Theoretical Computer Science 4 (1977) 321–349.

A DIRECT ALGORITHM FOR CHECKING EQUIVALENCE OF LL(k) GRAMMARS

Tmima OLSHANSKY and Amir PNUELI
Computer Science Division, Tel Aviv University, Tel Aviv, Israel

Communicated by Maurice Nivat
Received August 1976
Revised January 1977

Abstract. We deal with the problem of testing equivalence of two LL(k) grammars. The problem had been shown to be decidable for general k by Rosenkrantz and Stearns [2], who solved it by reduction into an equivalence problem for special DPDA's. In a paper by Korenjak and Hopcroft [1] the equivalence problem for LL(1) grammars is solved by a branching algorithm operating directly on the grammars. Our work presents a direct branching algorithm for the general LL(k) grammars equivalence problem.

1. Introduction

While the equivalence problem for general context free grammars is known to be undecidable, the same problem for deterministic context free grammars is still an open problem. There has been a considerable effort to identify the highest type of deterministic grammars, for which equivalence is decidable. (See Friedman [3] and Valiant [4] for such efforts.)

The equivalence problem for LL(k) grammars is known to be solvable for quite a while. The first result in this direction was given by Korenjak and Hopcroft [1]. They propose a branching algorithm for deciding equivalence of two LL(1) grammars in Greibach Normal Form. The basic idea in a branching algorithm is to construct a tree, containing equivalence statements at its nodes. The root of the tree contains the equality $S_1 \approx S_2$, where S_i is the initial variable of the grammar G_i. Branching is accomplished by considering different terminal letters as initial letter in words generated from S_1 and S_2. For any letter $a \in T$ there can be by the LL(1) property at most one rule in each grammar, which produces from S_i a word starting with a. Let these rules be

$$S_1 \to a\alpha_1, \qquad S_2 \to a\alpha_2.$$

Then one consequence of the equality $S_1 \approx S_2$ must be the equality $\alpha_1 \approx \alpha_2$, which we add as a node, branching from the root. Similarly, we consider other initial letters and then apply the branching process to the newly generated nodes. In order

to keep the tree finite, there is an alternate process called splitting, which has to be applied to nodes, whose equality contains words which have become too long. This process will ensure a bounded length of the equalities in the nodes, and hence a finitely terminating algorithm. It is proved by the authors that if during construction of this comparison tree no contradiction is discovered, this proves the equivalence of the grammars.

The decidability of equivalence for general LL(k) grammars was demonstrated by Rosenkrantz and Stearns [2]. However, their equivalence testing algorithm does not operate directly on the grammars. Instead, they reduce the grammars to corresponding special DPDA machines, and then construct a product machine, which simulates the joint action of the two automata, searching for a word which differentiates between the two. The final step is to test this joint machine for emptiness.

The interest in equivalence problems for context free grammars and LL(k) grammars in particular, has increased when the theory of recursive program schemas was shown to be closely related to these grammars, having very similar algorithm, and often reducible to language theory. See references [5, 6, 7, 3, 8, 9, 13, 14].

The motivation for searching for a more "natural" (branching) equivalence test for the general LL(k) case was therefore twofold:

(a) A direct equivalence test for LL(k) may suggest a corresponding branching algorithm for schemas, which will enable direct testing in schemas terms. (See [12] for a corresponding algorithm.)

(b) Having an alternative equivalence test algorithm might suggest additional subdeterministic class of grammars, whose equivalence will be decidable.

The described algorithm is essentially an extension of the Korenjak and Hopcroft [1] algorithm, and uses the basic ideas of comparison tree, branching and splitting processes.

The following exposition is divided into six sections: The first and second sections are an introduction, and some definitions. The third section gives a detailed description of the algorithm, outlining the main steps and situations that may arise, including the final conclusion, which is that either a contradiction (inconsistency) is discovered, or the algorithm terminates, and the grammars are equivalent.

In the Section 4 we give a proof of the correctness and termination (in finite time) of the algorithm. The proof relies on some properties of the phrases generated throughout the algorithm, whose proof is deferrred to the last section.

The Section 5 provides proofs of the necessary properties of those generated phrases. In particular we have to extend the property of being LL(k) grammars to mixed words, containing variables from the two grammars. It also formally justifies the basic steps of branching and splitting by showing that the equivalences generated as descendant nodes hold if and only if the equivalence at the father node holds.

The last section completes the proof of the termination of the algorithm.

2. Definitions and notations

We shall use capital Latin-alphabet letters for variables. Lower case letters at the beginning of the Latin alphabet are used for terminals. Strings of terminals are denoted by lower case letters near the end of the Latin alphabet, and strings of variables are denoted by lower case Greek letters.

We use the equivalence sign "$\approx$" between words to denote equality of the respectively generated languages.

$$\alpha \approx \beta \leftrightarrow L(\alpha) = L(\beta).$$

We will also introduce a formal addition on words to denote union of the respective languages, thus:

$$L(\alpha + \beta) = L(\alpha) \cup L(\beta),$$

which is also used for multiple sums:

$$L\left(\sum_{i \in I} \beta_i\right) = \bigcup_{i \in I} L(\beta_i).$$

The components of a context free grammar are denoted as usual

$$G = \langle N, T, P, S \rangle.$$

Definition 2.1. Let $G = \langle N, T, P, S \rangle$ be a context free grammar. A word $\alpha \in N^*$ is an LL(k) word for some positive integer k if for every two derivations:

(1) $\alpha \Rightarrow^*_{\text{lm}} wA\delta \Rightarrow_{\text{lm}} w\beta\delta \Rightarrow^* wx$,

(2) $\alpha \Rightarrow^*_{\text{lm}} wA\delta \Rightarrow_{\text{lm}} w\gamma\delta \Rightarrow^* wy$,

where "lm" means leftmost derivation, then if $x/k = y/k$, it follows that $\beta = \gamma$. Here

$$x/k = \begin{cases} x & |x| \leqslant k, \\ \begin{pmatrix}\text{The prefix of length } k \\ \text{of } x\end{pmatrix} & |x| > k. \end{cases}$$

We also denote $L/k = \{x/k \mid x \in L\}$.

In other words, given a variable A within a derivation, it can have at most one direct rule, leading ultimately to a word with prefix x/k.

A grammar $G = \langle N, T, P, S \rangle$ is an LL(k) grammar if the initial symbol S is an LL(k) word.

From now on we will call a cfg simply a grammar. In the sequel we will fix our

grammars to be in GNF form. Recall that any LL(k) grammar can be transformed into an LL($k+1$) GNF form grammar ([2]).

We use the notation '\' for prefix removal:

$$u \backslash L = \{v \mid v \in T^*,\ uv \in L\}.$$

Definition 2.2. Equivalence between words. $\alpha \approx \beta$ is defined as $L(\alpha) = L(\beta)$. We define also $\alpha \underset{w}{\approx} \beta$ for any $w \in T^*\$^*$ as

$$w \backslash (L(\alpha) \cdot \$^{|w|}) = w \backslash (L(\beta) \cdot \$^{|w|})$$

i.e. the set of words with prefix w generated by $\alpha \cdot \$^{|w|}$ (α concatenated with $|w|$ \$-signs) is equal to the set of words with this prefix generated by $\beta \cdot \$^{|w|}$.

We add the $\$^{|w|}$ in order to be able to keep a look ahead of $|w|$ letters. If we want to derive from a given word α a terminal word x, we'll derive $x \cdot \$^{|w|}$ from $\alpha \cdot \$^{|w|}$, so when α derives the last letter of x, we still have a $|w|$-look ahead of $\$^{|w|}$.

Note that if $w \in T^*$ then $\alpha \approx_w \beta$ is equivalent to the simpler statement

$$w \backslash (L(\alpha)) = w \backslash (L(\beta)).$$

Definition 2.3. For any given α, the norm of $\alpha, \|\alpha\|$, is defined as:

$$\|\alpha\| = \min\{|t| \mid \alpha \Rightarrow^* t,\ t \in T^*\}.$$

Note that in a grammar whose productions are in GNF, for each variable $A, \|A\| \geq 1$.

Definition 2.4. For a $w \in T^*\$^*$ and $\alpha \in N^*\$^*$, $\|\alpha\| \geq |w| \geq |\alpha|$ (the restriction is always realized if $|\alpha| = |w|$ in a grammar whose rules are in GNF) we define a prefix restricted variable (PRV), written as $[\![w, \alpha]\!]$. This variable is an artificial variable, whose derivation rules are defined using the participating variables productions:

Let $[\![a_1 a_2 \cdots a_m, A_1 A_2 \cdots A_s]\!]$, $s \leq m$ be a PRV. Then for each derivation $A_1 \to a_1 B_1 \cdots B_l$, $l \geq 0$, in P we let

$$[\![a_1 a_2 \cdots a_m,\ A_1 A_2 \cdots A_s]\!] \to a_1 [\![a_2 \cdots a_m,\ C_1 C_2 \cdots C_{m-1}]\!] C_m \cdots C_{s+l-1}$$

be a derivation for this PRV, where $C_1 \cdots C_{s+l-1} = B_1 \cdots B_l A_2 \cdots A_s$.

In the case that $s + l \leq m$ the string $C_s \cdots C_{s+l-1}$ is obviously empty. In the case that $m = 1$ we omit the empty $[\![\Lambda, \Lambda]\!]$. Let us denote the set of all PRV variables over a set of variables N with $|w| \leq k$ by N^R. It is obvious that for any given finite N, N^R is also finite. Note that under the assumption that G is in GNF form, all the derivation rules of N_G^R are also in GNF.

For $\|\alpha\| > |w|$ we also define a prefix restricted word (PRW) denoted by $[w, \alpha]$ as follows:

If $|w| \geq |\alpha|$ then $[w, \alpha] = [\![w, \alpha]\!]$. Otherwise, for $s \geq m$ we define:

$$[a_1 a_2 \cdots a_m, A_1 A_2 \cdots A_s] = [\![a_1 a_2 \cdots a_m, A_1 A_2 \cdots A_m]\!] A_{m+1} \cdots A_s.$$

This consists of a word of length $s - m$, whose first variable is a PRV and the rest are variables of N.

Note that if $|\alpha| \geq |w|$ then

$$[w, \alpha\beta] = [w, \alpha]\beta.$$

Using the notation of PRW's it is possible to write the derivation rules for PRV's in a more compact manner:

If $A \to a\beta$ is a derivation in P we let $[\![aw, A\gamma]\!] \to a[w, \beta\gamma]$ for any $\|A\gamma\| \geq |aw| \geq |A\gamma|$. This will also imply that for any $\|A\gamma\| \geq |aw|$

$$[aw, A\gamma] \Rightarrow a[w, \beta\gamma].$$

Definition 2.5. For any $w \in T^*\$^*$ and $\alpha \in N^*\$^*$ we denote by $L(w, \alpha)$ the language defined by $L(w, \alpha) = w \cdot (w \setminus L(\alpha))$. This is the set of all words out of $L(\alpha)$ beginning with the prefix w.

Definition 2.6. In a grammar, whose productions are in GNF we can define grouped variables representing sums of words:

$$M = \sum_{i=1}^{n} [w_i, \alpha_i]$$

where each $[w_i, \alpha_i]$ is a PRW and $w_i \neq w_j$ for $i \neq j$. If $\alpha_i \in N^*$, we will denote the set of grouped variables over N by N^G.

A grouped variable is also attributed derivation rules by letting

$$M \to [w_i, \alpha_i] \quad \text{for } i = 1, 2, \ldots, n.$$

Note that the derivation rules for grouped variables are no longer in GNF form.

Definition 2.7. The set of prefixes of length l of a language L will be denoted by $\Theta_l(L)$.

Thus

$$\Theta_l(L) = L/l \cap T^l.$$

Lemma 2.8.

$$\alpha \underset{w}{\approx} \beta \supset \alpha \underset{ww'}{\approx} \beta.$$

Proof.

$$\begin{aligned}
\alpha \underset{w}{\approx} \beta &\Rightarrow w\backslash(L(\alpha)\cdot\$^{|w|}) = w\backslash(L(\beta)\cdot\$^{|w|}) \\
&\Rightarrow w\backslash(L(\alpha)\cdot\$^{|w|+|w'|}) = w\backslash(L(\beta)\cdot\$^{|w|+|w'|}) \\
&\Rightarrow w'\backslash(w\backslash(L(\alpha)\cdot\$^{|w|+|w'|})) = w'\backslash(w\backslash(L(\beta)\cdot\$^{|w|+|w'|})) \\
&\Rightarrow (ww')\backslash(L(\alpha)\cdot\$^{|w|+|w'|}) = (ww')\backslash(L(\beta)\cdot\$^{|w|+|w'|}) \\
&\Rightarrow \alpha \underset{ww'}{\approx} \beta. \quad \square
\end{aligned}$$

Lemma 2.9.

$$\alpha \underset{w}{\approx} \beta \iff L(w,\alpha) = L(w,\beta) \quad \textit{for } \alpha,\beta \in N^*,\ w \in T^*.$$

Proof.

$$\begin{aligned}
\alpha \underset{w}{\approx} \beta \supset w\backslash L(\alpha) &= w\backslash L(\beta) \supset w\cdot(w\backslash L(\alpha)) \\
&= w\cdot(w\backslash L(\beta)) \supset L(w,\alpha) = L(w,\beta).
\end{aligned}$$

The other direction is similar. $\square$

Lemma 2.10. *For a PRW* $[w,\alpha]$, $L([w,\alpha]) = L(w,\alpha)$.

Proof. By induction on the length of w If $|w| = 1$, $w = a$. We will show $L([a,\alpha]) = L(a,\alpha)$. Rewrite $\alpha = A\gamma$.

If $\alpha \Rightarrow^* x$, and $x/1 = a$, the first derivation rule used in the derivation $\alpha \Rightarrow^* x$ was of the form $A \to a\beta$. Hence one of the derivation rules of $[\![a,\alpha]\!]$ is $[\![a,\alpha]\!] \to a\beta\gamma$.

We have then:

$$\begin{aligned}
&\alpha \Rightarrow a\beta\gamma \overset{*}{\Rightarrow} x, \\
&[\![a,\alpha]\!] = [\![a,A]\!]\gamma \Rightarrow a\beta\gamma \overset{*}{\Rightarrow} x.
\end{aligned}$$

Conversely, if $[a,A\gamma] \Rightarrow^* x$ then clearly its first step has to be $[\![a,A]\!]\gamma \Rightarrow a\beta\gamma \Rightarrow^* x$ where $A \to \alpha\beta$ is a derivation rule. Consequently, $\alpha = A\gamma \Rightarrow a\beta\gamma \Rightarrow^* x$.

Induction assumption: for $|w| = n$

$$L([w,\alpha]) = L(w,\alpha).$$

We will show the property for a word of length $n+1$, $a_1 w$ for example.

Again we rewrite $\alpha = A\gamma$.

$$L([a_1 w, A\gamma]) = \bigcup_{A \to a_1\beta_{1i} \in P} a_1 \cdot L([w, \beta_{1i}\gamma])$$

$$= \bigcup_{A \to a_1\beta_{1i} \in P} a_1 \cdot L(w, \beta_{1i}\gamma)$$

by the induction assumption.

On the other hand

$$L(A\gamma) = \bigcup_{A \to a_j\beta_{ji} \in P} a_j \cdot L(\beta_{ji}\gamma)$$

$$a_1 w \setminus L(A\gamma) = \bigcup_{A \to a_1\beta_{1i} \in P} w \setminus L(\beta_{1i}\gamma)$$

$$L(a_1 w, A\gamma) = \bigcup_{A \to a_1\beta_{1i} \in P} a_1 \cdot L(w, \beta_{1i}\gamma).$$

Hence

$$L([a_1 w, A\gamma]) = L(a_1 w, A\gamma). \quad \square$$

Lemma 2.11. *Let M be a grouped variable defined as:*

$$M = \sum_{i=1}^{n} [w_i, \eta_i]$$

$$\forall i, j \; |w_i| = |w_j|; \quad \text{if } i \neq j \text{ then } w_i \neq w_j; \quad \forall i \; w_i \in T^*.$$

Claim. $M\alpha \approx_{w_i} \eta_i\alpha$ *for any* α.

Proof.

$$w_i \setminus L(M\alpha) = w_i \setminus (L(M)L(\alpha)) = w_i \setminus \left(\bigcup_{j=1}^{n} L(w_j, \eta_j)L(\alpha) \right)$$

$$= w_i \setminus \left(\bigcup_{j=1}^{n} w_j \cdot (w_j \setminus L(\eta_j))L(\alpha) \right)$$

$$= w_i \setminus (L(\eta_i)L(\alpha)) = w_i \setminus L(\eta_i\alpha). \quad \square$$

Lemma 2.12. *If* $\|\beta\| \geq |w|$ *and* $w \in T^*$ *then*

$$\alpha \underset{w}{\approx} \beta \iff \alpha \underset{w}{\approx} [w, \beta].$$

Proof.

$$w \setminus L(\alpha) = w \setminus L(\beta) = w \setminus (w \cdot (w \setminus L(\beta)) = w \setminus L([w, \beta]).$$

(For every L, $L = w \setminus (w \cdot L)$.)

Hence

$$\alpha \underset{w}{\approx} [w, \beta]. \quad \square$$

3. The equivalence checking algorithm

The algorithm operates on two LL(k) grammars G_1 and G_2 in GNF form, where for $i = 1, 2$

$$G_i = \langle N_i, T, P_i, S_i \rangle.$$

In order that $S_1 \approx S_2$ (i.e. $L(S_1) = L(S_2)$) the two following conditions are necessary and sufficient:

(a) $\Theta_{k-1}(S_1\$^k) = \Theta_{k-1}(S_2\$^k)$ (i.e. the set of all terminal prefixes of length $k-1$ derivable from $S_1\$^k$ is equal to the similar set for $S_2\$^k$).

(b) For each $w \in \Theta_{k-1}(S_1\$^k)$, $S_1 \approx_w S_2$.

Correspondingly, we start an equivalence checking algorithm by constructing a comparison tree whose root node contains the relation $S_1 \approx_w S_2$ for each $w \in \Theta_{k-1}(S_1\$^k)$.

A comparison tree is a labeled tree whose nodes are labeled by equivalence relations of the form

$$\alpha \underset{w}{\approx} \beta$$

where $\alpha \in N^+$, $\beta \in N_2^+$ and $w \in (T^*\$^*)/(k-1)$. (We denote $N = N_1 \cup N_2 \cup N_2^G \cup N_2^R$.)

Altogether there will be $|\Theta_{k-1}(S_1\$^k)|$ such trees.

Each comparison tree is constructed by applying two basic steps to its nodes: the branching step and the splitting step. Both operate on a parent node to produce a set of successor nodes, such that the relations labeling the successors hold if and only if the relation at the parent node holds. Nodes whose labels appeared elsewhere in the tree or which contain equality between the empty words on both sides are not developed further. If during construction any of the steps fails the relation at the root of the tree is concluded to be invalid (proved in Section 4.1). It will also be proved (Section 4.2) that the construction of the tree must terminate, and that if it has terminated successfully (no failure detected in any step) the relation at the root is proved to be valid (proved in Section 4.1).

We will now describe in detail the two steps used in developing the tree:

3.1. Branching

Let the relation to be developed be:

(B) $$A\alpha \underset{w}{\approx} B\beta$$

where $w = a_1 \cdots a_{k-1} \in T^*\$^*$, $A\alpha \in (N_1 \cup N_2)N^*$ and $B\beta \in N_2^+$. It will be proved below (Section 4.2) that all generated relations must be of that form. Moreover, it is assumed (and proved in Section 4.1) that both $A\alpha$ and $B\beta$ are LL(k) words.

Let $b \in \{T \cup \$\}$ such that $wb \in \Theta_k(A\alpha\$^k)$. By the LL($k$) property there is exactly one derivation rule for A such that starting with this derivation $A\alpha\$^k$ can derive a terminal word with prefix wb. Let this rule be

$$A \to a_1\gamma \qquad \gamma \in (N_1 \cup N_2)^*.$$

If (B) is to hold, then also $wb \in \Theta_k(B\beta\$^k)$. If wb is not generable from $B\beta\$^k$ the step fails.

If wb is generable, then there must be exactly one rule for B which will lead to generation of the prefix wb. Let it be

$$B \to a_1\delta, \qquad \delta \in N_2^*.$$

Since $A\alpha \approx_w B\beta$, it must also be true that $A\alpha \approx_{wb} B\beta$. However, since for both A and B there is only one derivation under the restriction of requiring the prefix wb, this is the same as

$$a_1\gamma\alpha \underset{a_1\cdots a_{k-1}o}{\approx} a_1\delta\beta, \qquad \text{(Lemma 2.8)}$$

which is the same as

$$\gamma\alpha \underset{a_2\cdots a_{k-1}b}{\approx} \delta\beta. \qquad \text{(Assertion 2)}$$

This process can be repeated for each $b \in w \setminus \Theta_k(A\alpha\$^k)$. If we denote $\{b_i \mid i = 1, \ldots, s\} = w \setminus \Theta_k(A\alpha\$^k)$ we get for each $i = 1, \ldots, s$ the successor relation

(Bi) $$\gamma_i\alpha \underset{a_2\cdots a_{k-1}b_i}{\approx} \delta_i\beta.$$

A special simplification is required in the case that any of the γ_i has as its leftmost variables a grouped variable. Let $\gamma_i = M\kappa_i$ where

$$M = \sum_{j=1}^{n} [w_j, \eta_j], \qquad \eta_j \in N_2^*,$$

then the relevant relation is

(Bi) $$M\kappa_i\alpha \underset{u_i}{\approx} \delta_i\beta \quad \text{where } u_i = a_2 \cdots a_{k-1}b_i.$$

Since for $l \neq r$, $w_l \neq w_r$ there can be at most one j such that $w_j = u_i$. Since u_i is a generable prefix for the left hand side there must be exactly one.

We claim that in this case (Bi) is equivalent to

(Bi)′ $$[w_j, \eta_j]\kappa_i\alpha \underset{u_i}{\approx} \delta_i\beta \qquad \text{(Lemma 2.11)}$$

which in turn is equivalent to

(Bi)″ $$\eta_j\kappa_i\alpha \underset{u_i}{\approx} \delta_i\beta \qquad \text{(Lemma 2.12)}$$

We can thus formally summarize the branching step as follows:

B – step

Let (B) $A\alpha \approx_w B\beta$ be the parent node

$$(w = a_1 \cdots a_{k-1} \in T^*\$^*, \qquad A\alpha \in (N_1 \cup N_2)N^*, \qquad B\beta \in N_2^+).$$

1. Check that $w \setminus \Theta_k(A\alpha\$^k) = w \setminus \Theta_k(B\beta\$^k)$. If they are not equal report a failure.

2. For each b_i $i = 1, \ldots, s$, $b_i \in w \setminus \Theta_k(A\alpha\$^k)$.

Let $A \to a_i\gamma_i$ and $B \to a_i\delta_i$ be the uniquely determined rules for prefix wb_i. The set of successor nodes to the given parent is given by

(Bi) $$\gamma_i\alpha \underset{a_2 \cdots a_{k-1}b_i}{\approx} \delta_i\beta$$

3. (Replacement Substep). Replace every relation of (Bi) of the form

$$\left\{\sum_{j=1}^{n} [w_j, \eta_j]\right\} \cdot \alpha_i \underset{u_i}{\approx} \delta_i\beta.$$

By (Bi)″

$$\eta_j\alpha_i \underset{u_i}{\approx} \delta_i\beta,$$

where j is the (uniquely determined) index such that $w_j = u_i$.

Justification of the B-step

It is obvious that (B) implies the set (Bi) $i = 1, \ldots, s$. To observe the converse assume that the set of (Bi) holds. We wish to show that $A\alpha \approx_w B\beta$. Let $x \in T^*\$^*$ be any word generated by $A\alpha\$^k$ with prefix w. Then x must be representable as $x = wb_iy$ where $b_i \in w \setminus \Theta_k(A\alpha\$^k)$.

Correspondingly its derivation must start by

$$A\alpha\$^k \Rightarrow a_1\gamma_i\alpha\$^k \overset{*}{\Rightarrow} a_1a_2 \cdots a_{k-1}b_iy.$$

Hence $\gamma_i\alpha\$^k$ generates $a_2 \cdots a_{k-1}b_iy$. By (Bi)

$$\delta_i\beta\$^k \overset{*}{\Rightarrow} a_2 \cdots a_{k-1}b_iy$$

and we have the derivation:

$$B\beta\$^k \Rightarrow a_1\delta_i\beta\$^k \overset{*}{\Rightarrow} a_1 \cdots a_kb_iy = x.$$

All this shows that $w \setminus L(A\alpha\$^k) \subseteq w \setminus L(B\beta\$^k)$. By a symmetric argument we obtain $A\alpha \approx_w B\beta$.

3.2. Splitting

The splitting step is to be applied to relations which contain a word which have

grown undesirably long. Its purpose is to keep the length of the relations bounded. Let, therefore, the parent relation be:

$$(S) \qquad A\gamma \underset{w}{\approx} \psi, \qquad A\gamma \in (N_1 \cup N_2)\cdot N^*, \qquad \psi \in N_2^+, \qquad w \in T^{k-1}$$

where we explicitly require that $\|\gamma\| > k-1$. This requirement implies that w contain no \$ letters.

Choose any $x \in T^*$ which is one of the shortest words derivable from A, which satisfy $|x| \geq k-1$, $x/(k-1) = w$. Obviously, since $A\gamma \Rightarrow^* x\gamma$, the right hand side ψ (if they are to be equivalent under w) must also be able to derive a word with a prefix x. However, it might do it in several different ways, generating different sentential forms, all beginning with x.

Consider therefore, any $w_i \in \Theta_{k-1}(\gamma)$, $|w_i| = k-1$. From the given formula it should also follow that

$$A\gamma \underset{xw_i}{\approx} \psi \qquad \text{(Lemma 2.8)}$$

We can now let both sides derive sentential forms with prefix x, looking ahead at w_i. This derivation must be unique on both sides, and the left hand side must derive $x\gamma$. Let us partition ψ into three parts: Let α be its initial part, which consists of all the variables, participating in the derivation of x for any of the $w_i \in \Theta_{k-1}(\gamma)$. Let ρ be the next $k-1$ variables and β the remaining variables (it is assumed that ψ is also sufficiently long). We can thus rewrite the equivalence as

$$A\gamma \underset{xw_i}{\approx} \alpha\rho\beta.$$

Let the right hand side derive x, looking ahead at w_i, yielding:

$$\alpha\rho\beta \overset{*}{\underset{\text{lm}}{\Rightarrow}} x\delta_i\rho\beta.$$

We have therefore

$$x\gamma \underset{xw_i}{\approx} x\delta_i\rho\beta$$

which is equivalent to

$$(S)_i \qquad \gamma \underset{w_i}{\approx} \delta_i\rho\beta \qquad \text{(Assertion 2)}$$

These equalities are derived for all $w_i \in \Theta_{k-1}(\gamma)$.

Since the equality in $(S)_i$ is significant only for words with prefix w_i, we can restrict the right hand side to derive only such words, and obtain for each i:

$$\gamma \underset{w_i}{\approx} [w_i, \delta_i\rho]\beta \qquad \text{(by Lemma 2.12)}$$

We claim now that these equalities can be summed, omitting the w_i under the equality to get:

(S3) $$\gamma \approx \sum_{w_i \in \Theta_{k-1}(\gamma)} [w_i, \delta_i \rho]\beta.$$

Obviously, any word generated by γ must start with one of the $w_i \in \Theta_{k-1}(\gamma)$, and then must be derivable from $[w_i, \delta_i\rho]\beta$. In the same manner any word derivable from any of the $[w_i, \delta_i\rho]\beta$ is by $(S)_i$ derivable from γ.

If we introduce now a grouped variable (see Definition [2.6])

$$M = \sum_{w_i \in \Theta_{k-1}(\gamma)} [w_i, \delta_i \rho]$$

it is easy to show that

$$M\beta \approx \sum_{w_i \in \Theta_{k-1}(\gamma)} [w_i, \delta_i \rho]\beta.$$

We can thus replace the set of $(S)_i$ by

(S1) $$\gamma \approx M\beta.$$

By substitution of (S1) in (S) for γ we get

(S2)' $$AM\beta \underset{w}{\approx} \alpha\rho\beta$$

or equivalently, by Assertion 2 of Section 5 below

(S2) $$AM \underset{w}{\approx} \alpha\rho.$$

The splitting step can therefore be summarized as follows:

S-step

Let the father node be

(S) $$A\gamma \underset{w}{\approx} \alpha\rho\beta,$$

where

$$\alpha\rho\beta \in N_2^*, \qquad A\gamma \in (N_1 \cup N_2)N^*, \qquad w \in T^{k-1}, \quad \text{and } \|\gamma\| \geq k-1.$$

Choose any $x \in T^*$ such that $A \Rightarrow^* x$, $x/(k-1) = w$, and x is one of the shortest words satisfying these conditions. Then the sons relations consist of the set of equivalences:

$(S)_i$ $$\gamma \underset{w_i}{\approx} \delta_i\rho\beta \quad \text{for each } w_i \in \Theta_{k-1}(\gamma)$$

where

$$\alpha \overset{*}{\Rightarrow}_{xw_i} x\delta_i.$$

And the single node

(S2) $AM \underset{w}{\approx} \alpha\rho$

where

$$M = \sum_{w_i \in \Theta_{k-1}(\gamma)} [w_i, \delta_i\rho].$$

Justification

The fact that $(S)_i$ and (S2) follow from (S) is implied by the derivation process. To show the converse let $(S)_i$ hold for each $w_i \in \Theta_{k-1}(\gamma)$. Then summing up we obtain $\gamma \approx M\beta$. Concatenating β on the right to both sides of (S2) we obtain

$$AM\beta \underset{w}{\approx} \alpha\rho\beta$$

which by substituting $M\beta \approx \gamma$ yields (S).

What is the advantage gained by the splitting step? Obviously, in all equalities of the $(S)_i$ type the left hand side is shortened by one variable, while in (S2) the left hand side is of size 2. It is true that in the process we introduced a new grouped variable M which actually stands for a sum of words but in the discussion on termination we will see that the number of such variables is bounded and so is the length of the words they represent.

To the above two operational rules we add a stopping rule which checks for imbalance between the two sides of an equivalence relation.

3.3. Stopping check

If the right hand side of an equivalence relation is longer than r times the length of the left hand side, stop and report a failure.

r denotes here the maximal length of the shortest terminal word derivable from any variable of N, with a given $k-1$ prefix. Thus, given any variable A of N, and a $k-1$ prefix w there is always a terminal word x, $A \Rightarrow^* x$ such that w is a prefix of x and $|x| \leq r$.

The stopping check should be applied to any node as soon as it is generated.

To justify the stopping rule let the offending relation be

$$A_1 \cdots A_t \underset{w}{\approx} B_1 \cdots B_s$$

where $s > rt$.

We claim that the two sides cannot be equivalent. We can let each of the A_i derive its shortest terminal word under the initial obligation w. By the definition of r there is a terminal word y derivable from $A_1 \cdots A_t$ whose length does not exceed rt. On the other hand all of the B_i are out of N_2 and their derivation rules are in GNF form. Consequently, the right hand side cannot derive a terminal word shorter than s, and in particular cannot derive y.

3.4. Examples

To illustrate the application of the algorithm, we bring two examples.

Example A. In this example there is no need for the splitting step and the checking is completed by using branching alone.

Consider the following two candidates for equivalence. (Both are LL(2).)

$S_1 \to abC \mid acE$ $\quad$ $S_2 \to aX$

$C \to cB \mid c$ $\quad$ $X \to bZ \mid cY$

$E \to bA \mid b$ $\quad$ $Z \to cV \mid c$

$A \to abC$ $\quad$ $Y \to bU \mid b$

$B \to acE$ $\quad$ $V \to aQ$

$U \to aP$

$Q \to cY$

$P \to bZ.$

The comparison tree generated by these grammars is given in Fig. 1. The leaves

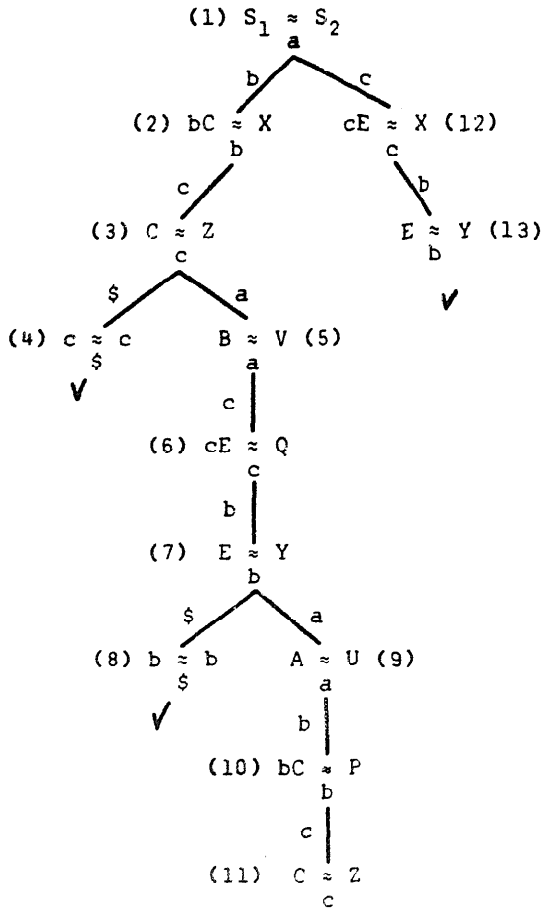

Fig. 1.

4, 8 are terminal identities. The leaves 11,13 contain equalities which were previously generated (nodes 3, 7 respectively) and hence require no further developement.

Unfortunately, we cannot hope that any comparison tree, constructed by branching only, will be finite. Actually, a finite branching tree will be generated for bounded grammars only. Thus in the next example we must use splitting steps to avoid unbounded proliferation of the tree.

Example B. Let us check the equivalence of the two LL(2) grammars:

$G_1 = \langle N_1, T, P_1, S_1 \rangle$; $N_1 = \{S_1, B, D\}$ $T = \{(,)\}$

P_1: $S_1 \to (D$

$D \to (BD \mid) \mid)S_1$

$B \to (BB \mid)$

$G_2 = \langle N_2, T, P_2, S_2 \rangle$; $N_2 = \{S_2, X\}$

P_2: $S_2 \to (X \mid (S_2X$

$X \to) \mid)S_2$.

The tree generated by the algorithm is given in Fig. 2. The leaves 6, 22, 23 are

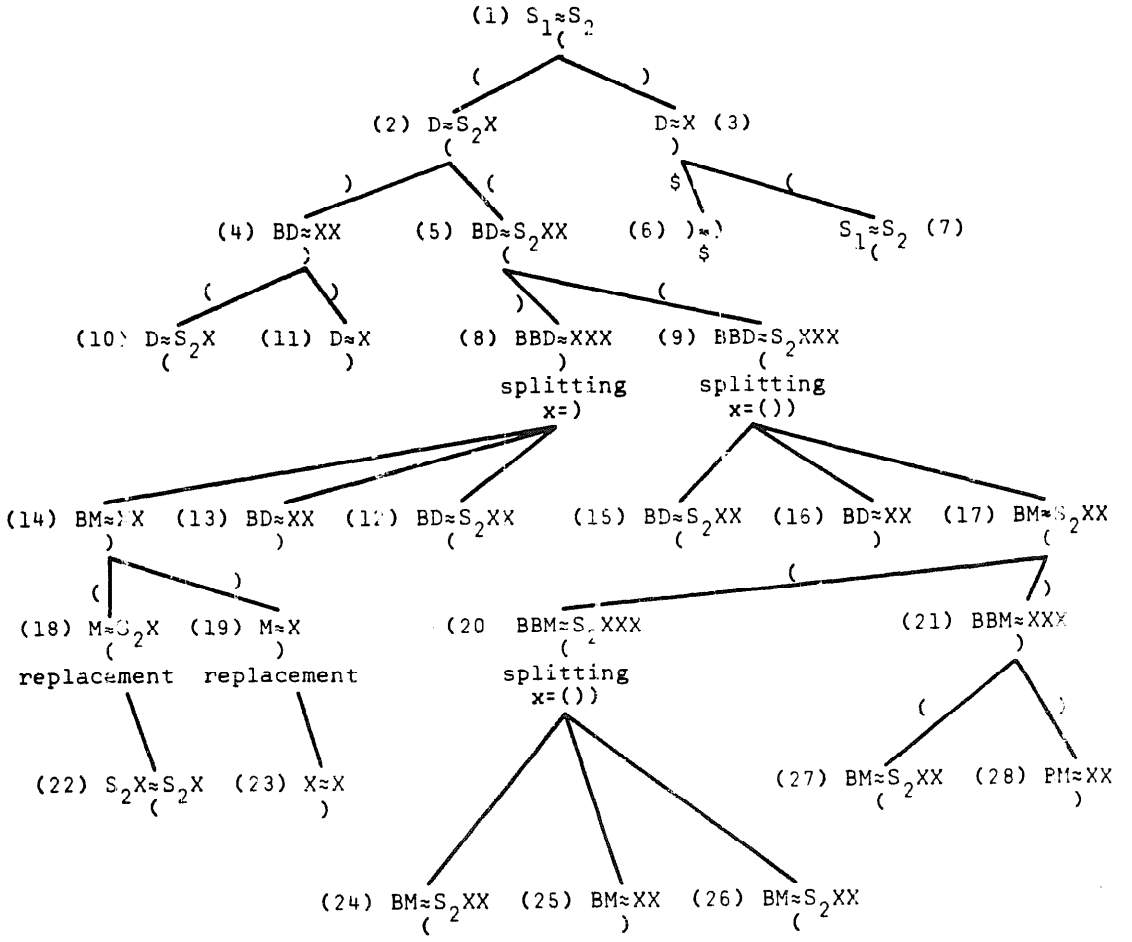

Fig. 2.

identities. The leaves 7, 10, 11, 12, 13, 15, 16, 24, 25, 26, 27, 28 contain equalities, which were previously generated.

Let us follow the process of splitting of node (9): $BBD \approx_{(} S_2XXX$;

$$x = (\,)), \quad \gamma = BD, \quad \alpha = S_2X, \quad \rho = X, \quad \beta = X.$$

$$BBD \overset{*}{\Rightarrow} (\,))BD, \quad S_2XXX \overset{*}{\Rightarrow} (\,))[\,), XX] \text{ and } (\,))[(, S_2XX].$$

We define now $M = [\,), X] + [(, S_2X]$ and get 15, 16, and 17.

Let us follow now the splitting of (8): $BBD \approx_{)} XXX$. Here

$$x =), \quad \gamma = BD, \quad \alpha = X, \quad \rho = X, \quad \beta = X.$$

$$BBD \Rightarrow)BD, \quad XXX \Rightarrow)[\,), XX] \text{ and })[\,(, S_2XX].$$

We've got here the same $M = [\,), X] + [(, S_2X]$ and get 12, 13, and 14.

On the node (18) $M \approx_{(} S_2X$ we apply replacement: we replace M by the summand, whose restriction is "(", i.e. S_2X, and get 22.

As no contradiction has been discovered during the algorithm, the languages have been verified to be equivalent.

4. Correctness and termination of the algorithm

4.1. Correctness

In order to prove the correctness of the algorithm described above, we show that a contradiction is found in at least one of the trees if and only if the grammars are not equivalent. If the algorithm terminates without encountering any contradictions, then we can conclude that the grammars are equivalent.

1. If the grammars are not equivalent, then a contradiction must be found in one of the trees.

Assume to the contrary, that no contradiction was detected during the process. If the grammars are not equivalent, there are some words which separate them, i.e. can be generated by one of the grammars but not by the other.

This means that there exist nodes in the comparison trees which contain invalid relations, i.e. there exist words which separate the left hand side of such a relation from its right hand side. If w separates S_1 from S_2, then if $w \in L(S_1)$ and $w \notin L(S_2)$, $S_1 \approx_u S_2$ with $u = w/(k-1)$ is such a node. so that we have at least one such node.

Among all nodes which have separating words let us pick a node which has a separating word w of a minimal length, i.e. one of the shortest words separating any of the nodes in the trees.

Let the node associated with the word w be

$$\alpha \underset{w_1}{\approx} \beta.$$

We may assume with no loss of generality that it is not a terminal node, i.e. it has successors. Otherwise it has an earlier appearance elsewhere in the tree which has successors and we may consider it instead.

Consider first the case that the node is developed by branching. In this case let $w = a_1 a_2 \cdots a_{k-1} a_k y$ and $w_i = a_1 a_2 \cdots a_{k-1}$. By the successful application of the branching step we know that α and β each have a unique derivation of a word starting with $a_1 \cdots a_k$. Let these derivations be respectively $\alpha \Rightarrow a_1\gamma$, $\beta \Rightarrow a_1\delta$. Thus the considered node has a successor of the form:

$$\gamma \underset{a_2 \cdots a_k}{\approx} \delta.$$

Obviously, if w separates α from β under w_i so will $a_1 \backslash w$ separate γ from δ under $a_2 \cdots a_k$ contradicting w's minimality.

A special case is when $|w| \leq k-1$. Note first, that $w \neq \Lambda$ since the node $\alpha \approx_{w_i} \beta$ is constructed only if both sides can derive each at least one word with prefix w_i. Thus when $w = \Lambda$, $w_i = \$^{k-1}$ and both sides must be identical.

Assuming therefore, that $w = a_1 \cdots a_l$, $l \leq k-1$ we will have $w_i = w\$^{k-1-l}$. As before w will separate the node under w_i only if $a_2 \cdots a_l$ will separate one of its successors under $a_1 \backslash (w_i \$)$.

Consider now the case that the node

$$A\gamma \underset{u}{\approx} \alpha\rho\beta \quad \text{where } u = w/(k-1)$$

is developed by splitting.

The descendants after splitting are:

(S1) $\gamma \approx M\beta$ (represents several nodes).

(S2) $AM \underset{u}{\approx} \alpha\rho$.

If w is supposed to separate $A\gamma$ from $\alpha\rho\beta$, the following possibilities exist:

(a) $A\gamma$ can generate w and $\alpha\rho\beta$ cannot. Let us assume therefore that

$$A\gamma \overset{*}{\Rightarrow} w = vy \quad \text{where } A \overset{*}{\Rightarrow} v \quad \text{and} \quad \gamma \overset{*}{\Rightarrow} y.$$

If now $M\beta$ cannot generate y, we will proceed with node (S1) (or actually the $(S)_i$ corresponding to $w_i = y/(k-1)$), with y as a separating word (shorter than w).

If $M\beta$ does generate y, we get:

$$M\beta \overset{*}{\Rightarrow} y = y_1 y_2 \quad \text{where} \quad M \overset{*}{\Rightarrow} y_1, \qquad \beta \overset{*}{\Rightarrow} y_2,$$

which gives the possibility

$$AM \overset{*}{\Rightarrow} vy_1.$$

If now $\alpha\rho$ cannot generate vy_1, we will proceed with node (S2) with a shorter

separating word vy_1. If $\alpha\rho$ can generate vy_1 then $\alpha\rho\beta \Rightarrow^* vy_1y_2 = w$, which contradicts our original hypothesis that $\alpha\rho\beta$ cannot generate w.

(b) $\alpha\rho\beta$ can generate w but $A\gamma$ cannot. By a similar argument, it can be shown that one of the implied descendant relations (S1) or (S2) is separated by a word shorter that w.

We have thus shown, that if we enter a node with a word known to be separating, then either this node generates a contradiction, which should have been detected in the construction of its descendants or one of its descendants must be separated by a shorter subword of the original separating word. Contradicting the minimality of w.

2. The other direction of the correctness statement claims that if an explicit contradiction arises during construction, then the grammars are inequivalent.

The justification of this claim follows from the fact that anywhere in the constructed trees, the validity of the descendants is equivalent to the validity of the father node. Therefore any contradiction within the tree makes the equivalence formula at the root's node invalid, and hence the equivalence of the grammars invalid.

While the current presentation of the checking alogithm assumed that it was a priori known that the grammars are LL(k) grammars, it is also possible to apply the algorithm to any two arbitrary grammars in GNF form. At any branching step we then verify that for the assumed $w \in (T^*\$^k)/k$ there exists only a single applicable rule on both sides. The algorithm may then terminate successfully, establishing not only the equivalence of the grammars but also proving both to be LL(k) grammars. Alternately the algorithm may fail, showing either that one of the grammars is not an LL(k) grammar, or that the grammars are inequivalent. In order to justify this claim one has to show that all possible combinations of a variable with a right hand context and generated terminal prefix do appear in the comparison tree, and this is indeed the case.

We are thankful to the referee for this paper for suggesting the above observation.

4.2. Termination

We want to show that we can limit the length of the words in the left hand sides of the equalities. The length of those words can increase by applying a branching or a replacement step.

In a branching step, one non-terminal is replaced by a string of non-terminals, which is a right hand side of the derivation rule we have just used. Using the bound on the longest right hand side of any of the production rules, we can bound the amount of increases in each branching step.

The other case of length increase is replacement of a grouped variable M by one of its summands. As is shown in Section 6, the length of these summands is also bounded. Consequently, the length increase in any single step is bounded.

Let us choose any number s (at least $s \geq k$) as a criterion for applying the splitting step, i.e. when we reach a node in which the length of the word on the left hand side of the equality has grown beyond s, we apply the splitting step.

We have already seen, that the left hand sides of the equalities in the nodes generated during splitting are shorter than that of the father's, and if one of the sons has got a left hand side word still longer than s, we simply apply to it another splitting step. Consequently, with the splitting level set at s, and the maximal length increase due to branching and replacements is bounded by b, no left hand side can ever grow beyond $s + b$.

The above rule guarantees boundedness of the left hand sides of the equalities.

By continuous application of the stopping rule (3.3) it is ensured that the length of the right hand sides of the equalities is also bounded. The only thing left to be shown is that the number of variables, in particular the artificial variables generated during the algorithm, is finite. Observing the conditions under which these variables are generated, we see that they are constructed out of variables taken exclusively from N_2.

The grouped variables have the form

$$M = \sum_{w_i \in \Theta_{k-1}(\gamma)} [w_i, \delta_i \rho], \qquad \delta_i, \rho \in N_2^*, \qquad |\rho| = k - 1.$$

The length of each δ_i depends on the grammar only, and can be calculated (see Section 6). It is clearly bounded, and so is the number of items in any such a sum, (not more than $|T|^{k-1}$, which is the number of distinct terminal words of length $k - 1$). Consequently, the number of distinct sums is finite, and so is the number of grouped variables.

We have shown that the number of distinct variables is finite, and that the length of both sides of the equalities is bounded, hence the number of distinct equalities which can be generated in the trees is finite. Since an equality which appears more than once is not developed further, the trees are finite, and the algorithm must terminate.

Because of a wish to keep the exposition as simple as possible, all the bounds are by no means the best possible, and are probably very exaggerated. In all trial cases we made on different grammars, the actual number of distinct aritficial variables was very small. If one insists on deriving a bound on the number of possible nodes in a comparison tree, an upper bound is given by the constant

$$C = (|T| + 1)^{k-1} \cdot |N_2^G| \cdot (|N_1 \cup N_2|)^{(r+1)(b+s)},$$

where b, s, r were given before (r is the maximal length of the shortest terminal word derivable from any variable of N with a given $k - 1$ prefix). For bounding purpose s can be taken as k.

The existence of this bound and a close examination of the comparison algorithm actually establishes the existence of another bound D, dependent only on the

structure of the two grammars, and computable from them, such that if the grammars are inequivalent, there exists a word shorter than D which separates them.

This in priniciple gives an alternative checking algorithm (also implied by the results in [1]), namely check whether any of the words over T shorter than D separate the grammars. However, since our estimate for D is overby pessimistic we believe the comparison algorithm to be far superior.

5. LL(k) words and their properties

Justification of the algorithm depends on the fact that every side in the equalities generated has the LL(k) property. This property states that any word α appearing on any side of an equality can have at most one direct derivation, leading to a terminal word with prefix w for a given w of length k.

The sentential forms of both grammars obviously have the LL(k) property, but during the construction of the trees we get mixed words, consisting of variables from both grammars, including artificial variables. In order to ensure for such words the property of unique branching, we must extend the LL(k) property of the sentential forms to the mixed words generated in our trees.

The more rigorous treatment in this chapter will present a sequence of simple lemmas, leading to the two required main results: (a) Whether we apply a branching, splitting or a replacement step to the father, the validity of the resulting sons is equivalent to the validity of the father node. (b) All mixed words appearing in formulas at all nodes are LL(k) words, and therefore has a unique derivation rule for each k prefix.

In the sequel, when we do not explicitly state otherwise, all words are assumed to be mixed words over $N = N_1 \cup N_2 \cup N_2^G \cup N_2^R$; and all productions used are from $P_1 \cup P_2$. We assume $N_1 \cap N_2 = \emptyset$ and hence a production for a variable in N_i will always be from P_i.

Definition 5.1. A word $\eta \in N^*$ is defined to be an LL(k)-word if for any two derivations

$$\eta \overset{*}{\underset{\text{lm}}{\Rightarrow}} wA\alpha \underset{\text{lm}}{\Rightarrow} w\gamma_1\alpha \overset{*}{\Rightarrow} wx,$$

$$\eta \overset{*}{\underset{\text{lm}}{\Rightarrow}} wA\alpha \underset{\text{lm}}{\Rightarrow} w\gamma_2\alpha \overset{*}{\Rightarrow} wy,$$

such that $x/k = y/k$, it follows that $\gamma_1 = \gamma_2$ (i.e. the rule applied to A is unique).

Note that this is exactly the definition of LL(k) words extended to mixed words.

The following simple properties can be verified for LL(k) words. Some of them follow directly from the definition, the others require a proof which is identical to the proof given for the same property for LL(k) grammars. (See for example Hopcroft and Ullman [10].)

Lemma 5.2. *If $\alpha \in N^*$ is an LL(k) word, and $\alpha \underset{\text{lm}}{\Rightarrow}^* \beta$, then β is also an LL(k) word.*

Lemma 5.3. *An LL(k) word is unambiguous. In other words, if $\eta \underset{\text{lm}}{\Rightarrow}^* x$ and η is an LL(k) word, the derivation is unique.*

Similarly to LL(k) grammars, the notion of unambiguity extends to uniqueness in derivation of two words with equal prefixes.

Lemma 5.4. *For an LL(k) word η which has two derivations*

$$\eta \overset{*}{\underset{\text{lm}}{\Rightarrow}} w_1 A\alpha \underset{\text{lm}}{\Rightarrow} w_1\gamma_1\alpha \overset{*}{\Rightarrow} w_1 x,$$

$$\eta \overset{*}{\underset{\text{lm}}{\Rightarrow}} w_1 A\beta \underset{\text{lm}}{\Rightarrow} w_1\gamma_2\beta \overset{*}{\Rightarrow} w_1 y,$$

such that $x/k = y/k$, it follows that $\gamma_1 = \gamma_2$ and $\alpha = \beta$.

Lemma 5.5. *In a single LL(k) grammar, all the sentential forms (including the initial variable S) are LL(k) words.*

Lemma 5.6. *If $\eta\zeta$ is an LL(k) word, so is η.*

Proof. Let z be any terminal word derivable from ζ. The truth of the lemma follows from the observation that for every two derivations for η:

$$\eta \overset{*}{\underset{\text{lm}}{\Rightarrow}} wA\beta \overset{*}{\Rightarrow} wx,$$

$$\eta \overset{*}{\underset{\text{lm}}{\Rightarrow}} wA\alpha \overset{*}{\Rightarrow} wy,$$

such that $x/k = y/k$, we can consider the two derivations

$$\eta\zeta \overset{*}{\underset{\text{lm}}{\Rightarrow}} wA\beta\zeta \overset{*}{\Rightarrow} wxz,$$

$$\eta\zeta \overset{*}{\underset{\text{lm}}{\Rightarrow}} wA\alpha\zeta \overset{*}{\Rightarrow} wyz \quad \text{for any } z \text{ derivable from } \zeta,$$

where $(xz)/k = (yz)/k$. The uniqueness of the latter implies the uniqueness of the initial derivation. □

Lemma 5.7. *If α is an LL(k) word so is $[w, \alpha]$. Since the derivations of $[w, \alpha]$ are part of the derivations of α, the result follows.*

Lemma 5.8. *If $w\eta$ is an LL(k) word ($w \in T^*$, $\eta \in N^*$), so is η.*

Sometimes it is easier to verify that a word is an LL(k) word by using the auxiliary concept of an LL(k) context:

Definition 5.9. A word $\zeta \in N^*$ is an LL(k) context of the variable $A \in N$ if for every two derivations

$$A\zeta \underset{lm}{\overset{*}{\Rightarrow}} \alpha\zeta \overset{*}{\Rightarrow} w_1,$$

$$A\zeta \underset{lm}{\overset{*}{\Rightarrow}} \beta\zeta \overset{*}{\Rightarrow} w_2,$$

such that $w_1/k = w_2/k$, it follows that $\alpha = \beta$.

We have the following obvious lemma:

Lemma 5.10. *A word η is an LL(k) word if and only if for every derivation*

$$\eta \underset{lm}{\overset{*}{\Rightarrow}} wA\alpha$$

α is an LL(k) context of A.

Lemma 5.11. *If α is an LL(k) context of A, and if $L(\alpha)/(k-1) = L(\beta)/(k-1)$ then β is also an LL(k) context of A.*

Proof. Let $A\beta$ have two derivations:

$$A\beta \underset{lm}{\Rightarrow} \gamma_1\beta \overset{*}{\Rightarrow} w_1w_2 \qquad (\beta \overset{*}{\Rightarrow} w_2)$$

$$A\beta \underset{lm}{\Rightarrow} \gamma_2\beta \overset{*}{\Rightarrow} w_3w_4 \qquad (\beta \overset{*}{\Rightarrow} w_4),$$

and $(w_1w_2)/k = (w_3w_4)/k$.

Since $L(\alpha)/(k-1) = L(\beta)/(k-1)$, there exist two words w_2', w_4' both generated by α such that $w_2/(k-1) = w_2'/(k-1)$, $w_4/(k-1) = w_4'/(k-1)$. As we deal with GNF grammars, $|w_1|, |w_3| \geq 1$ so that the above implies $(w_1w_2')/k = (w_3w_4')/k$. Consequently, we have the two derivations:

$$A\alpha \underset{lm}{\Rightarrow} \gamma_1\alpha \overset{*}{\Rightarrow} w_1w_2'$$

$$A\alpha \underset{\text{lm}}{\Rightarrow} \gamma_2\alpha \overset{*}{\Rightarrow} w_3w_4'$$

which by α being an LL(k) context of A lead to $\gamma_1 = \gamma_2$ as required. □

Corollary. *If $L(\alpha) = L(\beta)$, then if α is an LL(k) context of A, so is β.*

The following sequence of assertions is intended to justify the operations done in the branching and splitting steps of the algorithm and to show that all words generated at tree nodes are LL(k) words.

Assertion 1. Right and left concatenation.
Let $\gamma \in N^*$, $\alpha, \beta \in T^*N^*$, $w \in T^*$.

a. Right concatenation
If $\|\alpha\|, \|\beta\| \geq |w|$ then

$$\alpha \underset{w}{\approx} \beta \supset \alpha\gamma \underset{w}{\approx} \beta\gamma.$$

b. Left concatenation
Given any two words $\alpha, \beta \in N^*$ and a word $x \in T^*$, the following holds:

$$\alpha \underset{w}{\approx} \beta \supset x\alpha \underset{xw}{\approx} x\beta.$$

Assertion 2. Right and left cancellation.
Let $\alpha, \beta \in T^*N^*$, $\gamma \in N^*$, $w \in T^*$, $\alpha\gamma$ and $\beta\gamma$ are LL(k) words, $\|\alpha\|, \|\beta\| \geq |w|$.

a. Right cancellation

$$\alpha\gamma \underset{w}{\approx} \beta\gamma \supset \alpha \underset{w}{\approx} \beta$$

(and from Lemma 5.6 it follows that both α and β are LL(k) words).

Proof. Assume $\alpha \not\approx \beta$, then there exist, for example, words that are in $L(w, \alpha)$ but not in $L(w, \beta)$.

Let x be one of the shortest words separating $L(w, \alpha)$ from $L(w, \beta)$. It is obvious that $x/|w| = w$.

Assume $\alpha \overset{*}{\underset{\text{lm}}{\Rightarrow}} x$ and $\beta \not\overset{*}{\Rightarrow} x$. Let z be one of the shortest words derivable from γ. Obviously, $\alpha\gamma \overset{*}{\underset{\text{lm}}{\Rightarrow}} xz$, and since $\alpha\gamma \approx_w \beta\gamma$, it follows that $\beta\gamma \overset{*}{\underset{\text{lm}}{\Rightarrow}} xz$.

However, since $\beta \not\overset{*}{\Rightarrow} x$, there are only two possibilities:

(1) $\beta \overset{*}{\underset{\text{lm}}{\Rightarrow}} xu$, $|u| > 0$. But then γ must derive $\gamma \overset{*}{\underset{\text{lm}}{\Rightarrow}} v$ such that $uv = z$, and since $|u| > 0$, it follows that $|v| < |z|$ which is impossible, for z was chosen as one of the shortest words derivable from γ.

(2) $\beta \overset{*}{\underset{\text{lm}}{\Rightarrow}} y$ and $\gamma \overset{*}{\underset{\text{lm}}{\Rightarrow}} x'z$ where $yx' = x$, and $|x'| > 0$. Notice that $y/|w| = w$ since $\|\beta\| \geq |w|$.

(a) If we assume $\alpha \Rightarrow^*_{lm} y$, it follows that $\alpha\gamma$ can derive xz in two different ways, contrary to $\alpha\gamma$ being an LL(k) word.

(b) Assume therefore, that $\alpha \not\Rightarrow^* y$. Then y separates α from β. But $|y| < |x|$, and x was chosen as one of the shortest words separating $L(w, \alpha)$ from $L(w, \beta)$, again a contradiction.

The only possibility left is that $\beta \Rightarrow^*_{lm} x$ and $\gamma \Rightarrow^*_{lm} z$, thus $\beta\gamma \Rightarrow^*_{lm} xz$. Hence it is impossible to find a word x such as $x /\!/ w = w$, separating α from β, and it follows that $\alpha \approx_w \beta$. □

b. Left cancellation

We cancel only terminal words.

$$x\alpha \underset{xw}{\approx} x\beta \supset \alpha \underset{w}{\approx} \beta$$

where

$$\alpha, \beta \in N^*, \; xw \in T^*.$$

Proof. Immediate from the definitions. □

Assertion 3. Right substitution.

Let $\alpha, \beta, \gamma, \delta \in N^*$, $w \in T^*$, $\alpha, \beta, \gamma, \delta$ are LL(k) words. Then

(a) $\alpha\beta \approx_w \gamma \cap \beta \approx \delta \supset \alpha\delta \approx_w \gamma$,

(b) $\alpha\delta$ is an LL(k) word.

Proof. (a) $w \backslash (L(\gamma)\$^k) = w \backslash (L(\alpha\beta)\$^k) = w \backslash (L(\alpha)L(\beta)\$^k) = w \backslash (L(\alpha)L(\delta)\$^k) = w \backslash (L(\alpha\delta)\$^k)$. From the definition this implies $\alpha\beta \approx_w \gamma$.

(b) In order to prove that $\alpha\delta$ is an LL(k) word, we prefer to invoke Lemma 5.10 and prove that in any word $wA\eta$ derivable from $\alpha\delta$, η is an LL(k) context of A.

Let $\alpha\delta \Rightarrow^*_{lm} wA\eta$. We distinguish two subcases:

1. $w = w_1 w_2$ where $\alpha \Rightarrow^*_{lm} w_1$, $\delta \Rightarrow^*_{lm} w_2 A\eta$.

In this case we use the LL(k) property of δ to conclude that η is an LL(k) context of A.

2. $\eta = \psi\delta$ and $\alpha \Rightarrow^*_{lm} wA\psi$. Consequently we also have $\alpha\beta \Rightarrow^*_{lm} wA\psi\beta$.

Since $\alpha\beta$ is known to be an LL(k) word, $\psi\beta$ is an LL(k) context of A. Since $\beta \approx \delta$, we get from Lemma 5.11 that $\psi\delta$ is an LL(k) context of A. □

Assertion 4. In the splitting step applied on the equality (S) $A\gamma \approx_w \alpha\rho\beta$ (see Section 3 for details), a grouped variable M is defined as

$$M = \sum_{w_i \in \Theta_{k-1}(\gamma)} [w_i, \delta_i \rho]$$

and we get the equalities

(S1) $\gamma \approx M\beta$

and

(S2) $\quad AM \underset{w}{\approx} \alpha\rho.$

It is also known that $[w_i, \delta_i\rho]$ is an LL(k) word for each i.

Claim. *AM is an LL(k) word.*

We'll show first that M is an LL(k) word.

(a) $\quad M \underset{\text{lm}}{\overset{*}{\Rightarrow}} wE\eta.$

All the derivation rules of M are of the form $M \to [w_i, \delta_i\rho]$, so the derivation $M \Rightarrow^*_{\text{lm}} wE\eta$ can be written as follows:

$$M \Rightarrow [w_j, \delta_j\rho] \underset{\text{lm}}{\overset{*}{\Rightarrow}} wE\eta$$

and as $[w_j, \delta_j\rho]$ is an LL(k) word, η is an LL(k) context for E.

(b) If there exist two derivations

$$M \Rightarrow [w_i, \delta_i\rho] \overset{*}{\Rightarrow} x$$

$$M \Rightarrow [w_j, \delta_j\rho] \overset{*}{\Rightarrow} y$$

and

$$x/k = y/k, \quad \text{then} \quad w_i = w_j,$$

hence $i = j$, and the first step of the derivation is unique.

Let us show now that AM is an LL(k) word.

Assume $AM \Rightarrow^*_{\text{lm}} wE\zeta$.

We distinguish two subcases:

(1) $\quad wE\zeta = w_1w_2E\zeta,$

where

$$A \overset{*}{\Rightarrow} w_1 \quad \text{and} \quad M \underset{\text{lm}}{\overset{*}{\Rightarrow}} w_2E\zeta.$$

In this case ζ is an LL(k) context of E because M is an LL(k) word.

(2) $\quad wE\zeta = wE\eta M,$

where

$$A \underset{\text{lm}}{\overset{*}{\Rightarrow}} wE\eta.$$

We have to show that ηM is an LL(k) context of E. From the above it follows that:

$$AM\beta \underset{\text{lm}}{\overset{*}{\Rightarrow}} wE\eta M\beta.$$

On the other hand we also have

$$A\gamma \underset{\text{lm}}{\overset{*}{\Rightarrow}} wE\eta\gamma.$$

Since $\gamma \approx M\beta$ and $A\gamma$ is an LL(k) word, we get that $\eta\gamma$ is an LL(k) context of E, and, by Lemma 5.11, that $\eta M\beta$ is also an LL(k) context of E. Since $\|M\| \geq k-1$ it is also true that $L(\eta M\beta)/(k-1) = L(\eta M)/(k-1)$, which implies, again through Lemma 5.11, that ηM is also an LL(k) context of E.

Conclusions. We use now the previously established results to prove that all words participating in the constructed trees are LL(k) words.

Consider first the branching process. Assertion 2 implies that if the father contains LL(k) words, so do the sons.

Review next the splitting step. The descendant nodes of type $(S)_i$ contain relations of the form

$$(S)_i \qquad \gamma \underset{w_i}{\approx} \delta_i\rho\beta$$

each derived from a father node of the form

$$A\gamma \underset{xw_i}{\approx} \alpha\rho\beta.$$

Therefore, by assertion 2, both sides of each $(S)_i$ are again LL(k) words.

By summing the equalities $(S)_i$ we get

$$\gamma = \sum_{w_i \in \Theta_{k-1}(\gamma)} [w_i, \delta_i\rho]\beta.$$

We next define a grouped variable $M = \Sigma_{w_i \in \Theta_{k-1}(\gamma)}[w_i, \delta_i\rho]$.

By Assertion 4 AM is an LL(k) word, so that the other descendant node

$$(S1) \qquad AM \underset{w}{\approx} \alpha\rho$$

is also a relation between two LL(k) words.

Another occurrence in the tree construction is the replacement step, in which we replace $M \approx_{w_i} \xi$ by $\delta_i\rho \approx_{w_i} \xi$. $\delta_i\rho$ has been derived from a previous LL(k) word $\alpha\rho\beta$ (the β cancelled on the right) and is therefore an LL(k) word too.

6. Bounds for termination

In this section we will complete the missing details of the termination proof. Its essential part is deriving several bounds on the lengths of generated words and

devising a criterion for application of the splitting step in a way ensuring boundedness of all generated words.

We first analyze the different types of equalities that can arise in the nodes. These types differ by the presence of different alphabets on each side of the equality relation.

The first type is that of $N_1^+ = N_2^+$. Equalities of this type have variables out of N_1 on their left hand side, and variables out of N_2 on their right hand side. The initial equalities $S_1 \approx_w S_2$ are of this type.

Application of branching on type 1 nodes produces again type 1 nodes.

Application of splitting to type 1 nodes generates some nodes (the $(S)_i$ $\gamma \approx_{w_i} \delta_i\rho\beta$ nodes) which are also of type 1, but generates also a type 2 node, described symbolically by $N_1^+N_2^G = N_2^+$. Here N_2^G stands for grouped variables over N_2. These are equalities of the form $AM \approx_w \alpha\rho$ or $\theta M \approx_w \psi$.

Application of branching to type 2 equalities can result in other type 2 equalities, but can also lead to disappearance of all N_1^+ variables from the left hand side. This is type 3 equality of the form $N_2^G = N_2^+$. (Note that from this node on we are exploring interrelations within G_2).

Application of splitting to a type 2 equality will yield equalities $\gamma \approx_w \delta_i\rho\beta$ which can be of types 2 or 3, and to an equality $AM \approx_w \alpha\rho$ which is of type 2.

Equalities of type 3 will have only replacements applied to them, each generating a type 4 equality of the form $N_2^+ = N_2^+$.

By following these various types and their possible transitions under branching, splitting and replacements, we can arrive at the following transition diagram.

Type	Form	Under branching	Under splitting	Under replacement
1	$N_1^+ = N_2^+$	1	1,2	———
2	$N_1^+N_2^G = N_2^+$	2,3	2,3	———
3	$N_2^G = N_2^+$	———	———	4
4	$N_2^+ = N_2^+$	4	4,5	———
5	$N_2^+N_2^G = N_2^+$	3,5	3,5	———

Every grouped variable defines a finite sum of words. We have to show that the length of any element in this sum is bounded.

Denote by r_1 the maximal length of the shortest terminal word derivable from any variable of $N_1 \cup N_2$, with a given $k-1$ prefix. Denote by t the maximal number of variables in the right hand side of any of the production rules of the variables of N_2.

Recollect now how a grouped variable is created: It always happens in a splitting step, when the equality is $A\gamma \approx_w \alpha\rho\beta$. We derive from A a word x, where $|x| \leq r_1$. Any $w_i \in \Theta_{k-1}(\gamma)$ contributes an element to the sum defining the grouped variable.

Now, α is the maximal part of the word in the right hand side of the equality, which participates in the derivation of x for at least one $w_i \in \Theta_{k-1}(\gamma)$, hence $|\alpha| \leq r_1$.

Consider the case of a w_i in which only a part of α participates in the derivation of x. In order to derive x from α, we must apply $|x|$ derivations, i.e. not more that r_1. Every such application can cause δ_i to increase by not more than $t-1$ variables, since we use a production rule for a variable out of N_2, whose right hand side might be of length t.

Consequently, application of r_1 rules may increase δ_i by not more than $(t-1)r_1$ variables, and together with the initial length of α (which is contained in δ_i) we get a bound of $(t-1)r_1 + r_1 - 1$. We know that $|\rho| \leq k-1$, so the length of $\delta_i\rho$ is bounded by

$$k_0 = (t-1)r_1 + r_1 - 1 + k - 1 = t \cdot r_1 + k - 2.$$

This result ensures two important things:

(a) Every summand in a definition of a grouped variable cannot be longer than k_0. This and the fact that such a sum contains a finite number of summands ensures together that the number of distinct grouped variables will be finite. In fact, we can bound N_2^G by

$$|N_2^G| \leq |T|^{k-1} \cdot (|N_2| + 1)^{k_0}.$$

We assume here that every grouped variable is a sum of exactly $|T|^{k-1}$ summands, each of the form $[w, \alpha]$ where $w \in T^{k-1}$, $\alpha \in (N_2 \cup \Lambda)^{k_0}$, and the case in which the grouped variable does not contain a summand $[w_i, \alpha_i]$ for a particular w_i is counted by letting α_i be Λ.

(b) In a replacement step, we get an equality of type 4 ($N_2^+ = N_2^+$) where the left hand side term is not longer than k_0.

In summary we have shown that the number of artificial variables introduced during the process is bounded. Consequently the number of possible distinct nodes in the tree is finite.

Acknowledgements

We would like to thank Dr. Rina Cohen of the Technion, Israel for helpful suggestions and improvements. We would also like to thank the referee for this paper for a very careful reading of the manuscript and numerous helpful corrections.

References

[1] A.J. Korenjak and J.E. Hopcroft, Simple deterministic languages, IEEE Conf. Record of 7th Annual Symposium on Switching and Automata Theory, pp. 36–46.

[2] D.J. Rosenkrantz and R.E. Stearns, Properties of deterministic top down grammars *Information and Control* 17 (3) (1970) 226–256.
[3] E.P. Friedman, Deterministic languages and monadic recursion schemes, Center for Research in Computing Technology, Harvard University (1974).
[4] L.G. Valiant, Decision procedures for families of deterministic pushdown automata, Ph.D. Thesis, Department of Computer Science, University of Warwick, Coventry, England (1973).
[5] E. Ashcroft, Z. Manna and A. Pnueli, Decidable properties of monadic functional schemes, *J. ACM* **20** (3) (1973) 489–499.
[6] Z. Galil, Functional schemas with nested predicates, *Information and Control* 27 (4) (1975) 349–368.
[7] J. Engelfriet, Some program schemes and formal languages, *Lecture Notes in Computer Science*, No 20 (Springer-Verlag, Berlin, 1974).
[8] S.J. Garland and D.C. Luckham, Program schemes, recursion schemes and formal languages, *J. Comp. System Sci.* 7 (1973) 119–160.
[9] M. Nivat, Sur l'interpretation des schemas de programme monadiques, Symposium IRIA — *Automata, Languages and Programming* (North-Holland, Amsterdam, 1973).
[10] J.E. Hopcroft and J.D. Ullman, *Formal Languages and their Relation to Automata* (Addison-Wesley, Reading, MA, 1969).
[11] A.V. Aho and J.D. Ullman, *The Theory of Parsing, Translation and Compiling* (Prentice-Hall, Inc., Englewood Cliffs, NJ, 1972).
[12] T. Olshansky and A. Pnueli, A direct algorithm for checking equivalence of free $N(k)$ schemes, to be published.
[13] B. Courcelle and J. Vuillemin, Completeness results for the equivalence of recursive schemes, *J. Comput. System Sci.* 12 (1976) 179–197.
[14] B. Courcelle, Recursive schemes, algebraic trees and deterministic languages, 15th Annual Symposium SWAT (1974) 52–62.