JOURNAL OF COMPUTER AND SYSTEM SCIENCES 7, 66–96 (1973)

LR-Regular Grammars—an Extension of LR(k) Grammars*

KAREL ČULIK II AND RINA COHEN†

Department of Applied Analysis and Computer Science, University of Waterloo, Waterloo, Ontario, Canada

Received October 16, 1971

LR-regular grammars are defined similarly to Knuth's LR(k) grammars, with the following exception: arbitrarily long look-ahead is allowed before making a parsing decision during the bottom-up syntactical analysis; however, this look-ahead is restricted in that the essential "look-ahead information" can be represented by a finite number of regular sets, thus can be computed by a finite state machine. LR-regular grammars can be parsed deterministically in linear time by a rather simple two-scan algorithm. Efficient parsers are constructed for given LR-regular grammars. The family of LR-regular languages is studied; it properly includes the family of deterministic CF languages and has similar properties. Necessary and sufficient conditions for a grammar to be LR-regular are derived and then utilized for developing parser generation techniques for arbitrary grammars.

INTRODUCTION

In the last decade or so, several classes of context free grammars have been introduced, whose grammars can be deterministically parsed in linear time by a single left-to-right scan. Among these are the Precedence grammars [3], LL(k) grammars [15], Bounded Context and Bounded Right Context grammars [4] and LR(k) grammars [1]. In particular, the LR(k) class is the most general class of grammars of the above type, that can be parsed bottom-up using a left-to-right scan with k symbols look-ahead. It has been generally agreed that for "well designed" programming languages, the above classes of grammars are adequate to specify all of the syntactic features that can be specified by context free grammars; as DeRemer puts it, "if a designer sets out to design an unambiguous CF grammar to specify the "structural properties" of a language, his result will be an LR(k) grammar" [2].

There exist examples of statements in today's programming languages, whose left-to-right analysis may require an unlimited amount of look-ahead (e.g., PL/I statements of the form "IF (...) = ... THEN ..."). But these are usually cases in which

* The research reported here was supported by the National Research Council of Canada, Grants No. A-7403 and A-5255.

† Present address: Department of Computer Science, Technion, Israel Institute of Technology, Haifa, Israel.

the look-ahead is needed during the lexical analysis phase, after which no ambiguity remains and the syntactical analysis may then be carried out in an LR(k) manner. But even if we restrict ourselves to grammars generating the intermediate language obtained after the lexical scan, there are still cases where LR(k) grammars are inadequate. Such cases arise with the use of extendable languages [9, 10], where the user is allowed to extend the syntax of the basic programming language by the so-called "syntax macros", thus taking part in the design of the language. Then we cannot expect the user to extend the syntax always in such a manner as to yield a "well-designed" language. It is therefore important to develop syntactical parsers capable of parsing efficiently as large a class of grammars as possible, including languages in which the structure of a subexpression may depend on an unlimited context both on the left and on the right.

We will now describe a grammar of a simple programming language with the above feature, which is a typical example of a language arising when using syntax macros. Informally, a program in our language is a sequence of (possibly labeled) statements; there are assignment statements as well as conditional **goto** statements. The right-hand side of an assignment statement can either be an arithmetic expression or a set expression. Similarly, the condition in a "jump" statement is a relation either between two arithmetic expressions or between two set expressions. The point in this example is that both arithmetic and set expressions are created from identifiers, constants and operators of the same form; only the context determines whether a given string is to be interpreted as an arithmetic expression (in which case the context is an equals sign "$=$" occuring somewhere in the assignment statement or **if** statement), or a set expression (where the context is an equivalence sign "$\equiv$" occuring somewhere in the assignment, or **if** statement). For instance, 101 could be interpreted either as a binary constant or as a singleton set containing the string 101; the symbol $*$ could mean arithmetic multiplication or concatenation, etc. The structure of an expression depends on its context ("$=$" or "$\equiv$") due to the fact that the same operators have different priorities when used as arithmetic operators or as set operators.

A grammar for the above programming language is given below. In this grammar the nonterminals are of the form $\langle\cdots\rangle$, the set of terminals is $T = \{a, b, 0, 1, +, -, *, =, \equiv, :, ;, (,)$, **if**, **then**, **goto**$\}$, the start symbol is $\langle$program$\rangle$ and the productions are as follows:

$$\langle\text{program}\rangle \rightarrow \langle\text{statement}\rangle \mid \langle\text{program}\rangle; \langle\text{statement}\rangle$$

$$\langle\text{statement}\rangle \rightarrow \langle\text{assign stat}\rangle \mid \langle\text{jump}\rangle \mid \langle\text{ident}\rangle : \langle\text{statement}\rangle$$

$$\langle\text{assign stat}\rangle \rightarrow \langle\text{ident}\rangle = \langle\text{arith exp}\rangle \mid \langle\text{ident}\rangle \equiv \langle\text{set exp}\rangle$$

⟨jump⟩ → **if** ⟨relation⟩ **then goto** ⟨ident⟩

⟨relation⟩ → ⟨arith exp⟩ = ⟨arith exp⟩ |
⟨set exp⟩ ≡ ⟨set exp⟩

⟨arith exp⟩ → ⟨arith exp⟩ + ⟨arith term⟩ |
⟨arith exp⟩ − ⟨arith term⟩ |
⟨arith term⟩

⟨arith term⟩ → ⟨arith term⟩ ∗ ⟨arith primary⟩ |
⟨arith primary⟩

⟨arith primary⟩ → (⟨arith exp⟩) | ⟨ident⟩ | ⟨const⟩

⟨set exp⟩ → ⟨set exp⟩ + ⟨set term⟩ | ⟨set term⟩

⟨set term⟩ → ⟨set term⟩ ∗ ⟨set factor⟩ |
⟨set factor⟩

⟨set factor⟩ → ⟨set factor⟩ − ⟨set primary⟩ |
⟨set primary⟩

⟨set primary⟩ → (⟨set exp⟩) | ⟨ident⟩ | ⟨const⟩

⟨ident⟩ → ⟨ident⟩ ⟨letter⟩ | ⟨letter⟩

⟨const⟩ → ⟨const⟩ ⟨digit⟩ | ⟨digit⟩

⟨letter⟩ → *a* | *b*

⟨digit⟩ → 0 | 1

The above grammar is clearly non-LR(k). However, this grammar is LR-regular—the type of grammars considered in this paper.

We will focus our attention on CF grammars for which the "look-ahead information" essential for determining the handle in any right sentential form can assume only finitely many different values. Then it may be feasible to compute this information in advance by a right-to-left pre-scan of the given string. During the pre-scan, some labels will be attached to the string symbols, representing the auxiliary look-ahead information required later on during the left-to-right "main scan" to perform the parsing. Such a two-scan parsing algorithm may be most appropriate in cases where it is feasible to carry out the lexical scan and the syntactical parsing in reverse directions, for then the pre-scan may be incorporated into the lexical scan.

In what follows, we will be concerned merely with syntactical analysis, i.e., we assume that the given CF grammar generates the intermediate language obtained after the lexical scan.

We shall study in detail a new class of grammars, called LR-regular (abbreviated LRR), which includes all LR(k) grammars as well as many practically interesting non-LR(k) grammars, yet allows the construction of efficient linear-time bottom-up parsers, using a rather simple two-pass parsing scheme as described above. For an LRR grammar, the look-ahead information essential for determining the handle in any right sentential form can be represented by a finite number of regular sets. Thus the parsing procedure calls for a finite-state sequential machine to recognize these sets during the pre-scan and a modified LR(0) parser to perform the actual parsing. The sequential machine reads the given string from right to left, at each step attaching a label representing its current output to the next symbol scanned. The labeled string thus obtained can now be parsed essentially as if the grammar were LR(0), because the modified LR(0) parser uses an extended stack alphabet and whenever a parsing decision would not be unique in the usual LR(0) parser, it is made unique using the auxiliary information attached to the top symbol of the stack.

The parsing method described above applies to all LR(k) grammars as well as to many non-LR(k) grammars or grammars generating non-deterministic CF languages, which cannot be deterministically parsed by a strictly left-to-right process. Even if the given grammar is LR(k) for some larger k, our method may provide a parser more efficient than the optimized LR(k) parser [5]. It may be easier to prepare and store some auxiliary information during the pre-scan than to "look-ahead", particularly in case it is feasible to incorporate the pre-scan into the lexical scan. Our method then can be interpreted as a guide for systematically modifying the intermediate context-free language and its grammar to make it LR(0).

The reader is assumed familiar with the basic notions of language theory and with LR(k) grammars [1]. For general background the reader is referred to [6–8].

All grammars and languages considered in this paper are assumed to be context-free.

1. LRR Grammars

Before proceeding to define LRR grammars, we introduce some notation and review briefly the definitions of context-free grammars and LR(k)-ness. A **context-free grammar** (CFG) is a four-tuple $G = (N, T, P, S)$, where N and T are finite disjoint sets of non-terminals and terminals, respectively, S, in N, is the start symbol and P is a finite set of productions of the form $A \to \alpha$, where A is in N and α in $(N \cup T)^*$. Let $V = N \cup T$ and let the relations $\Rightarrow$ and $\Rightarrow^*$ be defined in the usual way. The language generated by G is $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$.

We may assume without loss of generality that the starting symbol does not occur at the right side of any production in P. We will do it throughout the paper to avoid complications in the definitions of LR(k) and LR(π) grammars otherwise shown to be necessary in [8].

Usually, upper case Latin letters will denote nonterminals, a, b, c will denote terminals, x, y, z, u, v, w will denote terminal strings and small Greek letters will be used for strings in V^*. ϵ will denote the empty string. The length of a string α will be denoted by $|\alpha|$. For any string $x = a_1 a_2 \cdots a_n$, let the **reverse** of x be $x^R = a_n \cdots a_2 a_1$ and let $\epsilon^R = \epsilon$. For a language L, let $L^R = \{x^R \mid x \in L\}$. Let ϕ denote the empty language.

We shall use $\overset{R}{\Rightarrow}$ and $\overset{R}{\Rightarrow}^*$ to denote rightmost derivation, i.e. one in which at each step the rightmost nonterminal is replaced. A **right (canonical) sentential form** is any string α in V^* such that $S \overset{R}{\Rightarrow}^* \alpha$. If $A \to \alpha$ is in P and if $S \overset{R}{\Rightarrow}^* \beta A w \Rightarrow \beta\alpha w$ for some $\beta \in V^*$ and $w \in T^*$, then α is called a **handle** of $\beta\alpha w$.

For any string w and for any integer $k = 0, 1, \ldots$, define $\delta_k(w)$ to be the first k symbols of w, in case $|w| \geqslant k$, or w, in case $|w| < k$. A CFG $G = (N, T, P, S)$ is said to be **LR(k)** if given any two right-most derivations of the form

$$S \overset{R}{\Rightarrow}^* \alpha_1 A_1 y_1 \overset{R}{\Rightarrow} \alpha_1 \gamma y_1 ,$$

and

$$S \overset{R}{\Rightarrow}^* \alpha_2 A_2 y_3 \overset{R}{\Rightarrow} \alpha_1 \gamma y_2 ,$$

where $\alpha_i, \gamma \in V^*$, $y_i \in T^*$ and $\delta_k(y_1) = \delta_k(y_2)$, then we may conclude that $A_1 = A_2$, $\alpha_1 = \alpha_2$, and $y_2 = y_3$.

Let $\pi = \{R_1, \ldots, R_n\}$ denote a partition of T^* into a finite number n of disjoint sets R_i. π is called a **regular partition** of T^* if all sets R_i are regular. If two strings x and y belong to the same set R_i then we write $x \equiv y \pmod{\pi}$. Partition π is said to be a **left (right) congruence** if for any strings x, y, z in T^*, $x \equiv y \pmod{\pi}$ implies $zx \equiv zy \pmod{\pi}$ $(xz \equiv yz \pmod{\pi})$.

We now introduce LRR grammars. Informally, a CFG is LRR if there exists a regular partition $\pi = \{R_1, \ldots, R_n\}$ of T^* such that the handle in any right sentential form is uniquely determined by the string to its left and the set R_i containing the terminal string to its right.

DEFINITION 1.1. Let $\pi = \{R_1, \ldots, R_n\}$ be any regular partition of T^*. A CFG $G = (N, T, P, S)$ is called **LR(π)** if given any two right-most derivations of the form

$$S \overset{R}{\Rightarrow}^* \alpha_1 A_1 y_1 \overset{R}{\Rightarrow} \alpha_1 \gamma y_1 ,$$

and

$$S \overset{R}{\Rightarrow}^* \alpha_2 A_2 y_3 \overset{R}{\Rightarrow} \alpha_1 \gamma y_2$$

and $y_1 \equiv y_2 \pmod{\pi}$, then we may conclude that $A_1 = A_2$, $\alpha_1 = \alpha_2$ and $y_2 = y_3$.

A CFG G is called **LR-regular (LRR)** if G is LR(π) for some regular partition π of T^*. A language L is LRR iff $L = L(G)$ for some LRR grammar G.

Clearly every LR(k) grammar is LRR with respect to the regular partition

$\pi_k = \{\{u_1\},\ldots \{u_n\}, w_1T^* \ldots, w_mT^*\}$, where w_i, $1 \leqslant i \leqslant m$, are all terminal strings of length k and u_i, $1 \leqslant i \leqslant n$, are all terminal strings of length less than k, including the empty string.

EXAMPLE 1.1. The grammar $G = (N, T, P, S)$ of the simple programming language presented in the Introduction is clearly LR(π), where $\pi = \{(T - \{=\})^*\{=\}T^*, (T - \{=\})^*\{\equiv\}T^*, (T - \{=, \equiv\})^*\}$.

In fact grammar G is also LR(π'), where π' is the decomposition of T^* into $(T - \{\equiv\})^*\{=\}T^*$ and its complement; however, decomposition π allows for earlier error detection.

EXAMPLE 1.2. Consider the linear CFG $G = (\{S, T, U\}, \{a, b\}, P, S)$ where $P = \{S \to aTb; S \to bTa; S \to aUa; S \to bUb; T \to aTaa; T \to b; U \to aUa; U \to b\}$. One can see that G is LRR w.r.t. the regular partition $\pi = \{T^*a; T^*b\}$ but G is not LR(k) for any k; moreover, the language

$$L = \{aa^nba^{2n}b, ba^nba^{2n}a; aa^nba^na; ba^nba^nb \mid n = 0, 1, 2,\ldots\},$$

generated by G, is not a deterministic PDA language, nor is its reverse language L^R.

We shall now study the relations of the LRR languages to other families of languages. First we note that, by their definition, LRR grammars are unambiguous. However, the unambiguous language $L = \{ww^R \mid w \in \{a, b\}^*\}$ is clearly not LRR; hence the LRR languages are properly contained in the unambiguous CFL's.

By Examples 1.1 and 1.2, the LRR languages properly contain the deterministic CFL's. Moreover, the LRR language in Example 1.2 is neither a deterministic CFL nor the mirror image (reverse) of one.

Considering reversal, we can also define the symmetric notion of right-to-left regular (RLR) grammar, which corresponds to left-most derivations. Clearly the RLR languages are the mirror images of the LRR languages. As in the case of deterministic languages, these two families are incomparable; moreover, the family of RLR languages is incomparable with the family of deterministic CFL's, as is shown by the following example:

EXAMPLE 1.3. Consider the grammar $G = (\{S, A, B\}, \{a, b\}, P, S)$ where $P = \{S \to AB; S \to BCB: A \to aAbb; A \to abb; B \to aBb; B \to ab; C \to aC; C \to a\}$. The left-most derivations are of the forms:

$$S \overset{L}{\Rightarrow} AB \overset{L}{\Rightarrow}{}^* a^{m-1}Ab^{2m-2}B \overset{L}{\Rightarrow} a^mb^{2m}B \overset{L}{\Rightarrow}{}^* a^mb^{2m}a^{n-1}Bb^{n-1} \overset{L}{\Rightarrow} a^mb^{2m}a^nb^n$$

$$S \overset{L}{\Rightarrow} BCB \overset{L}{\Rightarrow}{}^* a^{m-1}Bb^{m-1}CB \overset{L}{\Rightarrow} a^mb^mCB \overset{L}{\Rightarrow}{}^* a^mb^ma^kB$$

$$\overset{L}{\Rightarrow}{}^* a^mb^ma^ka^{n-1}Bb^{n-1} \overset{L}{\Rightarrow} a^mb^ma^{n+k}b^n$$

and one can see that G is RL(0) (i.e., reverse LR(0)). However, for right-most derivations we have:

$$S \overset{R}{\Rightarrow} AB \overset{R}{\Rightarrow}{}^{*} Aa^nb^n \overset{R}{\Rightarrow}{}^{*} a^{m-1}Ab^{2m-2}a^nb^n \overset{R}{\Rightarrow} a^mb^{2m}a^nb^n$$

$$S \overset{R}{\Rightarrow} BCB \overset{R}{\Rightarrow}{}^{*} BCa^nb^n \overset{R}{\Rightarrow}{}^{*} BCa^{k-1}a^nb^n \overset{R}{\Rightarrow}{}^{*} Ba^{n+k}b^n$$

$$\overset{R}{\Rightarrow}{}^{*} a^{m-1}Bb^{m-1}a^{n+k}b^n \overset{R}{\Rightarrow} a^mb^ma^{n+k}b^n$$

where $m, n, k = 1, 2,...$ and thus the handle in any terminal string cannot be determined without first matching the a's with the b's in the right part of the string and checking whether there are more a's than b's, an operation which cannot be accomplished by a finite automaton. Hence G is not LRR (a formal proof of this will be presented in Section 5). Furthermore, one can show that the language $L(G) = \{a^mb^{2m}a^nb^n, a^mb^ma^{n+k}b^n \mid k, m, n \geqslant 1\}$ cannot be generated by any LRR grammar.

The relations among the various families of grammars and languages shown above are summed up in the following theorem.

THEOREM 1.1. (a) *The family of* LRR [RLR] *grammars properly contains the family of* LR(k) [RL(k)] *grammars, it is incomparable with the family of* RL(k) [LR(k)] *grammars (and thus also with the family of* RLR [LRR] *grammars), and it is properly contained in the family of unambiguous grammars.*

(b) *The family of* LRR [RLR] *languages properly contains the family of deterministic [reverse deterministic] languages, is incomparable with the family of reverse deterministic [deterministic] languages (and thus also with the family of* RLR [LRR] *languages) and is properly contained in the family of unambiguous languages.*

2. A PARSING ALGORITHM FOR LRR GRAMMARS

We now present a construction which yields a practical parsing method for LRR grammars. For each regular partition π of T^* and for each grammar G, a new grammar G_π over an extended alphabet is constructed, such that if G is LR(π) then G_π is LR(0). Every string x generated by G can be converted into a corresponding string x' generated by G_π by a sequential machine which scans x from right to left and yields x' as its output. The parsing tree of x' w.r.t. G_π is essentially the same as that of x w.r.t. G and can be easily obtained by the LR(0) parser of G_π.

DEFINITION 2.1. Let $G = (N, T, P, S)$ be any CFG and let $\pi = \{R_1, ..., R_n\}$

be a regular partition of T^*. Let g_π be the sequential mapping[1] from T^* into $\{1, 2,..., n\}^*$ defined by: $g_\pi(\epsilon) = \epsilon$, $g_\pi(a_1a_2 \cdots a_k) = i_1i_2 \cdots i_k$, where for each j, $1 \leqslant j \leqslant k$, $a_ja_{j-1} \cdots a_1 \in R_{i_j}$. To see that g_π is a sequential mapping, consider the partition $\pi^R = \{R_1{}^R,..., R_n{}^R\}$. Now let π' be the right congruence of T^* with the minimum number of blocks, which is a refinement of π^R. Then π' defines the states and transition function of a sequential machine which can recognize the sets $R_i{}^R$. Associating the output j with each block of π' contained in set $R_j{}^R$, we obtain a minimum-state Moore machine[1] $M_\pi = (K, T, \Delta, \delta, \lambda, q_0)$ realizing the sequential mapping g_π; here K is the set of states, q_0 the initial state, $\Delta = \{1, 2,..., n\}$, $\delta: K \times T \to K$ is the transition function and $\lambda: K \to \Delta$ is the output function. Now define a grammar $G_\pi = (N', T' \cup \{\$\}, P', S')$ as follows: let $\vdash$, \$ be new symbols not in V; let $N' = \{S'\} \cup (K \times (V \cup \{\vdash\}) \times K)$; $T' = (T \cup \{\vdash\}) \times \Delta$ and let P' consist of productions of the following four types:

(i) $S' \to (p, \vdash, p)(p, S, q_0)\$$ [2] for all $p \in K$.

(ii) If $A \to X_1X_2 \cdots X_r$ is in P then

$$(p, A, q) \to (p, X_1, p_1)(p_1, X_2, p_2) \cdots (p_{r-2}, X_{r-1}, p_{r-1})(p_{r-1}, X_r, q)$$

is in P' for any $p, q, p_1,..., p_{r-1}$ in K.

[1] A (nondeterministic) generalized sequential machine (gsm) is a 6-tuple $g = (K, \Sigma, \Delta, \delta, p_1, F)$ where K is a finite set of states, Σ and Δ are the input alphabet and output alphabet, respectively, $p_1 \in K$ is the initial state, $F \subseteq K$ is the set of final states and δ is a mapping from $K \times \Sigma$ into finite subsets of $K \times \Delta^*$. The domain of δ is extended to $K \times \Sigma^*$ as follows: $\delta(p, \epsilon) = \{(p, \epsilon)\}$ and for any x in Σ^* and a in Σ, $\delta(p, xa) = \{(q, w) \mid w = w_1w_2$ and for some q' in K, (q', w_1) is in $\delta(p, x)$ and (q, w_2) in $\delta(q', a)\}$. The gsm mapping g associated with gsm g is a mapping from 2^{Σ^*} into 2^{Δ^*} defined by: $g(L) = \{y \mid (p, y)$ is in $\delta(p_1, x)$ for some $p \in F$ and $x \in L\}$.

A gsm g as above is said to be deterministic if for each state p and for each a in Σ, $\delta(p, a)$ contains exactly one element. For deterministic gsm, we will use a slightly different notation, i.e., the above defined function δ, which, in this case, has range $K \times \Delta^*$, will be separated into two functions, $\delta: K \times \Sigma \to K$ and $\lambda: K \times \Sigma \to \Delta^*$ (the transition function and output function resp.). δ and λ are extended to $K \times \Sigma^*$ in the usual way. Thus a deterministic gsm will be represented by a 7-tuple $(K, \Sigma, T, \delta, \lambda, p_1, F)$. In some cases F will be omitted, and will be understood to be K. A (Mealy type) sequential machine is any deterministic gsm in which the range of λ is Δ. The gsm mapping of a sequential machine is called a sequential mapping. A Moore machine is defined similarly to a Mealy type machine, except that the output function λ is a function from K to Δ. The sequential mapping g of a Moore machine $M = (K, \Sigma, \Delta, \delta, \lambda, p_1)$ is then defined as follows: for any $x = a_1 \cdots a_k$ in Σ^*, $g(x) = \lambda(\delta(p_1, a_1))\ \lambda(\delta(p_1, a_1a_2)) \cdots \lambda(\delta(p_1, a_1 \cdots a_k))$, and $g(L) = \{g(x) \mid x \in L\}$.

[2] The endmarker \$ is inessential for the proof of Theorem 2.1 and has been added just for convenience in later proofs. One could instead assume (without loss of generality) that the partition π has a separate block $\{\epsilon\}$ for the empty string only, thus turning the last symbol $[a, \lambda(q_0)]$ into an endmarker, since $\lambda(q_0)$ would then be distinct from all other outputs occuring inside the string.

(iii) $(p, a, q) \to [a, i]$ for any $a \in T$, $p, q \in K$ and $i \in \Delta$ such that $\delta(q, a) = p$ and $\lambda(q) = i$.

(iv) $(p, \vdash, p) \to [\vdash, i]$ for any $p \in K$, $i \in \Delta$ such that $i = \lambda(p)$.

We note that in the above construction for G_π, for any string $x \in L(G)$ there corresponds a "modified version" $f_\pi(x) \in L(G_\pi)$ of x which is obtained by attaching to the original symbols of x additional "output labels" corresponding to the output sequence produced by M_π when scanning the string x from right to left. The last output produced by M_π is attached to the special begin symbol $\vdash$ added in front of x.

Formally, define the function $f_\pi: T^* \to (T')^*\$$ as follows: For any $x = a_1 a_2 \cdots a_k \in T^*$, let $y = g_\pi(x^R) = i_1 i_2 \cdots i_k \in \Delta^*$. Define

$$f_\pi(x) = [\vdash, i_k][a_1, i_{k-1}] \cdots [a_{k-1}, i_1][a_k, i_0]\$,$$

where $i_0 = \lambda(q_0)$, $i_1 = \lambda(\delta(q_0, a_k))$, and $i_j = \lambda(\delta(q_0, a_k \cdots a_{k-j+1}))$ for $2 \leqslant j \leqslant k$.

Clearly f_π is a 1-1 gsm mapping whose inverse can be extended to a homomorphism $h_\pi: (T')^*\$ \to T^*$ defined by $h_\pi([a, i]) = a$ and $h_\pi([\vdash, i]) = h_\pi(\$) = \epsilon$, for any $a \in T$ and $i \in \Delta$. Let R_π denote the range of f_π, i.e., $R_\pi = f_\pi(T^*) \subset (T')^*\$$; then f_π^{-1} is the restriction of h_π to R_π. Hence h_π is 1-1 on R_π. One can easily verify that for any grammar G over terminal alphabet T, $f_\pi(L(G)) = L(G_\pi)$ and $h_\pi(L(G_\pi)) = L(G)$. Let us extend h_π to a homomorphism from $(V')^*$ into V^* by defining

$$h_\pi(S') = S$$

$$h_\pi(p, X, q) = X \text{ for all } X \in V,\ p, q \in K$$

and

$$h_\pi(p, \vdash, q) = \epsilon \text{ for all } p, q \in K.$$

Remark. We note that if a grammar G is LR(π) for some partition π, then it is also LR(π') for any refinement π' of π. It is well known that any partition π of T^* has a refinement which is a left congruence [6, 14]. Therefore for any LR(π) grammar, we may assume without loss of generality that π is a left congruence.

Note also that in the case that π is a left congruence, the Moore machine M_π defined above has distinct outputs for distinct states.

THEOREM 2.1. *Let $G = (N, T, P, S)$ be a* CFG *and π be a left congruence on T^*. Then G is* LR(π) *iff the grammar G_π defined above is* LR(0).

Proof. (a) Suppose G is LR(π). To show that G_π is LR(0), consider the following two derivations in G_π:

$$(*) \qquad \begin{aligned} S' &\overset{R}{\Rightarrow}{}^* \alpha_1'(p_1, A_1, q_1)y_1' \overset{R}{\Rightarrow} \alpha_1'\gamma' y_1' \\ S' &\overset{R}{\Rightarrow}{}^* \alpha_2'(p_2, A_2, q_2)y_3' \overset{R}{\Rightarrow} \alpha_1'\gamma' y_2' \end{aligned}$$

where $\alpha_i', \gamma' \in (V')^*$, $y_i' \in (T')^*\$$, $A_i \in N$ and $p_i, q_i \in K$. For G_π to be LR(0) we must have $\alpha_1' = \alpha_2'$, $(p_1, A_1, q_1) = (p_2, A_2, q_2)$ and $y_3' = y_2'$. Let us consider the corresponding derivations in G, namely,

$$(**) \qquad \begin{aligned} S &\overset{R}{\Rightarrow}{}^* \alpha_1 A_1 y_1 \overset{R}{\Rightarrow} \alpha_1 \gamma y_1 \\ S &\overset{R}{\Rightarrow}{}^* \alpha_2 A_2 y_3 \overset{R}{\Rightarrow} \alpha_1 \gamma y_2 \end{aligned}$$

where α_i, y_i and γ are obtained from α_i', y_i' and γ' resp. by replacing each nonterminal (p, X, q) in N' by X, each terminal $[a, i]$, $a \in T$, of G_π by a, and erasing the endmarkers. We claim that $y_1 \equiv y_2 (\mathrm{mod}\ \pi)$. To see this, observe that if we have in G_π $S' \overset{R}{\Rightarrow}{}^* \eta(p, X, q)w\$$ for some $w \in (T')^*$, $\eta \in (V')^*$, $p, q \in K$ and $X \in N$, then as one can show by simple induction, $\delta(q_0, h_\pi(w^R)) = q$. It follows that $h_\pi(w) \in R_i$ where $i = \lambda(q)$. From the above derivations (*) and (**) we obtain $y_1, y_2 \in R_j$, where j is the output associated with the last component of the last symbol of $\alpha_1'\gamma'$ (this string is nonempty by construction of G_π). Hence $y_1 \equiv y_2 (\mathrm{mod}\ \pi)$ and since G is LR(π) we have $A_1 = A_2$, $\alpha_1 = \alpha_2$ and $y_3 = y_2$. Since M_π is a deterministic machine whose states are in one-to-one correspondence with its outputs, strings $\alpha_1', \alpha_2', y_2', y_3'$ are uniquely determined by strings $\alpha_1, \alpha_2, y_2, y_3$ and therefore $\alpha_1' = \alpha_2'$, $y_3' = y_2'$ and $(p_1, A_1, Q_1) = (p_2, A_2, q_2)$ as required.

(b) Now suppose G_π is LR(0) and assume that π is a left congruence on T^*. This implies that in the corresponding Moore machine M_π distinct states have distinct outputs. Thus consider two derivations in G as in (**) above and suppose $y_1 \equiv y_2 (\mathrm{mod}\ \pi)$. Then in M_π we have $\delta(q_0, y_1{}^R) = \delta(q_0, y_2{}^R) = q_j$ for some $q_j \in K$ such that $\lambda(q_j) = j$. Now construct two corresponding derivations in G_π:

$$\begin{aligned} S' &\overset{R}{\Rightarrow}{}^* \alpha_1'(p_1, A_1, q_1)y_1' \Rightarrow \alpha_1'\gamma' y_1' \\ S' &\overset{R}{\Rightarrow}{}^* \alpha_2'(p_2, A_2, q_2)y_3' \Rightarrow \alpha_1''\gamma'' y_2' \end{aligned}$$

where y_1' and y_2' are obtained from y_1 and y_2 resp. by adding endmarker \$ and simulating the behavior of M_π on the sequences $y_1{}^R$ and $y_2{}^R$. Since $\delta(q_0, y_i) = q_j$, $i = 1, 2$, the symbol immediately preceding $y_1'(y_2')$ in $\alpha_1'\gamma' y_1'(\alpha_1''\gamma'' y_2')$ must be of the form (p, X, q_j) or, if it is terminal, of the form $[a, j]$, for some $a \in T \cup \{\vdash\}$, $X \in V \cup \{\vdash\}$ and $p \in K$. Thus the last symbol of both $\alpha_1'\gamma'$ and $\alpha_1''\gamma''$ will have last component q_j or j, and since both strings are obtained from $\alpha_1\gamma$ by adding some state sequences and outputs in M_π to the symbols of $\alpha_1\gamma$ and adding an extra symbol $(p, \vdash, p)$ at the beginning, $\alpha_1'\gamma'$ and $\alpha_1''\gamma''$ can be made equal by choosing the same state sequences for both. Then we have $\gamma' = \gamma''$, $\alpha_1' = \alpha_1''$ and the above derivations in G_π are of the form (*). Since by assumption G_π is LR(0) we get $\alpha_1' = \alpha_2'$, $y_2' = y_3'$ and

$(p_1, A_1, q_1) = (p_2, A_2, q_2)$. Consequently, $\alpha_1 = \alpha_2$, $y_2 = y_3$ and $A_1 = A_2$. This concludes the proof.

Utilizing Theorem 2.1, efficient parsers for LRR grammars can be constructed. For a given LR(π) grammar $G = (N, T, P, S)$, a parser will consist of the deterministic Moore machine M_π (constructed in Definition 2.1) and an LR(0) parser for the modified grammar G_π. The sequential machine will process the input string x from right to left during the pre-scan; at each step, after having scanned a symbol, it will attach a label representing its current output to the next symbol scanned. The labeled string obtained after completion of the pre-scan (i.e., $f_\pi(x)$) can now be parsed by the LR(0) parser of G_π, yielding essentially the same parsing tree as that of the original string x with respect to G.

Optimization

The above parsing scheme works due to the fact that the auxiliary labels attached to the input string symbols during the prescan represent the "look ahead" information essential for determining the (unique) handle in x or any subsequent right sentential forms occuring during the bottom-up parsing of x. The modified LR(0) parser maintains these auxiliary labels on its stack and uses them for determining the reductions to be performed during the parsing. However, some of these labels may actually never be used during the parsing process and can therefore be omitted. In particular, we need labels only in the situations when a parsing decision cannot be done in LR(0)-manner. Therefore we can describe, independently on a decomposition π, subset T' of terminal alphabet T containing all the symbols which need to be labeled. We will do it formally using the notion of characteristic finite state machine (CFSM) from [2, p. 44]. Given $G = (N, T, P, S)$ let $T' = \{a \in T \mid$ there is X in $N \cup \{a\}$ such that $X \Rightarrow^* wa$ for some w in V^* and vX transfers the CFSM of G from the starting state into an inadequate state for some $v \in V^*\}$.

3. Properties of LRR Languages

Theorem 2.1 also has some interesting theoretical implications. First we note that the theorem provides an algorithm for deciding whether an arbitrary grammar G is LR(π) for some given left congruence π on T^*. One simply constructs the grammar G_π and checks to see whether or not it is LR(0). However, this algorithm applies only in case the given partition π is a left congruence. In Section 5 we will derive another algorithm which works also if π is not a left congruence.

Theorem 2.1 reflects the close relationship between LRR languages and deterministic languages.

COROLLARY 3.1. *Every* LR(π) *language can be obtained by homomorphism* h_π (*see Definition* 2.1) *from a deterministic language.*

Proof. Let $L = L(G)$ for some LR(π) grammar G. Then $L = h_\pi(L(G_\pi))$, where G_π is as in Definition 2.1, and by Theorem 2.1, $L(G_\pi)$ is a deterministic language.

Theorem 2.1 allows us to generalize many of the known results on deterministic CF languages to LRR languages. In particular, the LRR languages have similar closure properties, and also form an "AFDL" (Abstract Family of Deterministic Languages [12]).

LEMMA 3.2. *Let* π *be a left congruence on* T^* *and let* $G' = (N, T' \cup \{\$\}, \bar{P}, S)$ *be any grammar generating a language* $L(G') \subseteq R_\pi$, *where* T', R_π, \$ *and* h_π *are as in Definition* 2.1. *Let* $G = (N, T, P, S)$ *be the grammar obtained from* G' *by applying the homomorphism* h_π *to all terminal symbols occuring in the productions of* G', *and leaving the nonterminals unchanged, i.e., if* $A \to X_1X_2 \cdots X_m$ *is a production in* $\bar{P}$, *then* $A \to Y_1Y_2 \cdots Y_m$ *is in* P, *where for* $1 \leqslant i \leqslant m$, $Y_i = X_i$ *if* $X_i \in N$ *and* $Y_i = h_\pi(X_i)$ *if* $X_i \in T' \cup \{\$\}$[3]. *If* G' *is* LR(0) *then* G *is* LR(π).

Proof. Clearly $h_\pi(L(G')) = L(G)$ and since $L(G') \subseteq R_\pi$ we also have $L(G') = f_\pi(L(G))$. Now construct the grammar $G_\pi = (N', T' \cup \{\$\}, P', S')$ for G, as in Definition 2.1. Clearly $L(G_\pi) = f_\pi(L(G)) = L(G')$. Moreover, we claim that G_π is also LR(0). To see this, recall that G_π has productions of types (i)–(iv) defined in Definition 2.1 above; in particular, each production of type (ii) of G_π is obtained from a production $A \to Y_1Y_2 \cdots Y_m$[3] of G, which, in turn, corresponds to a production $A \to X_1X_2 \cdots X_m$ of G' such that $Y_i = h_\pi(X_i)$ if $X_i \in T'$ and $Y_i = X_i$ otherwise. Now consider any right sentential form η of G_π; one can see immediately that if $\eta \in (T')^*\$$ then the handle in η is the first symbol $[\vdash, i]$, which is to be replaced by $(p, \vdash, p)$ as in production (iv). Similarly, a handle corresponding to production (i) can be readily recognized and will occur only in the last step of the parsing. Thus suppose η contains at least one nonterminal and is not identical with the right-hand side of production (i). Then the handle in η can still be determined without look-ahead in the following way. Convert η into the corresponding right sentential form η' of G', by replacing each nonterminal $(p, a, q) \in K \times (T \cup \{\vdash\}) \times K$ by $[a, i]$, where $i = \lambda(q)$, and each $(p, A, q) \in K \times N \times K$ should be replaced by A. Then find the handle α' in η' w.r.t. G' (which can be determined without look-ahead, since G' is LR(0)), and let α be the corresponding substring of η. If α contains any terminal symbol $[a, i]$, then the handle in η is the left-most such terminal in α, which is to be reduced to (p, a, q), where $\lambda(q) = i$ (q is unique because π is a left congruence) and $p = q$ if $a = \vdash$, else $p = \delta(q, a)$. If α is made exclusively of nonterminal symbols, then the handle in η is α itself and the reduction to be used is of type (ii) and is determined

[3] Note that Y_1 and/or Y_m may be ϵ.

by the production corresponding to handle α' in η'. It follows that G_π is also an LR(0) grammar, and by Theorem 2.1, G is LR(π).

THEOREM 3.3. *The family of* LRR *languages is closed under complementation. In particular, if language* L *is* LR(π) *for some left congruence* π *on* T^*, *then* $T^* - L$ *is also* LR(π).

Proof. Let L be generated by an LR(π) grammar G for some left congruence π on T^*, and let G_π be the grammar constructed in Definition 2.1. Then G_π is LR(0); hence $L(G_\pi)$ is a deterministic CFL. Since deterministic CFL's are closed under complementation and intersection with regular sets, also the language

$$L' = ((T' \cup \{\$\})^* - L(G_\pi)) \cap R_\pi$$

is a deterministic CFL, and since L' is a subset of $(T')^*\$$, it can be generated by an LR(0) grammar, say $G' = (\bar{N}, T' \cup \{\$\}, \bar{P}', \bar{S})$. Let $\bar{G} = (\bar{N}, T, \bar{P}, \bar{S})$ be the grammar obtained from G' by applying the homomorphism h_π to all terminal symbols occuring in the productions of G', as in Lemma 3.2. Since G' is LR(0), $\bar{G}$ is LR(π) by this lemma. Furthermore, using the fact that h_π is 1-1 on R_π, one can easily verify that $L(\bar{G}) = h_\pi(L') = T^* - L$; hence $T^* - L$ is also LR(π).

THEOREM 3.4. *The family of* LRR *languages is closed under intersection with regular sets. In particular, for any fixed left congruence* π *of* T^*, *the family of all* LR(π) *languages is closed under intersection with regular sets.*

Proof. Let $R \subseteq T^*$ be any regular set and let $G = (N, T, P, S)$ be an arbitrary LR(π) grammar for some left congruence π of T^*. Let $G_\pi = (N', T' \cup \{\$\}, P', S')$ be the LR(0) grammar and f_π the gsm mapping defined in Definition 2.1. Let $R' = f_\pi(R) \subseteq R_\pi$; since f_π is a gsm mapping, R' is also regular. Since deterministic CFL's are closed under intersection with regular sets, the language $L' = L(G_\pi) \cap R'$ is also a deterministic CFL, and since $L' \subseteq (T')^*\$$, L' is generated by an LR(0) grammar, say $G' = (\bar{N}, T' \cup \{\$\}, \bar{P}', \bar{S})$. Let $\bar{G} = (\bar{N}, T, \bar{P}, \bar{S})$ be the grammar obtained from G' by applying the homomorphism h_π to all productions of G', as in Lemma 3.2. By this latter lemma $\bar{G}$ is an LR(π) grammar. Furthermore, using the fact that h_π is 1-1 on R_π and is the inverse of f_π, we get:

$$L(\bar{G}) = h_\pi(L(G')) = h_\pi(L(G_\pi) \cap R') = h_\pi(L(G_\pi)) \cap h_\pi(R') = L(G) \cap R$$

as required.

One can show, using the same proofs as for deterministic languages [11], that the family of LRR languages is also not closed under most of the standard operations, like union, concatenation, star, reversal and homomorphism. (In all these proofs, the non-

deterministic language obtained is $L = \{a^i b^i a^j \mid i, j \geqslant 1\} \cup \{a^i b^j a^j \mid i, j \geqslant 1\}$ which is an inherently ambiguous language, and therefore also non-LRR). However, the family of LRR languages is closed under the "marked" operations.

DEFINITION 3.1. For any two languages $L_1, L_2 \subseteq T^*$ and for any two symbols a, $b \notin T$, *the set* $L_1 a L_2$ *is the* **marked product** of L_1 and L_2, $aL_1 \cup bL_2$ is the **marked union** of L_1 and L_2 and $(aL_1)^*$ is the **marked** * of L_1.

THEOREM 3.5. *The family of* LRR *languages is closed under marked union and marked* *.

Proof. Let $L_i = L(G_i)$, where $G_i = (N_i, T, P_i, S_i)$ is an LR(π_i) grammar, $i = 1, 2$, for regular partitions $\pi_1 = \{U_1, ..., U_n\}$ and $\pi_2 = \{V_1, ..., V_m\}$ of T^*.

For marked union, define the grammar $G = (N', T', P, S)$, where $T' = T \cup \{a, b\}$, $N' = N_1 \cup N_2 \cup \{S\}$ and $P = P_1 \cup P_2 \cup \{S \to aS_1, S \to bS_2\}$. Then clearly $L(G) = aL_1 \cup bL_2$ and one can easily verify that G is an LR(π) grammar, for

$$\pi = \{U_i \cap V_j \mid 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m\} \cup \{aT^*, bT^*, ((T')^* - \{a, b, \epsilon\}T^*)\}.$$

For marked *, let $G' = (N'', T'', P', S')$ where $T'' = T \cup \{a\}$, $N'' = N_1 \cup \{S', S''\}$ and $P' = P_1 \cup \{S' \to S'', S'' \to aS_1S'', S'' \to \epsilon\}$. Then $L(G') = (aL_1)^*$ and clearly G' is LR(π') where $\pi' = \{U_1 a(T'')^*, ..., U_n a(T'')^*, U_1, ..., U_n\}$.

We now define a "marked gsm", which is an extension of the notion of a deterministic gsm to that of a machine with a right end-marker [12].

DEFINITION 3.2. Let $\mathfrak{c}$ be an abstract symbol. A **marked generalized sequential machine** (mgsm) is a 6-tuple $g = (K, \Sigma, \Delta, \delta, \lambda, p_1)$, where

(1) K, Σ, Δ and p_1 are as defined for a gsm[1]

(2) $\Sigma \cup \Delta$ does not contain $\mathfrak{c}$

(3) δ is a mapping from $K \times (\Sigma \cup \{\mathfrak{c}\})$ into K (next state function) and

(4) λ is a mapping from $K \times (\Sigma \cup \{\mathfrak{c}\})$ into $\Delta^* \cup \Delta^*\mathfrak{c}$ such that $\lambda(K, \Sigma) \subset \Delta^*$ and $\lambda(K, \mathfrak{c}) \subset \Delta^*\mathfrak{c}$ (output function).

The functions δ and λ are extended to mappings from $K \times (\Sigma \cup \{\mathfrak{c}\})^*$ as for a gsm. The **mgsm mapping** g associated with mgsm g is a mapping from 2^{Σ^*} into 2^{Δ^*} defined by $g(L) = \{y \mid \lambda(p_1, x\mathfrak{c}) = y\mathfrak{c}$ for some $x \in L\}$. The function from 2^{Δ^*} into 2^{Σ^*} defined by $g^{-1}(L') = \{x \mid \lambda(p_1, x\mathfrak{c}) \in L'\mathfrak{c}\}$ is called an **inverse mgsm mapping.**

DEFINITION 3.3[12]. An abstract family of deterministic languages (AFDL) is a family of languages closed under (1) marked union, (2) marked *, and (3) inverse mgsm mapping.

In order to show that the family of LRR languages forms an AFDL, it remains to establish its closure under inverse mgsm mapping.

THEOREM 3.6. *The family of* LRR *languages is closed under inverse* mgsm *mapping.*

Proof. Let L be any LRR language; then L is generated by some LR(π) grammar $G = (N, T, P, S)$ for some regular partition $\pi = \{R_1, ..., R_n\}$ of T^* and we may assume without loss of generality that π is a left congruence. Let $L_\pi = f_\pi(L) \subseteq (T')^*\$$, where f_π and T' are as in Definition 2.1; by Theorem 2.1 L_π is a deterministic language. Now consider any arbitrary mgsm $g = (K, \Sigma, T, \delta, \lambda, p_1)$, and let $K = \{p_1, ..., p_t\}$. Define a relation π' on Σ^* as follows: for any two strings u, u' in Σ^*, $u \equiv u' (\text{mod } \pi')$ if and only if for every $1 \leqslant j \leqslant t$, $\lambda(p_j, u) \equiv \lambda(p_j, u') \pmod{\pi}$. Clearly π' is a left congruence, whose equivalence classes are of the form

$$R_{(i_1,...,i_t)} = \{u \in \Sigma^* \mid \lambda(p_j, u) \in R_{i_j} \text{ for each } j = 1, ..., t\},$$

where $1 \leqslant i_1, ..., i_t \leqslant n$. We will now construct an LR(π') grammar generating $g^{-1}(L)$.

Define a new mgsm $g' = (K, \Sigma', T' \cup \{\$\}, \delta', \lambda', p_1)$, where K, p_1 are as for g, $T' \cup \{\$\}$ is the alphabet of L_π, $\Sigma' = (\{\vdash\} \cup \Sigma) \times \{1, 2, ..., n\}^t$, δ' is defined by $\delta'(p_j, (a, i_1, ..., i_t)) = \delta(p_j, a)$, $\delta'(p_j, (\vdash, i_1, ..., i_t)) = p_j$, $\delta'(p_j, \text{¢}) = \delta(p_j, \text{¢})$, and λ' is defined by $\lambda'(p_j, (a, i_1, ..., i_t)) = (b_1, k_1) \cdots (b_s, k_s) \in (T')^*$ where $\lambda(p_j, a) = b_1 \cdots b_s$, $k_s = i_z$, where $p_z = \delta(p_j, a)$, $b_s R_{k_s} \subseteq R_{k_{s-1}}$ and for each

$$r = 1, ..., s - 2, b_{s-r} \cdots b_s R_{k_s} \subseteq R_{k_{s-r-1}};$$

if $\lambda(p_j, a) = \epsilon$ then define $\lambda'(p_j, (a, i_1, ..., i_t)) = \epsilon$; also let $\lambda'(p_j, (\vdash, i_1, ..., i_t)) = (\vdash, i_j)$ and $\lambda'(p_j, \text{¢}) = (b_1, k_1) \cdots (b_s, k_s)\$\text{¢}$, where $\lambda(p_j, \text{¢}) = b_1 \cdots b_s\text{¢}$, $k_1, ..., k_s$ are defined by: $k_s = i_0$, where R_{i_0} is the regular set of π containing the empty string ϵ, and for each $r = 0, 1, ..., s-2, b_{s-r} \cdots b_s \in R_{k_{s-r-1}}$; in case $\lambda(p_j, \text{¢}) = \text{¢}$ let $\lambda'(p_j, \text{¢}) = \$\text{¢}$. In the above, $a \in \Sigma$, $b_i \in T$, $p_j \in K$ and $k_i \in \{1, 2, ..., n\}$.

Now consider the language $L' = (g')^{-1}(L_\pi)$. Since L_π is deterministic and deterministic languages are preserved by inverse mgsm mappings [11], L' is also a deterministic language. Hence the language $L'\$$ is also a deterministic language and is generated by some LR(0) grammar $G' = (N', \Sigma' \cup \{\$\}, P', S')$. Let $\bar{G} = (N', \Sigma, \bar{P}, S')$ be the grammar obtained from G' in the following way. Let $h_{\pi'}$ be the homomorphism from $(\Sigma' \cup \{\$\})^*$ into Σ^* defined by $h_{\pi'}((a, i_1, ..., i_t)) = a$ and $h_{\pi'}((\vdash, i_1, ..., i_t)) = h_{\pi'}(\$) = \epsilon$, for each $a \in \Sigma$ and $1 \leqslant i_1, ..., i_t \leqslant n$. Let $\bar{P}$ be obtained from P' by applying the homomorphism $h_{\pi'}$ to all terminal symbols in the productions of P' and leaving the nonterminals unchanged, as in Lemma 3.2. One can easily verify that $L(\bar{G}) = h_{\pi'}(L'\$) = g^{-1}(L)$. Furthermore, since the regular sets of the partition π' are represented by the t-tuples $(i_1, ..., i_t)$, the homomorphism $h_{\pi'}$ just defined is clearly the same as the one in Definition 2.1 with respect to partition π' and alphabet Σ, and the language $L'\$$

is contained in $f_{\pi'}(\Sigma^*) = R_{\pi'}$ where $f_{\pi'}$ and $R_{\pi'}$ are again as defined in Definition 2.1. Lemma 3.2 therefore applies to G' and $\bar{G}$ with respect to alphabets Σ, Σ' and the left congruence π', and we may deduce that $\bar{G}$ is an LR(π') grammar. Hence $g^{-1}(L) = L(\bar{G})$ is an LR(π') language.

COROLLARY 3.7. *The family of* LRR *languages is an* AFDL.

Using some general results about AFDL's (Theorem 1.1 in [12]), we obtain:

COROLLARY 3.8. *The family of* LRR *languages is also closed under the operations of inverse* gsm *mapping, marked product, and left, or right quotient over a single word.*

We can also generalize some well-known decidability results on deterministic languages to LRR languages. In particular, we have:

THEOREM 3.9. *It is decidable whether a given* LRR *language is regular.*

Proof. Let L be any LRR language generated by some LR(π) grammar $G = (N, T, P, S)$ for some left congruence π on T^*. Let G_π be the LR(0) grammar constructed in Definition 2.1. Then $L(G_\pi) = f_\pi(L)$ and $L = h_\pi(L(G_\pi))$, where f_π and h_π are a gsm mapping and a homomorphism, respectively. Consequently, by the closure of regular sets under gsm mappings (and homomorphisms as a special case), L is regular if and only if $L(G_\pi)$ is regular. The latter is decidable since $L(G_\pi)$ is a deterministic CFL [13].

4. LRRC GRAMMARS

As Knuth has shown [1], during the parsing of an LR(k) grammar, only some restricted information about the string preceding the handle is essential for recognizing the handle. This information can be represented by one out of a finite number of regular sets to which the string up to and including the handle belongs. Similar situation occurs in parsing LRR grammars; thus the information needed for recognizing a handle in any given canonical sentential form can be represented by two regular sets, one containing the string up to and including the handle and the other containing the string following the handle. This leads to the following definition of left-to-right regular context (LRRC) grammar, which can also be regarded as a generalization of Floyd's bounded right context grammar [4].

DEFINITION 4.1. Let $\pi = \{R_i\}_{1 \leqslant i \leqslant n}$ and $\tau = \{Q_j\}_{1 \leqslant j \leqslant m}$ be any two regular

partitions of T^* and V^*, respectively. A grammar $G = (N, T, P, S)$ is said to be **LRRC**(τ, π) if and only if given any two right-most derivations of the form

$$S \overset{R}{\Rightarrow}{}^* \alpha_1 A_1 y_1 \overset{R}{\Rightarrow} \alpha_1 \gamma y_1$$

$$S \overset{R}{\Rightarrow}{}^* \alpha_3 A_2 y_3 \overset{R}{\Rightarrow} \alpha_3 \gamma' y_3 = \alpha_2 \gamma y_2$$

and the conditions $\alpha_1 \gamma \equiv \alpha_2 \gamma \pmod{\tau}$, $y_1 \equiv y_2 \pmod{\pi}$ and $| \alpha_2 \gamma | \leqslant | \alpha_3 \gamma' |$, we may conclude that $A_1 = A_2$, $\gamma = \gamma'$, $\alpha_2 = \alpha_3$ and $y_2 = y_3$.

A grammar is **LR Regular Context** (LRRC) if and only if it is LRRC(τ, π) for some partitions τ, π as above.

Clearly every LRRC grammar is unambiguous. The following result is not surprising.

THEOREM 4.1. *Let π be a regular partition of T^*. A grammar $G = (N, T, P, S)$ is* LR(π) *if and only if it is* LRRC(τ, π) *for some regular partition τ of V^*.*

Proof. (a) Suppose G is LRRC(τ, π) for some regular partition τ of V^*. Let η, ξ be two strings satisfying the assumptions of Definition 1.1, i.e., $\eta = \alpha_1 \gamma y_1$, $\xi = \alpha_1 \gamma y_2$ such that

$$S \overset{R}{\Rightarrow}{}^* \alpha_1 A_1 y_1 \overset{R}{\Rightarrow} \alpha_1 \gamma y_1$$

$$S \overset{R}{\Rightarrow}{}^* \alpha_2 A_2 y_3 \overset{R}{\Rightarrow} \alpha_1 \gamma y_2$$

and $y_1 \equiv y_2 \pmod{\pi}$. Since $\alpha_1 \equiv \alpha_1 \pmod{\tau}$ the pair η, ξ also satisfies the assumptions of Definition 4.1 and hence $\alpha_1 = \alpha_2$, $A_1 = A_2$ and $y_2 = y_3$.

(b) Suppose G is LR(π). For each production p: $A \to \gamma$ and for each $i = 1, \ldots, n$, define the set

$$Q_{p,i} = \{\alpha\gamma \mid S \overset{R}{\Rightarrow}{}^* \alpha A y \overset{R}{\Rightarrow} \alpha\gamma y \text{ and } y \in R_i\}.$$

We claim that for each i, the sets $\{Q_{p,i} \mid p \in P\}$ are disjoint. For suppose $Q_{p,i} \cap Q_{p',i} \neq \phi$ where $p = A \to \gamma$ and $p' = A' \to \gamma'$; then there exist derivations

$$S \overset{R}{\Rightarrow}{}^* \alpha A y \overset{R}{\Rightarrow} \alpha\gamma y$$

$$S \overset{R}{\Rightarrow}{}^* \alpha' A' y' \Rightarrow \alpha'\gamma' y' = \alpha\gamma y'$$

where $y, y' \in R_i$. But then $y \equiv y'(\pi)$ and since G is LR(π) we have $p = p'$. Hence for each i, $\tau_i = \{Q_{p,i} \mid p \in P\} \cup \{V^* - \bigcup_{p \in P} Q_{p,i}\}$ is a partition of V^*. Define τ to be the

refinement of all partitions τ_i, $i = 1,\ldots, n$. We claim that G is LRRC(τ, π). To see this, let

(1) $S \overset{R}{\Rightarrow}{}^* \alpha_1 A_1 y_1 \overset{R}{\Rightarrow} \alpha_1\gamma y_1$

(2) $S \overset{R}{\Rightarrow}{}^* \alpha_3 A_3 y_3 \overset{R}{\Rightarrow} \alpha_3\gamma' y_3 = \alpha_2\gamma y_2$

such that $\alpha_1\gamma \equiv \alpha_2\gamma \pmod{\tau}$, $y_1 \equiv y_2 \pmod{\pi}$ and $|\alpha_2\gamma| \leqslant |\alpha_3\gamma'|$. Let $y_1, y_2 \in R_i$; then we get $\alpha_1\gamma \in Q_{p,i}$, where $p = A_1 \to \gamma$, and thus also $\alpha_2\gamma \in Q_{p,i}$. Hence there exists a derivation

$$S \overset{R}{\Rightarrow}{}^* \alpha_2 A_1 y' \overset{R}{\Rightarrow} \alpha_2\gamma y'$$

which, together with derivation (2) above, the fact that $y_2 \equiv y' \pmod{\pi}$, and the assumption that G is LR(π), imply $A_1 = A_2$, $\alpha_2 = \alpha_3$, $y_2 = y_3$ and hence also $\gamma = \gamma'$. Therefore G is also LRRC(τ, π).

COROLLARY 4.2. *A grammar is* LRR *if and only if it is* LRRC.

Remark. As a generalization of Floyd's bounded context grammars [4], we could define here the analogous notion of "regular context grammars". Such a grammar would be defined with respect to some given regular partitions τ, π of V^* so that one can decide whether or not to reduce any sentential form (not necessarily right-most) $\alpha\gamma\beta$ using the production $A \to \gamma$ if one knows the equivalence classes of τ and π containing $\alpha\gamma$ and β respectively. Clearly every such grammar must be both LR(π) and RL(τ) and hence by Theorem 1.1 these grammars form a proper subfamily of both families of LRR and RLR grammars. The same statement is also true for the corresponding families of languages. Furthermore, such grammars can generate languages which are neither deterministic nor reverse deterministic CF languages, as can be seen from Example 1.2, in which the grammar G is linear and LRR and therefore also regular context grammar.

5. A CRITERION FOR LRR GRAMMARS

In this section we derive necessary and sufficient conditions for a grammar to be LRR. Before stating the main result we need some preliminary definitions and lemmas.

Notation. Throughout the rest of this paper let # denote an auxiliary symbol not in V.

In what follows, let $G = (N, T, P, S)$ denote an arbitrary CF grammar which will remain fixed throughout the discussion. Thus all definitions and lemmas presented below are stated with respect to G.

DEFINITION 5.1. For any production $p = A \to \gamma$ of G let L_p^{yes} and L_p^{no} be the languages defined by:

$$L_p^{\text{yes}} = \{\alpha\gamma\#y \mid \alpha \in V^*, y \in T^*, S \overset{R}{\Rightarrow}{}^* \alpha Ay \overset{R}{\Rightarrow} \alpha\gamma y\},$$

$$L_p^{\text{no}} = \{\alpha\gamma\#y \mid \alpha \in V^*, y \in T^*, S \overset{R}{\Rightarrow}{}^* \beta Bu \overset{R}{\Rightarrow} \beta\gamma' u = \alpha\gamma y,$$

such that $|\alpha\gamma| \leqslant |\beta\gamma'|$ and if $A = B$ and $\gamma = \gamma'$ then $|\alpha| < |\beta|\}$.

LEMMA 5.1. *For any production $p = A \to \gamma$ of G, L_p^{yes} and L_p^{no} are* CF *languages and can be effectively found.*

Proof. Let $G_1 = (N', T', P_1, \bar{S})$, where $N' = \{B' \mid B \in N\} \cup \{\bar{B} \mid B \in N\}$, $T' = T \cup N \cup \{\#\}$ and P_1 is defined as follows. Let h be the homomorphism from V^* into $(N' \cup T)^*$ defined by $h(a) = a$ for $a \in T$ and $h(B) = B'$ for $B \in N$. P_1 consists of the following three types of rules;

(i) For any $\alpha, \beta \in V^*$, $B, C \in N$ such that $B \to \alpha C\beta$ is a production in P, the production $\bar{B} \to \alpha\bar{C}h(\beta)$ is in P_1;

(ii) For any production $B \to \beta$ in P, $B' \to h(\beta)$ is in P_1;

(iii) $\bar{A} \to \gamma\#$ is in P_1.

One can easily verify that $L(G_1) = L_p^{\text{yes}}$.

Now consider the grammar $G_2 = (N', T', P_2, \bar{S})$ where N' and T' are the same as for G_1 and P_2 consists of all productions of types (i) and (ii) as above as well as productions of the form

(iii)′ $\bar{B} \to \gamma'\#$, where $B \to \gamma'$ is any production in P other than p.

Now suppose we have Case (a): $p = A \to \gamma \neq \epsilon$; then $\gamma = \gamma' Z$ for some $Z \in V$ and the following production will also be included in P_2:

(iv) $\bar{A} \to \gamma'\#Z$.

Define a gsm g_1 as follows: Let $\gamma = Z_1 Z_2 \cdots Z_k$ where $Z_i \in V$, $Z_k = Z$ and $k > 0$. Let $g_1 = (\{q_0, q_1, \ldots, q_{k+1}\}, T', T', \delta, q_0, \{q_{k+1}\})$ where δ is defined by

$$\begin{aligned}
\delta(q_0, X) &= \{(q_0, X)\} \text{ for each } X \in V - \{Z_1\},\\
\delta(q_0, Z_1) &= \{(q_0, Z_1), (q_1, Z_1)\},\\
\delta(q_{i-1}, Z_i) &= \{(q_i, Z_i)\} \text{ for } 2 \leqslant i \leqslant k-1,\\
\delta(q_{k-1}, Z_k) &= \{(q_k, Z_k\#)\},\\
\delta(q_k, \#) &= \{(q_{k+1}, \epsilon)\},\\
\delta(q, X) &= \{(q, X)\} \text{ for each } X \in V \text{ and } q \in \{q_k, q_{k+1}\}.
\end{aligned}$$

Clearly

$$L(G_2) = \{\alpha\beta\#x\colon S \overset{R}{\underset{G}{\Rightarrow}}{}^{*}\ \alpha\beta x \overset{R}{\underset{G}{\Rightarrow}}\ \alpha\beta x \text{ for } p \neq B \to \beta\}$$

$$\cup\ \{\alpha\gamma'\#Zx\colon S \overset{R}{\underset{G}{\Rightarrow}}{}^{*}\ \alpha Ax \overset{R}{\underset{G}{\Rightarrow}}\ \alpha\gamma' Zx \text{ for } p = A \to \gamma' Z\}$$

and for any α, β, δ in $V^*g_1(\alpha\gamma\beta\#\delta)$ contains $\alpha\gamma\#\beta\delta$ where $A \to \gamma$ is the fixed production p. It can be verified that $g_1(L(G_2)) = L_p^{\text{no}}$ and since context-free languages are closed under gsm mappings, L_p^{no} is context-free.

Now for Case (b): $p = A \to \epsilon$. Let $G_3 = (N', T'', P_3, \bar{S})$, where $T'' = V \cup \{\#'\}$ where $\#' \notin N' \cup T'$ and P_3 consists of all productions of the types (i), (ii) and (iii)′ as above, with $\#$ replaced by $\#'$. Now define a gsm $g_2 = (\{q_0, q_1, q_2\}, V \cup \{\#, \#'\}, T', \delta', q_0, \{q_2\})$ where δ' is as follows:

$$\delta'(q_0, X) = \{(q_0, X), (q_1, \#X)\} \text{ for all } X \in T,$$

$$\delta'(q_0, X) = \{(q_0, X)\} \text{ for all } X \in N,$$

$$\delta'(q_0, \#') = \{(q_2, \#)\},$$

$$\delta'(q_1, X) = \{(q_1, X)\} \text{ for all } X \in T,$$

$$\delta'(q_1, Y) = \{(q_2, \epsilon)\} \text{ for } Y \in \{\#, \#'\},$$

$$\delta'(q_2, X) = \{(q_2, X)\} \text{ for all } X \in T.$$

One can easily verify that $L_p^{\text{no}} = g_2(L(G_3) \cup L_p^{\text{yes}})$ and hence L_p^{no} is a CF language.

DEFINITION 5.2. Let p be a fixed production of the grammar $G = (N, T, P, S)$. Define the following sets:

$$X_p^{\text{yes}} = L_p^{\text{yes}}/\{\#\}T^*(=\{\alpha \in V^* \mid \alpha\#y \in L_p^{\text{yes}}\})\ ^{4}$$

$$X_p^{\text{no}} = L_p^{\text{no}}/\{\#\}T^*(=\{\alpha \in V^* \mid \alpha\#y \in L_p^{\text{no}}\})$$

Also let $K_p^{\text{yes}} = L_p^{\text{yes}} \cap (X_p^{\text{no}}\{\#\}T^*)$ and $K_p^{\text{no}} = L_p^{\text{no}} \cap (X_p^{\text{yes}}\{\#\}T^*)$. Now we proceed to show the regularity of X_p^{no}; for X_p^{yes} this was already shown in [1].

LEMMA 5.2. *For any production p: $A \to \gamma$ of G, the sets X_p^{yes} and X_p^{no} are regular.*

Proof. The regularity of X_p^{yes} was shown in [1].

[4] For any two languages L, L', the left quotient of L by L' is $L'\backslash L = \{y \mid xy \in L \text{ for some } x \in L'\}$. The right quotient of L by L' is $L/L' = \{x \mid xy \in L \text{ for some } y \in L'\}$.

For X_p^{no} consider the right-linear grammar $G_2 = (N_1, N \cup T, P_2, \bar{S})$ where P_2 consists of productions of type (i) as for G_1 as well as productions of type (ii)′ and (iii) defined as follows:

(ii)′ For any production $B \to \beta$ in $P - \{p\}$, let $\bar{B} \to \beta$ be in P_2.

We distinguish two cases: Case (a): $\gamma \neq \epsilon$; then $\gamma = \gamma' Z$ for some $Z \in V$, and the following production is also to be included in P_2:

(iii) $\bar{A} \to \gamma'$.

Now define a gsm g as follows: Let $\gamma = Z_1 Z_2 \cdots Z_k$, where $Z_i \in V$, $Z_k = Z$ and $k > 0$. Let $g = (\{q_0, q_1, \ldots, q_k\}, V, V, \delta, q_0, \{q_k\})$, where δ is defined by:

$$\begin{aligned}
\delta(q_0, X) &= \{(q_0, X)\} \text{ for each } X \in V - \{Z_1\};\\
\delta(q_0, Z_1) &= \{(q_0, Z_1), (q_1, Z_1)\};\\
\delta(q_{i-1}, Z_i) &= \{(q_i, Z_i)\} \text{ for } 2 \leqslant i \leqslant k.\\
\delta(q_k, X) &= \{(q_k, \epsilon)\} \text{ for } X \text{ in} T.
\end{aligned}$$

It can be easily verified that $g(L(G_2)) = X_p^{\text{no}}$ and since $L(G_2)$ is a regular language, so is X_p^{no}.

Case (b): $p\colon A \to \epsilon$. Let P_2 consist of productions of type (i) and (ii)′ only. Then

$$\begin{aligned}
X_p^{\text{no}} &= \text{Init}^5(L(G_2) \cup (X_p^{\text{yes}}/V))\\
&= \{\alpha \in V^* \mid \text{there exists } \beta \in V^* \text{ such that } \alpha\beta \in L(G_2) \text{ or } \alpha\beta Z \in X_p^{\text{yes}} \text{ for some } Z \in V\}
\end{aligned}$$

and since both $L(G_2)$ as well as X_p^{yes}/V are regular languages, and the operation Init preserves regularity [7], X_p^{no} is also regular.

COROLLARY 5.3. *For any grammar G and production p of G, the sets K_p^{yes} and K_p^{no} are* CF *languages and can be effectively found.*

Proof. Lemmas 5.1 and 5.2.

DEFINITION 5.3. Let p be a fixed production of the grammar $G = (N, T, P, S)$; a set $M_p \subseteq V^*\{\#\}T^*$ is a *separating set* for p iff

(i) $K_p^{\text{yes}} \subseteq M_p$ and

(ii) $K_p^{\text{no}} \subseteq V^*\{\#\}T^* - M_p$.

Note. If $X_p^{\text{yes}} \cap X_p^{\text{no}} = \phi$ then any subset of $V^*\{\#\}T^*$ is a separating set for p.

[5] For any language L over alphabet T, $\text{Init}(L) = L/T^*$.

LEMMA 5.4. *For any production p of the grammar G and for any given regular set M, it is decidable whether M is a separating set for p.*

Proof. M is a separating set for p iff

$$L = (M \cap K_p^{\text{no}}) \cup ((V^*\{\#\}T^* - M) \cap K_p^{\text{yes}}) = \phi.$$

Since K_p^{yes} and K_p^{no} are CF languages, by well-known closure properties L is also a CF language which can be effectively found and whose emptiness problem is, therefore, decidable.

THEOREM 5.5. *A grammar $G = (N, T, P, S)$ is* LRRC *iff there exists a regular separating set for each production $p \in P$.*

Proof. For each $p \in P$, let M_p be a regular separating set for p. Since $M_p \subseteq V^*\{\#\}T^*$, we can decompose M_p as follows: $M_p = \bigcup_{1 \leqslant i \leqslant n_p} Q_i^p\{\#\}R_i^p$, where Q_i^p, R_i^p, $1 \leqslant i \leqslant n_p$, are regular subsets of V^* and T^*, respectively. Let $\tau = \{Q_i\}$ be the partition of V^* which is the refinement (i.e., intersection) of all the regular partitions

$$\{X_p^{\text{yes}}, V^* - X_p^{\text{yes}}\}, \{X_p^{\text{no}}, V^* - X_p^{\text{no}}\}, \{Q_i^p, V^* - Q_i^p\},$$

$p \in P$, $i = 1,..., n_p$, of V^*. Similarly let $\pi = \{R_i\}$ be defined as the refinement of all regular partitions $\{R_i^p, T^* - R_i^p\}$, $p \in P$, $i = 1,..., n_p$, of T^*. We claim that G is LRRC(τ, π). To see this assume the contrary, i.e., there exist γ, γ', α_1, α_2, $\alpha_3 \in V^*$, A_1, $A_2 \in N$ and y_1, y_2, $y_3 \in T^*$ such that

$$S \overset{R}{\Rightarrow}{}^* \alpha_1 A_1 y_1 \overset{R}{\Rightarrow} \alpha_1\gamma y_1$$

$$S \overset{R}{\Rightarrow}{}^* \alpha_3 A_2 y_3 \overset{R}{\Rightarrow} \alpha_3\gamma' y_3 = \alpha_2\gamma y_2$$

$\alpha_1\gamma \equiv \alpha_2\gamma \pmod{\tau}$, $y_1 \equiv y_2 \pmod{\pi}$, $|\alpha_2\gamma| \leqslant |\alpha_3\gamma'|$ and either $A_1 \neq A_2$ or $\gamma \neq \gamma'$ or $\alpha_2 \neq \alpha_3$. Consider the production p: $A_1 \to \gamma$. Clearly either $\alpha_1\gamma \in X_p^{\text{yes}} - X_p^{\text{no}}$ or else $\alpha_1\gamma\#y_1 \in M_p$. If $\alpha_1\gamma \in X_p^{\text{yes}} - X_p^{\text{no}}$ then $\alpha_1\gamma \equiv \alpha_2\gamma \pmod{\tau}$ and the definition of τ imply that also $\alpha_2\gamma \in X_p^{\text{yes}} - X_p^{\text{no}}$; but this is impossible since $\alpha_2\gamma \in X_p^{\text{no}}$ by the above derivations. Thus assume that $\alpha_1\gamma \notin X_p^{\text{yes}} - X_p^{\text{no}}$ and $\alpha_1\gamma\#y_1 \in M_p$; then $\alpha_1\gamma\#y_1 \in Q_j^p\#R_j^p$ for some $1 \leqslant j \leqslant n_p$, i.e., $\alpha_1\gamma \in Q_j^p$ and $y_1 \in R_j^p$. From $\alpha_1\gamma \equiv \alpha_2\gamma \pmod{\tau}$, $y_1 \equiv y_2 \pmod{\pi}$ and the definitions of τ and π it follows that $\alpha_2\gamma \in Q_j^p$ and $y_2 \in R_j^p$. Hence $\alpha_2\gamma\#y_2 \in Q_j^p\#R_j^p \subseteq M_p$. Since $\alpha_1\gamma \notin X_p^{\text{yes}} - X_p^{\text{no}}$ we must have $\alpha_1\gamma \in X_p^{\text{yes}} \cap X_p^{\text{no}}$ and therefore also $\alpha_2\gamma \in X_p^{\text{yes}} \cap X_p^{\text{no}}$; thus we get $\alpha_2\gamma\#y_2 \in M_p \cap L_p^{\text{no}} \cap (X_p^{\text{yes}}\{\#\}T^*) = M_p \cap K_p^{\text{no}}$ which contradicts our assumption that M_p is a separating set for p.

Now suppose that G is LRRC, that is, there exist regular partitions $\tau = \{Q_i\}_{1 \leqslant i \leqslant n}$ and $\pi = \{R_j\}_{1 \leqslant j \leqslant m}$ of V^* and T^* resp. such that G is LRRC(τ, π). Let p be any

production in P. Clearly for each pair (i, j), $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$, either $Q_i\{\#\}R_j \cap L_p^{\text{yes}} = \phi$ or $Q_i\{\#\}R_j \cap L_p^{\text{no}} = \phi$. Let $I_p = \{(i, j) \mid 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m$ and $Q_i\{\#\}R_j \cap L_p^{\text{yes}} \neq \phi\}$. Then $M_p = \bigcup_{(i,j)\in I_p} Q_i\{\#\}R_j$ is clearly a regular separating set for p.[6]

COROLLARY 5.6. *A grammar G is* LRR *iff there exists a regular separating set for each of its productions.*

COROLLARY 5.7. *For given regular partitions τ and π of V^* and T^* resp., it is decidable whether an arbitrary* CF *grammar G is* LRRC(τ, π).

Proof. For each production p of the grammar $G = (N, T, P, S)$, define M_p as in the second half of the proof of Theorem 5.5 above. Clearly G is LRRC(τ, π) iff for each $p \in P$, M_p is a separating set. The latter is decidable by Lemma 5.4.

COROLLARY 5.8. *For a given regular partition π of T^*, it is decidable whether an arbitrary* CF *grammar G is* LR(π).

Proof. For the given grammar $G = (N, T, P, S)$ and the partition π, define a regular partition τ of V^* as in the second half of the proof of Theorem 4.1. Then G is LR(π) iff G is LRRC(τ, π), which was shown to be decidable.

We now illustrate the above results by the following example.

EXAMPLE 5.1. Consider again the grammar G from Example 1.3. As noted above,

$$L(G) = \{a^m b^{2m} a^n b^n,\ a^m b^m a^{n+k} b^n \mid k, m, n \geqslant 1\}.$$

Let us number its productions in the following way:

(1) $S \to BCB$

(2) $S \to AB$

(3) $A \to aAbb$

(4) $A \to abb$

(5) $B \to aBb$

(6) $B \to ab$

(7) $C \to Ca$

(8) $C \to a$

[6] In the expression for M_p, we can restrict the Q_i's appearing in the union only to those which intersect both X_p^{yes} and X_p^{no}, thus obtaining smaller separating sets.

The sets L_p^{yes} and L_p^{no} for these productions will now be described; in all descriptions below, let $m, n, k = 1, 2, \ldots$ and $i, j = 0, 1, \ldots$.

$$L_1^{\text{yes}} = \{BCB\#\}; L_1^{\text{no}} = \phi;$$

$$L_2^{\text{yes}} = \{AB\#\}; L_2^{\text{no}} = \phi;$$

$$L_3^{\text{yes}} = \{a^{i+1}Abb\#b^{2i}a^nb^n\}; L_3^{\text{no}} = \phi;$$

$$L_4^{\text{yes}} = \{a^mbb\#b^{2m-2}a^nb^n\}; L_4^{\text{no}} = \phi;$$

$$L_5^{\text{yes}} = \{a^{i+1}Bb\#b^ia^{k+n}b^n\} \cup \{BCa^{j+1}Bb\#b^j\} \cup \{Aa^{j+1}Bb\#b^j\}; L_5^{\text{no}} = \phi;$$

$$L_6^{\text{yes}} = \{a^mb\#b^{m-1}a^{n+k}b^n\} \cup \{BCa^nb\#b^{b-1}\} \cup \{Aa^nb\#b^{n-1}\};$$

$$L_6^{\text{no}} = \{a^mb\#b^{2m-2}a^nb^n\};$$

$$L_7^{\text{yes}} = \{BCa\#a^{i+n}b^n\}; L_7^{\text{no}} = \phi;$$

$$L_8^{\text{yes}} = \{Ba\#a^{i+n}b^n\};$$

$$L_8^{\text{no}} = \{a^m\#a^ib^{j+m}a^{n+k}b^n\} \cup \{BCa^m\#a^{j+n}b^n\} \cup \{BCa^m\#a^jb^{j+m}\}$$
$$\cup \{a^m\#a^ib^{2m+2i}a^nb^n\} \cup \{Aa^m\#a^ib^{m+i}\}.$$

Clearly for all productions p with $L_p^{\text{no}} = \phi$, K_p^{yes}, K_p^{no} are both empty and no separation is necessary. The handle corresponding to such productions can be determined solely by its appearance in the string to the left of any other (possible) handle, thus obviously no look-ahead is needed in such cases. Now consider production (8): we have $X_8^{\text{yes}} = \{Ba\}$ and $X_8^{\text{no}} = a^+ \cup Aa^+ \cup \{BCa\}$; these two sets are disjoint and hence $K_8^{\text{yes}} = K_8^{\text{no}} = \phi$ and no look-ahead is necessary. Now consider (6): since $X_6^{\text{yes}} \cap X_6^{\text{no}} = (a^+b \cup BCa^+b \cup Aa^+b) \cap a^+b = a^+b$, we have:

$$K_6^{\text{yes}} = \{a^mb\#b^{m-1}a^{n+k}b^n\}$$

and

$$K_6^{\text{no}} = \{a^mb\#b^{2m-2}a^nb^n\}.$$

In order for G to be LRR, we must be able to separate K_6^{yes} from K_6^{no} by a regular set, i.e., find a regular set M such that $K_6^{\text{yes}} \subset M$ and $M \cap K_6^{\text{no}} = \phi$. Without loss of generality we may assume that M is a subset of $a^*b\#b^*a^*b^*$. However, using well-known results from automata theory, one can easily show that there exists no regular set satisfying the above requirements. Hence the grammar G above is not LRR.

Nevertheless, with a minor modification, G can be converted into an LRR grammar. Let G' be the grammar obtained from G by replacing production (7) above by (7') $C \to Caa$. The language then becomes $L(G') = \{a^m b^{2m} a^n b^n\} \cup \{a^m b^m a^{n+k} b^n\}$, same as $L(G)$ except that $k = 1, 3, 5,...$ rather than $k = 1, 2, 3,...$ as in $L(G)$. All sets $L_p^{\text{yes}}, L_p^{\text{no}}$, $1 \leqslant p \leqslant 7$, as well as L_8^{yes}, are given by the same descriptions as above, with k representing an odd number only; L_8^{no} has to be slightly modified but still retains the property that $X_8^{\text{yes}} \cap X_8^{\text{no}} = \phi$, thus no lookahead is needed for the recognition of (8). As for rule (6), the sets K_6^{yes} and K_6^{no} can be re-written as

$$K_6^{\text{yes}} = a^{i+1} b \# b^i a^{n+2j+1} b^n\} \qquad \text{and} \qquad K_6^{\text{no}} = \{a^{i+1} b \# b^{2i} a^n b^n\},$$

where $i, j = 0, 1,...$ and $n = 1, 2,...$. These sets can be distinguished by comparing the parity of the number of a's with that of the number of b's on the right part of the string. In fact, the regular set $M = a^* b \# b^* ((a^2)^* a (b^2)^* \cup (a^2)^* (b^2)^* b)$ contains K_6^{yes} and is disjoint from K_6^{no}, hence is a separating set for rule (6). Consequently G' is an LRR grammar. Specifically, G' is LR(π) for $\pi = \{R_1, R_2\}$, where

$$R_1 = b^*((a^2)^* a (b^2)^* \cup (a^2)^* (b^2)^* b) \qquad \text{and} \qquad R_2 = \{a, b\}^* - R_1 .$$

6. Generating LRR Parsers for Arbitrary Grammars

As was shown above, for any given CFG G and for any given regular partition $\pi = \{R_1, ..., R_n\}$ of T^*, one can effectively decide if G is LR(π); moreover, if indeed G turns out to be LR(π), a practical parsing method for G has been described. The problem remains, however, for a given grammar G, to find a regular partition π such that G is LR(π), if such a partition exists. We conjecture that the question of whether or not an arbitrary CFG is LRR is, in general, unsolvable. However, Theorem 5.5 and Corollary 5.6 provide some clues for answering this question in particular cases, and, when the answer is affirmative, for finding a regular partition π such that G is LR(π) (e.g., Example 5.1 above). According to these results, the question of whether or not G is LRR is equivalent to the question of whether or not each rule p of G has a regular separating set. The latter amounts to deciding, for CF languages K_p^{yes} and K_p^{no}, given by their grammars, whether or not there exists a regular set M_p separating them, i.e. containing K_p^{yes} and disjoint from K_p^{no}. This latter problem appears to be

[7] After completion of this paper it has been brought to our attention that the undecidability of the question of whether a CFG is LRR has been recently established by W. F. Ogden [17]. This result, together with Corollary 4.2 and Theorem 5.5, implies the undecidability of the general Open Problem stated in the text. However, a direct proof of this latter undecidability result would be of interest.

undecidable, though we are not aware of any proof to this effect.[7] Moreover the following problem is open.

Open Problem. Is it decidable whether for two given context-free languages L_1 and L_2 there exists a regular set containing L_1 and disjoint from L_2?

We conjecture that the above question is undecidable. However, we can develop some techniques for obtaining such regular separating sets, whenever such sets exist, and from these sets the desired regular partitions τ and π such that G is LRRC(τ, π) can be obtained using the construction in the proof of Theorem 5.5. One can start by checking, for each production $p: A \to \alpha$ in the given grammar G, whether $X_p^{\text{yes}} \cap X_p^{\text{no}} = \phi$. If this happens to be true then both sets K_p^{yes} and K_p^{no} are empty, which amounts to the fact that in any right sentential form, a handle α corresponding to rule p is uniquely determined solely by the string to its left; in such case of course no look-ahead information is needed.

Now suppose the above condition does not hold for rule p. Then construct the grammars G_1 and G_2 generating the languages K_p^{yes} and K_p^{no} resp. using Lemma 5.1. Now proceed to find, if possible, a "regular envelope" for K_p^{yes} disjoint from K_p^{no}, i.e., regular set M_p, containing K_p^{yes} and as close as possible to K_p^{yes}, and disjoint from K_p^{no}. Similarly, one can construct such a regular envelope N_p for K_p^{no} and check its disjointness from K_p^{yes}. If such a set N_p disjoint from K_p^{yes} is found, then its complement $\bar{N}_p = V^*\{\#\}T^* - N_p$ is also a regular separating set for rule p. In fact, if both such regular envelopes M_p and N_p are found, and they are disjoint, then any regular set R_p such that $M_p \subseteq R_p \subseteq \bar{N}_p$ is also a separating set for production p. This allows some flexibility in choosing the separating sets R_p for the productions of G so as to optimize, in some sense, the resulting partitions π and τ (obtained as in the proof of Theorem 5.5) to yield the most efficient LRRC parser. Therefore it is desirable to construct regular envelopes for both languages K_p^{yes} and K_p^{no}, and check their disjointness.

There are a few approaches one can take when trying to construct a good "regular approximation" for a given CF language. For instance, one can try to modify the corresponding push-down automaton so as to "forget" all but a bounded amount of information on its push-down stack, thus turning it into a finite-state machine. Another approach would be to modify the CF grammar so that it would generate a larger regular language. If the regular envelope thus obtained is still "too big", a smaller envelope can be obtained by a refinement of the construction. In the example presented below we have chosen the second approach.

As already mentioned, once the required regular envelopes M_p and $\bar{N}_p$ as above have been found for every production p of G, it is desirable to optimize the resulting partitions π and τ so as to make the parsing procedure, including the pre-scan, most efficient. Naturally one would try to minimize the number of blocks in partition π so that the number of distinct labels attached to the input string during the pre-scan

would be as small as possible. It would be also desirable to obtain the smallest possible number of states in the Moore machine M recognizing the regular sets of π, used during the pre-scan, as well as optimize the modified LR(0) parser. Such optimization can be carried out with the use of some known automata theory methods.[8]

The techniques discussed above will now be illustrated by the following example.

EXAMPLE 6.1. Let $G = (\{A, B, C, D, S\}, \{a, b, d\}, P, S)$ where $P = \{S \to AB;$ $S \to CD;$ $A \to bAb;$ $A \to ab;$ $B \to dBa;$ $B \to dBb;$ $B \to da;$ $C \to aC;$ $C \to bC;$ $C \to \epsilon;$ $D \to dDa;$ $D \to dDb;$ $D \to b\}$. The right sentential forms for G are: S; Ad^kBx; $b^nAb^nd^{k+1}ax$; $b^nab^{n+1}d^{k+1}ax$; Cd^kDy; zCd^kby; zd^kby, for any k, $n \geqslant 0$ and any words $x, y, z \in \{a, b\}^*$ such that $|x| = |y| = k$. One can easily verify that for all right sentential forms containing at least one nonterminal, the (unique) handle can be determined by the string to its left. This is due to the fact that for all productions p in P except for q: $A \to ab$ and r: $C \to \epsilon$, the set L_p^{no} is empty. Thus in order for G to be LRR, there must exist regular separating sets for both rules q and r. Before proceeding to find such sets, we observe that G is not LR(k) for any k. This is seen from the following two rightmost derivations in G:

$$S \overset{R}{\Rightarrow}{}^* b^nAb^nd^{k+1}ab^k \overset{R}{\Rightarrow} b^nabb^nd^{k+1}ab^k$$

$$S \overset{R}{\Rightarrow}{}^* b^nabb^nCd^{k+1}b^{k+2} \overset{R}{\Rightarrow} b^nabb^nd^{k+1}b^{k+2}$$

where k, n are any arbitrary non-negative integers.

Now consider the sets K_t^{yes} and K_t^{no} for $t = q, r$. From the above derivations we have: $b^nab\#b^nd^{k+1}ab^k \in K_q^{\text{yes}}$ and $b^nab\#b^nd^{k+1}b^{k+2} \in K_q^{\text{no}}$. Also, as can be easily verified, for all $k = 0, 1, 2, \ldots$, $ab\#d^{k+1}ab^k \in K_r^{\text{no}}$ and $ab\#d^kb^{k+1} \in K_r^{\text{yes}}$. Hence both productions q and r require separating sets. Thus let us construct grammars G_1 and G_2 generating K_q^{yes} and K_q^{no} resp. We obtain:

$$G_1 = (\{S_1, A, B\}, \{a, b, d, \#\}, P_1, S_1)$$

and

$$G_2 = (\{S_2, C, D, E\}, \{a, b, d, \#\}, P_2, S_2),$$

where

$$P_1 = \{S_1 \to AB;\ A \to bAb;\ A \to ab\#;\ B \to dBa;\ B \to dBb;\ B \to da\}$$

and

$$P_2 = \{S_2 \to CD;\ C \to bC;\ C \to ab\#E;\ E \to aE;\ E \to bE;\ E \to \epsilon;$$
$$D \to dDa;\ D \to dDb;\ D \to b\}.$$

[8] Specifically, this optimization problem is closely related to the minimization problem of incompletely specified machines [14, 16], as is demonstrated in Example 6.1.

Let us now convert both G_1 and G_2 into grammars generating (possibly) larger regular sets. Thus in G_1, replace the self-embedding production $A \to bAb$ by the two productions $A \to bA$; $A \to Ab$. Similarly, replace $B \to dBb$ and $B \to dBa$ by $B \to dB$, $B \to Ba$ and $B \to Bb$. The new grammar thus obtained generates the regular set $M_0 = b^*ab\#b^*d^+a\{a, b\}^*$. Similarly for G_2, replace $D \to dDa$ and $D \to dDb$ by the three productions $D \to dD$, $D \to Da$ and $D \to Db$. Then the resulting grammar generates the regular set $N_q = b^*ab\#\{a, b\}^*d^*b\{a, b\}^*$. Clearly $K_q^{\text{yes}} \subset M_q$ and $K_q^{\text{no}} \subset N_q$ and it can be easily verified that $M_q \cap N_q = \phi$ as desired.

Now do the same for the rule r: $C \to \epsilon$. The grammars G_3 and G_4 generating the sets K_r^{yes} and K_r^{no} resp. are as follows: $G_3 = (\{S_3, C, D\}, \{a, b, d, \#\}, P_3, S_3)$ where $P_3 = \{S_3 \to CD;\ C \to aC;\ C \to bC;\ C \to \#;\ D \to dDa;\ D \to dDb;\ D \to b\}$ and $G_4 = (\{S_4, A, A', B, C, C', D\}, \{a, b, d, \#\}, P_4, S_4)$, where

$$P_4 = \{S_4 \to AB;\ S_4 \to CD;\ A \to ab\#;\ A \to a\#b;$$

$A \to bAb$; $A \to \#A'$; $A' \to bA'b$; $A' \to ab$; $B \to dBa$; $B \to dBb$; $B \to da$; $C \to aC$; $C \to bC$; $C \to \#C'$; $C' \to aC'$; $C' \to bC'$; $C' \to a$; $C' \to b$; $D \to dDa$; $D \to dDb$; $D \to b\}$. Now modify G_3 by replacing the productions $D \to dDa$ and $D \to dDb$ by the productions $D \to dD$; $D \to Da$ and $D \to Db$. The modified grammar thus obtained generates the regular envelope $M_r = \{a, b\}^*\#d^*b\{a, b\}^*$ of K_r^{yes}. Similarly modify G_4 by replacing the self-embedding productions $\{A \to bAb;\ A' \to bA'b;$ $B \to dBa;\ B \to dBb;\ D \to dDa;\ D \to dDb\}$ by the productions $\{A \to bA;\ A \to Ab;$ $A' \to bA';\ A' \to A'b;\ B \to dB;\ B \to Ba;\ B \to Bb;\ D \to dD;\ D \to Da;\ D \to Db\}$. The regular envelope for K_r^{no}, generated by the modified grammar, is:

$$N_r = (b^*\#b^*ab^+ \cup b^*a\#b^+ \cup b^*ab\#b^*)\, d^+a\{a, b\}^* \cup \{a, b\}^*\#\{a, b\}^+\, d^*b\{a, b\}^*.$$

However, checking for disjointness of the two envelopes, we find that $M_r \cap N_r = \{a, b\}^*\#b\{a, b\}^*\, b\{a, b\}^* \neq \phi$. Hence the above envelopes are too big and better regular approximations for K_r^{yes} and K_r^{no} are required. Thus replace the self-embedding rules $D \to dDa$ and $D \to dDb$ in G_3 by the rules: $D \to dD'a$; $D \to dD'b$; $D' \to dD'$; $D' \to D'a$; $D' \to D'b$; $D' \to b$, adding a new nonterminal D' to G_3 and leaving all other rules in P_3 unchanged. This new modified grammar generates the following better regular approximation of

$$K_r^{\text{yes}}\colon M_r' = \{a, b\}^*\#b \cup \{a, b\}^*\#d^+b\{a, b\}^+.$$

As can be readily seen, $M_r' \cap N_r = \phi$ and we have thus found the required envelopes.

We are now ready to search for "optimal" separating sets for productions q and r, namely, regular sets M and M', $M_q \subseteq M \subseteq \bar{N}_q$ and $M_r \subseteq M' \subseteq \bar{N}_r$, whose recognition by a finite state machine will require the minimum number of states. We can treat this

problem systematically be converting it into a problem of minimization of an incompletely specified machine [14, 16] i.e., construct the smallest possible sequential machine which distinguishes M_q from N_q (M_r from N_r) and is unspecified otherwise. A general solution of such a problem is presented in [14]. However, for the sake of brevity, we shall omit the detailed solution for the above example and simply indicate the final result, that is, the optimal separating sets M and M'. These are: $M = T^*\#T^*\,daT^*$ and $M' = T^*\#(b \cup dT^*)$. Clearly, in this case, no distinction is needed for the string on the left of "#", thus the partition τ is independent of rules q and r and is determined by the other rules of the grammar. To obtain τ, one has to construct DeRemer's characteristic finite-state machine (CFSM) [2], whose states will correspond to the sets of τ; this is because, in DeRemer's terminology, the look-ahead needed for parsing our grammar does not require any „state splitting" in the CFSM (thus the above grammar G can be called "LALRR" as an extension of DeRemer's LALR(k) grammars).

As for the partition π, we simply intersect the sets $R_1 = T^*\,daT^*$, $R_2 = \{b\} \cup dT^*$ and their complements to obtain:

$$\pi = \{R_1 - R_2;\ R_2 - R_1\,,\ R_1 \cap R_2\,,\ T^* - R_1 - R_2\}.$$

However, looking at the sets $M_q\,, N_q\,, M_r$ and N_r again, we observe that $T^*\#R_1$ is disjoint not only from N_q but also from N_r; hence no distinction need be made between $R_1 - R_2$ and $R_1 \cap R_2$ and the partition $\pi' = \{R_1\,, R_2 - R_1\,, T^* - R_1 - R_2\} = \{R_1', R_2', R_3'\}$ is sufficient. The sequential machine M_{π}, (Definition 2.1) used for the pre-scan, can now be constructed. We obtain $M_{\pi'} = (\{q_0\,, q_1\,, q_2\,, q_3\,, q_4\}, T, \{1, 2, 3\}, \delta, \lambda, q_0)$ where δ is defined as follows:

$$\delta(q_0\,, a) = \delta(q_1\,, a) = \delta(q_2\,, a) = \delta(q_3\,, a) = q_2;$$

$$\delta(q_4\,, a) = q_4;\ \delta(q_0\,, b) = \delta(q_0\,, d) = q_1;$$

$$\delta(q_1\,, b) = \delta(q_2\,, b) = \delta(q_3\,, b) = q_3;\ \delta(q_4\,, b) = q_4;$$

$$\delta(q_1\,, d) = \delta(q_3\,, d) = q_1;\ \delta(q_2\,, d) = q_4 \text{ and } \delta(q_4\,, d) = q_4\,.$$

The output function λ is given by: $\lambda(q_1) = 2$; $\lambda(q_2) = \lambda(q_3) = 3$ and $\lambda(q_4) = 1$; $\lambda(q_0)$ can be arbitrarily defined since q_0 is not a re-entrant state. The optimal LR(0) parser for $G_{\pi'}$ can be constructed using DeRemer's method, and will not be presented here.

We note that for the above example, a better parsing algorithm can be obtained, if we are willing to use an LR(2) parser rather than an LR(0) parser for the "main scan". Then there is no need for labeling the string symbols during the pre-scan, and it suffices to remember the final output of the sequential machine after having scanned

the whole input string. This can be seen if we observe the following facts about terminal strings w generated by G:

(1) If $w \in (T - \{d\})^* = M_1$, then the handle in w corresponds to production r: $C \to \epsilon$ is located before the last letter of w.

(2) If $w \in T^* daT^* = M_2$, then the handle in w corresponds to rule q: $A \to ab$ and is found at the left most occurence of "ab" in w.

(3) If none of the above conditions is satisfied (i.e. $w \in T^* - M_1 - M_2 = M_3$), then the handle in w corresponds to rule r and is located just before the first occurence of the letter "d".

Consequently, an LR(2) parser can locate the handle in each terminal string w, provided the information about the set M_i containing w is supplied by the sequential machine pre-scanner. In this case, of course, the pre-scan may be performed from left to right as well. The sequential machine recognizing the sets M_i has four states, and the LR(2) parser requires look-ahead only in two cases, corresponding to cases (1) and (3) above, thus it is "almost LR(0)". Clearly here this latter parsing procedure is by far more efficient than the usual LRR one, involving labeling the input string. However, this method applies only to a proper subset of the family of LRR grammars and cannot be used in general.

7. Possible Extension

The idea of a two-scan parsing algorithm described above can be extended to the case where the right-to-left pre-scan is performed by a more powerful type of deterministic transducer; for instance, a deterministic push down transducer (PDT). The non-LRR grammar (and language) presented in an Example 1.3 (and further discussed in Example 5.1), can be parsed by such a scheme; (in fact, all RL(k) grammars can be parsed by using a two-scan process with a push-down machine pre-scanner, simply because they can be parsed entirely during the pre-scan from right to left). However, by a modification of Example 1.3 we can obtain a grammar generating the language

$$L = \{a^m b^m a^n b^n a^p b^p,\ a^m b^{2m} a^{n+k} b^n a^p b^{2p} \mid m, n, p, k \geqslant 1\}$$

which is neither LRR nor RLR but is "LR(PDT)" parsable. Thus the class of "LR(PDT)" grammars seems to be another, yet larger, class of grammars, which properly includes the LRR grammars, yet still consists entirely of unambiguous grammars and can be parsed by a deterministic 2-pass process.

Acknowledgment

The authors would like to thank the referee for his helpful comments.

REFERENCES

1. D. E. KNUTH, On the translation of languages from left to right, *Information and Control* **8** (1965), 607–639.
2. F. DEREMER, "Practical Translators for LR(k) Languages," Project MAC Report MAC TR-65, October 1969, Cambridge, MA. See also *Simple* LR(k) *grammars*, *Comm. Assoc. Comp. Mach.* **14** (1971), 453–460.
3. R. W. FLOYD, Syntactic analysis and operator precedence, *J. Assoc. Comp. Mach.* **10** (1963), 313–333.
4. R. W. FLOYD, Bounded context syntactic analysis, *Comm. Assoc. Comp. Mach.* **7** (1964), 62–66.
5. D. PAGER, A solution to an open problem by Knuth, *Information and Control* **17** (1970), 462–472.
6. J. E. HOPCROFT AND J. D. ULLMAN, "Formal Languages and their Relation to Automata," Addison Wesley, Reading, MA, 1969.
7. S. GINSBURG, "The Mathematical Theory of Context-Free Languages," McGraw-Hill, New York, 1966.
8. A. V. AHO AND J. D. ULLMAN, "The theory of parsing, translation and compiling," Prentice-Hall, Englewood Cliffs, 1972.
9. T. E. CHEATHAM, The introduction of definitional facilities into higher level programming languages, Proc. AFIPS 1966, FJCC, Vol. 29, pp. 623–637.
10. B. GALLER AND A. J. PERLIS, A proposal for definitions in ALGOL, *Comm. Assoc. Comp. Mach.* **10** (1967), 204–219.
11. S. GINSBURG AND S. GREIBACH, Deterministic context-free languages, *Information and Control* **9** (1966), 620–648.
12. W. J. CHANDLER, "Abstract Families of Deterministic Languages," SDC Scientific Report No. 25, 1969.
13. R. E. STEARNS, A regularity test for pushdown machines, *Information and Control* **11** (1967), 323–340.
14. J. A. BRZOZOWSKI, Class notes for Math 894, Department of Applied Analysis and Computer Science, University of Waterloo, Waterloo, Ontario, 1971.
15. D.J. ROSENKRANTZ AND R.E. STEARNS, Properties of deterministic top-down grammars, *Information and Control* **17** (1970), 226–256.
16. M. C. PAULL AND S. H. UNGER, Minimizing the number of states in incompletely specified switching functions, *IRE Trans. Electronic Computers* **EC8** (1959), 356–367.
17. W. F. OGDEN, personal communication.