in V \}.$$

Note that only tree adresses $i_1 i_2 \ldots i_n \in dom(t)$ ending in a *leaf node* (labeled with a variable) are considered. For this reason, the language $Brch(t)$ does not "represent" the entire tree t if t has some subtree with no leaf. However, if t is *locally finite*, that is, if every node belongs to some finite branch going from the root to some *leaf*, then $Brch(t)$ characterizes t completely. Indeed, for any two locally finite trees t and t',

$$t = t' \quad \text{if and only if} \quad Brch(t) = Brch(t').$$

For example, if t is the infinite tree

$$t = f(v_1, g(v_2, g(v_2, g(v_2, \ldots)))),$$

then

$$Brch(t) = \{ f_1 v_1,\ f_2 g_1 v_2,\ f_2 g_2 g_1 v_2,\ f_2 g_2 g_2 g_1 v_2, \ldots \} = \{ f_1 v_1 \} \cup f_2 g_2^* g_1 v_2.$$

The infinite tree associated with a trim RPS S satisfying (2.1.1) is always locally finite, hence the equivalence for two RPS's S and S' reduces to the equality of the branch languages $Brch(S^\nabla)$ and $Brch(S'^\nabla)$.

We now review the different ways of defining branch languages by context-free grammars or automata.

2.2. Branch languages and strict deterministic grammars

The first technique used to define branch languages [4, 5] is to construct a strict deterministic grammar $Gram(S)$ for a given RPS $S = (\Sigma, \tau)$.

Let $\Sigma = \langle \varphi(v_1, \ldots, v_{r(\varphi)}) = \alpha_\varphi;\ \varphi \in \Phi \rangle$ and $\tau \in T(\Phi, V_k)$. We let $Gram(S)$ be the context-free grammar with

the set $\bar{F} \cup V_k$ of terminal symbols,

the set $\bar{\Phi} \cup \{\tau_0\}$ of nonterminal symbols,

the following set of production rules:

$\tau_0 \to u$, for all $u \in Brch(\tau)$

$\varphi_i \to u$, for all $\varphi_i \in \bar{\Phi}$, all $u \in (\bar{F} \cup \bar{\Phi})^*$, such that $uv_i \in Brch(\alpha_\varphi)$.

This grammar is π-complete deterministic for the partition

$$\{\{\tau_0\}, \{v_1\}, \ldots, \{v_k\}\} \cup \{[f] \mid f \in F \cup \Phi\},$$

where $[f]$ denotes $\{f_1, \ldots, f_{r(f)}\}$, for all $f \in F \cup \Phi$. Hence it is strict deterministic and $Brch(S^\nabla) = L(Gram(S))$ is a deterministic context-free language.

Example 2.2.1: Let $S = (\Sigma, \tau)$ be the following RPS:

$$\tau = \varphi(v_1, \psi(v_2))$$
$$\varphi(v_1, v_2) = f(v_1, \varphi(v_2, \varphi(\psi(v_1), v_2)))$$
$$\psi(v_1) = g(v_1, \psi(v_1)).$$

$Gram(S)$ is the following grammar with axiom τ_0 (using the abbreviation $A \to \alpha_1 \mid \ldots \mid \alpha_n$ for grouping productions $A \to \alpha_1, \ldots, A \to \alpha_n$ with the same lefthand side A):

$$\tau_0 \to \varphi_1 v_1 \mid \varphi_2 \psi_1 v_2$$
$$\varphi_1 \to f_1 \mid f_2 \varphi_2 \varphi_1 \psi_1$$
$$\varphi_2 \to f_2 \varphi_1 \mid f_2 \varphi_2 \varphi_2$$
$$\psi_1 \to g_1 \mid g_2 \psi_1.$$

2.3. Construction of the reduced DPDA associated with a RPS

The second technique for defining $Brch(S^\nabla)$ is to construct a DPDA denoted as $DPDA(S)$, using a variant of the construction used in Gallier [10] adapted to branch languages.

2.3.1. *Construction of the DPDA associated with a RPS*

The basic idea of the construction of $DPDA(S)$ is to mimic the implementation of recursive calls using a pushdown store. Let $S = (\Sigma, \tau)$ be a weakly trim recursion scheme, where Σ consists of N recursive definitions $\varphi_i(v_1, \ldots,$

$v_{m_i}) = \alpha_i$. The construction uses the trees $\alpha_1, \ldots, \alpha_N, \tau$ as finite automata, as long as the current tree address is not labeled with a nonterminal or a variable. Reaching a node u labeled with a nonterminal φ_m in tree α_k, the "function" φ_m is called and a jump to the root of α_m is made, the "return address" (u, k) being saved on top of the stack. Upon completion of a "function call", that is when a leaf labeled v_i is reached, we "jump back" to the i-th successor of the node from where the call originated, this address being currently on top of the stack and now being popped.

Formally, $DPDA(S)$ is constructed as follows. To simplify the notation, let us assume that τ is also named by α_0 and that $\varphi_0(v_1, \ldots, v_{n_0}) = \tau$. A state is either a pair of the form (u, i), where i is the index of a tree α_i, $0 \leqq i \leqq N$, and u is a tree-address in α_i, or $accept_i$, or *stop*. The initial state is $(e, 0)$, the bottom of stack symbol is 0, and the other stack symbols are all "push states", that is, states of the form (u, k), where $\alpha_k(u)$ is a nonterminal.

(1) *read moves:* For each k, $0 \leqq k \leqq N$, for each tree address $u \in dom(\alpha_k)$ such that $\alpha_k(u) = f \in F$ (base functions), for each i s.t. $ui \in dom(\alpha_k)$, we have:

$$(u, k) \xrightarrow{\text{read } f_i} (ui, k).$$

(2) *push moves:* For each k, $0 \leqq k \leqq N$, tree address $u \in dom(\alpha_k)$ such that $\alpha_k(u) \in \Phi$, say $\alpha_k(u) = \varphi_i$, we have:

$$(u, k) \xrightarrow{\text{push}} (e, i).$$

(3) *pop moves:* For each i, $0 \leqq i \leqq N$, each tree address $v \in \text{dom}(\alpha_i)$ such that $\alpha_i(v)$ is a variable, say $\alpha_i(v) = v_j$, from state (v, i), for every (u, k) such that $\alpha_k(u) = \varphi_i$, there is a transition:

$$(v, i) \xrightarrow{\text{pop } (u, k)} (uj, k).$$

For each variable v_j, for every state of the form $(ui, 0)$ such that $\tau(ui)$ is the variable v_j, there are transitions

$$(ui, 0) \xrightarrow{\text{pop } 0} accept_j$$

and

$$ccept_j \xrightarrow{\text{read } v_j} stop \quad \square$$

The following examples illustrate the above construction. Since the schemes below consist of a single definition and $\tau = \varphi(x_1, x_2)$ in Example 2.3.1, $\tau = \varphi(x_1)$ in Example 2.3.2, there is no need for states corresponding to τ and we have simplified the construction by denoting the states by their tree address component u rather than the pair (u, i) since $i = 1$.

Example 2.3.1. Let S_1 be the RPS:

$$\varphi(v_1, v_2) = f(v_1, \varphi(\varphi(v_2, v_1), v_1)).$$

The DPDA D_1 accepting the language $Brch(S_1^{\nabla})$ by final state and empty store is shown in Figure 1. Note that input $(f_2 f_1)^n f_1 v_1$ causes the DPDA to enter a sequence of n pop moves.

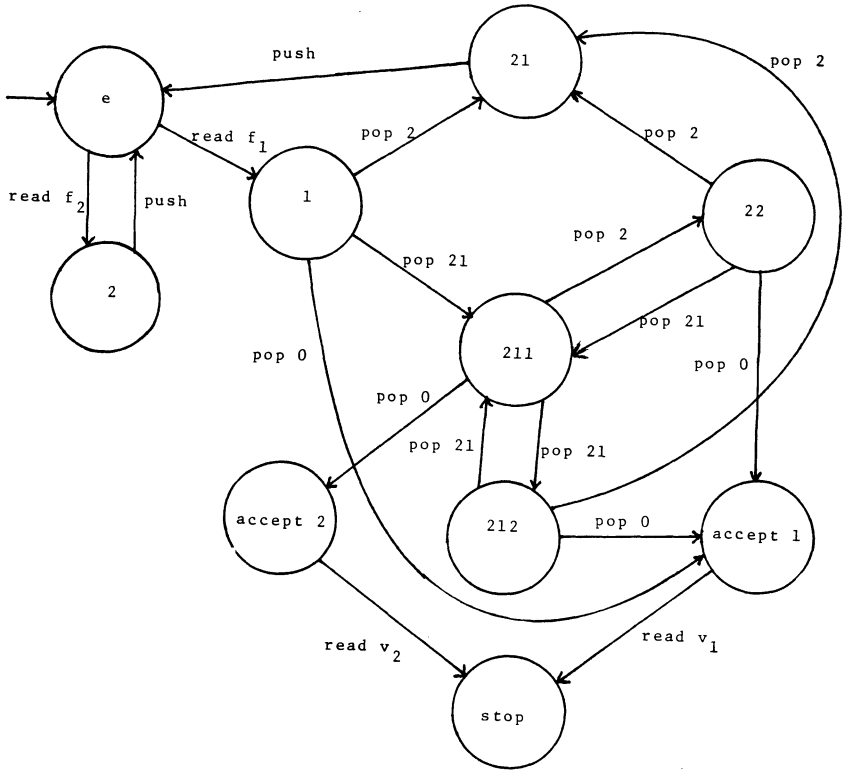

Figure 1

Example 2.3.2: Let S_2 be the RPS shown below:

$$\varphi(v_1) = f(v_1, \varphi(\varphi(v_1))).$$

The DPDA D_2 accepting the language $Brch(S_2^{\nabla})$ by final state and empty store is shown in Figure 2. Note that the DPDA D_2 enters a sequence of n pop moves on input $(f_2 f_1)^n f_1 v_1$. However, in the second example, note that the push moves in state 21 can be replaced by change state moves. Hence, D_2 can be transformed to the following equivalent e-limited DPDA shown in Figure 3.

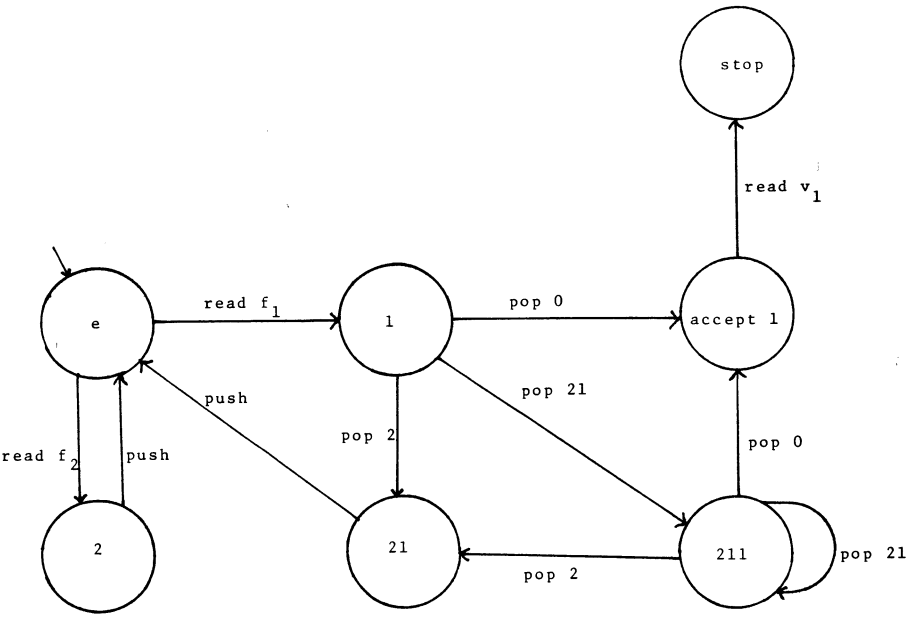

Figure 2

Instead of making a redundant push in state 21, control is passed directly to the entry of the function involved by performing a change state move from state 21 to state e, and state 211 and the transitions to and from it are eliminated. As we shall see later, it is not possible to construct an e-limited DPDA equivalent to D_1. However, the two previous examples illustrate the fact that our present construction performs redundant push moves for addresses corresponding to "tail-recursion". We will say than an occurrence u of a nonterminal φ in the right-hand side of a definition is a *tail-recursion* if the subtree rooted at u is of the form $\varphi(v_{i_1}, \ldots, v_{i_m})$ where $m = r(\varphi)$, the arity of φ. Note that redundant push moves corresponding to tail-recursions can also introduce unbounded sequences of pop moves. It is possible to eliminate these redundant push and pop moves by handling tail-recursion occurrences more carefully.

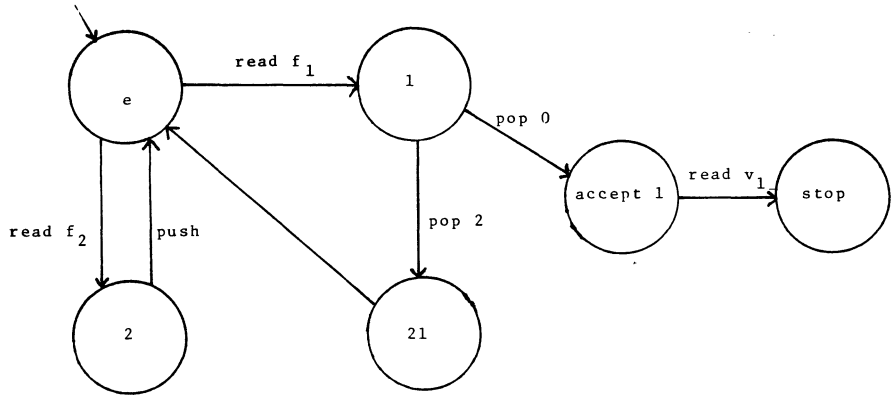

Figure 3

The construction below is a modification of our previous construction, and tail-recursion calls are handled using transfers instead of push moves. The resulting DPDA will be called the *reduced DPDA* associated with S and will be denoted as $RDPDA(S)$. Before giving the construction of $RDPDA(S)$, note the following fact which is the key to the construction.

If $\varphi_i = \alpha_i$ and $\varphi_j = \alpha_j$ are definitions with tail-recursion occurrences

$$\varphi_j(v_{\eta(1)}, \ldots, v_{\eta(n)}) \quad \text{in } \alpha_i$$

and

$$\varphi_k(v_{\theta(1)}, \ldots, v_{\theta(p)}) \quad \text{in } \alpha_j,$$

where $m = r(\varphi_i)$, $n = r(\varphi_j)$, $p = r(\varphi_k)$ and η and θ are functions

$$\eta: \{1, \ldots, n\} \to \{1, \ldots, m\}$$

and

$$\theta: \{1, \ldots, p\} \to \{1, \ldots, n\},$$

then we have $\varphi_i \underset{\Sigma}{\Rightarrow}^{+} t$, where t has the subtree

$$\varphi_k(v_{\eta(\theta(1))}, \ldots, v_{\eta(\theta(p))})$$

as a tail-recursion occurrence. Hence the indices of the variables in the tail-recursion occurrence φ_k in this last tree are given by the function

$$\theta \circ \eta: \{1, \ldots, p\} \to \{1, \ldots, m\}.$$

(Note that composition of functions is denoted from *left* to *right*, as in the diagrammatic order). It is also necessary to encode in the states when a non tail-recursion call is made. The full construction is now given.

2.3.2. *Construction of the Reduced DPDA RDPDA(S)*

Let $S=(\Sigma, \tau)$ be a scheme with Σ of the form:

$$\varphi_1(v_1, \ldots, v_{m_1})=\alpha_1$$

$$\ldots$$

$$\varphi_N(v_1, \ldots, v_{m_N})=\alpha_N$$

and assume that Σ is weakly trim. To simplify the notation, let us assume that τ is also named by α_0 and that $\varphi_0(v_1, \ldots, v_{n_0})=\tau$.

A state is either a quadruple of the form (u, h, η, i), or $accept_i$, or *stop*, where i is the index of a tree α_i, $0 \leqq i \leqq N$, u is a tree-address in α_i, h is the index of an ancestor of i in a sequence of calls, and η is a function, either the identity $I_m: \{1, \ldots, m\} \to \{1, \ldots, m\}$, or a function $\eta: \{1, \ldots, n\} \to \{1, \ldots, m\}$, with $n=r(\varphi_i)$, and $m=r(\varphi_h)$ [note that $\eta(k)$ is not necessarily equal to v_{i_k}].

The initial state is $(e, 0, I_{r(\varphi_0)}, 0)$, the bottom of stack symbol is 0, and the other stack symbols are all "push states", that is, states of the form (u, h, η, k), where $\alpha_k(u)$ is a nonterminal of positive arity and the subtree at u is not a tail-recursion.

(1) *read moves:* For each k, $0 \leqq k \leqq N$, for each tree address $u \in dom(\alpha_k)$ such that $\alpha_k(u)=f \in F$ (base functions), for each h, $0 \leqq h \leqq N$, for each i s.t. $ui \in dom(\alpha_k)$, we have:

$$(u, h, \eta, k) \xrightarrow{\text{read } f_i} (ui, h, \eta, k)$$

Note that for a trim scheme, $u=e$.

(2) *push moves:* For each k, $0 \leqq k \leqq N$, tree address $u \in dom(\alpha_k)$ such that $\alpha_k(u) \in \Phi$, say $\alpha_k(u)=\varphi_i$ and $r(\varphi_i)=p$, we have:

(i) If the subtree at u is *not* a tail-recursion then we have:

$$(u, h, \eta, k) \xrightarrow{\text{push}} (e, i, I_p, i)$$

(e must have $p > 0$).

(ii) The subtree at u is a tail-recursion occurrence $\varphi_i(v_{\theta(1)}, \ldots, v_{\theta(p)})$, where $p = r(\varphi_i)$. Assume that

$$\theta\colon \quad \{1, \ldots, p\} \to \{1, \ldots, n\}$$

and

$$\eta\colon \quad \{1, \ldots, n\} \to \{1, \ldots, m\},$$

where $p = r(\varphi_i)$, $n = r(\varphi_k)$, $m = r(\varphi_h)$. Then, we have a "change state" move:

$$(u, h, \eta, k) \to (e, h, \theta \circ \eta, i)$$

(3) *pop moves:* For each i, $0 \leqq i \leqq N$, each tree address $v \in dom(\alpha_i)$ such that $\alpha_i(v)$ is a variable, say $\alpha_i(v) = v_j$, from state (v, m, θ, i), for every (u, h, η, k) such that $\alpha_k(u) = \varphi_m$ and the occurrence of φ_m at u in α_k is not a tail-recursion, then there is a transition:

$$(v, m, \theta, i) \xrightarrow{\text{pop } (u, h, \eta, k)} (u\theta(j), h, \eta, k).$$

For each variable v_j, for every state of the form $(ui, 0, \eta, 0)$ such that $\tau(ui)$ is a variable and $\eta(i) = j$, there are transitions

$$(ui, 0, \eta, 0) \xrightarrow{\text{pop } 0} accept_j$$

and

$$accept_j \xrightarrow{\text{read } v_j} stop$$

(The bottom of stack symbol is 0). □

Example 2.3.3: The reduced DPDA accepting $Brch(S_1^{\nabla})$ by final state and empty store is shown in Figure 4. Since the scheme has a single definition and $\tau = \varphi(x_1, x_2)$, there is no need for states for τ and for simplicity, states are denoted by pairs (u, η) rather than quadruples (u, h, η, i) since $h = i = 1$. Furthermore, η is omitted when it is the identity. Note that input $(f_2 f_2 f_1)^n f_1 v_1$ still causes n consecutive pop moves.

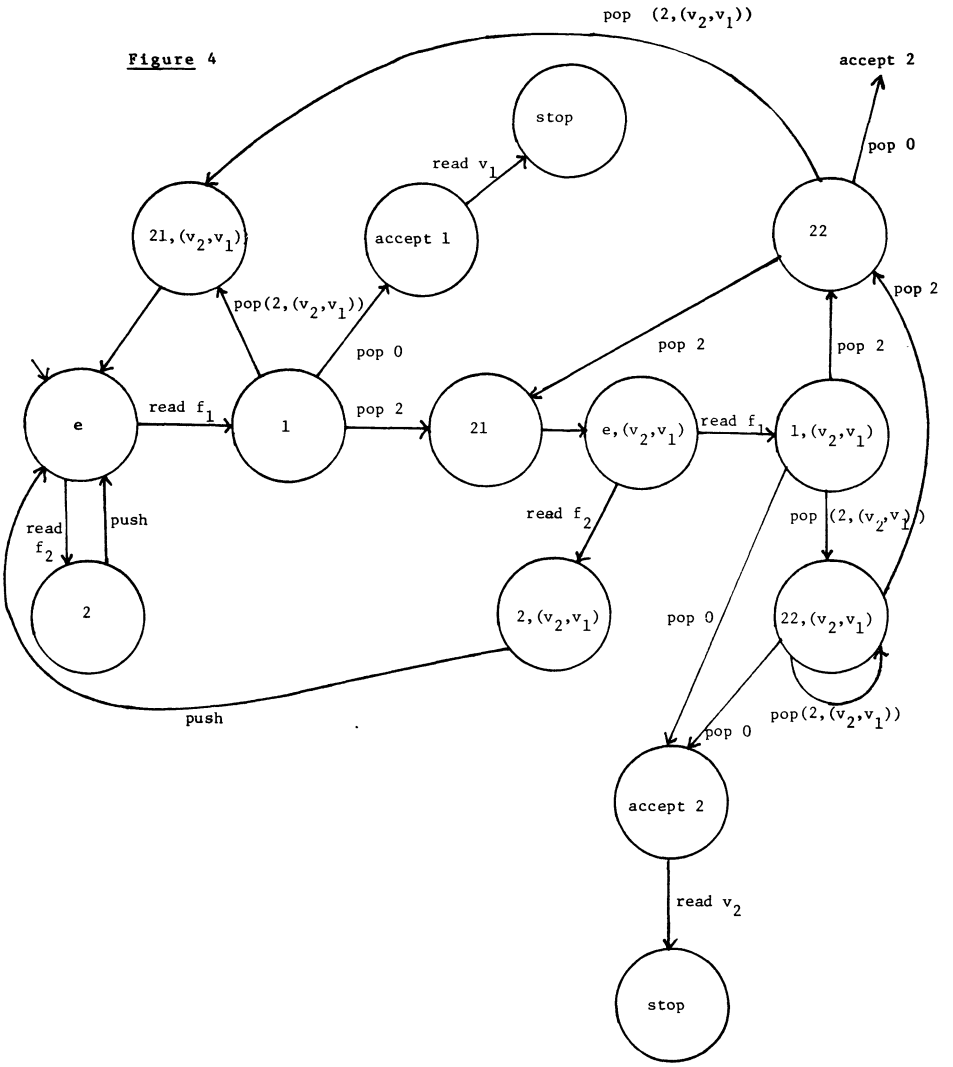

Figure 4

The above construction is now briefly justified.

Let us recall from Courcelle [7] that if (Σ, τ) is a scheme such that $\tau^{\nabla} \in CT(F, V)$, then τ^{∇} is the unique tree T in $CT(F, V)$ such that for every $w \in (\omega - \{0\})^*$, $T(w) = x \in F \cup V$ iff there exists some finite tree t such that,

$\tau \underset{\Sigma}{\Rightarrow^*} t$, $t(w)=x$, and $t(w') \in F \cup V$ for every prefix w' of w. It is also known that for every derivation $\tau \underset{\Sigma}{\Rightarrow^*} t$, there is an OI derivation $\tau \underset{OI}{\Rightarrow^*} t$ (Nivat [19]).

In order to state the following lemma, some definitions are needed. If

$$\bar{F}=\{f_i \mid f \in F,\ 1 \leqq i \leqq r(f)\} \cup \{a \mid a \in F,\ r(a)=0\},$$

let $label(f_i)=f$ and $path(f_i)=i$, with $label(a)=a$ and $path(a)=e$ for a constant a. The homomorphic extension of *path* to $\bar{F}^*$ is also denoted by *path*. Then, the correct behaviour of $RDPDA(S)$ is justified by the following lemma.

LEMMA 2.3.1 :*Given a weakly trim RPS* $S=(\Sigma, \tau)$, *for every computation*

$$((e, 0, I, 0), uv, 0) \vdash^* ((x, h, \eta, k), v, \gamma)$$
$$\vdash ((e, m, I, m), v, (x, h, \eta, k)\gamma) \vdash^{\otimes} ((y, m, \theta, i), e, (x, h, \eta, k)\gamma)$$
$$\vdash ((x\theta(j), h, \eta, k), e, \gamma)$$

with $\gamma \in \Phi^*$, $\alpha_k(x)=\varphi_m$ and $\alpha_i(y)=v_j$, and where the stack has $(x, h, \eta, k)\gamma$ as a prefix during the subcomputation $\otimes$, there is a derivation

$$\tau \underset{\Sigma}{\Rightarrow^*} \beta[path(u) \leftarrow \varphi_m(t_1, \ldots, t_n)] \underset{\Sigma}{\Rightarrow^+} \beta[path(u) \leftarrow \delta(t_1, \ldots, t_n)]=T,$$

for some tree β, and the subtree at $path(v)$ in $\delta(t_1, \ldots, t_n)$ is $t_{\theta(j)}$ and, for every prefix $w\sigma$ of uv, where $\sigma \in \bar{F}$, $T(path(w))=label(\sigma)$.

The proof is obtained as a simple modification of Lemma 3.3 in Gallier [10]. □

COROLLARY 2.3.2: *For any trim scheme S satisfying* (2.1.1), $RDPDA(S)$ *accepts the language* $Brch(S^\nabla)$ *by empty store and final state. Furthermore,* $RDPDA(S)$ *does not have push or pop moves corresponding to tailrecursion occurrences.* □

Note that we can obtain a stronger result if, instead of considering the branch language $Brch(S^\nabla)$, we consider the language $PBrch(S^\nabla)$ which is defined as follows:

$$PBrch(S^\nabla)=\{u \in \bar{F}^* \cup \bar{F}^* V \mid u \text{ codes any prefix of any finite or infinite branch of } S^\nabla\}.$$

It is no longer necessary to require S^∇ to be locally finite since for every two

trees t and t' in $CT(F, V)$:

$$t = t' \quad \text{if and only if} \quad PBrch(t) = PBrch(t').$$

Note that if S^∇ is locally finite then $PBrch(S^\nabla)$ is the prefix-closure of $Brch(S^\nabla)$. Furthermore, (2.1.1) can be relaxed, but we can not require acceptance by empty store. It is necessary to add the following moves:

If $p = r(\varphi_i) = 0$, then we have the change state move:

$$(u, h, \eta, k) \to (e, i, I_p, i)$$

We also need read moves

$$(u, h, \eta, k) \xrightarrow{\text{read } a} stop$$

for every constant a, and every $u \in dom(\alpha_k)$, $0 \leqq k \leqq N$, with $\alpha_k(u) = a$.

COROLLARY 2.3.3: *The language $PBrch(S^\nabla)$ is accepted by final state by $RDPDA(S)$ with all states final.* □

2.4. Branch languages and context-free tree grammars

The third method for defining $Brch(S^\nabla)$ uses a context-free tree grammar and was introduced in [3].

Given an RPS $S = (\Sigma, \tau)$, the tree grammar $Tgram(S)$ consists of the following components:

the terminal alphabet F, each symbol being considered as a monadic function symbol,

the set of variables V_n, with $n = Max\{r(\varphi) \mid \varphi \in \Phi\}$,

the nonterminal alphabet Φ (with no change in arities),

the axiom τ,

the following set P of productions:

(2.4.1) $\varphi(v_1, \ldots, v_{r(\varphi)}) \to \alpha$, if $\alpha \in T(\Phi, V)$,

(2.4.2) $\varphi(v_1, \ldots, v_{r(\varphi)}) \to f_i(t_i)$, for i, $1 \leqq i \leqq r(f)$, if $\alpha = f(t_1, \ldots, t_{r(f)})$, where $f \in F$.

The tree language $L(Tgram(S))$, that is, the set of trees $t \in T(\bar{F}, V)$ such that $\tau \underset{P}{\Rightarrow}^* t$, can be identified with a subset of $\bar{F}^* V_n$, since $\bar{F}$ consists only of monadic symbols. This subset is precisely $Brch(S^\nabla)$. □

The reader is referred to Engelfriet and Schmidt [8] for more details on context-free tree grammars and to Courcelle [3] for the above construction.

Let us only recall that *OI* (outside-in) derivations are sufficient to generate $L(Tgram(S))$.

The tree grammar $Tgram(S)$ can be converted into a jump-DPDA $JDPDA(S)$ accepting $Brch(S^{\nabla})$. The method consists in encoding the trees in $T(F, V)$ into stack words, so that a derivation of $Tgram(S)$:

$$\tau \underset{OI}{\Rightarrow} u_1(\tau_1) \underset{OI}{\Rightarrow} u_2(\tau_2) \underset{OI}{\Rightarrow} \ldots \underset{OI}{\Rightarrow} u_{n-1}(\tau_{n-1}) \underset{OI}{\Rightarrow} u_n(v_j),$$

where $u_1, \ldots, u_n \in \bar{F}^*$, $\tau_1, \ldots, \tau_{n-1} \in T(\Phi, V_k)$, is represented by a computation of $JDPDA(S)$ which recognizes the word $u_n u_j$ with successive stack contents: $m, m_1, m_2, \ldots, m_n$, which are the respective encodings of $\tau, \tau_1, \ldots, \tau_{n-1}, v_j$.

Example 2.4.1: The tree grammar $Tgram(S)$ for the RPS of example 2.2.1 is given below:

The axiom is $\varphi(v_1, \psi(v_2))$ and the rules are:

$$\varphi(v_1, v_2) \to f_1(v_1)$$
$$\varphi(v_1, v_2) \to f_2(\varphi(v_2, \varphi(\psi(v_1), v_2))$$
$$\psi(v_1) \to g_1(v_1)$$
$$\psi(v_1) \to g_2(\psi(v_1))$$

The string $f_2 f_1 g_2 g_1 v_2$ can be derived as follows:

$$\tau \underset{OI}{\Rightarrow} f_2(\varphi(\psi(v_2), \varphi(\psi(v_1), \psi(v_2)))) \underset{OI}{\Rightarrow} f_2 f_1(\psi(v_2))$$
$$\underset{OI}{\Rightarrow} f_2 f_1 g_2(\psi(v_2)) \underset{OI}{\Rightarrow} f_2 f_1 g_2 g_1(v_2). \square$$

Hence, each of the three above methods for defining $Brch(S^{\nabla})$ shows that $EQ(RPS) \leqq EQ(DPDA)$. We shall use these methods to find classes of RPS's whose equivalence problem is decidable. In the remainder of this section, the proof of the reduction $EQ(DPDA) \leqq EQ(RPS)$ given by Courcelle [4, 5] is sketched..

2.5. Reducing $EQ(DPDA)$ to $EQ(RPS)$

The method consists in constructing for every DPDA A (accepting by empty store) a RPS $S_A = (\Sigma_A, \varphi_A(v_1))$ such that, for any two DPDA's A and A', $N(A) = N(A')$ if and only if $S_A \equiv S_{A'}$. The construction is performed in several steps.

STEP 1: Given a DPDA A recognizing a deterministic language L, one constructs a faithful DPDA A_1 such that $L=N(A)=N(A_1)$, using lemma 1.1.

The next step requires the following definitions. For any finite alphabet X, let

$$\tilde{X}=\{(a, s) \mid a\in s,\ s\subseteq X,\ s\neq\varnothing\},$$

and let π be the partition of $\tilde{X}$ whose blocks are the sets $\{(a, s) \mid a\in s\}$, for all nonempty subsets $s\subseteq X$. To any prefix-free language L over X corresponds the prefix-free language L over X defined as follows:

$$\tilde{L}=\{(a_1, s_1)(a_2, s_2)\ldots(a_n, s_n) \mid a_1 a_2\ldots a_n\in L,$$
$$s_i=\{b\in X \mid a_1 a_2\ldots a_{i-1} b X^*\cap L\neq\varnothing\} \text{ for all } i,\ 1\leqq i\leqq n\}.$$

Note that for any two prefix-free languages L, L', $L=L'$ if and only if $\tilde{L}=\tilde{L}'$. It is assumed that X is totally ordered. The ranked alphabet $F=\{\bar{s} \mid s\subseteq X,\ s\neq\varnothing\}$, with $r(\bar{s})=Card(s)$, is defined so that $\tilde{X}$ is in bijection with $\bar{F}$: (a, s) corresponds to that element $\bar{s}_i$ such that a is the i-th element of s in the ordering chosen for X.

It follows from [4] (Prop. 5.10) that for every prefix-free π-complete language L' over $\tilde{X}=\bar{F}$, there exists a locally finite tree $t_{L'}$ in $CT(F, \{v_1\})$ such that $Brch(t_{L'})=L' v_1$.

STEP 2: Given a faithful DPDA A_1 accepting $L\subseteq X^*$, one can construct a faithful DPDA A_2 accepting $\tilde{L}$.

For a proof, see [4], Theorem 2.14.

STEP 3: Given A_2 as above such that $N(A_2)=\tilde{L}$, one constructs the canonical grammar G_{A_2}. Let T be its axiom. This grammar is $\pi\cup\pi'$-complete deterministic and $\{T\}$ is a block of π' (where π' is a partition of the nonterminal alphabet of G_{A_2}).

For a proof, see [4], Theorem 3.20.

STEP 4: Let G_{A_2} be as above. Let T' be a new nonterminal symbol, let G'_{A_2} be G_{A_2} augmented with the rule $T'\to T v_1$ and with T' as axiom. There exists a RPS $S=(\Sigma, \varphi(v_1))$ with set of base function symbols F such that $Gram(S)$ is isomorphic to G'_{A_2} via the bijection $\bar{F}\to\tilde{X}$ defined above and some bijection of the nonterminal alphabets.

For a proof, see [4], Theorem 5.12.

It follows that

$$Brch(\varphi(v_1)^\nabla)=L(G'_{A_2})=L(G_{A_2})v_1=\tilde{L}v_1=N(A)v_1.$$

(up to the bijection between $\bar{F}$ and $\tilde{X}$).

Let us denote by S_A the RPS $(\Sigma_A, \varphi_A(v_1))$ obtained from A using the above construction. Then, for any two DPDA's A and A' (over the same alphabet X), we have

$$S_A \equiv S_{A'} \quad \text{if and only if} \quad N(A)=N(A'),$$

establishing the reduction $EQ(DPDA) \leqq EQ(RPS)$. □

It is shown in Gallier [10] that $EQ(\text{LR}(1)\text{-parsers}) \leqq EQ(RPS)$, using a different method. Since every DPDA is equivalent to an LR(1)-parser, an alternate proof of the above reduction is obtained.

3. RECURSIVE PROGRAM SCHEMES AND REALTIME STRICT DPDA's

The class of realtime strict DPDA's is particularly interesting because it is decidable whether $N(A)=N(B)$, where A belongs to the class $\mathscr{R}_0$ of realtime strict deterministic DPDA's (following Valiant's notation [24]) and B is an arbitrary DPDA (Oyamaguchi, Inagaki and Honda [21]).

The realtime strict deterministic languages have been characterized by Harrison and Havel [17] as the class of languages generated by the *realtime* strict deterministic grammars, that is, the strict deterministic grammars such that, whenever

$$A \to \alpha T, \qquad A' \to \alpha T' \beta \qquad \text{and} \qquad A \equiv A' \quad \text{then} \quad \beta = e.$$

The class of RPS's corresponding to realtime strict deterministic grammars via the construction of $Gram(S)$ of 2.2 is given below. It is assumed that (2.1.1) holds.

DEFINITION 3.1: The subset $B(\Phi, V)$ of $T(\Phi, V)$ consisting of *balanced* trees is defined as follows: a tree $t \in T(\Phi, V)$ is *balanced* if either t is a variable in V, or $t=\varphi(t_1, \ldots, t_k)$ for some $\varphi \in \Phi$ of arity k and some $t_1, \ldots, t_k \in T(\Phi, V)$ such that, either all or none of the t_i are in V.

An RPS $S=(\Sigma, \tau)$ is *balanced* if each definition in Σ is of the form $\varphi(v_1, \ldots, v_k)=f(t_1, \ldots, t_m)$, with $t_1, \ldots, t_m \in B(\Phi, V)$, and $f \in F$ (Note that τ is not necessarily balanced). The class of balanced RPS's is denoted by $\mathscr{B}$.

Note that a balanced RPS is in Greibach normal form, and so it is necessarily proper (Courcelle [4,5]). From the definition of $Gram(S)$ and the

above, we have:

PROPOSITION 3.1: *Gram (S) is realtime strict deterministic if and only if S is balanced.* □

Example 3.1: Let $S=(\Sigma,\tau)$ where $\tau=\varphi(\psi(v_1,v_2),v_3)$ and Σ is:

$$\varphi(v_1,v_2)=f(v_1,v_2,\varphi(\psi(v_1,v_1),\varphi(v_2,v_1))$$
$$\psi(v_1,v_2)=g(v_1,\psi(v_2,v_1),\psi(\varphi(v_2,v_1),\psi(v_2,v_2)))$$

Then *Gram* (S) is the following grammar with axiom τ_0:

$$\tau_0\to\varphi_1\psi_1 v_1,\quad \tau_0\to\varphi_1\psi_2 v_2,\quad \tau_0\to\varphi_2 v_3$$
$$\varphi_1\to f_1,\quad \varphi_1\to f_3\quad \varphi_1\psi_1,\quad \varphi_1\to f_3\varphi_1\psi_2,\quad \varphi_1\to f_3\varphi_2\varphi_2$$
$$\varphi_2\to f_2,\quad \varphi_2\to f_3\varphi_2\varphi_1$$
$$\psi_1\to g_1,\quad \psi_1\to g_2\psi_2,\quad \psi_1\to g_3\psi_1\varphi_2$$
$$\psi_2\to g_2\psi_1,\quad \psi_2\to g_3\psi_1\varphi_1,\quad \psi_2\to g_3\psi_2\psi_1,\quad \psi_2\to g_3\psi_2\psi_2$$

Gram (S) generates $Brch(S^{\nabla})$.

COROLLARY 3.2: *It is decidable whether $S\equiv S'$ for any two schemes S, S', one of which is balanced.* □

This is abbreviated by saying that $EQ(\mathscr{B}:RPS)$ is decidable, where $EQ(\mathscr{C}:\mathscr{C}')$ denotes the problem of deciding whether $A\equiv B$, for any two classes $\mathscr{C}$ and $\mathscr{C}'$ of DPDA's or RPS's, and any $A\in\mathscr{C}$, $B\in\mathscr{C}'$. We also show that if a DPDA A is realtime strict, the scheme S_A obtained from steps 1-4 of the reduction $EQ(DPDA)\leqq EQ(RPS)$ given in 2.5 is a balanced scheme.

STEP 1: If A is realtime, so is A_1.

Proof: The construction of [4, lemma 1.1] does not introduce e-moves.

STEP 2: If A_1 is realtime, so is A_2.

Proof: The above remark also applies to the construction of [4], Theorem 2.14.

STEP 3: If A_2 is realtime, the grammar G_{A_2} is realtime strict deterministic for the partition π' of its nonterminal alphabet (A_2 accepts by empty store).

Proof: Straightforward (see also Harrison and Havel [17], Theorem 2.2).

STEP 4: If G_{A_2} is as above then, G'_{A_2} is also realtime and S_A is balanced.

Proof: By Proposition 3.1 above. □

Hence we have shown the following Theorem.

THEOREM 3.3: *A scheme S is balanced iff Gram (S) is realtime strict deterministic. If A is a realtime strict deterministic* DPDA, *then the scheme S_A is balanced. Consequently, the problems $EQ(\mathscr{R}_0 : DPDA)$ and $EQ(\mathscr{B} : RPS)$ are interreducible and decidable.* □

The reduction of $EQ(\mathscr{R}_0 : DPDA)$ to $EQ(\mathscr{B} : RPS)$ provided by the above construction might yield an alternate proof of the decidability of $EQ(\mathscr{R}_0 : DPDA)$ if a direct proof of the decidability of $EQ(\mathscr{B} : RPS)$ is found. This is left for further research.

Gallier's construction [10] is now used to improve the results of the previous section. The class $\mathscr{EB}$ of *extended balanced* schemes is obtained by relaxing condition (2.1.1) from the definition of the class $\mathscr{B}$. The class $\mathscr{B}'$ of *e-limited* schemes is defined in terms of a certain acyclicity condition and it is shown that $EQ(\mathscr{B}' : RPS)$ is decidable. The class $\mathscr{EB}'$ of *extended e-limited* schemes is obtained by relaxing condition (2.1.1) from the definition of the class $\mathscr{B}'$. The class $\mathscr{B}'$ contains properly the class $\mathscr{B}$ but it will also be shown that every scheme in $\mathscr{B}'$ is equivalent to a scheme in $\mathscr{B}$.

e-limited RPS's

The construction of the reduced DPDA associated with a scheme S given in section 2.3 can be used to study a class of schemes for which the reduced DPDA's obtained are e-limited. Since every e-limited DPDA can be transformed into an equivalent realtime DPDA, this reduces the equivalence problem for such schemes to the equivalence problem for realtime strict DPDA's, which is known to be decidable [21]. Such schemes, will be called *e-limited schemes.* The following definitions are needed in order to define e-limited schemes.

DEFINITION 3.2: Let $S = (\Sigma, \tau)$ be a trim scheme, with $\Sigma = (\varphi_1(v_1, \ldots, v_{n_1}) = \alpha_1, \ldots, \varphi_N(v_1, \ldots, v_{nN}) = \alpha_N)$. The *tail-recursion graph TL* is defined as follows: The nodes of *TL* are all the nonterminals in Φ having a tail-recursion occurrence in some α_i, and there is an edge from φ_i to φ_j if and only if φ_j has a tail-recursion occurrence in α_i. The graph G_Σ is defined as follows: Its nodes are all the nonterminals φ_i having some occurrence which is not a tail-recursion, and there is an edge from φ_i to φ_j if and only if either the rule for φ_i is of the form

$$\varphi_i(v_1, \ldots, v_{n_i}) = \beta[u \leftarrow \varphi_j(t_1, \ldots, t_{k-1}, v_m, t_{k+1}, \ldots, t_n)],$$

where not all $t_1, \ldots, t_{k-1}, t_{k+1}, \ldots, t_n$ are variables (hence $\varphi_j(t_1, \ldots, t_{k-1}, v_m, t_{k+1}, \ldots, t_n)$ is not balanced) or,

$$\varphi_i(v_1, \ldots, v_{n_i}) = \beta[u \leftarrow \varphi_{i'}(v_{i_1}, \ldots, v_{i_p})],$$

where $\varphi_{i'}$ is a tail-recursion occurrence, there is a (possibly null) path in *TL* from $\varphi_{i'}$ to some $\varphi_{j'}$, and the rule for $\varphi_{j'}$ is

$$\varphi_{j'}(v_1, \ldots, v_{n_{j'}}) = \delta[v \leftarrow \varphi_j(t_1, \ldots, t_{k-1}, v_m, t_{k+1}, \ldots, t_n)],$$

where not all t's are variables.

DEFINITION 3.3: A system Σ is *e-limited* if the graph G_Σ is *acyclic*. The class of *e*-limited schemes is denoted by $\mathscr{B}'$, and the class $\mathscr{EB}'$ of *extended e-limited* schemes is obtained from $\mathscr{B}'$ by relaxing condition (2.1.1).

Remarks: It is obvious that every balanced scheme is *e*-limited, and there are *e*-limited schemes that are not balanced. For example, the scheme S_1:

$$\varphi(v_1) = f(v_1, \varphi(\varphi(v_1)))$$

is balanced, the scheme S_2:

$$\varphi(v_1, v_2) = f(v_1, \psi(\varphi(v_2, v_1), v_1))$$
$$\psi(v_1, v_2) = f(v_1, \psi(v_2, v_2))$$

is *e*-limited but not balanced, and the scheme S_3

$$\varphi(v_1, v_2) = f(v_1, \varphi(\varphi(v_2, v_1), v_1))$$

is not *e*-limited.

However, it can be shown that S_2 is equivalent to a balanced scheme. This property is true in general, as we shall prove shortly.

THEOREM 3.4: *If S is an e-limited trim scheme then, the reduced DPDA RDPDA (S) accepting Brch* (S^∇) *by empty store is e-limited.*

Proof: First, since S is trim it is proper, and so, every push move or change state move is followed by a read move (since the root of every tree is labeled by a symbol in F). Hence, push moves and change state moves cannot introduce unbounded chains of *e*-moves. To take care of *e*-moves, we proceed by contradiction. Assume that for every $n > 1$, there is a string uv such that

$$((e, 0, I, 0), uv, e) \vdash^* ((y_1, m_1, \eta_1, j_1), v, \gamma_1\, \delta) \vdash^n ((z, h, \theta, k), v, \delta),$$

where $\alpha_{j_1}(y_1)=v_{k_1}$, $|\gamma_1|=n$, and where the sequence of n moves is a sequence of pop moves. Then, since tail-recursion calls have been eliminated, we must have:

$$\gamma_1=(z_2, m_2, \eta_2, j_2)\,\gamma_2, \qquad \alpha_{j_2}(z_2)=\varphi_{m_1}$$

and

$$((y_1, m_1, \eta_1, j_1), v, (z_2, m_2, \eta_2, j_2)\,\gamma_2) \vdash ((z_2\,\eta_1(k_1), m_2, \eta_2, j_2), v, \gamma_2).$$

Let $z_2\,\eta_1(k_1)=y_2$, and let $\alpha_{j_2}(y_2)=v_{k_2}$. Similarly, for $1\leqq i\leqq n$, we obtain the transition

$$((y_i, m_i, \eta_i, j_i), v, (z_{i+1}, m_{i+1}, \eta_{i+1}, j_{i+1})\,\gamma_{i+1}) \vdash ((z_{i+1}\,\eta_i(k_i), m_{i+1}, \eta_{i+1}, j_{i+1}), v, \gamma_{i+1}),$$

with $\alpha_{j_i}(y_i)=v_{k_i}$, $y_{i+1}=z_{i+1}\,\eta_i(k_i)$, $\alpha_{j_{i+1}}(z_{i+1})=\varphi_{m_i}$ and $\gamma_{n+1}=e$.

But then, note that we have a path

$$\varphi_{m_n} \to \varphi_{m_{n-1}} \to \ldots \to \varphi_{m_2} \to \varphi_{m_1}$$

in the graph G_Σ. If $n>N$, this path must contain a cycle, contrary to the assumption that the graph is acyclic. Hence, the length of sequences of consecutive e-moves is bounded by N, showing that the DPDA is e-limited. □

COROLLARY 3.5: *The equivalence problem for e-limited schemes is decidable.*

Proof: S_1 is equivalent to S_2 if and only if $Brch(S_1^\nabla)=Brch(S_2^\nabla)$, if and only if the reduced DPDA's $RDPDA(S_1)$ and $RDPDA(S_2)$ are equivalent. By Theorem 3.4, $RDPDA(S_1)$ and $RDPDA(S_2)$ are e-limited, and so they are equivalent to realtime DPDA's D_1 and D_2. But the equivalence problem for realtime DPDA's is decidable [21, 23], which concludes the proof. □

Let $\mathscr{R}$ denote the class of realtime DPDA's accepting by final state. We can state the following result.

PROPOSITION 3.6:

$$EQ(\mathscr{E}\mathscr{B}) \leqq EQ(\mathscr{E}\mathscr{B}') \leqq EQ(\mathscr{R})$$

and

$$EQ(\mathscr{E}\mathscr{B}: RPS) \leqq EQ(\mathscr{E}\mathscr{B}': RPS) \leqq EQ(\mathscr{R}: DPDA).$$

Proof: The proof follows from Corollary 2.3.3 since, given a scheme S_1 in $\mathscr{E}\mathscr{B}'$, $RDPDA(S_1)$ accepts $PBrch(S_1^{\nabla})$ by final state, and the argument of Theorem 3.4 shows that $RDPDA(S_1)$ is e-limited. □

We also state the following result.

PROPOSITION 3.7:

$$EQ(\mathscr{R}) \leqq EQ(\mathscr{E}\mathscr{B}).$$

The proof is a modification of the proof of Theorem 3.3. It is omitted since it is very technical and does not bring any new ideas. □

M. Oyamaguchi has proved that $EQ(\mathscr{R}: DPDA)$ is decidable [23]. Hence, the equivalence problems mentioned in propositions 3.6 and 3.7 are all decidable. We conclude by proving that every scheme in $\mathscr{B}'$ is equivalent to a scheme in $\mathscr{B}$. Technically, we prove the following proposition.

PROPOSITION 3.8: *The classes $\mathscr{B}$ and $\mathscr{B}'$ define the same set of infinite trees.*

Proof: Let S be a scheme in $\mathscr{B}'$. The e-limited DPDA accepting $Brch(S^{\nabla})$ produced by Theorem 3.4 can be converted into an equivalent realtime DPDA A by Theorem 11.7.3 of Harrison [15]. The canonical grammar G_A is realtime strict deterministic ([4], Theorem 2.2) generating a π-complete language (for the canonical partition π on $\bar{F} \cup V$ whose equivalences classes are $\{\{v\} \mid v \in V\}$ and $\{\{f_1, \ldots, f_{r(f)}\} \mid f \in \bar{F}\}$. Proposition 3.14 of Courcelle [4] shows that G_A is π'-complete deterministic for a partition π' whose restriction to $\bar{F} \cup V$ is π. Hence, G_A is isomorphic to $Gram(S_A)$, by [5] (proof of Theorem 5.15) for some S_A, and S_A is balanced by Proposition 3.1. Since

$$Brch(S^{\nabla}) = N(A) = L(G_A) = L(Gram(S_A)) = Brch(S_A^{\nabla})$$

and S^{∇} is locally finite, $S^{\nabla} = S_A^{\nabla}$ (*see* Courcelle [5], Proposition 5.9). □

A direct transformation of a scheme in $\mathscr{B}'$ into an equivalent one in $\mathscr{B}$ using neither grammars nor DPDA's would be of interest. However, we do not know of such a transformation. We also conjecture that $\mathscr{E}\mathscr{B}$ and $\mathscr{E}\mathscr{B}'$ have the same power.

4. FINITE-TURN AND ORDERED PROGRAM SCHEMES

Our purpose in this section is to characterize the following class of program schemes:

$\mathscr{F}\mathscr{T}\mathscr{S}$ = the class of RPS's such that $JDPDA(S)$ is finite-turn.

Since $EQ(\mathscr{F}\mathscr{T}: DPDA)$ is decidable [22], the equivalence problem for a pair (S, S') of *RPS's* where S belongs to $\mathscr{F}\mathscr{T}\mathscr{S}$ is decidable. Conversely, we

shall give a direct construction of the reduction $EQ(\mathcal{FT}:DPDA) \leqq EQ(\mathcal{FTS}:RPS)$.

DEFINITION 4.1: The *finite-turn property* is defined for arbitrary computing devices using computation sequences. A *computation* γ is a sequence of *moves* $\gamma = \langle m_1, m_2, \ldots, m_k \rangle$. Each move m is of the form $ID \vdash ID'$, where ID and ID' are *instantaneous descritpions* of the machine, before and after the move (typically, an ID is a triple (q, u, β), where q is the current state, u the remaining input to be processed and β represents the storage configuration). For simplicity of notation, a computation is denoted in string form as $\gamma = m_1 \ldots m_k$. It is assumed that each move is either *increasing*, *decreasing* or *steady*. Correspondingly, given a move m we let $\bar{m} = +$ if m is increasing, $\bar{m} = -$ if m is decreasing and $\bar{m} = e$ if m is steady. A computation $\gamma = m_1 \ldots m_k$ has profile $\bar{\gamma} = \bar{m}_1 \ldots \bar{m}_k \in \{+, -\}^*$. This word can be written in a unique way as

$$\bar{\gamma} = -^{n_1} +^{n_2} -^{n_3} \ldots +^{n_{h-1}} -^{n_h},$$

with $n_1 \geqq 0$, $h \geqq 1$, $n_2, \ldots, n_{h-1} \geqq 1$, $n_h \geqq 0$. We say that the number of blocks of $+$'s in $\bar{\gamma}$ is the *number of ups* of γ. Each i such that $n_i \neq 0$ and $n_{i+1} \neq 0$ is called a *turn* of γ. The number of turns of γ is at most $h-1$ and at least $h-3$.

For a DPDA, a move $m = (q, au, Z\beta) \vdash (p, u, m\beta)$ defined by a transition $\delta(q, a, Z) = (p, m)$ is *increasing* if $|m| \geqq 2$, steady if $|m| = 1$ and decreasing if $|m| = 0$ (that is, $m = e$).

A DPDA is *finite-turn* if there exists a constant M such that every accepting computation γ [with starting ID (q_0, u, Z_0) and accepting ID (q, e, m)] makes at most M turns (equivalently, there is a constant M' such that every such computation has at most M' ups).

For a jump-DPDA, a move $(q, au, m) \vdash (q', u, m')$ induced by a rule $\delta(q, a, Z) = (q', E, T)$ is defined as *decreasing*. Hence, a jump-DPDA is finite-turn if and only if the DPDA that simulates it is finite-turn.

When defining finite-turn DPDA's we can require that all computations with starting ID (q_0, u, Z_0) (even "hopeless") make at most M turns, as shown in the following technical lemma.

LEMMA 4.1: *The equivalence problem for finite-turn DPDA's reduces to the equivalence problem for finite-turn DPDA's which accept by empty store and are faithful.*

Proof: Given a finite-turn DPDA A accepting L, one constructs a finite-turn DPDA A' which accepts L by empty store. By Lemma 1.1, A' can be

transformed into an equivalent faithful DPDA A'' accepting L by empty store and which is also finite-turn. Since for any two DPDA's A and B, $A \equiv B$ if and only *if* $A'' \equiv B''$, the proof is complete. □

DEFINITION 4.2: Let Σ be a trim algebraic system with set Φ of non-terminal symbols and set F of base function symbols. Let $R(\Sigma)$ be the set of rules $s \to u$, with s, u terms in $T(\Phi, V)$ defined as follows:

For every rule $\varphi(v_1, \ldots, v_k) = u$ in Σ:

If $u \in T(\Phi, V)$ then $\varphi(v_1, \ldots, v_k) \to u$ is in $R(\Sigma)$;

If $u = f(u_1, \ldots, u_h)$ for some $f \in F_h$, then each $\varphi(v_1 \ldots, v_k) \to u_i$, $1 \leqq i \leqq h$, is in $R(\Sigma)$ [note that $u_i \in T(\Phi, V)$].

The *OI*-derivation relation associated with $R(\Sigma)$ is denoted by $\Rightarrow$. Hence,

$$u \Rightarrow u' \quad \text{if and only if} \quad u = \varphi(u_1, \ldots, u_k),$$
$$u' = t[u_1/v_1, \ldots, u_k/v_k] \quad \text{for some } \varphi(v_1, \ldots, v_k) \to t \text{ in } R(\Sigma).$$

Each derivation step $u \Rightarrow u'$ as above can be viewed as a move $u \vdash u'$ of a machine. A move is defined as

decreasing if $t \in V_k$,

steady if $t \in \Phi(V)$, where $\Phi(V)$ denotes the set of terms of the form $\varphi(w_1, \ldots, w_k)$, with $\varphi \in \Phi$ and $w_i \in V$, and

increasing otherwise.

We will identify a derivation $u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k$ and the corresponding computation $(u_1 \vdash u_2)\ (u_2 \vdash u_3) \ldots (u_{k-1} \vdash u_k)$.

DEFINITION 4.3: A program scheme $S = (\Sigma, \tau)$ where Σ is trim and $\tau \in T(\Phi, V)$ is *finite-turn* if the number of turns in every derivation $u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k$ is uniformly bounded. We say that Σ is *finite-turn* if and only if $(\Sigma, \varphi(v_1, \ldots, v_{r(\varphi)}))$ is finite-turn for every $\varphi \in \Phi$.

The (easy) proof of the following lemma is legt to the reader.

LEMMA 4.2: *Let Σ be a trim algebraic system. If Σ is finite-turn then (Σ, τ) is finite-turn for every $\tau \in T(\Phi, V)$. Conversely, if (Σ, τ) is finite-turn and every non-terminal φ occurs in some computation starting from τ, then Σ is finite-turn.* □

In the sequel, we shall only consider algebraic systems that are trim and satisfy condition (2.1.1), that is, that $r(f) \geqq 1$ for all $f \in F \cup \Phi$.

Recall from section 2.4 that a jump-DPDA *JDPDA* (S) can be associated with any RPS S, by encoding the elements of $T(\Phi, V)$ as pushdown-store words. This can be done in such a way that the sequence of rewrite steps

$u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k$ corresponds to the computation steps of the DPDA via the encoding. The details of this construction can be found in Courcelle [3]. By examining this consstruction, one notices that the increasing, decreasing and steady moves of $JDPDA(S)$ correspond exactly to increasing, decreasing and steady steps of $\Rightarrow$ respectively. Hence, the following lemma holds.

LEMMA 4.3: *A trim RPS S is finite-turn if and only if JDPDA (S) is finite-turn.* □

Our next goal will be to give a decidable characterization of finite-turn algebraic systems.

DEFINITION 4.4: Let Σ be an algebraic system [trim and satisfying condition (2.1.1) according to our previous assumption]. Let φ be a non-terminal symbol of arity k. We say that φ is *rational* it, for every t such that $\varphi(v_1, \ldots, v_k) \Rightarrow^* t, t \in V_k \cup \Phi(V_k)$. If i is an integer, $1 \leqq i \leqq k$, φ is *i-rational* if, for every t such that $\varphi(v_1, \ldots, v_k) \Rightarrow^* t$ and $v_i \in \mathrm{Var}(t)$, then $t \in V_k \cup \Phi(V_k)$.

(We denote by $\Phi(V_k)$ *the set of terms of the form* $\varphi(v_{i_1}, \ldots, v_{i_m})$, $\varphi \in \Phi$, $m = r(\varphi)$, $1 \leqslant i_1, \ldots, i_m \leqslant k$).

Recall that $\bar{\Phi}$ denotes the set $\{\varphi_i \mid \varphi \in \Phi, 1 \leqq i \leqq k\}$, where $k = r(\varphi)$. If $t \in T(\Phi, V)$ and $v \in V$, $Above(t, v)$ denotes the set of symbols φ_i such that t has a subterm of the form $\varphi(t_1, \ldots, t_k)$ and $v \in Var(t_i)$. In other words, $Above(t, v)$ is the set of symbols φ_i such that there is a path from an occurrence of φ to a leaf labeled by v in the i-th subtree rooted at this occurrence of φ.

We say that t is *rational with respect to v* if φ is i-rational for every φ_i in $Above(t, v)$.

It is clear that φ is rational if and only if it is i-rational for every i, $1 \leqq i \leqq r(\varphi)$.

The following lemma is easily verified.

LEMMA 4.4: *A tree t is rational with respect to v if and only if every computation* $t \Rightarrow^* t'$ *such that* $v \in \mathrm{Var}(t')$ *has only steady and decreasing steps.* □

DEFINITION 4.5: Assume that $\psi(v_1, \ldots, v_n) \Rightarrow^* \varphi(t_1, \ldots, t_k)$, with $v_j \in Var(t_i)$. The pair (i, j) $(1 \leqq i \leqq k, 1 \leqq j \leqq n)$ is *iterable* for $(\psi(v_1, \ldots, v_n), \varphi(v_1, \ldots, v_k))$, if there exists a derivation

$$\varphi(v_1, \ldots, v_k) \Rightarrow^* \psi(s_1, \ldots, s_n),$$

such that $v_i \in Var(s_j)$.

An algebraic system Σ is *ordered* if there exists a preorder (a reflexive and transitive relation) $\leqq$ on the set Φ of nonterminals such that, for every rule of the form

$$\psi(v_1, \ldots, v_n) \to \varphi(t_1, \ldots, t_k) \text{ in } R(\Sigma):$$

(4.5.1) $\varphi \leqq \psi$ and every symbol $\theta \in \Phi$ occurring in $t_1, \ldots, t_k$ is such that $\theta < \psi$ (that is, $\theta \leqq \psi$ and $\psi \not\leqq \theta$).

(4.5.2) If $\varphi \cong \psi$ (that is, $\varphi \leqq \psi$ and $\psi \leqq \varphi$) and (i, j) is iterable for $(\psi(v_1, \ldots, v_n), \varphi(v_1, \ldots, v_k))$ then t_i is rational with respect to v_j.

The class of ordered schemes is denoted by $\mathcal{OR}$.

If we replace condition (4.5.2) by the following stronger (and simpler) condition.

(4.5.2′) If $\varphi \cong \psi$, every $\theta \in \Phi$ occurring in $t_1, \ldots, t_k$ is rational, we obtain *ultralinear* systems. The class of ultralinear schemes is denoted by $\mathcal{UL}$.

Most of the remainder of this section will be devoted to showing that $\mathcal{FTS} = \mathcal{OR}$. We start with the "easy" direction.

LEMMA 4.5: *A finite-turn algebraic system is ordered.*

Proof: Let Σ be an algebraic system. Since it is assumed trim, for every φ of arity k in Φ, for each i, $1 \leqq i \leqq k$, there exists a derivation $\varphi(v_1, \ldots, v_k) \Rightarrow^* v_i$. The preorder $\leqq$ on Φ is defined as follows:

$\varphi \leqq \psi$ if and only if:

there exists a derivation $\psi(v_1, \ldots, v_{r(\psi)}) \Rightarrow^* t$, where φ occurs in t.

The above condition is equivalent to the existence of a derivation of the form

$$\psi(v_1, \ldots, v_{r(\psi)}) \quad \Rightarrow^* \quad \varphi(t_1, \ldots, t_{r(\varphi)}).$$

We say that $\varphi \cong \psi$ if and only if $\varphi \leqq \psi$ and $\psi \leqq \varphi$, and that $\varphi < \psi$ if and only if $\varphi \leqq \psi$ but $\psi \not\leqq \varphi$.

We now prove that (4.5.1) and (4.5.2) hold. Let

$$\psi(v_1, \ldots, v_n) \to \varphi(t_1, \ldots, t_k)$$

be a rule in $R(\Sigma)$. By definition of $\leqq$, $\varphi \leqq \psi$. For every symbol $\theta \in \Phi$ occurring in $t_1, \ldots, t_k$, we also have $\theta \leqq \psi$. We have to show that $\psi \not\leqq \theta$. For the sake of contradiction, assume that $\theta(t'_1, \ldots, t'_h)$ is a subtree of t_i for some i and that there exists a derivation

$$\gamma: \quad \theta(v_1, \ldots, v_h) \quad \Rightarrow^* \quad \psi(u_1, \ldots, u_n).$$

In order to simplify the presentation, let us introduce the following notation: For a derivation

$$\gamma: \quad t_1 \Rightarrow t_2 \Rightarrow \ldots \Rightarrow t_n,$$

with $t_1, t_2, \ldots, t_n \in T(\Phi, V_k)$ and $\bar{u}$ a k-tuple in $T(\Phi, V)^k$, the derivation

$$t_1[\bar{u}] \Rightarrow t_2[\bar{u}] \Rightarrow \ldots \Rightarrow t_n[\bar{u}]$$

is denoted by $\gamma[\bar{u}]$.

For $w \in T(\Phi, V_k)$ and $\bar{t}=(t_1, \ldots, t_k) \in T(\Phi, V)^k$, $w[t_1, \ldots, t_k]$ is abbreviated by $w.t$ and $(u_1.\bar{t}, \ldots, u_n.t)$ by $\bar{u}.t$, where $\bar{u}=(u_1, \ldots, u_n)$. Since the operation . is associative, parentheses are omitted in expressions like $\bar{u}.\bar{w}.\bar{t}$. Note that $(\gamma[\bar{u}])[\bar{t}].\gamma[\bar{u}.\bar{t}]$.

Going back to the proof, we have a derivation:

$$\gamma': \quad \psi(v_1, \ldots, v_n) \underset{(1)}{\Rightarrow^*} \varphi(t_1, \ldots, t_k) \underset{(2)}{\Rightarrow^*} \theta(t'_1, \ldots, t'_h) \underset{\gamma[\bar{t}']}{\Rightarrow^*} \psi(u'_1, \ldots, u'_n),$$

where $u'_i = u_i[t'_1, \ldots, t'_h] = u_i[\bar{t}']$. It makes at least one turn since part (1) is increasing and part (2) contains decreasing steps [since $\theta(t'_1, \ldots, t'_h)$ is a subterm of t_i]. This derivation can be "iterated" as follows: one can build a derivation

$$\gamma'^n: \quad \psi(v_1, \ldots, v_n) \underset{\gamma'}{\Rightarrow^*} \psi[\bar{u}'] \underset{\gamma'[\bar{u}']}{\Rightarrow^*} \psi[\bar{u}'^2] \underset{\gamma'[\bar{u}'^2]}{\Rightarrow^*} \ldots \Rightarrow^* \psi[\bar{u}'^n].$$

The derivation γ'^n clearly makes at least n turns. But then, Σ is not finite-turn, a contradiction. Having established (4.5.1), let us now consider (4.5.2).

If (4.5.2) does not hold, then $\varphi \cong \psi$ and there is a θ_p in *Above* (t_i, v_j), where (i, j) is iterable in the derivation

$$\psi(v_1, \ldots, v_n) \Rightarrow \varphi(t_1, \ldots, t_k)$$

and θ is not p-rational.

There exists a tree s in $T(\Phi, V_k)$ but not in $\Phi(V_k) \cup V_k$ such that

$$\theta(v_1, \ldots, v_h) \Rightarrow^* s$$

and $v_p \in Var(s)$.

Hence the following computation makes at least one turn (for some appropriate δ'):

$$\delta'' : \quad \theta(v_1, \ldots, v_h) \underset{\delta}{\Rightarrow}^* s \underset{\delta'}{\Rightarrow}^* v_p.$$

Let us define computations as follows:

$$\eta_1 : \quad \psi(v_1, \ldots, v_n) \Rightarrow \varphi(\bar{t})$$
$$\eta_2 : \quad \varphi(v_1, \ldots, v_k) \Rightarrow^* \psi(\bar{u})$$
$$\eta_3 : \quad u_j \Rightarrow^* v_i,$$

abbreviating $(t_1, \ldots, t_k)$ as $\bar{t}$ and $(u_1, \ldots, u_n)$ as $\bar{u}$;

$$\eta_4 : \quad \varphi(v_1, \ldots, v_k) \Rightarrow^* v_i.$$

The term t_i has a subterm of the form $\theta(t'_1, \ldots t'_h)$, and we have computations:

$$\eta_5 : \quad t_i \Rightarrow^* \theta(\bar{t}') \qquad \text{where } \bar{t}' = (t'_1, \ldots, t'_h),$$
$$\eta_6 : \quad t'_p \Rightarrow^* v_j.$$

The computation

$$\gamma : \quad \psi(v_1, \ldots, v_n) \underset{\eta_1}{\Rightarrow}^* \varphi[\bar{t}] \underset{\eta_2[\bar{t}]}{\Rightarrow}^* \psi[\bar{u}.\bar{t}]$$

can be iterated and gives for all $m \geqq 1$:

$$\psi(v_1, \ldots, v_n) \Rightarrow^* \psi[(\bar{u}.\bar{t})^m].$$

Using $\eta_1[(\bar{u}.\bar{t})^m]$ followed by $\eta_4[\bar{t}.(\bar{u}.\bar{t})^m]$, we obtain:

$$\psi[(\bar{u}.\bar{t})^m] \Rightarrow^* t_i.(\bar{u}.\bar{t})^m.$$

Let γ' be the following computation:

$$t_i.(\bar{u}.\bar{t}) \underset{\eta_5[\bar{u}.\bar{t}]}{\Rightarrow} \theta[\bar{t}'.(\bar{u}.\bar{t})] \underset{\delta''[\bar{u}.\bar{t}]}{\Rightarrow}^* t'_p.(\bar{u}.\bar{t}) \underset{\eta_6[\bar{u}.\bar{t}]}{\Rightarrow}^* u_j.\bar{t} \underset{\eta_3[\bar{t}]}{\Rightarrow}^* t_i.$$

It makes at least one turn since δ'' does. Then, we get

$$t_i.(\bar{u}.\bar{t})^m \underset{\gamma'[(\bar{u}.\bar{t})^{m-1}]}{\Rightarrow}^* t_i.(\bar{u}.\bar{t})^{m-1} \underset{\gamma'[(\bar{u}.\bar{t})^{m-2}]}{\Rightarrow}^* \ldots \underset{\gamma'}{\Rightarrow}^* t_i.$$

which makes at least $m-1$ turns. Since m is arbitrary, S is not finite-turn, a contradiction. This shows that (4.5.2) holds, and completes the proof. □

In order to prove the converse, namely that $\mathcal{OR}$ is a subset of $\mathcal{FTS}$, we need some auxiliary definitions and lemmas.

DEFINITION 4.6: Given an ordered system Σ with preorder $\leqq$ and equivalence $\cong$ on Φ, let us define relations $\underline{\prec}$ and $\approx$ on $\bar{\Phi}$ as follows:

$\varphi_i \underline{\prec} \psi_j$ if and only if $\varphi \leqq \psi$ and there exists $t_1, \ldots, t_k$ such that $\psi(v_1, \ldots, v_n) \Rightarrow^* \varphi(t_1, \ldots, t_k)$ and $v_j \in Var(t_i)$;

$\varphi_i \approx \psi_j$ if and only if $\varphi_i \underline{\prec} \psi_j$ and $\psi_j \underline{\prec} \varphi_i$;

$\varphi_i \prec \psi_j$ if $\varphi_i \underline{\prec} \psi_j$ and $\psi_j \not\underline{\prec} \varphi_i$.

It is obvious that $\underline{\prec}$ is a preorder.

LEMMA 4.6: *Let $\psi(v_1, \ldots, v_n) \to t$ be a rule in $R(\Sigma)$, with $t = \varphi(t_1, \ldots, t_k)$.*

(1) *If $\varphi_i \in Above(t, v_j)$ then $\varphi_i \underline{\prec} \psi_j$. If $\theta_p \in Above(t_i, v_j)$ for some i, $1 \leqq i \leqq k$ then $\theta_p \prec \psi_j$.*

(2) *If $\varphi_i \approx \psi_j$ then t_i is rational w. r. t. v_j.*

Proof: (1) If $\varphi_i \in Above(t, v_j)$ then

$$\psi(v_1, \ldots, v_n) \quad \Rightarrow \quad \varphi(t_1, \ldots, t_k),$$

with $v_j \in Var(t_i)$. By (4.5.1), $\varphi \leqq \psi$ and so $\varphi_i \underline{\prec} \psi_j$.

If $\theta_p \in Above(t_i, v_j)$ then

$$\psi(v_1, \ldots, v_n) \quad \Rightarrow^* \quad \theta(u_1, \ldots, u_n),$$

with $v_j \in Var(u_p)$. By (4.5.1), $\theta < \psi$ and so $\theta_p \underline{\prec} \psi_j$. Now, if $\theta_p \approx \psi_j$ then $\theta \cong \psi$, which would contradict (4.5.1). Hence $\theta_p \prec \psi_j$.

(2) Assume $\varphi_i \approx \psi_j$. This implies $\varphi \cong \psi$. In order to apply (4.5.2) it is sufficient to show that (i, j) is iterable for $(\psi(v_1, \ldots, v_n), \varphi(v_1, \ldots, v_k))$. But this follows from the definition of $\underline{\prec}$ and $\approx$. □

Let $\gamma : t_1 \Rightarrow t_2 \Rightarrow \ldots \Rightarrow t_n$ be a computation. Let $U(\gamma)$ denote the number of ups in γ. For $t \in T(\Phi, V)$ and $v \in V$, let $U(t, v)$ denote the least upper

bound of the numbers $U(\gamma)$ for all computations $\gamma : t \Rightarrow^* v$. This least upper bound may be $+\infty$, an integer or $-\infty$ if there is no computation as above [i. e. if $v \notin Var(t)$].

LEMMA 4.7: *$U(t, v) < +\infty$ for all $t \in T(\Phi, V)$ and $v \in V$.*

Proof: The result follows from the following observations:

Observation 1: $U(t, v) \leqq 0$ if t is rational w. r. t. v (this follows from Lemma 4.4).

Observation 2: $U(y, v) = 0$ if $y = v$, $U(y, v) = -\infty$ if $y \in V$, $y \neq v$.

Observation 3: Let us abbreviate $U(\varphi(v_1, \ldots, v_n), v_i)$ as $U(\varphi, i)$. Then,

$$U(\varphi(t_1, \ldots, t_n), v) = \text{Max}\{ U(\varphi, i) + U(t_i, v) \mid 1 \leqq i \leqq n \},$$

with the following rules for $+$:

$$+\infty + v = v + \infty = +\infty, \qquad \text{for} \quad v \in \{0, 1, \ldots\} \cup \{+\infty\}$$

$$v + -\infty = -\infty + v = -\infty, \qquad \text{for} \quad v \in \{0, 1, \ldots\} \cup \{+\infty, -\infty\},$$

and $u + v$ is defined as usual otherwise.

Observation 4: $U(t, v) < +\infty$ if $U(\varphi, i) < +\infty$ for all $\varphi_i \in$ *Above* (t, v).

Observation 5: Let $\gamma : \psi(v_1, \ldots, v_n) \Rightarrow^* v_j$ be a computation and let $u_1, \ldots, u_n$ be rational w. r. t, x_h. Let γ' be a computation

$$\psi(u_1, \ldots, u_n) \quad \Rightarrow^* \quad u_j \quad \Rightarrow^* \quad v_h,$$

the first half of which being $\gamma[u_1, \ldots, u_n]$. Then, $U(\gamma') = U(\gamma)$. □

We will also need the following technical definitions.

DEFINITION 4.7: *Given an ordered system* Σ, a rule $\psi(v_1, \ldots, v_n) \to t$ in $R(\Sigma)$ is of *type* I if, for all θ occurring in t, $\theta < \psi$ (this is the case in particular if $t \in V_n$). It is of *type* II if $t = \varphi(t_1, \ldots, t_k)$ and $\varphi \cong \psi$ [*see* (4.5.2)].

For a computation γ, we define $U^+(\gamma)$ as $U(\gamma)$ if the profile of γ begins with $+$, and as $1 + U(\gamma)$ if the profile of γ is empty or begins with $-$. The effect of such a definition is that $U^+(m\gamma) = U^+(\gamma)$ if m is a steady or increasing move. $U^+(t, v)$ is defined as the least upper bound of the set of $U^+(\gamma)$'s such that some computation $\gamma : t \Rightarrow^* v$ exists, and as $-\infty$ otherwise.

LEMMA 4.8: *If a trim algebraic system* [*satisfying* (2.1.1)] *is ordered, then it is finite-turn.*

Proof: It is sufficient to show that $U(\varphi, i) < +\infty$, for all $\varphi_i \in \bar{\Phi}$.

The case of rational symbols is taken care of by Observation 1. Otherwise, it suffices to establish the statement for all the elements of a class $[\varphi_i]$ of the equivalence $\approx$, using it as an induction hypothesis assumed to hold for all ψ_j such that $\psi_j \prec \varphi_i$ (This will be called the *induction hypothesis*).

Let $\mathscr{C}$ be such a class and let $\psi_j \in \mathscr{C}$. Let a be the least upper bound of the set of numbers $U^+(t, v_j)$ for all t such that $\psi(v_1, \ldots, v_n) \to t$ is a rule of $R(\Sigma)$ of type I for some $\psi_j \in \mathscr{C}$. By observations 3 and 4 and by the induction hypothesis, $a < +\infty$. Let also b be the least upper bound of the set of values of the form

$$U^+(\varphi, i) + U(t_i, v_j),$$

for all $\varphi_i \prec \psi_j$ and all rules $\psi(v_1, \ldots, v_n) \to \varphi(t_1, \ldots, t_k)$ of $R(\Sigma)$ of type II for some $\psi_j \in \mathscr{C}$. Once again, the induction hypothesis, property (4.5.1) and observations 3 and 4 show that $b < +\infty$.

Claim: If $\psi_j \in \mathscr{C}$, then $U^+(\psi, j) \leqq Max\{a, b\}$.

Proof of Claim: Every computation $\psi(v_1, \ldots, v_n) \Rightarrow^* v_j$ starts with a certain number of applications of rules of type II. Let m be this number. We perform an induction on m (called the *inner induction*, as opposed to the induction on $\prec$).

Case $m = 0$:

$$\gamma: \quad \psi(v_1, \ldots, v_n) \quad \Rightarrow \quad t \quad \Rightarrow^* \quad v_j$$

and the first rule is of type I. But then, $U^+(\gamma) \leqq U^+(t, v_j) \leqq a$.

Case $m = m' + 1$:

$$\gamma: \quad \psi(v_1, \ldots, v_n) \quad \Rightarrow \quad \varphi(t_1, \ldots, t_k) \quad \Rightarrow^* \quad v_j.$$

The first part of γ is an application of a rule of type II and its second part can be written as:

$$\gamma'': \quad \varphi(t_1, \ldots, t_k) \quad \Rightarrow^* \quad t_i \quad \Rightarrow^* \quad v_j,$$

where the first computation is $\gamma'[t_1, \ldots, t_k]$, with γ' the computation $\varphi(v_1, \ldots, v_k) \Rightarrow^* v_i$ and the second computation is called γ''_1. There are two subcases:

(1) $\varphi_i \approx \psi_j$ and t_i is rational w.r.t. v_j. By Observation 5, $U(\gamma'') = U(\gamma')$ and similarly for U^+. On the other hand, $U^+(\gamma') \leqq Max\{a, b\}$, by the inner induction hypothesis. Hence,

$$U^+(\gamma) = U^+(\gamma'') \leqq Max\{a, b\}.$$

(2) $\varphi_i \prec \psi_j$. Then,

$$U^+(\gamma'') = U^+(\gamma'[t_1, \ldots, t_k]) + U(\gamma_1'') = U^+(\gamma') + U(\gamma_1'')$$
$$\leqq U^+(\varphi, i) + U(t_i, v_j) \leqq b.$$

Hence, $U^+(\gamma) = U^+(\gamma'') \leqq b$. This concludes the inner induction. □

This concludes the main induction and the proof of Lemma 4.8. □

The following is obtained as a corollary.

PROPOSITION 4.9: *A trim algebraic system is finite-turn if and only if it is ordered. Furthermore, this is a decidable property.*

Proof: The fact that $\mathscr{FTS} = \mathscr{OR}$ follows from Lemma 4.5 and Lemma 4.8. One can decide whether a symbol φ is rational w. r. t. v_i, and since there are only finitely many preorders on Φ, the existence of a preorder satisfying (4.5.1) and (4.5.2) can be tested. □

LEMMA 4.10: *Given a finite-turn* DPDA A, *the construction of section* 2.5 *yields a schme* S_A *in* $\mathscr{FTS}$.

Proof: Let us review how steps 1 to 4 of the reduction recalled in section 2.5 apply to finite-turn DPDA's. Let A be a finite-turn DPDA accepting the language L_0 by final state,. We can construct a faithul DPDA A_1 accepting the prefix-free language $L = L_0 \$$ by empty store and which is faithful, by combining the standard construction with Lemma 1.1 of [4]. These constructions preserve the finite-turn property of the DPDA's to which they are applied.

The construction of [4] recalled in step 2 of 2.5 yields a DPDA A_2 which is also finite-turn and such that $N(A_2) = \tilde{L}$. In steps 3 and 4 of 2.5, we recall how a RPS $S_A = (\Sigma_A, \varphi_A(v_1))$ can be constructed from A_2 in such a way that $Brch(S_A^{\nabla}) = \tilde{L} v_1$. Our next goal is to establish that S_A is finite-turn.

Let γ be a computation of S_A:

$$\varphi(v_1) \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_m \Rightarrow v_1.$$

By the method in which Σ_A is associated with A_2, every right-hand side t in a rule $s \to t$ in $R(\Sigma_A)$ is a tree all branches of which have the same length denoted by $h(t)$. It follows that $u_1, u_2, \ldots, u_m$ have the same property (by an easy induction on m). Moreover, a unique computation γ' of A_2 of the form

$$(q_0, x, Z_0) \vdash (q_1, x_1, w_1) \vdash \ldots \vdash (q_m, x_m, w_m) \vdash (q', e, e)$$

corresponds to γ, and $|w_i| = h(u_i) - 1$ for all i, $1 \leqq i \leqq m$.

Hence, γ and γ' have exactly the same profile. Since A_2 is finite-turn, so is S_A. □

If one starts with two DPDA's A and B, where A is finite-turn and B is arbitrary, the above constructions reduce the equality $L(A) = L(B)$ to the equality $S_A^{\nabla} = S_B^{\nabla}$, that is, to the equivalence of two RPS's one of which (S_A) is finite-turn.

THEOREM 4.11: *The equivalence problem* $EQ(\mathscr{FTS} : RPS)$ *is decidable. Futhermore, given a finite-turn DPDA A, the construction of section* 2.5 *yields a scheme* S_A *which is finite-turn.*

Proof: By Lemma 4.3, $EQ(\mathscr{FTS} : RPS) \leqq EQ(\mathscr{FT} : \text{DPDA})$. The decidabity of $EQ(\mathscr{FT} : DPDA)$ has been shown in Oyamaguchi *et al.* [22, Theorem 2]. The second part of the Theorem is Lemma 4.10. □

COROLLARY 4.12: *The equivalence problem for ultralinear algebraic systems is decidable.* □

Open Question: Find a direct transformation that, given a finite-turn DPDA's A yields an ultralinear system S_A such that, for any two finite-turn DPDA's A and B, $L(A) = L(B)$ iff $S_A \equiv S_B$.

We shall now improve corollary 4.12 by introducing the class of *quasi-ultralinear* systems which properly includes the class of ultralinear systems but is actually not stronger (in the sense that for every quasi-ultralinear system, there is an equivalent ultralinear system).

DEFINITION 4.8: *Let* Σ be a trim algebraic system with set of nonterminals Φ. An element $\varphi \in \Phi$ is *bounded* if the set of $t \in T(\Phi, V_k)$ such that

$$\varphi(v_1, \ldots, v_k) \Rightarrow^* t$$

is finite.

The subset $\Phi_b \subseteqq \Phi$ of bounded symbols can be effectively determined. Every rational symbol is bounded. Let $\Phi_{nb} = \Phi - \Phi_b$. For any subset $\Phi' \subseteqq \Phi_{nb}$, we shall denote as $N(\Phi', V_k)$ the set of elements $t \in T(\Phi, V_k)$, every branch of which contains only symbols in $\Phi' \cup \Phi_b$ with at most one symbol of Φ'.

We say that Σ is *quasi-ultralinear* if there exists a preorder $\leqq$ on Φ such that for every rule $\varphi(v_1, \ldots, v_k) \to t$ in $R(\Sigma)$, with $\varphi \in \Phi_{nb}$:

(4.8.1) Either every symbol ψ occurring in t is such that $\psi < \varphi$, or

(4.8.2) t does not satisfy (4.8.1) and belongs to $N(\Phi_\varphi, V_k)$, where

$$\Phi_\varphi = \{ \varphi \in \Phi \mid \psi \leqq \varphi \} \cup \Phi_b.$$

PROPOSITION 4.13: *Let Σ be a quasi-ultralinear system with set of nonterminals Φ. One can build an ultralinear system Σ' with set of nonterminals Φ' such that $\Phi \subseteq \Phi'$ and $(\Sigma, \varphi(v_1, \ldots, v_k))$ and $(\Sigma', \varphi(v_1, \ldots, v_k))$ are equivalent for all $\varphi \in \Phi$. Hence, the equivalence problem for quasi-ultralinear systems is decidable.*

Sketch of Proof: The idea is to replace certain yerms of $T(\Phi_b, V)$ by new symbols in order to eliminate "small turns". Since the proof is long and technical, it is omitted. □

REFERENCES

1. C. BEERI, *An Improvement on Valiant's Decision Procedure for Equivalence of Deterministic Finite-Turn Pushdown Machines*, Theoret. Comput. Sci., Vol. 3, 1976, pp. 305-320.
2. D. CAUCAL, *Décidabilité de l'égalité des languages algébriques infinitaires simples*, 3rd Symposium on Theoretical Aspects of Computer Science, L.N.C.S., Vol. 210, Springer Verlag, 1986.
3. B. COURCELLE, *On Jump-Deterministic Pushdown Automata*, Math. Systems Theory, Vol. 11, 1977, pp. 87-109.
4. B. COURCELLE, *A Representation of Trees by Languages I*, Theoret. Comput. Sci., Vol. 6, 1978, pp. 255-279.
5. B. COURCELLE, *A Representation of Trees by Languages II*, Theoret. Comput. Sci., Vol. 7, 1978, pp. 25-55.
6. B. COURCELLE and J. VUILLEMIN. *Completeness Results for the Equivalence of Recursive Schemes*, J. Comput. System Sci., Vol. 12, (2), 1976, pp. 179-197.
7. B. COURCELLE, *Fundamental Properties of Infinite Trees*, Theoret. Comput. Sci., Vol. 25, (2), 1983, pp. 95-170.
8. J. ENGELFRIET and E. SCHMIDT, *IO and OI, I and II*, J. Comput. System Sci., Vol. 15, (3), 1977, and Vol. 16, (1), 1978, pp. 328-353 and 67-99.
9. E. P. FRIEDMAN, *Equivalence Problems for Deterministic Context-Free Languages and Monadic Recursion Schemes*, J. Comput. System Sci., Vol. 14, 1977, pp. 344-359.
10. J. H. GALLIER, *DPDA's in "Atomic Normal Form" and Applications to Equivalence Problems*, Theoret. Comput. Sci., Vol. 14, 1981 and Vol. 19, 1982, pp. 155-188 and 229.
11. S. GINSBURG and E. SPANIER, *Finite-Turn Pushdown Automata*, S.I.A.M. J. on Control, Vol. 4, (3), 1966, pp. 429-453.
12. J. GOGUEN, J. THATCHER, E. WAGNER and J. WRIGHT, *Initial Algebra Semantics*, J. A.C.M., Vol. 24, 1977, pp. 68-95.
13. S. GORN, *Explicit Definitions and Linguistic Dominoes*, in J. HART and S. TAKASU, Eds., Systems and Computer Science, University of Toronto Press, 1965.
14. I. GUESSARIAN, *Algebraic Semantics*, L.N.C.S., Vol. 99. Springer Verlag, 1981.
15. M. A. HARRISON, *Introduction to Formal Language Theory*, Addison Wesley, Reading, Mass, 1978.

16. M. A. HARRISON and I. M. HAVEL, *Strict Deterministic Grammars*, J. Comput. System Sci., Vol. 7, 1973, pp. 237-277.

17. M. A. HARRISON and I. M. HAVEL, *Realtime Strict Deterministic Languages*, S.I.A.M. J. on Comput., Vol. 1, 1972, pp. 333-349.

18. M. A. HARRISON and I. M. HAVEL, *On the Parsing of Deterministic Languages*, J. A.C.M., Vol. 21, 1974, pp. 525-548.

19. M. NIVAT, *On the Interpretation of Recursive Polyadic Program Schemes*, Symposia Mathematica, Vol. 15, Academic Press, New York, 1975, pp. 255-281.

20. M. OYAMAGUCHI and N. HONDA, *The Decidability of Equivalence for Deterministic Stateless Pushdown Automata*, Information and Control, Vol. 38, 1978, pp. 367-376.

21. M. OYAMAGUCHI, Y. INAGAKI and N. HONDA, *The Equivalence Problem for Real-Time Strict Deterministic Languages*, Information and Control, Vol. 45, 1980, pp. 367-376.

22. M. OYAMAGUCHI, Y. INAGAKI and N. HONDA, *The Equivalence Problem for Two DPDA's One of which is a Finite-Turn or One-Counter Machine*, J. Comput. System Sci., Vol. 23, (3), 1981, pp. 366-382.

23. M. OYAMAGUCHI, *The Equivalence Problem for Real-Time DPDA's*, submitted for publication, 1986.

24. L. G. VALIANT, *Decision Procedures for Families of Deterministic Pushdown automata*, Ph. D. thesis, University of Warwick, U.K., 1973.

25. L. G. VALIANT, *The Equivalence Problem for Deterministic Finite-Turn Pushdown Automata*, Information and Control, Vol. 25, 1975, pp. 123-133.