Comput. Lang. Vol. 16, No. 2, pp. 167–178, 1991
Printed in Great Britain.

0096-0551/91 $3.00 + 0.00

A UNIFYING MODEL FOR LOOKAHEAD LR PARSING

MANUEL E. BERMUDEZ
Computer and Information Sciences Department, University of Florida, Gainesville, FL 32611, U.S.A.

(*Received* 22 *January* 1990)

Abstract—We present a construction method for lookahead LR parsers that unifies several of the construction algorithms available in the literature. The model allows single, multiple and arbitrary symbol lookahead, each only when required. Three user-supplied parameters are used by the construction method; specific settings of these parameters yield well-known grammar classes such as LALR(k), SLR(k), and several subsets of the LR-Regular class, among others. Thus, rather than using several severely incompatible parser generators, or making unnatural changes to the grammar to accommodate the parsing technique, one may manipulate these parameters to obtain a suitable parser. The model captures the essence of the problem of computing lookahead for LR parsers, and provides a better understanding of the relationships among the corresponding classes of context-free grammars.

Algorithms Languages Theory Context-free grammar LR parsing LALR(k) SLR(k) NQLALR(1) Lookahead FSA LR–Regular

1. INTRODUCTION

The usual strategy for constructing a parser for a context-free language using LR techniques consists of two steps. First, the LR(0) automaton is constructed. Usually, some states in this automaton have shift/reduce conflicts and/or reduce/reduce conflicts. Second, lookahead sets are added to these so-called *inconsistent* states. Many algorithms have been proposed for adding such lookahead sets to an LR(0) parser. However, there are no techniques capable of describing two or more of these algorithms with a single notation. The variety of approaches to computing lookahead for LR parsers has brought about a confusing landscape of notations that makes the comparison of grammar classes a difficult task, and the simultaneous availability of several techniques an expensive proposition. For example, DeRemer in [4] showed that an LALR(1) parser can be built by first constructing a full LR(1) parser, and then merging states according to certain criteria. DeRemer also defined SLR(k) parsers in [5] as those built using ordinary Follow sets (of nonterminals) as lookahead sets. Kristensen and Madsen in [8] compute LALR(k) lookahead sets by resolving recursive equations. DeRemer and Pennello [6] now compute LALR(1) lookahead sets by performing two graph traversals, the graphs being induced by relations on nonterminal transitions in the LR(0) state diagram. They also formalized a technique called NQLALR(1), in which the relations are defined on nonterminals, rather than on nonterminal transitions. Their LALR(1) technique has been improved by Park *et al.* [9] by refining the definitions of the relations to include information contained in the LR(0) item-sets. The theory of arbitrary lookahead LR parsers is described by Culik and Cohen [3]; practical techniques have been proposed by Baker [1] and by Bermudez and Schimpf [2].

A general, unifying model that covers these techniques is desirable for several reasons. First, context-free grammars must be "debugged", because parser generators are rarely (if ever!) able to produce a working parser from one's first version of the grammar. Diagnostics provided by the typical parser generator are seldom of much help in the grammar debugging process. Hence it is usually very difficult for the user to identify problems in his grammar. Second, when the user finally does identify the problems, the required changes often wreak havoc with semantics, and yield a cluttered, unnatural, and unreadable syntax specification. Using several different parser generators does not alleviate this problem, because virtually all parser generators are completely incompatible with one another. This is due to the wide variety of algorithms and techniques, which are described using many different notations, as mentioned above.

A solution to this problem is the model presented in this paper, which unifies, and describes with a single notation, virtually all the techniques mentioned above. Its generality stems from a single

insight: the sets of lookahead strings can be obtained by "simulating-ahead" the LR(0) parser's moves, starting from the point of conflict. Doing so requires simulating the changes that would take place on the LR(0) parser's stack. The parsing technique that results from the simulation is determined by three criteria:

(1) How much of the stack one is willing to store during the simulation. We call this parameter M (Maximum stack size); it is a positive integer, but its value may also be infinity (∞) to indicate that no limit on stack size is imposed during the simulation. This parameter provides a generalization of the difference between NQLALR(1) (when M is finite), and LALR(1) (when M is infinite). As we shall see, there exists a similar difference between NQSLR and SLR parsers.
(2) Whether or not the context implied by the conflicting state is to be used during the simulation. We call this parameter C (left Context). C is a truthvalue: if C is false, then one obtains a "Simple" LR parser (in the spirit of the SLR(k) technique); if C is true, the technique obtained is akin to LALR.
(3) The maximum amount of lookahead L. This parameter determines the worst-case duration of the simulation; it is a positive integer, whose value may also be ∞ to indicate that arbitrary lookahead is desired.

The model, which we call LAR(M, C, L), defines an infinite number of grammar classes, since both M and L may have an infinite number of values, and C may be either true or false.

Our approach consists of constructing a "LookAhead Finite State Automaton" for each inconsistent LR(0) state. This machine is constructed so as to deterministically accept the first L symbols of any string that the LR(0) parser could recognize, if it were to begin parsing at the point of conflict. During parse time, when the LR(0) parser arrives at an inconsistent state, the main parser is temporarily suspended. The lookahead finite-state machine is invoked to scan the lookahead symbols, and to make the correct parsing decision based on that lookahead. The decision is transmitted back to the main parser, which acts upon that decision, and then resumes parsing. The model has the following advantages:

(1) Several important parsing techniques (most notably SLR(k), LALR(k), and a few subsets of LR-Regular) can be obtained using specific settings of the three aforementioned parameters. Instead of using several different (and severely incompatible) parser generators, or bending the grammar out of its natural shape to accommodate the one parser generator that is available, one may manipulate these parameters to find the technique that is most suitable for a given grammar.
(2) The technique progressively attempts single-symbol, multi-symbol and arbitrary lookahead, **only** where necessary.
(3) The model permits a clear understanding of the relationships among these various grammar classes. In fact, these relationships exhibit a rather elegant lattice structure, shown later.
(4) The model sheds light on the issue of decidability of the class membership problem. Specifically, we show that as long as either M or L are finite, the question of whether an arbitrary grammar is in the corresponding grammar class, is decidable. We also show that when L and M are infinite, the corresponding grammar classes can be described as SLR(∞) and LALR(∞). For these two classes, the grammar class membership problem is undecidable.

The remainder of this paper is organized as follows. In Section 2 we give some background and terminology. In Section 3 we describe the operation of the parser. In Sections 4 and 5 we give the method for constructing the parser, and discuss termination conditions. In Section 6 we prove the parser correct. In Section 7 we show how SLR(k) and LALR(k) parsers can be obtained from this model by making appropriate settings of the parameters. In Section 8 we discuss the relationships among the eight grammars classes defined by the model. Finally, in Section 9 we present conclusions.

2. BACKGROUND AND TERMINOLOGY

Here we briefly review context-free grammars, finite-state automata and LR(0) parsers. A detailed presentation is given in [7].

A **context-free grammar** (CFG) is a quadruple $G = (N, T, P, S)$, where N and T are finite disjoint sets of nonterminals and terminals respectively, $S \in N$ is the **start symbol**, and P is a finite set of productions of the form $A \to \omega$, where $A \in N$ and $\omega \in (N \cup T)^*$. The following (usual) conventions will hold:

$$\begin{aligned} A, B, C, \ldots &\in N; \\ t, a, b, c, \ldots &\in T; \\ \ldots, X, Y, Z &\in (N \cup T); \\ \ldots, x, y, z &\in T^*; \\ \alpha, \beta, \gamma, \ldots &\in (N \cup T)^*; \\ \epsilon &\text{ is the empty string;} \\ p, q, r, s &\text{ are states.} \end{aligned}$$

Relations $\Rightarrow$ and $\Rightarrow^*$ are defined in the usual way. The **language** generated by G, denoted L(G), is the set $L(G) = \{z \in T^* \mid S \Rightarrow^* z\}$. An element of L(G) is called a **sentence**. We assume all CFGs to be **reduced**, i.e. every production is used in the derivation of some sentence. We also require every CFG to contain two productions of the form $S \to S\perp$ and $S \to S'$, where S and $\perp$ do not appear in any other production. Every sentence is thus "padded" with an arbitrary number of "end-of-file" markers "$\perp$". These will be required later by the parser.

A **deterministic finite-state automaton (DFA)** D is a quintuple $D = (K, T, \Delta, \text{Start}, F)$, where K is a finite set of **states**, T is as above, $\Delta: K \times T \to K$ is a finite set of **transitions** of the form

$$p \xrightarrow{t} q,$$

$\text{Start} \in K$ is the **start state**, and $F \subseteq K$ is the set of **final states.** A **path** in DFA D is a sequence of states $q_0 q_1 \ldots q_n$ such that for some $t_1 t_2 \ldots t_n \in T^n$ $(n \geq 0)$,

$$q_0 \xrightarrow{t_1} q_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} q_n,$$

and is denoted $[q_0 : t_1 t_2 \ldots t_n]_D$. The subscript D will be omitted whenever the intent is clear. $\text{Top}[q_0 : t_1 t_2 \ldots t_n]$ denotes q_n, and the **length** of path $[q_0 : t_1 t_2 \ldots t_n]$ is defined as $n + 1$. The first state in a path is omitted if it is the start state of D; hence $[t_1 t_2 \ldots t_n]$ denotes $[\text{Start}: t_1 t_2 \ldots t_n]$. Our intent is to use paths as stacks, or as suffixes thereof. Hence paths can be "truncated" to a given maximum length M:

$$[q : tz]^M = \begin{cases} [\text{Top}[q : t] : z]^M, & \text{if } |tz| \geq M \\ [q : tz], & \text{otherwise.} \end{cases}$$

Note that when a path is truncated to length M, the **last** M states in the path are kept. The definition applies even when $M = \infty$, i.e. a path truncated to length ∞ is not truncated at all. The **language** recognized by FSA M, denoted L(M), is $\{z \in T^* \mid \text{Top}[z] \in F\}$. A FSA M is **reduced** if for all $q \in K$, there exists a path from Start to q, and a path from q to some final state.

An LR(0) parser for a CFG G is a quintuple $LR_0 = (G, K, \text{Start}, \text{Next}, \text{Reduce})$, where K is a finite set of **parse states,** $\text{Start} \in K$ is the **start state,** Next is the transition function for the characteristic finite-state automaton $CA = (K, N \cup T, \text{Next}, \text{Start}, K)$, and Reduce: $K \to 2^P$ is the **reduce function.** By construction (see [4] and [7]), CA is deterministic. However, LR_0 might be nondeterministic, as we show next.

A **configuration** of parser LR_0 is a pair denoted $[\alpha]z$, where $[\alpha]$ is the parse-time state stack (i.e. a path through CA, beginning at Start) and $z \in T^*$ is the remaining input to be parsed. The moves made by LR_0 are defined by relation "$\mapsto$" (pronounced "moves to"), as follows:

- **Shift:** $[\alpha]tz \mapsto [\alpha t]z$ iff Next(Top$[\alpha]$, t) is defined;
- **Reduce:** $[\alpha\omega]z \mapsto [\alpha A]z$ iff $A \to \omega \in$ Reduce (Top$[\alpha\omega]$).

Note that it is the top state of the stack that determines which move(s) apply. An LR(0) state is **inconsistent** if either both moves apply (shift/reduce conflict), or more than one reduce-move applies (reduce/reduce conflict). LR_0 is nondeterministic if it has one or more inconsistent states. Note that LR_0 is "built upon" CA, by adding the Reduce function and the parse-time stack. The nondeterminism is introduced by the Reduce function, since every conflict involves at least one reduction. The **language** recognized by LR_0, denoted $L(LR_0)$, is the set $L(LR_0) = \{z \in T^* \mid [\epsilon]z\perp \mapsto^+ [S]\perp\}$. The following is a well known result: $L(LR_0) = L(G)$, i.e. LR_0 (nondeterministically) accepts only "correct" strings.

LALR(k) lookahead sets are defined for each state q and each conflicting action (shift or reduce), as the set of strings of length k that the LR(0) parser could accept by (1) performing the conflicting action in question at state q, and (2) performing some sequence of applicable moves.

$$LA_k(q, A \to \omega) = \{t_1 t_2 \ldots t_k \mid [\alpha\omega]t_1 t_2 \ldots t_k y \mapsto [\alpha A]t_1 t_2 \ldots t_k y \mapsto^* [S]\perp, \text{Top}[\alpha\omega] = q\}.$$

$$LA_k(q, t) = \{t_1 t_2 \ldots t_k \mid [\alpha]t_1 t_2 \ldots t_k y \mapsto [\alpha t_1]t_2 \ldots t_k y \mapsto^* [S]\perp, \text{Top}[\alpha] = q, t = t_1\}.$$

A CFG G is LALR(k) if and only if for every inconsistent state q in its LR(0) automaton, and for each $\Omega_1, \Omega_2 \in (T \cup P)$ such that $\Omega_1 \neq \Omega_2$, $LA_k(q, \Omega_1)$ and $LA_k(q, \Omega_2)$ are disjoint.

3. LAR(M, C, L) PARSERS

We will define a CFG to be LAR(M, C, L) if its constructed LAR(M, C, L) **parser** satisfies a certain (rather simple) property. The LAR(M, C, L) parser is the corresponding LR(0) parser augmented with a collection of finite-state automata; there is one such **lookahead automaton** per inconsistent state in the LR(0) parser. Hence the LAR(M, C, L) parser is constructed "on top of" the LR(0) parser with the intent of providing it with the determinism it lacks. At parse time, whenever the LR(0) parser "lands" in an inconsistent state, it is temporarily suspended, and the corresponding lookahead automaton is invoked to perform the lookahead. After "looking ahead" L symbols or less, the lookahead automaton reaches a final state, and the correct action (either the symbol on which to shift or the production on which to reduce) is sent to the LR(0) parser via the **Decision** function, allowing it to resume parsing. There is one Decision function per lookahead automaton.

Definition 3.1: A LAR(M, C, L) parser for a CFG G, denoted $LAR_{M,C,L}$, is a pair (LR_0, LAA), where $LAA = \{(LAA_q, \text{Decision}_q) \mid q \in K \text{ is inconsistent}\}$. LR_0 is the LR(0) parser for G, and for each inconsistent state q, $LAA_q = (LAS_q, T, \text{Next}_q, q, FLAS_q)$. LAS_q contains **lookahead states**, $FLAS_q$ contains **final lookahead states**, and Next_q is LAA_q's transition function. $\text{Decision}_q: FLAS_q \to (T \cup P)$ returns either a terminal symbol or a production from each final lookahead state. ●

We will describe the construction of LAA_q in Section 4. For now, we assume that the lookahead automaton has been built somehow, and describe the operation of the parser in terms of the lookahead automaton.

Definition 3.2: A **configuration** of a LAR(M, C, L) parser is a triple denoted $[\alpha]z:n$, where n is the **current lookahead depth**, and $[\alpha]z$ is the same as for the LR(0) parser. ●

A configuration $[\alpha]z:n$ describes the current status of the parser in the following terms: the parser has reduced a certain initial portion of the input to string α, and the remaining input is z; α spells a path in the LR(0) parser from its start state to some state q; in attempting to decide the parser's

next move, q's lookahead automaton has been invoked, and so far it has scanned (i.e. performed lookahead on) the first n symbols of z; when $n = 0$, either the lookahead automaton has not yet been invoked, or q is consistent and there is no need for lookahead.

The LAR(M, C, L) parser can make five types of moves. Intuitively, these are (1) shift as LR_0 would, (2) reduce as LR_0 would, (3) look ahead one more symbol, (4) resolve a conflict in favor of a shift move, and (5) resolve a conflict in favor of a reduce move. These five are described by relation

$$\text{“}\overset{\text{lar}}{\vdash\!\!\rightarrow}\text{”}.$$

Definition 3.3: $\overset{\text{lar}}{\vdash\!\!\rightarrow}$ is the union of the following five moves:

(1) **shift move:** $[\alpha]tz{:}0 \overset{\text{lar}}{\vdash\!\!\rightarrow} [\alpha t]z{:}0$ iff $\text{Top}[\alpha]$ is consistent and $[\alpha]tz \vdash\!\!\rightarrow [\alpha t]z$;

(2) **reduce move:** $[\alpha\omega]z{:}0 \overset{\text{lar}}{\vdash\!\!\rightarrow} [\alpha A]z{:}0$ iff $\text{Top}[\alpha\omega]$ is consistent and $[\alpha\omega]z \vdash\!\!\rightarrow [\alpha A]z$;

(3) **LA-scan move:** $[\alpha]wtz{:}n \overset{\text{lar}}{\vdash\!\!\rightarrow} [\alpha]wtz{:}n+1$ iff $q = \text{Top}[\alpha]$ is inconsistent, $|w| = n$, and $\text{Lookahead}_q(\text{Top}[q{:}w],t)$ is defined;

(4) **LA-shift move:** $[\alpha]tyz{:}n \overset{\text{lar}}{\vdash\!\!\rightarrow} [\alpha t]yz{:}0$ iff $q = \text{Top}[\alpha]$ is inconsistent, $[\alpha]tyz \vdash\!\!\rightarrow [\alpha t]yz$, $|ty| = n$, and $\text{Decision}_q(\text{Top}[q{:}ty]_{LAA_q}) = t$;

(5) **LA-reduce move:** $[\alpha\omega]wz{:}n \overset{\text{lar}}{\vdash\!\!\rightarrow} [\alpha A]wz{:}0$ iff $q = \text{Top}[\alpha\omega]$ is inconsistent, $[\alpha\omega]wz \vdash\!\!\rightarrow [\alpha A]wz$, $|w| = n$, and $\text{Decision}_q(\text{Top}[q{:}w]_{LAA_q}) = A \rightarrow \omega$.

●

In the first two moves, no lookahead is necessary; no LAA_q of $LAR_{M,C,L}$ are invoked. In the LA-scan move, LAA_q has already scanned string w, and $|w| < L$; more input is needed before the conflict in state q can be resolved. LAA_q advances on t, and increments the lookahead depth. When $n = 0$, this move invokes LAA_q. In the LA-shift move, LAA_q has resolved the conflict by looking ahead n symbols in the input; the correct action is to shift on t. LAA_q is deactivated, n is reset to zero, and the parser proceeds. The LA-reduce move operates in a similar fashion: the correct action is to reduce using $A \rightarrow \omega$, and the parser acts accordingly. Clearly at most one of the previous five applies to a given configuration; hence $LAR_{M,C,L}$ is deterministic. It should be noted that **no** move might apply; it is also possible to scan the remainder of the input, and enter a "loop" of the form

$$r \overset{\perp}{\rightarrow} r,$$

with no other way out of r. In either case the parser will get "stuck"; we will later give conditions under which this cannot occur.

An example of an LAR parser is shown in Fig. 1. The grammar used contains the productions $S \rightarrow S\perp$, $S \rightarrow X$, $X \rightarrow afDd$, $X \rightarrow AfDc$, $D \rightarrow eb$, and $A \rightarrow a$. The only inconsistent state in the LR(0) parser is state 5, which has a shift/reduce conflict. Four symbols of lookahead are needed, since the parser must look beyond symbols "feb" to examine the deciding symbol, which is either "c" or "d". The lookahead automaton of state 5 is also depicted. A sample parse, of input string "afebd⊥", is shown in the figure. Each step in the parse is marked with its number from definition 3.3. After initially shifting on "a", the parser arrives in state 5. Here it invokes the lookahead automaton, and scans the input, examining symbols f, e, b, and d. Upon scanning the "d", the lookahead automaton enters a final state, from which it obtains the correct action, namely to shift on "f". The parser resumes at state 5, performs the shift, and the remainder of the parse proceeds uneventfully.

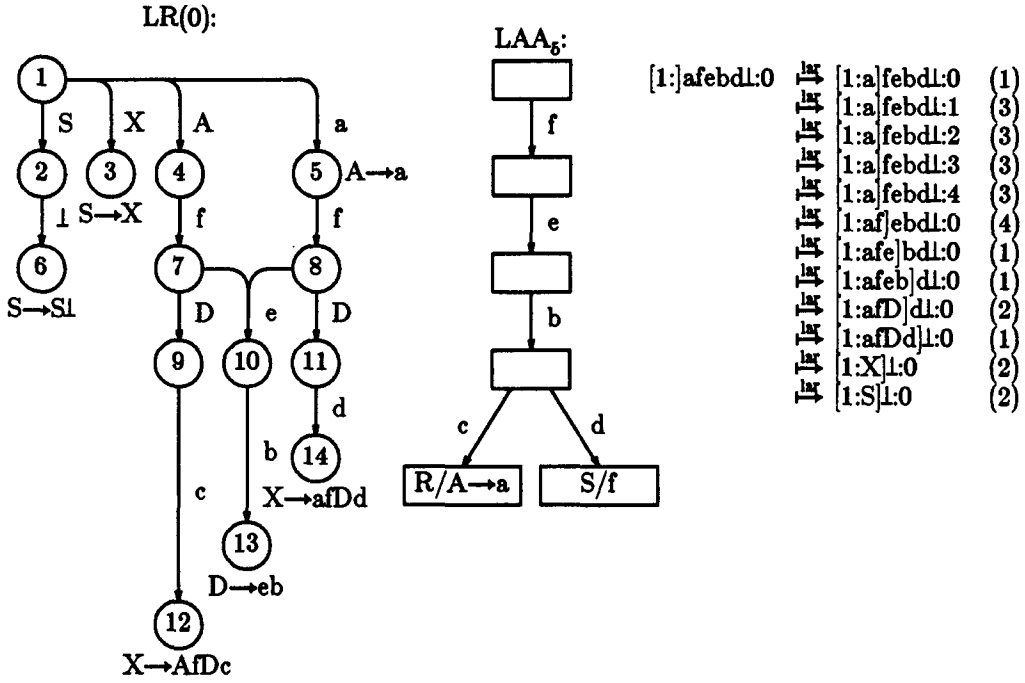

Fig. 1. An LR(0) parser, a lookahead automaton, and a sample parse.

Definition 3.4: The **language** recognized by $LAR_{M,C,L}$ is $\{z \in T^* | [\epsilon]z\perp:0 \overset{lar}{\mapsto}{}^* [S]\perp:0\}$.●

$LAR_{M,C,L}$'s moves are ultimately LR_0's moves, interspersed with sequences of lookahead scan moves that deterministically perform the lookahead and (hopefully) determine the correct action. Such sequences of lookahead scan moves must force LAA_q into a final state; otherwise no parsing decision will take place. Hence an important property of LAA_q is whether or not it is **reduced**, i.e. whether there is a path from every state to some final state. We later prove that if this property is satisfied by all lookahead automata, both the LAR(M, C, L) parser and the LR(0) parser accept the same language, and hence the LAR(M, C, L) parser is correct.

Definition 3.5: A CFG is LAR(M, C, L) iff LAA_q is reduced, for all inconsistent $q \in K$.●

Having described the operation of LAR(M, C, L) parsers, we now turn to the problem of constructing them.

4. CONSTRUCTION OF LAR(M, C, L) PARSERS

The LAR(M, C, L) parser is constructed from the LR(0) parser by constructing LAA_q and $Decision_q$, for each inconsistent state $q \in K$. The construction method is similar to the construction of LR(0) parsers: each state is an item set, and closure and successor operations are applied to each item set. Items in LAA_q describe suffixes of LR_0's stack. The construction method consists of

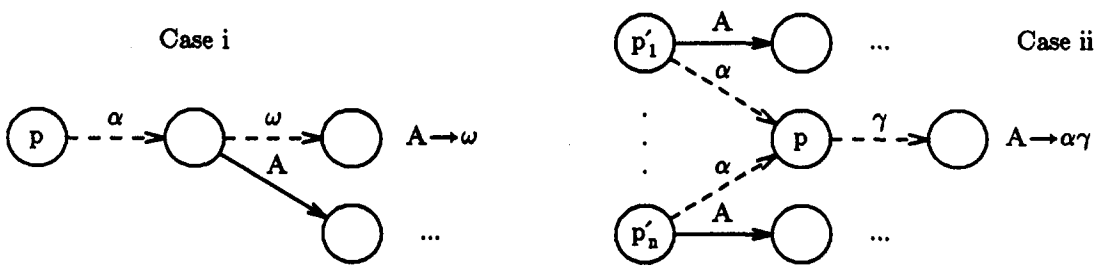

Fig. 2. The Reduces relation.

simulating the LR(0) parser's moves, and recording these moves using items. We first formally define lookahead items.

Definition 4.1: A **lookahead item** is a pair $([p:\alpha], \Omega)$. $\Omega \in (T \cup P)$ is called an **action**. A **lookahead state** is a set of lookahead items; it is called **final** iff all items in it have the same action.●

Two relations, called Reduces and Reads, describe the simulation of the LR(0) parser's moves.

Definition 4.2: For a parser $LAR_{M,C,L}$, Reduces $\subset (K^* \times K^*)$ is the union of all $Reduces_{A \to \omega}$, where for all $A \to \omega \in P$:

(i) $[p:\alpha\omega]$ $Reduces_{A \to \omega}$ $[p:\alpha A]^M$ iff $A \to \omega \in$ Reduce(Top $[p:\alpha w]$);

(ii) $[p:\gamma]$ $Reduces_{A \to \omega}$ $[p':A]^M$ iff $A \to \omega \in$ Reduce (Top$[p:\gamma]$), $\omega = \alpha\gamma$, and

(if C = true, then Top$[p':\alpha] = p$).●

Relation $Reduces_{A \to \omega}$ simulates a reduction of phrase ω to nonterminal A. The situation is shown pictorially in Fig. 2. In case (i), stack suffix $[p:\alpha\omega]$ is long enough to accommodate the entire phrase being reduced (ω). We backtrack on ω and then move forward on A: the resulting (unique) stack suffix is $[p:\alpha A]$, truncated to maximum stack length M. In case (ii), the phrase being reduced ($\alpha\gamma$) is at least as long as the stack suffix ($[p:\gamma]$), i.e. we must backtrack on γ (back to state p), **and further** on α (if such left context is to be used, i.e. if C is true) in all possible ways, before moving forward on A. If C is false, then the "further" backtracking does not take place, and we simply "jump" to any state with a transition on A, thereby ignoring the said left context. The resulting stack suffixes are of the form $[p':A]$, truncated to maximum stack length M. In case ii) in figure 2, the paths on α are required to be "backtracked on" only if C is true.

Definition 4.3: For a parser $LAR_{M,C,L}$, Reads $\subset (K^* \times K^*)$ is the union of $Reads_t$, where for all $t \in T$, $[p:\alpha]$ $Reads_t[p:\alpha t]^M$ iff Top$[p:\alpha t]$ is defined.●

The construction method of LAA_q is similar to that of LR(0) parsers: each state is an item-set, and Closure and Successor operations are applied to each item-set. The successors of the start state receive special treatment, in which the conflicting shift and reduce actions are simulated separately, to synchronize the simulation of subsequent moves.

We begin by defining LAA_q's start state as an empty set of items. We then compute the "First-successors", i.e. the successors of the start state (one successor per $t \in T$), as follows:

(1) Add to the t-successor state an item of the form $([q:t], t)$, if $[q:]Reads_t[q:t]$.

(2) Add (further) to the t-successor state an item of the form $([p':\alpha t], A \to \omega)$, if $[q:]Reduces_{A \to \omega}$ $[p:A]Reduces^*[p':\alpha]Reads_t[p':\alpha t]$.

Thus the t-successor of the start state contains items describing how the LR(0) parser could carry out (1) a conflicting shift move on t, and/or (2) a conflicting reduce move on $A \to \omega$, followed by a sequence of reduce moves, followed by a shift move on t. It should be evident that the separate treatment of conflicting shift and reduce moves is necessary to synchronize the remainder of the simulation.

We then simulate all applicable moves, as follows:

(1) A reduce-move is simulated by adding another item, i.e. a (path, action) pair, to the current lookahead state. The path is the resulting one from the reduce-move (described via Reduces), and the action is the same as before the reduce-move. The simulation of all reduce-moves constitutes the Closure operation.

(2) A shift-move (on symbol t) is simulated by creating a transition (on t) from the current lookahead state to another (possibly new) lookahead state. An item is added to the new state, in which the path is that produced by the shift-move (via Reads), and the action is the same as before the shift-move. The simulation of shift-moves constitutes the Successor operation. It is important to note that the new lookahead state may already exist, in which case a loop in the lookahead FSA may be introduced. Such loops allow our model to encompass arbitrary lookahead.

Two criteria are used to end the simulation:

(1) A lookahead state r will have no successors if it is final, i.e. if the lookahead strings that spell the various paths to r, resolve the conflict.
(2) A lookahead state r will have no successors if some path leading to it from the start state has length L or greater, i.e. if the lookahead limit L has been exceeded.

Summarizing, after special treatment of the first lookahead symbol, we simulate all applicable moves until either (1) a lookahead state is reached, all of whose items are derived from the same initial "forced" action, or (2) the maximum lookahead parameter L is exceeded. In the formal definition of the lookahead automaton construction (below), we write $X = {}_s F(X)$ to mean that X is the smallest set satisfying $X = F(X)$.

Definition 4.4: For each inconsistent state $q \in K$:

$$LAA_q = (LAS_q, T, Lookahead_q, q, FLAS_q);$$

$$LAS_q = {}_s\{q\} \cup \{First\text{-}succ(q, t) \mid t \in T\}$$
$$\cup \{Closure(r) \mid r \in Successor(r'), r' \in (LAS_q - FLAS_q)\};$$

$$First - succ(q, t) = Closure(Nucleus(Forced - reduce(q), t) \cup Forced - shift(q, t));$$

$$Forced - reduce(q) = Closure(\{([p:A], A \to \omega) \mid [q:\epsilon]Reduces_{A \to \omega}[p:A]\});$$

$$Forced - shift(q, t) = \{([q:t], t) \mid [q:\epsilon]Reads_t[q:t]\};$$

$$Closure(r) = r \cup \{([p:\alpha], \Omega) \mid ([p':\alpha'], \Omega) \in Closure(r), [p':\alpha']Reduces_{A \to \omega}[p:\alpha]\};$$

$$Successors(r) = \{Nucleus(r, t) \mid t \in T, r \notin FLAS_q, \text{ for all } t_1 \ldots t_n$$
$$\text{such that } Top[q:t_1 \ldots t_n]_{LAA_q} = r, n < L\};$$

$$Nucleus(r, t) = \{([p:\alpha t], \Omega) \mid ([p:\alpha], \Omega) \in r, [p:\alpha]Reads_t[p:\alpha t]\};$$

$$FLAS_q = \{r \in LAS_q \mid r \text{ is final}\};$$

$$Lookahead_q = (q \xrightarrow{t} First - succ(q, t) \mid t \in T\}$$

$$\cup \{r \xrightarrow{t} r' \mid r, r' \in LAS_q, t \in T, r' = Closure(Nucleus(r, t))\}.\bullet$$

The Decision function returns the common action Ω of all items in a given final lookahead state.

Definition 4.5: For all $f \in FLAS_q$, $Decision_q(f) = \Omega$ iff some $([p:\alpha], \Omega) \in f$.●

Four sample lookahead automata are shown in Fig. 3. All four are for state 5 in the LR(0) automaton in Fig. 1, and all are built using C = true. We examine them in turn, from left to right.

(1) Since L = 3, the construction stops after the third lookahead symbol, i.e. after creating the state accessed by "b". Three symbols of lookahead are not sufficient to solve the conflict. Thus no state in this automaton is final, and hence the automaton is not reduced.
(2) Since L = 4, the construction stops after four symbols of lookahead. These are sufficient to solve the conflict: the two actions (Shift on "f", Reduce on A → a) "migrate" to two different lookahead states, each of which is final. Since such final states are reachable from every state, the automaton is reduced.
(3) In this automaton we have M = 2. After scanning string "feb", the automaton has "forgotten" how it entered state 10 (from either 8 or 7), and must then "backtrack" out both ways for each action. Since L = 4, the construction is stopped after four lookahead symbols, but without ever creating a final lookahead state. Thus the automaton is not reduced. In contrast, automaton number 2 "remembers" this information, due its value of M = 5. It should be evident that a value of $M \geq 3$ is sufficient.

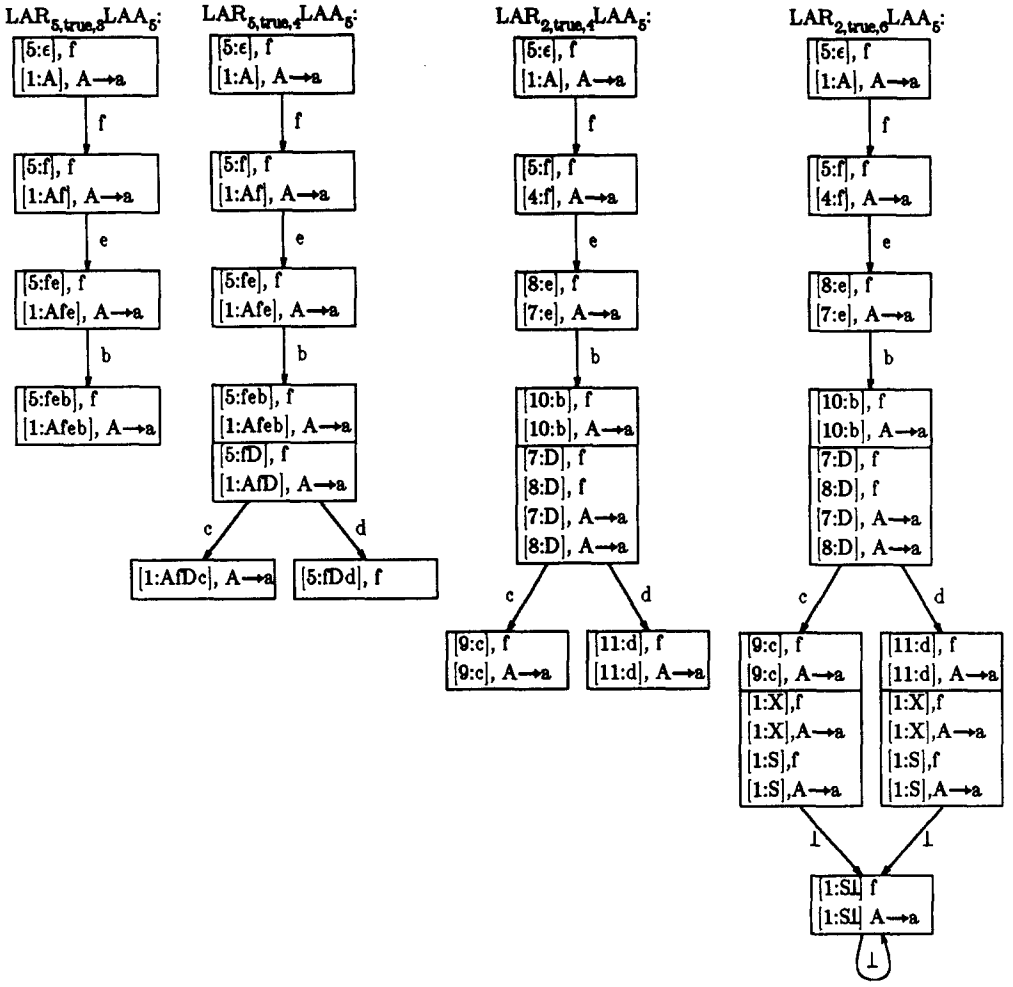

Fig. 3. Four lookahead automata for the LR(0) parser in Fig. 1.

(4) This automaton has the same problem as the previous one: contextual information is lost due to the value of M being too small. However, since L = 6, the construction continues until the end-of-file marker ⊥ is reached. A "trap" is created, and the automaton is not reduced.

In all three non-reduced automata (i.e. 1, 3 and 4), the problem is clearly caused by a value of M that is too small, or a value of L that is too small. The automata will be reduced if $M \geq 3$ and $L \geq 4$. For this grammar, the value of C makes no difference. Thus this grammar is LAR(M, C, L) if and only if $M \geq 3$ and $L \geq 4$.

5. CONSTRUCTION TERMINATION

At this point, the construction method shown is not yet an algorithm. Two reasons may prevent the above construction method from terminating.

(1) Some lookahead state r might have an infinite number of items, i.e. there may be an infinite sequence of reductions of the empty string (and nonterminals generating it). This can occur only if $M = \infty$, because if M is finite, then the number of items is also finite. The infinite sequence of reductions of the empty string can be pre-checked with a test described by DeRemer and Pennello (see [6]), who defined a relation among nonterminal transitions in the LR(0) parser called *reads*: (p, A) *reads* (r, C)

$$\text{iff } p \xrightarrow{A} r \xrightarrow{C} s,\ C \Rightarrow^* \epsilon,$$

for some s. They also proved that if the graph induced by *reads* contains a cycle, the grammar is not LR(k), for any k. From now on, we will assume that all LR(0) parsers have been pre-checked for this condition.

(2) The Successors operation may forever create new lookahead states. This will occur only if both $M = \infty$ and $L = \infty$, because: (1) if M is finite, then the number of possible items is finite, and hence the number of possible lookahead states is finite and (2) if L is finite, then no path of length greater than L will be constructed, and the worst-case number of lookahead states is $|T|^L$, i.e. the number of nodes in a tree of depth L in which each node has $|T|$ children. Thus the following result.

Lemma 5.1: The construction of $LAR_{M,C,L}$ will terminate if either M or L is finite.●

6. CORRECTNESS OF LAR(M, C, L) PARSERS

As mentioned, $LAR_{M,C,L}$'s moves are ultimately LR_0's moves, interspersed with sequences of lookahead scan moves that perform the lookahead and determine the correct action.

Lemma 6.1: $L(LAR_{M,C,L}) \subseteq L(LR_0)$●

Proof: Given a sequence of $LAR_{M,C,L}$ moves that accepts some string z, one may (1) remove all lookahead-scan-moves, and (2) replace all lookahead-shift and lookahead-reduce moves with shift and reduce moves respectively. The result is the corresponding sequence of LR_0 moves that also accepts z.●

The converse (i.e. $L(LR_0) \subseteq L(LAR_{M,C,L})$ is true only if the grammar is LAR(M, C, L), i.e. if all lookahead automata are reduced, as we show next.

Theorem 6.1: If a CFG is LAR(M, C, L), then $L(LR_0) = L(LAR_{M,C,L})$.●

Proof: We need only prove that $L(LR_0) \subseteq L(LAR_{M,C,L})$, for any CFG that is LAR(M, C, L). Let $z \in L(LR_0)$. Then $[\epsilon]z\perp \mapsto^* [S]\perp$. We will build a sequence of $LAR_{M,C,L}$ moves that parses z. First, replace every move of the form $[\alpha]x \mapsto [\beta]y$ such that $Top[\alpha]_{CA}$ is consistent, with the corresponding $LAR_{M,C,L}$ move

$$[\alpha]x:0 \overset{lar}{\mapsto} [\beta]y:0.$$

The remaining LR_0 moves in the sequence are of one of the following two forms: (1) $[\alpha]ty \mapsto [\alpha t]y$, $Top[\alpha]_{CA}$ is inconsistent; (2) $[\alpha\omega]y \mapsto [\alpha A]y$, $Top[\alpha\omega]_{CA}$ is inconsistent. Consider a move of type (1). By construction, and since all lookahead automata are reduced, $LAA_{Top[\alpha\omega]}$ accepts some prefix of y (of length L or less), so

$$[\alpha]ty:0 \overset{lar}{\mapsto} {}^n[\alpha]ty:n \overset{lar}{\mapsto} [\alpha t]y:0.$$

We replace the single LR_0 move with this sequence of $LAR_{M,C,L}$ moves. Analogous arguments apply to moves of type (2). The resulting sequence of $LAR_{M,C,L}$ moves deterministically parses z. Hence $z \in L(LAR_{M,C,L})$.●

This last result shows that we will obtain a correct, operational parser if (1) we provide values for M, C and L, (2) we build all the lookahead automata, and (3) once constructed, all the lookahead automata are reduced. It is important to note that if not all automata are reduced, other values for M, C and L may be tried. In particular, increasing M and/or L may be useful.

7. OBTAINING LALR(k) AND SLR(k) PARSERS

A useful feature of our model is that both LALR(k) and SLR(k) parsers can be obtained, by using specific settings of the three parameters. These two techniques are usually available in different, incompatible systems; in our model switching from one to the other is trivial. In addition, many other grammar classes (about which we will say more later) can also be obtained in this manner.

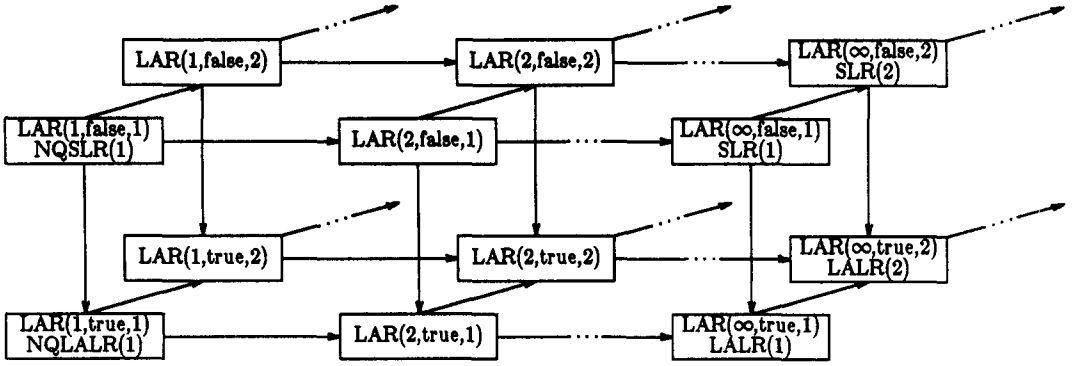

Fig. 4. Relationships among LAR(M, C, L) grammar classes.

Theorem 7.1: Let L = k be fixed. Then LAR(∞, true, k) ≡ LALR(k).●

Proof: We will prove that every LALR(k) grammar is LAR(∞, true, k), and vice-versa. Assume that G is LALR(k). Then for every inconsistent state q, $LA_k(q, \Omega_1)$ and $LA_k(q, \Omega_2)$ are disjoint, if $\Omega_1 \neq \Omega_2$. Consider a string s in any of these lookahead sets. String s is different from all the others in this set, and some prefix $t_1 \ldots t_n$ of s is accepted by $LAR_{\infty, true, k} LAA_q$. Furthermore, $r = Top[q: t_1 \ldots t_n]_{LAA_q}$ is a final lookahead state, and Decision(r) is the corresponding Ω. Thus a final lookahead state is always reachable in the lookahead automaton, which is therefore reduced. Hence G is LAR(∞, true, k). This argument applies in the reverse direction, i.e. if G is LAR(∞, true, k), then G is also LALR(k).●

Theorem 7.2: Let L = k be fixed. Then LAR(∞, false, k) ≡ SLR(k).●

Proof. The argument is analogous to that of the previous theorem, except for the validity of the SLR(k) lookahead set definitions, which we now justify. In Case ii, Definition 4.2, one may "jump" to any A-transition if C = false. This is exactly how one may compute ordinary Follow sets from the LR(0) parser, i.e. by disregarding the context implied by the conflicting state.●

8. RELATIONSHIPS AMONG GRAMMAR CLASSES

The LAR(M, C, L) grammar classes are related to each other in the rather elegant lattice structure shown in Fig. 4. The value of M varies from 1 to ∞, and from left to right the grammar classes asymptotically tend towards SLR(1) and LALR(1), for which M = ∞. The top side of the lattice contains the "simple" classes, for which C = false, that disregard the context preceding the conflict; the bottom side includes the "true" (C = true) lookahead techniques. Finally, from front to back the amount of lookahead L increases ad-infinitum, beginning with L = 1 on the front side, independently of the other two parameters.

The relationships among grammar classes are depicted with arrows, and justified with the three theorems below. Intuitively,

(1) Left-to-right arrows: A stack size of M + 1 yields a more powerful parsing technique than does a stack size of M, since truncation may "lose" contextual information. In particular, NQLALR(k) ⊆ LALR(k), a result first shown by DeRemer and Pennello [6], although they limited their analysis to k = 1. Here we see that between NQLALR(k) and LALR(k) there are an infinite number of grammar classes, and that LALR(k) is the limit as M tends towards infinity. This holds for any value of L (including ∞), and there is a similar relationship between NQSLR(k) and SLR(k).
(2) Top-to-bottom arrows: Using the left context that precedes the conflict (C = true) is more powerful than disregarding it (C = false). In particular, SLR(k) ⊆ LALR(k), a well-known result, generalized here to the "Not-Quite" techniques.
(3) Front-to-back arrows: Using L + 1 symbols of lookahead is more powerful than using only L.

Theorem 8.1: For any C and L, LAR(M, C, L) $\subseteq$ LAR(M + 1, C, L).●

Proof: The proof is by contradiction. Suppose a CFG G is LAR(M, C, L), but not LAR(M + 1, C, L). Then some $LAR_{M+1,C,L}$ lookahead automaton LAA_q is not reduced. Then there exists a lookahead state $r \in LAS_q - FLAS_q$ such that $Top[q: t_1 \ldots t_n]_{LAA_q} = r$, for some string $t_1 \ldots t_n$, and there is no path from r to a final lookahead state. Hence some $([p:\xi], \Omega_1) \in r$ and some other $([p':\xi'], \Omega_2) \in r$, where $\Omega_1 \neq \Omega_2$. Then [q:](Reduces*∘Reads)^n[p:ξ], and [q:](Reduces*∘Reads)^n [p′:ξ'], with the Reads steps on $t_1 \ldots t_n$. By definitions 4.2 and 4.3, the same Reduces and Reads steps lead to some lookahead state r′ in the corresponding $LAR_{M,C,L}$ lookahead automaton for state q, and there is no path from r′ to a final lookahead state. Hence $LAR_{M,C,L}$ contains a non-reduced lookahead automaton, and the grammar is not LAR(M, C, L), which contradicts our assumption.●

Theorem 8.2: For any M and L, LAR(M, false, L) $\subseteq$ LAR(M, true, L).●

Proof: The argument is analogous to that of theorem 8.1.●

Theorem 8.3: For any M and C, Then LAR(M, C, L) $\subseteq$ LAR(M, C, L + 1).●

Proof: Let G be a LAR(M, C, L) grammar. Then all lookahead automata in $LAR_{M,C,L}$ are reduced, and are constructed so they will accept no lookahead string of length greater than L. For the value L + 1, the lookahead automata will still be reduced. Hence G is LAR(M, C, L + 1).●

9. CONCLUSIONS

We have presented the class of LAR(M, C, L) grammars, which include well-known classes such as LALR(k), SLR(k), NQLALR(k), NQSLR(k), and their arbitrary lookahead counterparts. We have shown the operation of this type of parser, how to construct it, and its correctness. We have presented the elegant nature of the relationships among the grammar classes. Only two classes, the arbitrary lookahead versions of LALR(k) and SLR(k), pose construction termination problems. This is to be expected, since it is well-known that the question of whether a grammar is LALR(k) or SLR(k), for some k, is undecidable. These are also the two largest grammar classes, which illustrates a trade-off between parsing power and construction termination. This model describes, with a single notation and construction method, four finite lookahead and four arbitrary lookahead LR parsing techniques. The user of a parser generator based on this model may freely "navigate" among the various grammar classes, in search of a suitable parsing technique. The simultaneous availability of this many techniques is the main strength of the model.

REFERENCES

1. Baker, T. P. Extending lookahead for LR parsers. *J. Comput. Syst. Sci.* **22:** 243–259; 1981.
2. Bermudez, M. E. and Schimpf, K. A practical arbitrary lookahead LR parsing technique. *Proc. ACM SIGPLAN '86 Symposium on Compiler Construction*, pp. 136–144; 23–27 June 1986.
3. Culik, C. and Cohen, R. LR-regular grammars—an extension of LR(k) grammars. *J. Comput. System Sci.* **7:** 66–96; 1973.
4. DeRemer, F. L. Practical translators for LR(k) languages. Ph.D. thesis, Department of Electrical Engineering, MIT, Cambridge, MA; 1969.
5. DeRemer, F. L. Simple LR(k) grammars. *Commun. ACM* **14**(7): 453–460; July 1971.
6. DeRemer, F. L. and Pennello, T. J. Efficient computation of LALR(1) lookahead sets. *ACM TOPLAS* **4**(4): 615–649; October 1982.
7. Harrison, M. *An Introduction to Formal Language Theory*. Reading, MA: Addison–Wesley; 1978.
8. Kristenseen, B. B. and Madsen, O. L. Methods for computing LALR(k) lookahead. *ACM TOPLAS* **3**(1): 60–82; January 1981.
9. Park, J. C. H., Choe, K. M. and Chang, C. H. A new analysis of LALR formalisms. *ACM TOPLAS* **7**(1): 159–175; January 1985.

About the Author—MANUEL E. BERMUDEZ received the B.S. and Licenciado degrees in computer science in 1979 and 1980 from the University of Costa Rica, where he was then employed as an instructor. In 1981 he emigrated to the United States, and began graduate studies at the University of California at Santa Cruz, where he obtained the M.Sc. degree in 1982, and the Ph.D. in 1984, both in computer and information sciences. In 1984–85 he was a Visting Assistant Professor at the University of California, Santa Cruz. Since August 1985, he has been an Assistant Professor in the Department of Computer and Information Sciences at the University of Florida. Dr Bermudez's research has centered on the unification and simplification of LR parsing techniques. He is a member of ACM and IEEE.