# Generalized Left-Corner Parsing

**Mark-Jan Nederhof** *

*University of Nijmegen, Department of Computer Science*
*Toernooiveld, 6525 ED Nijmegen, The Netherlands*
markjan@cs.kun.nl

## Abstract

We show how techniques known from generalized LR parsing can be applied to left-corner parsing. The resulting parsing algorithm for context-free grammars has some advantages over generalized LR parsing: the sizes and generation times of the parsers are smaller, the produced output is more compact, and the basic parsing technique can more easily be adapted to arbitrary context-free grammars.

The algorithm can be seen as an optimization of algorithms known from existing literature. A strong advantage of our presentation is that it makes explicit the role of left-corner parsing in these algorithms.

**Keywords:** Generalized LR parsing, left-corner parsing, chart parsing, hidden left recursion.

## 1 Introduction

Generalized LR parsing was first described by Tomita [Tomita, 1986; Tomita, 1987]. It has been regarded as the most efficient parsing technique for context-free grammars. The technique has been adapted to other formalisms than context-free grammars in [Tomita, 1988].

A useful property of generalized LR parsing (henceforth abbreviated to *GLR parsing*) is that input is parsed in polynomial time. To be exact, if the length of the right side of the longest rule is $p$, and if the length of the input is $n$, then the time complexity is $\mathcal{O}(n^{p+1})$. Theoretically, this may be worse than the time complexity of Earley's algorithm [Earley, 1970], which is $\mathcal{O}(n^3)$. For practical cases in natural language processing however, GLR parsing seems to give the best results.

The polynomial time complexity is established by using a *graph-structured stack*, which is a generalization of the notion of *parse stack*, in which pointers are used to connect stack elements. If nondeterminism occurs, then the search paths are investigated simultaneously, where the initial part of the parse stack which is common to all search paths is represented only once. If two search paths share the state of the top elements of their imaginary individual parse stacks, then the top element is represented only once, so that any computation which thereupon pushes elements onto the stack is performed only once.

Another useful property of GLR parsing is that the output is a concise representation of all possible parses, the so called *parse forest*, which can be seen as a generalization of the notion of *parse tree*. (By some authors, parse forests are more specifically called *shared*, *shared-packed*, or *packed shared* (parse) forests.) The parse forests produced by the algorithm can be represented using $\mathcal{O}(n^{p+1})$ space. Efficient decoration of parse forests with attribute values has been investigated in [Dekkers *et al.*, 1992].

There are however some drawbacks to GLR parsing. In order of decreasing importance, these are:

- The parsing technique is based on the use of LR tables, which may be very large for grammars describing natural languages.[1] Related to this is the large amount of time needed to construct

*Supported by the Dutch Organization for Scientific Research (NWO), under grant 00-62-518

[1] [Purdom, 1974] argues that grammars for programming languages require LR tables which have a size which is about linear in the size of the grammar. It is generally considered doubtful that similar observations can be made for grammars for natural languages.

a parser. Incremental construction of parsers may in some cases alleviate this problem [Rekers, 1992].

- The parse forests produced by the algorithm are not as compact as they might be. This is because packing of subtrees is guided by the merging of search paths due to equal LR states, instead of by the equality of the derived nonterminals. The solution presented in [Rekers, 1992] implies much computational overhead.
- Adapting the technique to arbitrary grammars requires the generalization to *cyclic* graph-structured stacks [Nozohoor-Farshi, 1991], which may complicate the implementation.
- A minor disadvantage is that the theoretical time complexity worsens if $p$ becomes larger. The solution given in [Kipps, 1991] to obtain a variant of the parsing technique which has a fixed time complexity of $\mathcal{O}(n^3)$, independent of $p$, implies an overhead in computation costs which worsens instead of improves the time complexity in practical cases.

These disadvantages of generalized LR parsing are mainly consequences of the LR parsing technique, more than consequences of the use of graph-structured stacks and parse forests.

Lang [Lang, 1974; Lang, 1988c] gives a general construction of deterministic parsing algorithms from nondeterministic push-down automata. The produced data structures have a strong similarity to parse forests, as argued in [Billot and Lang, 1989; Lang, 1991]. The general idea of Lang has been applied to other formalisms than context-free grammars in [Lang, 1988a; Lang, 1988b; Lang, 1988d].

The idea of a graph-structured stack, however, does not immediately follow from Lang's construction. Instead, Lang uses the abstract notion of a table to store information, without trying to find the best implementation for this table.[2]

One of the parsing techniques which can with some minor difficulties be derived from the construction of Lang is *generalized left-corner parsing* (henceforth abbreviated to *GLC parsing*).[3] The starting-point is left-corner parsing, which was first formally defined in [Rosenkrantz and Lewis II, 1970]. Generalized left-corner parsing, albeit under a different name, has first been investigated in [Pratt, 1975]. (See also [Tanaka *et al.*, 1979; Bear, 1983; Sikkel and Op den Akker, 1992].) In [Shann, 1991] it was shown that the parsing technique can be a serious rival to generalized LR parsing with regard to the time complexities. (Other papers discussing the time complexity of GLC parsing are [Slocum, 1981; Wirén, 1987].)

A functional variant of GLC parsing for definite clause grammars has been discussed in [Matsumoto and Sugimura, 1987]. This algorithm does not achieve a polynomial time complexity however, because no "packing" takes place.

A variant of Earley's algorithm discussed in [Leiss, 1990] also is very similar to GLC parsing although the top-down nature of Earley's algorithm is preserved.

GLC parsing has been rediscovered a number of times (e.g. in [Leermakers, 1989; Leermakers, 1992], [Schabes, 1991], and [Perlin, 1991]), but without any mention of the connection with LC parsing, which made the presentations unnecessarily difficult to understand. This also prevented discovery of a number of optimizations which are obvious from the viewpoint of left-corner parsing.

In this paper we reinvestigate GLC parsing in combination with graph-structured stacks and parse forests. It is shown that this parsing technique is not subject to the four disadvantages of the algorithm of Tomita.

The structure of this paper is as follows. In Section 2 we explain nondeterministic LC parsing. This parsing algorithm is the starting-point of Section 3, which shows how a deterministic algorithm can be defined which uses a graph-structured stack and produces parse forests. Section 4 discusses how this generalized LC parsing algorithm can be adapted to arbitrary context-free grammars.

How the algorithm can be improved to operate in cubic time is shown in Section 5. The improved algorithm produces parse forests in a non-standard representation, which requires only cubic space. One more class of optimizations is discussed in Section 6.

Preliminary results with an implementation of our algorithm are discussed in Section 7.

## 2 Left-corner parsing

Before we define LC parsing, we first define some notions strongly connected with this kind of parsing.

We define a *spine* to be a path in a parse tree which begins at some node which is not the first son of its father (or which does not have a father), then proceeds downwards every time taking the leftmost son, and finally ends in a leaf.

We define the relation $\angle$ between nonterminals such that $B \angle A$ if and only if there is a rule $A \rightarrow B\ \alpha$, where $\alpha$ denotes some sequence of grammar symbols. The transitive and reflexive closure of $\angle$ is denoted by $\angle^*$, which is called the *left-corner relation*. Informally, we have that $B \angle^* A$ if and only if it is possible

[2] [Sikkel, 1990] argues that the way in which the table is implemented (using a two-dimensional matrix as in case of Earley's algorithm or using a graph-structured stack) is only of secondary importance to the global behaviour of the parsing algorithm.

[3] The term "generalized left-corner parsing" has been used before in [Demers, 1977] for a different parsing technique. Demers generalizes "left corner of a right side" to be a prefix of a right side which does not necessarily consist of one member, whereas *we* generalize LC parsing with zero lookahead to grammars which are not LC(0).

to have a spine in some parse tree in which B occurs below A (or B = A). We pronounce B $\angle^*$ A as "B is a left corner of A".

We define the set *GOAL* to be the set consisting of S, the start symbol, and of all nonterminals A which occur in a rule of the form B$\rightarrow \alpha$ A $\beta$ where $\alpha$ is not $\epsilon$ (the empty sequence of grammar symbols). Informally, a nonterminal is in *GOAL* if and only if it may occur at the first node of some spine.

We explain LC parsing by means of the small context-free grammar below. No claims are made about the linguistic relevance of this grammar. Note that we have transformed lexical ambiguity into grammatical ambiguity by introducing the nonterminals VorN and VorP.

S → NP VP
S → S PP
NP → "time"
NP → "an" "arrow"
NP → NP NP
NP → VorN
VP → VorN
VP → VorP NP
PP → VorP NP
VorN → "flies"
VorP → "like"

The algorithm reads the input from left to right. The elements on the parse stack are either nonterminals (the *goal elements*) or items (the *item elements*). *Items* consist of a rule in which a dot has been inserted somewhere in the right side to separate the members which have been recognized from those which have not.

Initially, the parse stack consists only of the start symbol, which is the first goal, as indicate in Figure 1. The indicated parse corresponds with one of the two possible readings of "time flies like an arrow" according to the grammar above.

We define a *nondeterministic LC parser* by the parsing steps which are possible according to the following clauses:

1a. If the element on top of the stack is the nonterminal A and if the first symbol of the remaining input is t, then we may remove t from the input and push an item [B → t • $\alpha$] onto the stack, provided B $\angle^*$ A.

1b. If the element on top of the stack is the nonterminal A, then we may push an item [B → •] onto the stack, provided B $\angle^*$ A. (The item [B → •] is derived from an *epsilon rule* B → $\epsilon$.)

2. If the element on top of the stack is the item [A → $\alpha$ • t $\beta$] and if the first symbol of the remaining input is t, then we may remove t from the input and replace the item by the item [A → $\alpha$ t • $\beta$].

3. If the top-most two elements on the stack are B [A → $\alpha$ •], then we may replace the item by an item of the form [C → A • $\beta$], provided C $\angle^*$ B.

4. If the top-most three elements on the stack are [B → $\beta$ • A $\gamma$] A [A → $\alpha$ •], then we may replace these three elements by the item [B → $\beta$ A • $\gamma$].

5. If a step according to one of the previous clauses ends with an item [A → $\alpha$ • B $\beta$] on top of the stack, where B is a nonterminal, then we subsequently push B onto the stack.

6. If the stack consists only of the two elements S [S→ $\alpha$ •] and if the input has been completely read, then we may successfully terminate the parsing process.

Note that only nonterminals from *GOAL* will occur as separate elements on the stack.

The nondeterministic LC parsing algorithm defined above uses one symbol of lookahead in case of terminal left corners. The algorithm is therefore deterministic for the LC(0) grammars, according to the definition of LC($k$) grammars in [Soisalon-Soininen and Ukkonen, 1979]. (This definition is incompatible with that of [Rosenkrantz and Lewis II, 1970].)

The exact formulation of the algorithm above is chosen to simplify the treatment of generalized LC parsing in the next section. The strict separation between goal elements and item elements has also been achieved in [Perlin, 1991], as opposed to [Schabes, 1991].

## 3 Generalizing left-corner parsing

The construction of Lang can be used to form deterministic table-driver parsing algorithms from nondeterministic push-down automata. Because left-corner parsers are also push-down automata, Lang's construction can also be applied to formulate a deterministic parsing algorithm based on LC parsing.

The parsing algorithm we propose in this paper does however not follow straightforwardly from Lang's construction. If we applied the construction directly, then not as much sharing would be provided as we would like. This is caused by the fact that sharing of computation of different search paths is interrupted if different elements occur on top of the stack (or just beneath the top if elements below the top are investigated).

To explain this more carefully we focus on Clause 3 of the nondeterministic LC parser. Assume the following situation. Two different search paths have at the same time the same item element [A → $\alpha$ •] on top of the stack. The goal elements (say B' and B") below that item element are different however in both search paths.

This means that the step which replaces [A → $\alpha$ •] by [C → A • $\beta$], which is done for both search paths (provided both C $\angle^*$ B' and C $\angle^*$ B"), is done separately because B' and B" differ. This is unfortunate

| Step | Parse stack | Input read |
|---|---|---|
| | S | |
| 1 | S [NP → "time" •] | "time" |
| 2 | S [NP → NP • NP] NP | |
| 3 | S [NP → NP • NP] NP [VorN → "flies" •] | "flies" |
| 4 | S [NP → NP • NP] NP [NP → VorN •] | |
| 5 | S [NP → NP NP •] | |
| 6 | S [S → NP • VP] VP | |
| 7 | S [S → NP • VP] VP [VorP → "like" •] | "like" |
| 8 | S [S → NP • VP] VP [VP → VorP • NP] NP | |
| 9 | S [S → NP • VP] VP [VP → VorP • NP] NP [NP → "an" • "arrow"] | "an" |
| 10 | S [S → NP • VP] VP [VP → VorP • NP] NP [NP → "an" "arrow" •] | "arrow" |
| 11 | S [S → NP • VP] VP [VP → VorP NP •] | |
| 12 | S [S → NP VP •] | |
| 13 | | |

Figure 1: One possible sequence of parsing steps while reading "time flies like an arrow"

because sharing of computation in this case is desirable both for efficiency reasons but also because it would simplify the construction of a most-compact parse forest.

Related to the fact that we propose to implement the parse table by means of a graph-structured stack, our solution to this problem lies in the introduction of goal elements consisting of *sets* of nonterminals from *GOAL*, instead of single nonterminals from *GOAL*.

As an example, Figure 2 shows the state of the graph-structured stack for the situation just after reading "time flies". Note that this state represents the states of two different search paths of a nondeterministic LC parser after reading "time flies", one of which is the state after Step 3 in Figure 1.

We see that the goals NP and VP are merged in one goal element so that there is only one edge from the item element labelled with [VorN → "flies" •] to those goals.

Merging goals in one stack element is of course only useful if those goals have at least one left corner in common. For the simplicity of the algorithm, we even allow merging of two goals in one goal element if these goals have anything to do with each other with respect to the left-corner relation $\angle^*$.

Formally, we define an equivalence relation $\sim$ on nonterminals, which is the reflexive, transitive, and symmetric closure of $\angle$. An equivalence class of this relation which includes nonterminal A will be denoted by [A]. Each goal element will now consist of a subset of some equivalence class of $\sim$.

In the running example, the goal elements consist of subsets of {S, NP, VP, PP}, which is the only equivalence class in this example.

Figures 3 and 4 give the complete generalized LC parsing algorithm. At this stage we do not want to complicate the algorithm by allowing epsilon rules in the grammar. Consequently, Clause 1b of the nondeterministic LC parser will have no corresponding piece of code in the GLC parsing algorithm. For the other clauses, we will indicate where they can be retraced in the new algorithm. In Section 4 we explain how our algorithm can be extended so that also grammars with epsilon rules can be handled.

The nodes and arrows in the parse forest are constructed by means of two functions:

MAKE_NODE (X) constructs a node with label X, which is a terminal or nonterminal. It returns (the address of) that node.

A node is associated with a number of lists of *sons*, which are other nodes in the forest. Each list represents an alternative derivation of the nonterminal with which the node is labelled. Initially, a node is associated with an empty collection of lists of sons. ADD_SUBNODE ($m$, $l$) adds a list of sons $l$ to the node $m$.

In the algorithm, an item element $el$ labelled with $[A \rightarrow X_1 \ldots X_m \bullet \alpha]$ is associated with a list of nodes deriving $X_1, \ldots, X_m$. This list is accessed by SONS ($el$). A list consisting of exactly one node $m$ is denoted by $<m>$, and list concatenation is denoted by the operator +.

A goal element $g$ contains for every nonterminal A such that A $\angle^*$ P for some P in $g$ a value NODE ($g$, A), which is the node representing some derivation of A found at the current input position, provided such a derivation exists, and NODE ($g$, A) is NIL otherwise.

In the graph-structured stack there may be an edge from an item element to a unique goal element, and from a goal in a goal element to a number of item elements. For item element $el$, SUCCESSOR ($el$) yields the unique goal element to which there is an edge from $el$. For goal element $g$ and goal P in $g$, SUCCESSORS ($g$, P) yields the zero or more item elements to which there is an edge from P in $g$.

The global variables used by the algorithm are the

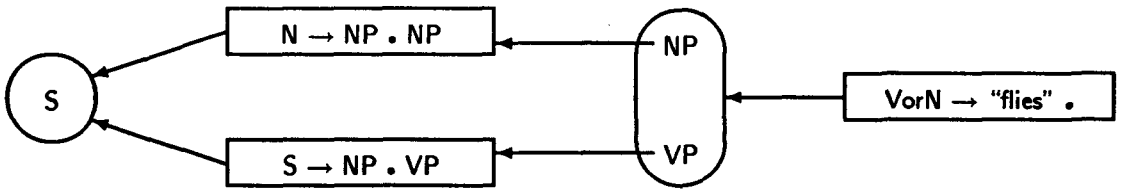


Figure 2: The graph-structured stack after reading "time flies"

following.

$a_0\ a_1 \ldots a_n$ The symbols in the input string.

$i$ The current input position.

$r$ The root of the parse forest. It has the value NIL at the end of the algorithm if no parse has been found.

$\Gamma$ and $\Gamma_{next}$ The sets of goal elements containing goals to be fulfilled from the current and next input position on, respectively.

I and $I_{next}$ The sets of item elements labelled with $[A \rightarrow \alpha \bullet t\ \beta]$ such that a shift may be performed through t at the current and next input position, respectively.

F The set of pairs $(g, A)$ such that a derivation from A has been found for $g$ at the current input position. In other words, F is the set of all pairs $(g, A)$ such that NODE $(g, A) \neq$ NIL.

The graph-structured stack (which is initially empty) and the rules of the grammar are implicit global data structures.

In a straightforward implementation, the relation $\angle^*$ is recorded by means of one large $s' \times s$ boolean matrix, where $s$ is the number of nonterminals in the grammar, and $s'$ is the number of elements in *GOAL*. We can do better however by using the fact that A $\angle^*$ B is never true if A $\not\sim$ B. We propose the storage of $\angle^*$ for every equivalence class of $\sim$ separately, i.e. we store one $t' \times t$ boolean matrix for every class of $\sim$ with $t$ members, $t'$ of which are in *GOAL*.

We furthermore need a list of all rules $A \rightarrow X\ \alpha$ for each terminal and nonterminal X. A small optimization of top-town filtering (see also Section 6) can be achieved by grouping the rules in these lists according to the left sides A.

Note that the storage of the relation $\angle^*$ is the main obstacle to a linear-sized parser.

The time needed to generate a parser is determined by the time needed to compute $\angle^*$ and the classes of $\sim$, which is quadratic in the size of the grammar.

## 4 Adapting the algorithm for arbitrary context-free grammars

The generalized LC parsing algorithm from the previous section is only specified for grammars without epsilon rules. Allowing epsilon rules would not only complicate the algorithm but would for some grammars also introduce the danger of non-termination of the parsing process.

There are two sources of non-termination for nondeterministic LC and LR parsing: cyclicity and hidden left-recursion. A grammar is said to be *cyclic* if there is some derivation of the form $A \rightarrow^+ A$. A grammar is said to be *hidden left-recursive* if $A \rightarrow B\ \alpha$, $B \rightarrow^* \epsilon$, and $\alpha \rightarrow^* A\ \beta$, for some A, B, $\alpha$, and $\beta$. Hidden left recursion is a special case of left recursion where the fact is "hidden" by an empty-generating nonterminal. (A nonterminal is said to be *nonfalse* if it generates the empty string.)

Both sources of non-termination have been studied extensively in [Nederhof and Koster, 1993; Nederhof and Sarbo, 1993].

An obvious way to avoid non-termination for nondeterministic LC parsers in case of hidden left-recursive grammars is the following. We generalize the relation $\angle$ so that B $\angle$ A if and only if there is a rule $A \rightarrow \mu\ B\ \beta$, where $\mu$ is a (possibly empty) sequence of grammar symbols such that $\mu \rightarrow^* \epsilon$. Clause 1b is eliminated and to compensate this, Clauses 1a and 3 are modified so that they take into account prefixes of right sides which generate the empty string:

1a. If the element on top of the stack is the nonterminal A and if the first symbol of the remaining input is t, then we may remove t from the input and push an item $[B \rightarrow \mu\ t \bullet \alpha]$ onto the stack, provided B $\angle^*$ A and $\mu \rightarrow^* \epsilon$.

3. If the top-most two elements on the stack are B $[A \rightarrow \alpha \bullet]$, then we may replace the item by an item of the form $[C \rightarrow \mu\ A \bullet \beta]$, provided C $\angle^*$ B and $\mu \rightarrow^* \epsilon$.

These clauses now allow for nonfalse members at the beginning of right sides. To allow for other nonfalse members we need an extra seventh clause: [4]

7. If the element on top of the stack is the item $[A \rightarrow \alpha \bullet B\ \beta]$, then we may replace this item by the item $[A \rightarrow \alpha\ B \bullet \beta]$, provided $B \rightarrow^* \epsilon$.

The same idea can be used in a straightforward way to make generalized LC parsing suitable for

[4] Actually, an eighth clause is necessary to handle the special case where S, the start symbol, is nonfalse, and the input is empty. We omit this clause for the sake of clarity.

PARSE:

- $r \Leftarrow$ NIL
- Create goal element $g$ consisting of S, the start symbol
- $\Gamma \Leftarrow \{g\}$
- $I \Leftarrow \emptyset$
- $F \Leftarrow \emptyset$
- **for** $i \Leftarrow 0$ **to** $n$ **do** PARSE_WORD
- **return** $r$, as the root of the parse forest

PARSE_WORD:

- $\Gamma_{next} \Leftarrow \emptyset$
- $I_{next} \Leftarrow \emptyset$
- **for all** pairs $(g, A) \in F$ **do**
  - NODE $(g, A) \Leftarrow$ NIL
- $F \Leftarrow \emptyset$
- $t \Leftarrow$ MAKE_NODE $(a_i)$
- FIND_CORNERS $(t)$
- SHIFT $(t)$
- $\Gamma \Leftarrow \Gamma_{next}$
- $I \Leftarrow I_{next}$

FIND_CORNERS $(t)$: /* cf. Clause 1a of the nondeterministic LC parser */

- **for all** goal elements $g$ in $\Gamma$ containing goals in class [B] **do**
  - **for all** rules $A \rightarrow a_i\ \alpha$ such that $A \in$ [B] **do**
    - **if** $A\ \angle^*\ P$ for some goal P in $g$ /* top-down filtering */
      **then**
      - MAKE_ITEM_ELEM ($[A \rightarrow a_i \bullet \alpha]$, $<t>$, g)

SHIFT $(t)$: /* cf. Clause 2 */

- **for all** item elements $el$ in I labelled with $[A \rightarrow \alpha \bullet a_i\ \beta]$ **do**
  - MAKE_ITEM_ELEM ($[A \rightarrow \alpha\ a_i \bullet \beta]$, SONS $(el)$ + $<t>$, SUCCESSOR $(el)$)

MAKE_ITEM_ELEM ($[A \rightarrow \alpha \bullet \beta]$, $l$, $g$):

- Create item element $el$ labelled with $[A \rightarrow \alpha \bullet \beta]$
- SONS $(el) \Leftarrow l$
- Create an edge from $el$ to $g$
- **if** $\beta = \epsilon$
  **then**
  - REDUCE $(el)$

  **elseif** $\beta = t\gamma$, where t is a terminal
  **then**
  - $I_{next} \Leftarrow I_{next} \cup \{el\}$

  **elseif** $\beta = B\gamma$, where B is a nonterminal /* cf. Clause 5 */
  **then**
  - MAKE_GOAL (B, $el$)

MAKE_GOAL (A, $el$):

- **if** there is a goal element $g$ in $\Gamma_{next}$ containing goals in class [A]
  **then**
  - Add goal A to $g$ (provided it is not already there)

  **else**
  - Create goal element $g$ consisting of A
  - Add $g$ to $\Gamma_{next}$
- Create an edge from A in $g$ to $el$

Figure 3: The generalized LC parsing algorithm

REDUCE ($el$):
- Assume the label of $el$ is [A $\rightarrow \alpha$ •]
- Assume SUCCESSOR ($el$) is $g$
- **if** NODE ($g$, A) = NIL
  **then**
  - $m \Leftarrow$ MAKE_NODE (A)
  - NODE ($g$, A) $\Leftarrow m$
  - F $\Leftarrow$ F $\cup$ {($g$, A)}
  - **for all** rules B $\rightarrow$ A $\beta$ **do** /* cf. Clause 3 */
    - **if** B $\angle^*$ P for some goal P in $g$ /* top-down filtering */
      **then**
      - MAKE_ITEM_ELEM ([B $\rightarrow$ A • $\beta$], <$m$>, $g$)
  - **if** A is a goal in $g$
    **then**
    - **if** SUCCESSORS ($g$, A) $\neq \emptyset$
      **then**
      - **for all** $el'$ $\in$ SUCCESSORS ($g$, A) labelled with [B $\rightarrow \beta$ • A $\gamma$] **do** /* cf. Clause 4 */
        - MAKE_ITEM_ELEM ([B $\rightarrow \beta$ A • $\gamma$], SONS ($el'$) + <$m$>, SUCCESSOR ($el'$))

      **elseif** $i = n$ /* cf. Clause 6 */
      **then**
      - $r \Leftarrow m$
- ADD_SUBNODE (NODE ($g$, A), SONS ($el$))

Figure 4: The generalized LC parsing algorithm (continued)

hidden left-recursive grammars, similar to the way this is handled in [Schabes, 1991] and [Leermakers, 1992]. The only technical problem is that, in order to be able to construct a complete parse forest, we need precomputed subforests which derive the empty string in every way from nonfalse nonterminals. This precomputation consists of performing $m_{\mathsf{A}} \Leftarrow$ MAKE_NODE (A) for each nonfalse nonterminal A, (where $m_{\mathsf{A}}$ are specific variables, one for each nonterminal A) and subsequently performing ADD_SUBNODE ($m_{\mathsf{A}}$, $<m_{\mathsf{B}_1}, \ldots, m_{\mathsf{B}_k}>$) for each rule A $\rightarrow$ $\mathsf{B}_1$ ... $\mathsf{B}_k$ consisting only of nonfalse nonterminals. The variables $m_{\mathsf{A}}$ now contain pointers to the required subforests.

GLC parsing is guaranteed to terminate also for cyclic grammars, in which case the infinite amount of parses is reflected by cyclic forests, which are also discussed in [Nozohoor-Farshi, 1991].

## 5 Parsing in cubic time

The size of parse forests, even of those which are optimally dense, can be more than cubic in the length of the input. More precisely, the number of nodes in a parse forest is $\mathcal{O}(n^{p+1})$, where $p$ is the length of the right side of the longest rule.

Using the normal representation of parse forests does therefore not allow cubic parsing algorithms for arbitrary grammars. There is however a kind of shorthand for parse forests which allows a representation which only requires cubic space.

For example, suppose that of some rule A $\rightarrow \alpha \beta$, the prefix $\alpha$ of the right side derives the same part of the input in more than one way, then these derivations may be combined in a new kind of packed node. Instead of the multiple derivations from $\alpha$, this packed node is then combined with the derivations from $\beta$ deriving subsequent input. We call packing of derivations from prefixes of right sides ***subpacking*** to distinguish this from normal packing of derivations from one nonterminal.

Subpacking has been discussed in [Billot and Lang, 1989; Leiss, 1990; Leermakers, 1991]; see also [Sheil, 1976].

Connected with cubic representation of parse forests is cubic parsing. The GLC parsing algorithm in Section 3 has a time complexity of $\mathcal{O}(n^{p+1})$. The algorithm can be easily changed so that, with a little amount of overhead, the time complexity is reduced to $\mathcal{O}(n^3)$, similar to the algorithms in [Perlin, 1991] and [Leermakers, 1992], and the algorithm produces parse forests with subpacking, which require only $\mathcal{O}(n^3)$ space for storage.

We consider how this can be accomplished. First we define the *underlying rule* of an item element labelled with [A $\rightarrow \alpha$ • $\beta$] to be the rule A $\rightarrow \alpha \beta$. Now suppose that two item elements $el_1$ and $el_2$ with the same underlying rule, with the dot at the same position and with the same successor are created at the same input position, then we may perform subpacking for the prefix of the right side before the dot. From then on, we only need one of the item elements $el_1$ and $el_2$ for continuing the parsing process.

Whether two item elements have one and the same goal element as successors cannot be efficiently veri-

fied. Therefore we propose to introduce a new kind of stack element which takes over the role of all former item elements whose successors are one and the same goal element and which have the same underlying rule.

We leave the details to the imagination of the reader.

## 6 Optimization of top-down filtering

One of the most time-costly activities of generalized LC parsing is the check whether for a goal element $g$ and a nonterminal A there is some goal P in g such that A $\angle^*$ P. This check, which is sometimes called *top-down filtering*, occurs in the routines FIND_CORNERS and REDUCE. We propose some optimizations to reduce the number of goals P in $g$ for which A $\angle^*$ P has to be checked.

The most straightforward optimization consists of annotating every edge from an item element labelled with $[A \rightarrow \alpha \bullet \beta]$ to a goal element $g$ with the subset of goals in $g$ which does not include those goals P for which A $\angle^*$ P has already been found to be false. This is the set of goals in $g$ which are actually useful in top-down filtering when a new item element labelled with $[B \rightarrow A \bullet \gamma]$ is created during a REDUCE (see the piece of code in REDUCE corresponding with Clause 3 of the nondeterministic LC parser). The idea is that if A $\angle^*$ P does not hold for goal P in $g$, then neither does B $\angle^*$ P if A $\angle$ B. This optimization can be realized very easily if sets of goals are implemented as lists.

A second optimization is useful if $\angle$ is such that there are many nonterminals A such that there is only one B with A $\angle$ B. In case we have such a nonterminal A which is not a goal, then no top-down filtering needs to be performed when a new item element labelled with $[B \rightarrow A \bullet \alpha]$ is created during a REDUCE. This can be explained by the fact that if for some goal P we have A $\angle^*$ P, and if A $\neq$ P, and if there is only one B such that A $\angle$ B, then we already know that B $\angle^*$ P.

There are many more of these optimizations but not all of these give better performance in all cases. It depends heavily on the properties of $\angle$ whether the gain in time while performing the actual top-down filtering (i.e. performing the tests A $\angle^*$ P for some P in a particular subset of the goals in a goal element $g$) outweighs the time needed to set up extra administration for the purpose of reducing those subsets of the goals.

## 7 Preliminary results

Only recently the author has implemented a GLC parser. The algorithm as presented in this paper has been implemented almost literally, with the treatment of epsilon rules as suggested in Section 4. A small adaptation has been made to deal with terminals of different lengths.

Also recently, some members of our department have completed the implementation of a GLR parser. Because both systems have been implemented using different programming languages, fair comparison of the two systems is difficult. Specific problems which occurred concerning the efficient calculation of LR tables and the correct treatment of epsilon rules for GLR parsing suggest that GLR parsing requires more effort to implement than GLC parsing.

Preliminary tests show that the division of nonterminals into equivalence classes yields disappointing results. In all tested cases, one large class contained most of the nonterminals.

The first optimization discussed in Section 6 proved to be very useful. The number of goals which had to be considered could in some cases be reduced to one fifth.

## Conclusions

We have discussed a parsing algorithm for context-free grammars called *generalized LC parsing*. This parsing algorithm has the following advantages over generalized LR parsing (in order of decreasing importance).

- The size of a parser is much smaller; if we neglect the storage of the relation $\angle^*$, the size is even linear in the size of the grammar. Related to this, only a little amount of time is needed to generate a parser.
- The generated parse forests are as compact as possible.
- Cyclic and hidden left-recursive grammars can be handled more easily and more efficiently (Section 4).
- As Section 5 shows, GLC parsing can more easily be made to run in cubic time for arbitrary context-free grammars. Furthermore, this can be done without much loss of efficiency in practical cases.

Because LR parsing is a more refined form of parsing than LC parsing, generalized LR parsing may at least for some grammars be more efficient than generalized LC parsing.[5] However, we feel that this does not outweigh the disadvantages of the large sizes and generation times of LR parsers in general, which renders GLR parsing unfeasible in some natural language applications.

GLC parsing does not suffer from these defects. We therefore propose this parsing algorithm as a reasonable alternative to GLR parsing. Because of the small generation time of GLC parsers, we expect this kind of parsing to be particularly appropriate during the development of grammars, when grammars

[5]The ratio between the time complexities of GLC parsing and GLR parsing is smaller than some constant, which is dependent on the grammar.

change often and consequently new parsers have to be generated many times.

As we have shown in this paper, the implementation of GLC parsing using a graph-structured stack allows many optimizations. These optimizations would be less straightforward and possibly less effective if a two-dimensional matrix was used for the implementation of the parse table. Furthermore, matrices require a large amount of space, especially for long input, causing overhead for initialization (at least if no optimizations are used).

In contrast, the time and space requirements of GLC parsing using a graph-structured stack are only a negligible quantity above that of nondeterministic LC parsing if no nondeterminism occurs (e.g. if the grammar is LC(0)). Only in the worst-case does a graph-structured stack require the same amount of space as a matrix.

In this paper we have not considered GLC parsing with more lookahead than one symbol for terminal left corners. The reason for this is that we feel that one of the main advantages of our parsing algorithm over GLR parsing is the small sizes of the parsers. Adding more lookahead requires larger tables and may therefore reduce the advantage of generalized LC parsing over its LR counterpart.

On the other hand, the phenomenon reported in [Billot and Lang, 1989] and [Lankhorst, 1991] that the time complexity of GLR parsing sometimes worsens if more lookahead is used, does possibly not apply to GLC parsing. For GLR parsing, more lookahead may result in more LR states, which may result in less sharing of computation. For GLC parsing there is however no relation between the amount of lookahead and the amount of sharing of computation.

Therefore, a judicious use of extra lookahead may on the whole be advantageous to the usefulness of GLC parsing.

## Acknowledgements

The author is greatly indebted to Klaas Sikkel, Janos Sarbo, Franc Grootjen, and Kees Koster, for many fruitful discussions. Valuable correspondence with René Leermakers, Jan Rekers, Masaru Tomita, and Dick Grune is gratefully acknowledged.

## References

[Bear, 1983] J. Bear. A breadth-first parsing model. In *Proc. of the Eighth International Joint Conference on Artificial Intelligence*, volume 2, pages 696–698, Karlsruhe, West Germany, August 1983.

[Billot and Lang, 1989] S. Billot and B. Lang. The structure of shared forests in ambiguous parsing. In *27th Annual Meeting of the ACL* [1], pages 143–151.

[Dekkers *et al.*, 1992] C. Dekkers, M.J. Nederhof, and J.J. Sarbo. Coping with ambiguity in decorated parse forests. In *Coping with Linguistic Ambiguity in Typed Feature Formalisms*, Proceedings of a Workshop held at ECAI 92, pages 11–19, Vienna, Austria, August 1992.

[Demers, 1977] A.J. Demers. Generalized left corner parsing. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages*, pages 170–182, Los Angeles, California, January 1977.

[Earley, 1970] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, February 1970.

[Kipps, 1991] J.R. Kipps. GLR parsing in time $\mathcal{O}(n^3)$. In [Tomita, 1991], chapter 4, pages 43–59.

[Lang, 1974] B. Lang. Deterministic techniques for efficient non-deterministic parsers. In *Automata, Languages and Programming, 2nd Colloquium*, Lecture Notes in Computer Science, volume 14, pages 255–269, Saarbrücken, 1974. Springer-Verlag.

[Lang, 1988a] B. Lang. Complete evaluation of Horn clauses: An automata theoretic approach. Rapport de Recherche 913, Institut National de Recherche en Informatique et en Automatique, Rocquencourt, France, November 1988.

[Lang, 1988b] B. Lang. Datalog automata. In *Proc. of the Third International Conference on Data and Knowledge Bases: Improving Usability and Responsiveness*, pages 389–401, Jerusalem, June 1988.

[Lang, 1988c] B. Lang. Parsing incomplete sentences. In *Proc. of the 12th International Conference on Computational Linguistics*, volume 1, pages 365–371, Budapest, August 1988.

[Lang, 1988d] B. Lang. The systematic construction of Earley parsers: Application to the production of $\mathcal{O}(n^6)$ Earley parsers for tree adjoining grammars. Unpublished paper, December 1988.

[Lang, 1991] B. Lang. Towards a uniform formal framework for parsing. In M. Tomita, editor, *Current Issues in Parsing Technology*, chapter 11, pages 153–171. Kluwer Academic Publishers, 1991.

[Lankhorst, 1991] M. Lankhorst. An empirical comparison of generalized LR tables. In R. Heemels, A. Nijholt, and K. Sikkel, editors, *Tomita's Algorithm: Extensions and Applications*, Proc. of the first Twente Workshop on Language Technology, pages 87–93. University of Twente, September 1991. Memoranda Informatica 91-68.

[Leermakers, 1989] R. Leermakers. How to cover a grammar. In *27th Annual Meeting of the ACL* [1], pages 135–142.

[Leermakers, 1991] R. Leermakers. Non-deterministic recursive ascent parsing. In *Fifth*

*Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference*, pages 63–68, Berlin, Germany, April 1991.

[Leermakers, 1992] R. Leermakers. A recursive ascent Earley parser. *Information Processing Letters*, 41(2):87–91, February 1992.

[Leiss, 1990] H. Leiss. On Kilbury's modification of Earley's algorithm. *ACM Transactions on Programming Languages and Systems*, 12(4):610–640, October 1990.

[Matsumoto and Sugimura, 1987] Y. Matsumoto and R. Sugimura. A parsing system based on logic programming. In *Proc. of the Tenth International Joint Conference on Artificial Intelligence*, volume 2, pages 671–674, Milan, August 1987.

[Nederhof, 1992] M.J. Nederhof. Generalized left-corner parsing. Technical report no. 92–21, University of Nijmegen, Department of Computer Science, August 1992.

[Nederhof and Koster, 1993] M.J. Nederhof and C.H.A. Koster. Top-down parsing of left-recursive grammars. Technical report, University of Nijmegen, Department of Computer Science, 1993. forthcoming.

[Nederhof and Sarbo, 1993] M.J. Nederhof and J.J. Sarbo. Increasing the applicability of LR parsing. Submitted for publication, 1993.

[Nozohoor-Farshi, 1991] R. Nozohoor-Farshi. GLR parsing for $\varepsilon$-grammars. In [Tomita, 1991], chapter 5, pages 61–75.

[Perlin, 1991] M. Perlin. LR recursive transition networks for Earley and Tomita parsing. In *29th Annual Meeting of the ACL* [2], pages 98–105.

[Pratt, 1975] V.R. Pratt. LINGOL - A progress report. In *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, pages 422–428, Tbilisi, Georgia, USSR, September 1975.

[Purdom, 1974] P. Purdom. The size of LALR (1) parsers. *BIT*, 14:326–337, 1974.

[Rekers, 1992] J. Rekers. *Parser Generation for Interactive Environments.* PhD thesis, University of Amsterdam, 1992.

[Rosenkrantz and Lewis II, 1970] D.J. Rosenkrantz and P.M. Lewis II. Deterministic left corner parsing. In *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata Theory*, pages 139–152, 1970.

[Schabes, 1991] Y. Schabes. Polynomial time and space shift-reduce parsing of arbitrary context-free grammars. In *29th Annual Meeting of the ACL* [2], pages 106–113.

[Shann, 1991] P. Shann. Experiments with GLR and chart parsing. In [Tomita, 1991], chapter 2, pages 17–34.

[Sheil, 1976] B.A. Sheil. Observations on context-free parsing. *Statistical Methods in Linguistics*, 1976, pages 71–109.

[Sikkel, 1990] K. Sikkel. Cross-fertilization of Earley and Tomita. Memoranda informatica 90-69, University of Twente, November 1990.

[Sikkel and Op den Akker, 1992] K. Sikkel and R. op den Akker. Head-corner chart parsing. In *Computing Science in the Netherlands*, Utrecht, November 1992.

[Slocum, 1981] J. Slocum. A practical comparison of parsing strategies. In *19th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 1–6, Stanford, California, June–July 1981.

[Soisalon-Soininen and Ukkonen, 1979] E. Soisalon-Soininen and E. Ukkonen. A method for transforming grammars into LL($k$) form. *Acta Informatica*, 12:339–369, 1979.

[Tanaka *et al.*, 1979] H. Tanaka, T. Sato, and F. Motoyoshi. Predictive control parser: Extended LINGOL. In *Proc. of the Sixth International Joint Conference on Artificial Intelligence*, volume 2, pages 868–870, Tokyo, August 1979.

[Tomita, 1986] M. Tomita. *Efficient Parsing for Natural Language.* Kluwer Academic Publishers, 1986.

[Tomita, 1987] M. Tomita. An efficient augmented-context-free parsing algorithm. *Computational Linguistics*, 13:31–46, 1987.

[Tomita, 1988] M. Tomita. Graph-structured stack and natural language parsing. In *26th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 249–257, Buffalo, New York, June 1988.

[Tomita, 1991] M. Tomita, editor. *Generalized LR Parsing.* Kluwer Academic Publishers, 1991.

[Wirén, 1987] Mats Wirén. A comparison of rule-invocation strategies in context-free chart parsing. In *Third Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference*, pages 226–233, Copenhagen, Denmark, April 1987.

[1] *27th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, Vancouver, British Columbia, June 1989.

[2] *29th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, Berkeley, California, June 1991.