AN AXIOMATIC APPROACH TO THE KORENJAK - HOPCROFT ALGORITHMS

B. Courcelle

Université de Bordeaux I
U.E.R. de Mathématiques-Informatique
351, cours de la Libération
33405 - T A L E N C E
FRANCE

Abstract : There exist many equivalence decision algorithms for classes of grammars, program schemes, transducers which follow the general pattern of the Korenjak-Hopcroft algorithm for deciding the equivalence of simple deterministic grammars. An axiomatic framework is presented which points out the essence of the Korenjak-Hopcroft algorithm and applies to numerous situations.

1. INTRODUCTION

Equivalence problems for context-free grammars, automata of various kinds, program schemes, string- and tree-transducers have received much attention (from a large number of papers we only cite (KH 66), (OP 77), (F 77), (C 78), (HHY 79), (Es 79) and (OHI 80)). The motivation of all these works is certainly not the applications for at least three reasons : first, a decidability result is often useless by itself (LL(k) grammars have been introduced for their ability to be parsed top-down but the decidability

Note : This work has been supported by the ATP contract 4275 of CNRS - The present paper is a shortened version of report AAI - 8081 (available on request).

of the equivalence of two LL(k)-grammars is of no utility at all for syntactic analysis; similar remarks apply to program schemes for which the strong equivalence usually considered is much too restrictive for any practical use; see (DB 76), (HL 78), (C 79)); secondly, because most interesting properties are undecidable (in particular for program schemes); thirdly, because most known algorithms for the above problems are super-exponential (in particular (OHI 80)) and even if needed, they are not pratically usable

But the motivation does exist and is twofold : first to draw the boundary between decidable and undecidable problems, and second to investigate the properties of the objects (grammars, automata, etc...) since a decidability result always rests upon some combinatorial property of the objects considered. In the same spirit, the decidability algorithm of (CV 76) (for the equivalence of certain program schemes) does not simply say "yes" or "no", but provides a set of relations from which one can construct a formal proof of the equivalence of the two considered program schemes (if they are equivalent) within a certain formal proof system.

It appears that certain of these algorithms are "similar" : for instance, the algorithms of (CV 76), (OP 77), (HHY 79) are explicitly modeled after the algorithm of Korenjak and Hopcroft for simple deterministic grammars (KH 66). It appears that this algorithm is quite fundamental since it can be "essentially" applied to a large number of different problems ((CF 80) and section 5 of the present paper propose new applications); it would be also interesting to understand why, in certain situations it is not applicable. The meanings of "similar" and "essentially" are very imprecise: is there anything more in common in these algorithms than the tree that one is naturally lead to draw when operating the algorithm by hand ?

The purpose of this paper is to design an axiomatic presentation of the Korenjak-Hopcroft algorithm, working with abstract assertions. In the applications these assertions will be properties (to decide) such that equivalence of two non-terminal strings of a context-free grammar, or two program schemes, etc...

A first attempt in this direction has been done by Harrison et al. (HHY 79). But their formalism is only applicable to context-free grammars in Greibach normal form. Our axiomatics is more abstract and is applicable to many other objects such as tree-grammars, string- and tree-transducers, program schemes of various types.

It provides a common way to formulate different algorithms solving the same problem, hence a precise means to compare them. It allows to point out in a precise way the similarities between algorithms solving different problems. Finally, the existence of abstract decidability results shortens and systematicizes the proofs of new concrete results : it reduces a proof to a number of verifications, many of them being trivial, the other ones being really important and showing the crux of the proof.

As applications, known algorithms can be reformulated and compared ((KH 66), (HHY 79)), known results can be proved in a different way (the equivalence problem for deterministic top-down tree-transductions, Esik (Es 79)) and a new result will be proved (the equivalence problem for ε-limited deterministic top-down tree-transductions of infinite trees).

An equivalence problem concerning program schemes and attribute systems will be shown decidable in this way in a forthcoming paper (Courcelle and Franchi-Zannettacci (CF 80)).

2. DECISION SYSTEMS AND GENERAL RESULTS.

The set of subsets of a set X is denoted by $P(X)$ and the set of finite subsets of X by $P_0(X)$.

We shall define objects called decision systems. They are composite objects of the form $\langle A, E, \mathbf{exp}, \mathbf{split}, \vdash\!\!\rightarrow \rangle$ the components of which are now defined.

(2.1) The set A is a countable set of objects called assertions. An assertion is either true or false.

A typical example is the set of all pairs of words on the non-terminal alphabet

of a given context-free grammar G . An assertion (α,β) is true if and only if $L(G,\alpha)=L(G,\beta)$ where $L(G,\alpha)$ denotes the language generated by from a non-terminal string α .

We shall assume that A is recursive i.e. that we can decide whether a given object is an assertion or not.

Our problem will be to decide whether some given assertion is true or false. This will be possible under a number of other assumptions.

Our assertions have various degrees of truth. For every integer $n\geq 0$, an assertion is either n-true or n-false.

We assume the following conditions :

(2.1.1) if an assertion is $(n+1)$-true, then it is n-true,

(2.1.2) an assertion is true if and only if it is n-true for all n .

In the above example, we say that (α,β) is n-true if and only if, for all word u of length less than n :

$$u \in L(G,\alpha) \Leftrightarrow u \in L(G,\beta)$$

A set of assertions is true (resp. n-true) if all its elements are true (resp. n-true). We denote by T (resp. n-T) the set of all true (resp. n-true) assertions.

We denote by Λ a distinguished 0-false assertion and we assume that :

(2.1.3) every assertion except Λ is 0-true.

(2.2.) Let also be given a subset of $P_0(A)\times A$ denoted by $\vdash$ and called a deduction relation. We shall write $C \vdash A$ instead of $(C,A)\in \vdash$. We say that A is a consequence of C if $C \vdash A$.

In the above example, we would define $\vdash$ in such a way that :

$$\{(\alpha,\beta),(\gamma,\delta)\} \vdash (\alpha\gamma,\beta\delta)$$

since $L(G,\alpha) = L(G,\beta)$ and $L(G,\gamma) = L(G,\delta)$ imply $L(G,\alpha\gamma) = L(G,\beta\delta)$. See (4.2) for more details.

We extend $\vdash$ into a relation on $P_0(A)$ by defining $C \vdash C'$ if and only if $C \vdash A$ for all A in C' .

We assume the following properties, for all n in N , all A in

A , all C and C' in $P_0(A)$:

(2.2.1) $A \in C$ implies $C \vdash\!\!\!- A$

(2.2.2) $C \vdash\!\!\!- \Lambda$ if and only if $\Lambda \in C$

(2.2.3) $C \vdash\!\!\!- C'$ and $C' \vdash\!\!\!- A$ imply $C \vdash\!\!\!- A$

(2.2.4) $C \subseteq n\text{-}T$ and $C \vdash\!\!\!- A$ imply $A \in n\text{-}T$

A canonical deduction relation $\vdash\!\!\!-_T$ can be defined as follows :

$C \vdash\!\!\!-_T A$ if and only if

either $A = \Lambda$ and $\Lambda \in T$

or $A \neq \Lambda$ and for all n , $C \subseteq n - T$

implies $A \in n - T$

It is easy to prove that $\vdash\!\!\!-_T$ satisfies (2.2.1) to (2.2.4).

We shall not use it in our decision systems and in the algorithms derived thereof since it is at least as difficult to determine as what we want to determine : A is true of and only if $\emptyset \vdash\!\!\!-_T A$.

The deduction relations $\vdash\!\!\!-$ that we shall use in concrete cases (see sections 4 and 5) will be syntactically characterizable parts of $\vdash\!\!\!-_T$

(2.3.) Certain assertions (and in particular Λ) are said elementary. We denote by E (resp. by TE) (resp. by $n\text{-}TE$) the set of elementary (resp. of true elementary) (resp. of n-true elementary) assertions.

(2.4) A partial recursive mapping exp : $A \rightarrow P_0(A)$ is given the domain of which is a recursive subset of A denoted by A_{exp} and such that $E \subseteq A_{exp}$.

We say that an assertion A in A_{exp} is expanded into a set exp(A) of assertions. (This mapping formalizes the type A-replacements of (KH 66)).

We assume the following properties, for all A in A_{exp}:

(2.4.1) $A \in T$ implies exp(A) $\subseteq T$,

(2.4.2) exp(Λ)=$\{\Lambda\}$,

(2.4.3) if exp(A) $\subseteq n\text{-}T$ then $A \in (n+1)\text{-}T$

(2.5) Let us define $A_{split} = (A-E) \cup \{\Lambda\}$,

We assume the existence of a multi-valued mapping split:$A_{split} \longrightarrow P_0(E)$.

Its purpose is to split a "complex" assertion A into a finite set of "elementary" ones. We shall allow this to be done in different possible ways. Hence split is multivalued in the general case.

The axioms concerning split are the following ones (where A is in A_{split} and C is any value of split(A)):

(2.5.1) $A \in T$ implies $C \subseteq TE$,

(2.5.2) if $A = \Lambda$ then $C = \{\Lambda\}$,

(2.5.3) if $\Lambda \notin C$ then $C \vdash A$.

Axiom (2.5.3) says that split is a kind of converse of $\vdash$. It reduces the proof of a goal assertion A to several proofs of "simpler" ones.

(2.6) A decision system is a 5-tuple $D = \langle A, E, exp, split, \vdash \rangle$ satisfying all conditions (2.1) to (2.5). It is said effective if TE is finite and split is computable (by means of a non-deterministic algorithm).

We shall also use a weaker concept. A weak decision system is an object $E = \langle A, exp, \vdash \rangle$ where A satisfies (2.4) with $A = A_{exp}$ and $\vdash$ satisfies (2.2).

An assertion will be proved or disproved by means of a tree defined from A, exp and split. This tree by itself is not essential but is a convenient way to organize the assertions constructed from A .

(2.7) Definitions :

Let $D = \langle A, E, exp, split, \vdash \rangle$ be a decision system.

We define a set $T(D)$ of finite and infinite trees associated with D . When D is known from the context, an element of $T(D)$ will be simply called a tree. A tree T is in $T(D)$ if and only if :

(i) each of its node is labeled by some assertion.

(ii) all internal nodes (i.e. the nodes which are not leaves) and certain leaves are "marked" as checked and this is subject to conditions given below.

A node ν in T labeled by A such that $A \neq \Lambda$ is checked if one of the following two cases holds :

(2.7.1) $A \in A_{exp}$, $exp(A)=\{B_1, \ldots, B_k\}$ and for each $i=1, \ldots, k$, the node ν has a successor labeled by B_i (no successor if $k=0$); we say that ν is an exp-node and that A is expanded in T .

(2.7.2) $A \in A_{split}$, $\{B_1, \ldots, B_k\}=split(A)$ and for each $i=1, \ldots, k$. the node ν has a successor labeled by B_i (none if $k=0$); We say that ν is a split-node and that A is splitted in T .

The assertion labeling a checked node is checked (in T).

A tree (in $T(D)$) is an A-tree if A is the label of its root . It is negative if it has some node labeled by Λ . It is positive if every node :

either is checked

or is a leaf and is not-to-be-checked. This means that its label is checked in T i.e. labels some other node of T which is checked.

(Hence a positive tree has no node labeled with Λ).

(2.8) Theorem : let D be a decision system and A be an assertion.

(1) A is false if and only if there exists a finite negative A-tree (in $T(D)$).

(2) A is true if and only if there exists a finite or infinite positive A-tree.

(3) Furthermore if TE is finite, A is true if and only if there exists a finite positive A-tree.

(2.13) Theorem : let D be a decision system.

(1) If $A=A_{exp}$ or if the mapping split is computable then the falsity of an assertion is semi-decidable (i.e. is a recursively enumerable property).

(2) If TE is finite and if $\vdash$ is recursively enumerable then the truth of an assertion is semi-decidable .

(3) If D is effective (i.e. TE is finite and the mapping split is computable) then the truth of an assertion is decidable.

Corollary : let $E = \langle A, exp, \vdash\!\!\rightarrow \rangle$ be a weak decision system such that $\vdash\!\!\rightarrow$ is recursively enumerable and there exists a finite set A_0 of true assertions such that $A_0 \vdash\!\!\rightarrow A$ for all true assertion A. Then the truth of an assertion is decidable.

3. SOME REFINEMENTS

In this section we shall formalize the possibility of short-cuts informally suggested in (KH 66). We shall refine our definition of decision systems to deal with cases where TE is infinite. This refined definition will also include a formalization of short-cuts.

(3.1) Definition :

A decision system with short-cuts is a pair $(D, \vdash\!\!\rightarrow')$ consisting of a decision system $D = \langle A, E, exp, split, \vdash\!\!\rightarrow \rangle$, and a subset $\vdash\!\!\rightarrow'$ of $\vdash\!\!\rightarrow$ such that :

(3.1.1) $C' \vdash\!\!\rightarrow' A$ and $C' \subseteq C$ imply $C \vdash\!\!\rightarrow' A$

It is said effective if D is effective and $\vdash\!\!\rightarrow'$ is recursive (we can decide whether $C \vdash\!\!\rightarrow' A$ or not).

A tree T in $T(D)$ is $\vdash\!\!\rightarrow'$-positive if each of its nodes :

either is checked

or is a leaf and is not-to-be-checked. This means that it is labeled by some assertion $A \neq \Lambda$ which is checked or is such that $C \vdash\!\!\rightarrow' A$, for some finite set C of expanded assertions.

Note that a positive tree is $\vdash\!\!\rightarrow'$-positive and that to be $\emptyset$-positive and positive are the same.

(3.2) Theorem : Let $(D, \vdash\!\!\rightarrow')$ be an effective decision system with short-cuts. An assertion A is true if and only if there exists a finite $\vdash\!\!\rightarrow'$-positive A-tree .

With help of a generalization of the mapping split (called rsplit) we define a new class of decision systems.

(3.4) Definition.

A generalized decision system is a 5-tuple $D=\langle A, E, \underline{exp}, \underline{rsplit}, \vdash\!\!\!\!- \rangle$ where A, $A_{\underline{exp}}$, $A_{\underline{split}}$, E, exp and $\vdash\!\!\!\!-$ are as in definition (2.6) and where rsplit is a multivalued mapping : $A_{\underline{split}} \times P_o(A) \longrightarrow P_o(E)$ satifying the following conditions where C' is any value of $\underline{rsplit}(A,C)$:

(3.4.1) if A and C are true, then C' is true,

(3.4.2) if $A = \Lambda$ or $\Lambda \in C$ then $C' = \{\Lambda\}$,

(3.4.3) if $\Lambda \notin C'$ then $C \cup C' \vdash\!\!\!\!- A$.

We shall use rsplit in the following way.

Having to prove an assertion A , we replace it by a set $C' = \underline{rsplit}(A,C)$ of elementary assertions which takes in account a set C of already proved (actually expanded that is partially proved) assertions. Axiom (3.4.3) shows that the larger C , the smaller C' will be.

This will tend to diminish the size of the trees we shall construct and this will also allow us to handle cases where a finite set TE is not easy to determine.

(3.5) Definition :

A generalized decision system as above is effective if :

(3.5.1) rsplit is computable,

(3.5.2) For every sequence $(C_i)_{i\geqslant o}$ in $P_o(TE)$, for every sequence $(A_i)_{i\geqslant 1}$ in $T \cap A_{\underline{split}}$ such that $C_i \cup C'_i \subseteq C_{i+1}$ for all i (where $C'_i = \underline{rsplit}(A_{i+1}, C_i)$) there exists j such that $C'_i \subseteq C_i$ for all $i \geqslant j$.

Remarks:

1 - Condition (3.5.2) holds if TE is finite : there exists j such that $C_i = C_j$ for all $i \geqslant j$. This implies $C'_i \subseteq C_i = C_j$ for all $i \geqslant j$.

We now define the family of trees associated with a generalized system.

(3.6) Definition : Let D be a generalized decision system .

We define $T(D)$ as in (2.7) except for (2.7.2) which is modified as follows :

(3.6.1) if $A \in A_{split}$, $\{B_1, \ldots, B_k\} = rsplit(A,C)$ for some finite set C of expanded assertions, then for each $i=1,\ldots,k$, the node ν has a successor labeled by B_i (if $k=o$, then ν has no successor) ; we say that ν is a split-node and is checked. We say that A is splitted.

(3.8) Theorem : let D be an effective generalized decision system. An assertion A is true if and only if there exists a finite positive A-tree. The truth of an assertion is decidable.

4 - APPLICATIONS TO CONTEXT-FREE GRAMMARS

The following (known) decidable cases of the equivalence problem $L(G_1,S_1) = L(G_2,S_2)$ can be proved as applications of theorems (2.13), (3.2) and (3.8).

(1) When G_1 and G_2 are simple deterministic grammars (KH 66). The short-cuts informally presented in (KH 66,p.43) can be justified.

(2) When G_1 is as above and G_2 is a strict-deterministic grammar. We obtain a simplification in the proof given in (HHY 79).

(3) When G_1 and G_2 are LL(k)-grammars (by formalizing the method of (OP 77)).

5 - APPLICATIONS TO TREE TRANSDUCTIONS

We shall give another proof of Esik's result showing the decidability of the equivalence problem for deterministic top-down tree transducers (Es 79)

(5.1) Definitions and notations.

Our definitions and notations concerning trees will be taken from Courcelle (C 79). Our notations concerning tree-transductions will be largely borrowed from Engelfriet (E 75).

A deterministic top-down tree transducer (we shall simply say a DT-transducer) is a 4-tuple $S=< P,F,\underline{\Phi},\Sigma >$ where:

(1) P and F are finite ranked alphabets such that $M(P) \neq \emptyset$ and $M(F) \neq \emptyset$ (they have symbols of arity 0),

(2) $\underline{\Phi}$ is a finite non empty set of unary symbols called the states of S,

(3) Σ is a finite set of rewriting rules, all of the form :

(5.1.1) $$\psi(p(v_1,\ldots,v_k)) \to t$$

for ψ in $\underline{\Phi}$, p in P, t in $M(F,\underline{\Phi}(V_k))$, (hence $t \in M(F)$ if $\rho(p)=0$),

(4) for all ψ in Φ and p in P there is at most one rule of the form (5.1.1)

The determinism is expressed by (4).

Such a system Σ is Noetherian (this can be proved from (3)) and confluent (a consequence of (4)). It defines a partial mapping $\psi_S : M(P) \to M(F)$ for each ψ in $\underline{\Phi}$

To every $k \geq 1$ and t in $M(F,\underline{\Phi}(V_k))$ corresponds a partial mapping $t_{S,k} : M(P)^k \to M(F)$ defined as follows :

$$t_{S,k}(\underline{u}) = \psi_S(u_i) \quad \text{if} \quad t=\psi(v_i)$$

$$t_{S,k}(\underline{u}) = f(t_{1S,k}(\underline{u}),\ldots,t_{nS,k}(\underline{u})) \quad \text{if} \quad t = f(t_1,\ldots,t_n) \text{ and } f \in F$$

where $\underline{u}$ denotes $(u_1,u_2,\ldots,u_k) \in M(P)^k$.

We shall write t_S instead of $t_{S,k}$ when k is known by the context.

For all ψ in Ψ, the domain of ψ_S i.e. the set of u in $M(P)$ such that $\psi_S(u)$ is defined is a deterministic recognizable tree-language (dr-tree language for short) i.e. a set of trees recognized by a deterministic top-down finite-state tree-automaton (dt-fta).

From this, the domain of $t_{S,k}$ is :

$$\mathrm{Dom}(t_{S,k}) = R_1 \times R_2 \times \ldots \times R_k$$

$$R_i = M(P) \cap \bigcap \{\mathrm{Dom}(\psi_S) / \psi \in \Psi \text{ and } \psi(v_i) \text{ is a subtern of } t\}.$$

and clearly, each set R_i is a deterministic recognizable subset of $M(P)$.

The problem is to decide whether the mappings ψ_S and ϕ_S are the same (for states ψ and ϕ of a given DT-transducer S).

When trying to define a decision system to solve this problem, it is natural to introduce assertions of the form (t,t') for t,t' in $M(F,\phi(v_k))$, which are true if the partial functions t_S and t'_S are the same (in particular have the same domain).

We want to split an assertion $f(t_1,t_2)$, $f(t'_1,t'_2)$ into the set of assertions $\{(t_1,t'_1), (t_2,t'_2)\}$. But the truth of the former does not imply the truth of the later, since for example :

$$f(t_1,t_2)_S = f(t'_1,t'_2)_S \quad \text{if and only if}$$

$$\mathrm{Dom}(t_{1S}) \cap \mathrm{Dom}(t_{2S}) = \mathrm{Dom}(t'_{1S}) \cap \mathrm{Dom}(t'_{2S}) \quad \text{(let us call id D)}$$

$$t_{iS} \restriction D = t'_{iS} \restriction D \quad \text{for } i=1,2 .$$

But this does not imply, say even the property

$$\mathrm{Dom}(t_{1S}) = \mathrm{Dom}(t'_{1S}) .$$

We remedy to this by considering assertions of the form $\langle D,t,t' \rangle$, and defined as true if $D \subseteq \mathrm{Dom}(t_S) \cap \mathrm{Dom}(t'_S)$ and $t_S \restriction D = t'_S \restriction D$.

Lacking of space, we can only partially define the decision system $D = \langle A, E, \underline{\mathrm{exp}}, \underline{\mathrm{split}}, \vdash\!\!\!\rightarrow \rangle$ that shows the decidability of our equivalence problem. (In particular $\mathcal{R}$ is a finite set of dr-tree languages that can be effectively determined).

An <u>assertion</u> is either Λ or an element of A_k for some $k>1$, where : A_k is the set of $(k+2)$-tuples $<R_1,\ldots,R_k,t,t'>$ such that $R_1,\ldots,R_k \in \mathcal{R}, t,t' \in M(F,\bar{\Phi}(V_k))$ and $R_1 \times \ldots \times R_k \subseteq Dom(t_S) \cap Dom(t'_S)$. Such an assertion will also be written $<\underline{\underline{R}},t,t'>$ with $\underline{\underline{R}} = (R_1, R_2,\ldots,R_k)$; the integer k will be denoted by $|\underline{\underline{R}}|$

We let $A = \{\Lambda\} \cup \bigcup \{ A_k / k>1 \}$ denote the set of all assertions. Note that we can decide whether a given object $<R_1,\ldots,R_k,t,t'>$ is an assertion or not.

The assertion Λ is defined as 0-false.

An assertion $<R_1,\ldots,R_k,t,t'>$ is n-<u>true</u> if for all $\underline{\underline{u}}=(u_1,u_2,\ldots u_k)$ in $R_1 \times \ldots \times R_k$ such that $\|\underline{\underline{u}}\| = Max\{|u_i| / 1\leq i\leq k\}<n$ then $t_S(\underline{\underline{u}}) = t'_S(\underline{\underline{u}})$. It is <u>true</u> if the same holds for all $\underline{\underline{u}}$ in $R_1 \times \ldots \times R_k$.

We define the deduction relation as the least subset $\vdash$ of $P_o(A) \times A$ which satisfies conditions (2.2.1), (2.2.2) and (2.2.3) together with the the following ones :

(5.3.1) $\emptyset \vdash <\underline{\underline{R}},t,t>$

(5.3.2) $<R_1,\ldots,R_\ell,t,t'> \vdash <R_1,\ldots,R_k,R_{k+1},\ldots R'_{\ell'},t,t'>$ if $Var(t) \cup Var(t') \subseteq V_k$.

(5.3.3) $<R_1,\ldots,R_k,t,t'> \vdash <R_{\sigma(1)},\ldots,R_{\sigma(k)},\mu(t),\mu(t')>$ if σ is a permutation of k and μ is the substitution such that $\mu(v_i)=v_{\sigma(i)}$.

(5.3.4) $<\underline{\underline{R}},t,t'> \vdash <\underline{\underline{R}},t',t>$

(5.3.5) $<\underline{\underline{R}},t,t'>,<\underline{\underline{R}},t',t''> \vdash <\underline{\underline{R}},t,t''>$

(5.3.6) $\{<\underline{\underline{R}},t_i,t'_i> \; ; \; 1\leq i\leq n\} \vdash <\underline{\underline{R}},f(t_1,\ldots,t_n),f(t'_1,..t'_n)>$

Next we define the set E of elementary assertions :

$E = \{\Lambda\} \cup \{<R_1,\ldots,R_k,\psi(v_1),t> / \psi \in \bar{\Phi}, \; v_i \in Var(t) \text{ for } i=2,3,\ldots,k\}$.

These techniques allow us to establish the decidability of the following decision problems :

(1) The equivalence and inclusion problems for deterministic top-down

finite-state tree-transducers (known from (Es 79)).

(2) The equivalence and inclusion problems for deterministic top-down finite-state tree-transducers with regular look-ahead (Es 79) hence for deterministic bottom-up finite-state tree-transducers by a result of (E 77).

(3) The equivalence and inclusion problems for deterministic top-down finite state ε-limited transducers of infinite trees (a new result).

Acknowledgements : I thank one of the referees who made especially helpful comments.

REFERENCES

(C 74) B. COURCELLE : Une forme canonique pour les grammaires simples déterministes , RAIRO, R-1 (1974) pp. 19-36

(C 78) B. COURCELLE : A representation of trees by languages . Theor. Comput. Sci. 6(1978) 255-279 and 7 (1978), 25-55.

(C79) B. COURCELLE : Infinite trees in normal form and recursive equations having a unique solution . Math. Systems Theory 13 (1979), 131-180

(CF 80) B. COURCELLE, P. FRANCHI-ZANNETTACCI : On the equivalence problem of attribute systems . Report AAI-80 Univ. of Bordeaux-I, 1980

(CV 76) B. COURCELLE, J. VUILLEMIN : Completeness results for the equivalence of recursive schemas. J. Comput. System Sci. 12 (1976), 179-197

(DB 76) J. DARLINGTON, R. BURSTALL : A system which automatically improves programs . Acta Informatica 6 (1976), 41-60

(E 75) J. ENGELFRIET : Bottom-up and Top-down tree transformations, a comparison . Math. Systems Theory, 9 (1975), 198-231.

(E 77) J. ENGELFRIET : Top-down transducers with regular look-ahead , Math. Systems Theory 10 (1977) 289-303

(Es 79) Z. ESIK : On functional tree-transducers , FCT Symposium 1979,

(F 77) E. FRIEDMAN : Equivalence problems for deterministic CFL and monadic recursion schemes , J. Comput. System Sci. 14 (1977) 344-359

(HHY 79) M. HARRISON, I. HAVEL, A. YEHUDAI : On equivalence of grammars through transformation trees , Theor. Comput. Sci. 9 (1979) 173-205

(HL 78) G. HUET, B. LANG : Proving and applying program transformations expressed with 2nd order patterns , Acta Informatica 11 (1978) 31-55

(KH 66) A. KORENJAK, J. HOPCROFT : Simple deterministic languages . 7th IEEE Annual Symp. on Swithing and Automata Theory, Berkeley, California (1966) 36-46

(OIH 80) M. OYAMAGUCHI, N.HONDA, Y. INAGAKI : The equivalence problem for real-time strict deterministic languages , Information and Control 45 (1980) 90-115.

(OP 77) T. OLSHANSKY, A. PNUELI : A direct algorithm for checking equivalence of LL(k)-grammars , Theor. Comput. Sci. 4 (1977) 321-349.