# How to cover a grammar

René Leermakers

Philips Research Laboratories, P.O. Box 80.000

5600 JA Eindhoven, The Netherlands

**ABSTRACT**

A novel formalism is presented for Earley-like parsers. It accommodates the simulation of non-deterministic pushdown automata. In particular, the theory is applied to non-deterministic LR-parsers for RTN grammars.

## 1 Introduction

A major problem of computational linguistics is the inefficiency of parsing natural language. The most popular parsing method for context-free natural language grammars, is the general context-free parsing method of Earley [1]. It was noted by Lang [2], that Earley-like methods can be used for simulating a class of non-deterministic pushdown automata(NPDA). Recently, Tomita [3] presented an algorithm that simulates non-deterministic LR-parsers, and claimed it to be a fast algorithm for practical natural language processing systems. The purpose of the present paper is threefold:

1 A novel formalism is presented for Earley-like parsers. A key rôle herein is played by the concept of bilinear grammars. These are defined as context-free grammars, that satisfy the constraint that the right hand side of each grammar rule have at most two non-terminals. The construction of parse matrices for bilinear grammars can be accomplished in cubic time, by an algorithm called C-parser. It includes an elegant way to represent the (possibly infinite) set of parse trees. A case in point is the use of predict functions, which impose restrictions on the parse matrix, if part of it is known. The exact form and effectiveness of predict functions depend on the bilinear grammar at hand. In order to parse a general context-free grammar $G$, a possible strategy is to define a cover for $G$ that satisfies the bilinear grammar constraint, and subsequently parse it with C-parser using appropriate predict functions. The resulting parsers are named Earley-like, and differ only in the precise description for deriving covers, and predict functions.

2 We present the Lang algorithm by giving a bilinear grammar corresponding to an NPDA. Employing the correct predict functions, the parser for this grammar is equivalent to Lang's algorithm, although it works for a slightly different class of NPDA's. We show that simulation of non-deterministic LR-parsers can be performed in our version of the Lang framework. It follows that Earley-like Tomita parsers can handle all context-free grammars, including cyclic ones, although Tomita suggested differently[3].

3 The formalism is illustrated by applying it to Recursive Transition Networks(RTN)[8]. Applying the techniques of deterministic LR-parsing to grammars written as RTN's has been the subject of recent studies [9,10]. Using this research, we show how to construct efficient non-deterministic LR-parsers for RTN's.

## 2 C-Parser

The simplest parser that is applicable to all context-free languages, is the well-known Cocke-Younger-Kasami (CYK) parser. It requires the grammar to be cast in Chomsky normal form. The CYK parser constructs, for the sentence $x_1..x_n$, a parse matrix $T$. To each part $x_{i+1}..x_j$ of the input corresponds the matrix element $T_{ij}$, the value of which is a set of non-terminals from which one can derive $x_{i+1}..x_j$. The algorithm can easily be generalized to work for any grammar, but its complexity then increases with the number of non-terminals at the right hand side of grammar rules. Bilinear grammars have the lowest complexity, disregarding linear grammars which do not have the generative power of general context-free grammars. Below we list the recursion relation $T$ must satisfy for general bilinear grammars. We write the grammar as as a four-tuple $(N, \Sigma, P, S)$, where $N$ is the set of non-terminals, $\Sigma$ the set of terminals, $P$ the set of production rules, and $S \in N$ the start symbol. We use variables $I, J, K, L \in N$, $\beta_1, \beta_2, \beta_3 \in \Sigma^*$, and $i, j, k_1..k_4$ as indices of the matrix $T$[1].

$$\begin{aligned}
I \in T_{ij} \equiv\ & \exists_{J,K \in N, i \leq k_1 \leq k_2 \leq k_3 \leq k_4 \leq j}(J \in T_{k_1k_2} \wedge \\
& \quad K \in T_{k_3k_4} \wedge I \rightarrow \beta_1 J \beta_2 K \beta_3 \wedge \beta_1 = x_{i+1}..x_{k_1} \\
& \quad \wedge \beta_2 = x_{k_2+1}..x_{k_3} \wedge \beta_3 = x_{k_4+1}..x_j) \\
& \vee \exists_{J \in N, i \leq k_1 \leq k_2 \leq j}(J \in T_{k_1k_2} \wedge I \rightarrow \beta_1 J \beta_2 \\
& \quad \wedge \beta_1 = x_{i+1}..x_{k_1} \wedge \beta_2 = x_{k_2}..x_j) \\
& \vee (I \rightarrow \beta_1 \wedge \beta_1 = x_{i+1}..x_j)
\end{aligned}$$

The relation can be solved for the diagonal elements $T_{ii}$, independently of the input sentence. They are equal to the set of non-terminals that derive $\epsilon$ in one or more

[1]Throughout the paper we identify a grammar rule $I \rightarrow \alpha$ with the boolean expression '$I$ directly derives $\alpha$'.

steps. Algorithms that construct $T$ for given input, will be referred to as C-parsers. The time needed for constructing $T$ is at most a cubic function of the input length $n$, while it takes an amount of space that is a quadratic function of $n$. The sentence is successfully parsed, if $S \in T_{0n}$. From $T$, one can simply deduce an output grammar $O$, which represents the set of parse trees. Its non-terminals are triples $< I, i, j >$, where $I$ is a non-terminal of the original bilinear grammar, and $i, j$ are integers between 0 and $n$.

$$\begin{aligned}
&< I,i,j > \rightarrow \beta_1 < J,k_1,k_2 > \beta_2 < K,k_3,k_4 > \beta_3 \equiv \\
&\quad I \in T_{ij} \wedge I \rightarrow \beta_1 J \beta_2 K \beta_3 \wedge J \in T_{k_1k_2} \wedge K \in T_{k_3k_4} \\
&\quad \wedge \beta_1 = x_{i+1}..x_{k_1} \wedge \beta_2 = x_{k_2+1}..x_{k_3} \wedge \beta_3 = x_{k_4+1}..x_j \\
&< I,i,j > \rightarrow \beta_1 < J,k_1,k_2 > \beta_2 \equiv I \in T_{ij} \wedge I \rightarrow \beta_1 J \beta_2 \\
&\quad \wedge J \in T_{k_1k_2} \wedge \beta_1 = x_{i+1}..x_{k_1} \wedge \beta_2 = x_{k_2+1}..x_j \\
&< I,i,j > \rightarrow \beta_1 \equiv I \in T_{ij} \wedge I \rightarrow \beta_1 \wedge \beta_1 = x_{i+1}..x_j
\end{aligned}$$

The grammar rules of $O$ are such that they generate only the sentence that was parsed. The parse trees according to the output grammar are isomorphic to the parse trees generated by the original grammar. The latter parse trees can be obtained from the former by replacing the triple non-terminals by their first element.

Matrix elements of $T$ are such that their members cover part of the input. This does not imply that all members are useful for constructing a possible parse of the input as a whole. In fact, many are useless for this purpose. Depending on the grammar, knowledge of part of $T$ may give restrictions on the possibly useful contents of the rest of $T$. Making use of these restrictions, one may get more efficient parsers, with the same functionality. As an example, one has the generalized Earley prediction. It involves functions $predict_k : 2^N \rightarrow 2^N$($N$ is the set of non-terminals), such that one can prove that the useful contents of the $T_{jk}$ are contained in the elements of a matrix $\theta$ related to $T$ by

$$\begin{aligned}
&\theta_{00} = \theta^c, \\
&\theta_{ij} = predict_{j-i}(\textstyle\bigcup_{k=0}^{i} \theta_{ki}) \cap T_{ij}, \text{ if } j > 0,
\end{aligned}$$

where $\theta^c$, called the initial prediction, is some constant set of non-terminals that derive $\epsilon$. It follows that $T_{ij}$ can be calculated from the matrix elements $\theta_{kl}$ with $i \leq k, l \leq j$, i.e. the occurrences of $T$ at the right hand side of the recurrence relation may be replaced by $\theta$. Hence $\theta_{ij}$, $j > 0$, can be calculated from the matrix elements $\theta_{kl}$, with $l \leq j$:

$$\begin{aligned}
&\theta_{ij} = predict_{j-i}(\textstyle\bigcup_k \theta_{ki}) \cap \\
&\quad \{I | \exists_{J,K \in N, i \leq k_1 \leq k_2 \leq k_3 \leq k_4 \leq j}(J \in \theta_{k_1k_2} \wedge \\
&\qquad K \in \theta_{k_3k_4} \wedge I \rightarrow \beta_1 J \beta_2 K \beta_3 \wedge \beta_1 = x_{i+1}..x_{k_1} \\
&\qquad \wedge \beta_2 = x_{k_2+1}..x_{k_3} \wedge \beta_3 = x_{k_4+1}..x_j) \\
&\quad \vee \exists_{J \in N, i \leq k_1 \leq k_2 \leq j}(J \in \theta_{k_1k_2} \wedge I \rightarrow \beta_1 J \beta_2 \\
&\qquad \wedge \beta_1 = x_{i+1}..x_{k_1} \wedge \beta_2 = x_{k_2}..x_j) \\
&\quad \vee (I \rightarrow \beta_1 \wedge \beta_1 = x_{i+1}..x_j)\}
\end{aligned}$$

The algorithm that creates the matrix $\theta$ in this way, scanning the input from left to right, is called a restricted C-parser. The above relation does not determine the diagonal elements of $\theta$ uniquely, and a restricted C-parser is to find the smallest solution. Concerning the gain of efficiency, it should be noted that this is very grammar-dependent. For some grammars, restriction of the parser reduces its complexity, while for others predict functions may even be counter-productive [4].

## 3 Bilinear covers

A grammar $G$ is said to be covered by a grammar $C(G)$, if the language generated by both grammars is identical, and if for each sentence the set of parse trees generated by $G$ can be recovered from the set of parse trees generated by $C(G)$. The grammar $C(G)$ is called a cover for $G$, and we will be interested in covers that are bilinear, and can thus be parsed by C-parser. It is rather surprising that at the heart of most parsing algorithms for context-free languages lies a method for deriving a bilinear cover.

### 3.1 Earley's method

Earley's construction of items is a clear example of a construction of a bilinear cover $C_E(G)$ for each context-free grammar $G$. The terminals of $C_E(G)$ and $G$ are identical, the non-terminals of $C_E(G)$ are the items (dotted rules[1]) $I_i^k$, defined as follows. Let the non-terminal defined by rule $i$ of grammar $G$ be given by $N_i$, then $I_i^k$ is $N_i \rightarrow \alpha \cdot \beta$, with $|\beta| + 1 = k$ ($\alpha, \beta$ are used for sequences of terminals and non-terminals). We assume that only one rule, rule 0, of $G$ rewrites the start symbol $S$. The length of the right-hand side of rule $i$ is given by $M_i - 1$. The rules of $C_E(G)$ are derived as follows.

- Let $I_i^k$ be an item of the form $A \rightarrow \alpha \cdot B\beta$, and hence $I_i^{k-1}$ be $A \rightarrow \alpha B \cdot \beta$. Then if B is a terminal, $I_i^{k-1} \rightarrow I_i^k B$, and if B is non-terminal then $I_i^{k-1} \rightarrow I_i^k I_j^0$, for all $j$ such that $N_j = B$.
- Initial items of the form $N_i \rightarrow \cdot\alpha$ rewrite to $\epsilon$: $I_i^{M_i} \rightarrow \epsilon$.
- For each $i$ one has the final rule $I_i^0 \rightarrow I_i^1$.

In [4] a similar construction was given, leading to a grammar in canonical two-form for each context-free grammar. Among other things it differs from the above in the appearance of the final rules, which are indeed superfluous. We have introduced them to make the extension to RTN's, in section 4, more immediate.

The description just given, yields a set of production rules consisting of sections $P_i$, that have the following structure:

$$P_i = \textstyle\bigcup_{j=2}^{M_i} \{I_i^{j-1} \rightarrow I_i^j x_i^j\} \cup \{I_i^{M_i} \rightarrow \epsilon\} \cup \{I_i^0 \rightarrow I_i^1\},$$

where $x_i^j \in \bigcup_j \{I_j^0\} \cup \Sigma$. Note that the start symbol of the cover is $I_0^0$. The construction of parse matrices $T$ by C-parser yields the Earley algorithm, without its prediction part. By restricting the parser by the $predict_0$ function satisfying

$$I_j^{M_j} \in predict_0(L) \equiv \exists_{i,k}((I_i^{k-1} \rightarrow I_i^k I_j^0) \wedge I_i^k \in L),$$

the initial prediction $\theta^c$ being the smallest solution of

$$\theta^c = predict_0(\theta^c) \cup \{I_0^{M_0}\},$$

one obtains a conventional Earley parser ($predict_k = \bigcup_{ij}\{I_i^j\}$ for $k > 0$). The cover is such that usually the predict action speeds up the parser considerably.

There are many ways to define covers with dotted rules as non-terminals. For example, from recent work by Kruseman Aretz [6], we learn a prescription for a bilinear cover for $G$, which is smaller in size compared to $C_E(G)$, at the cost of rules with longer right hand sides. The prescription is as follows ($\alpha$, $\beta$, $\gamma$, $\kappa$ are sequences of terminals and non-terminals, $\delta$ stands for sequences of terminals only, and $A$, $B$, $C$ are non-terminals):

- Let $I$ be an item of the form $A \rightarrow \alpha \cdot B\kappa$, and $K$ is an item $B \rightarrow \gamma\cdot$, then $J \rightarrow IK\delta$, where either

  $J$ is item $A \rightarrow \alpha B\delta \cdot C\beta$ and $\kappa = \delta C\beta$, or

  $J$ is item $A \rightarrow \alpha B\delta\cdot$ and $\kappa = \delta$.

- Let $I$ be an item of the form $A \rightarrow \delta \cdot B\alpha$ or $A \rightarrow \delta\cdot$, then $I \rightarrow \delta$.

## 3.2 Lang grammar

In a similar fashion the items used by Lang [2] in his algorithm for non-deterministic pushdown automata (NPDA) may be interpreted as non-terminals of a bilinear grammar, which we will call the Lang grammar. We adopt restrictions on NPDA's similarly to [2], the main one being that one or two symbols be pushed on the stack in a single move, and each stack symbol is removed when it is read. If two symbols are pushed on the stack, the bottom one must be identical to the symbol that is removed in the same transition. Formally we write an NPDA as a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, \zeta_0, F)$, where $Q$ is the set of state symbols, $\Sigma$ the input alphabet, $\Gamma$ the pushdown symbols, $\delta : Q \times (\Gamma \cup \{\epsilon\}) \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^{Q\times(\{\epsilon\}\cup\Gamma\cup(\Gamma\times\Gamma))}$ the transition function, $q_0 \in Q$ the initial state, $\zeta_0 \in \Gamma$ the start symbol, and $F \subseteq Q$ is the set of final states. If the automaton is in state p, and $\alpha$ is the top of the stack, and the current symbol on the input tape is $y$, then it may make the following eight types of moves:

if $(r, \epsilon) \in \delta(p, \epsilon, \epsilon)$: go to state $r$

if $(r, \epsilon) \in \delta(p, \alpha, \epsilon)$: pop $\alpha$, go to state $r$

if $(r, \gamma) \in \delta(p, \alpha, \epsilon)$: pop $\alpha$, push $\gamma$, go to state $r$

if $(r, \epsilon) \in \delta(p, \epsilon, y)$: shift input tape, go to state $r$

if $(r, \gamma) \in \delta(p, \epsilon, y)$: push $\gamma$, shift tape, go to $r$

if $(r, \epsilon) \in \delta(p, \alpha, y)$: pop $\alpha$, shift tape, go to $r$

if $(r, \gamma) \in \delta(p, \alpha, y)$: pop $\alpha$, push $\gamma$, shift tape, go to $r$

if $(r, \gamma\alpha) \in \delta(p, \alpha, y)$: push $\gamma$, shift tape, go to $r$

We do not allow transitions such that $(r, \gamma) \in \delta(p, \epsilon, \epsilon)$, or $(r, \gamma\alpha) \in \delta(p, \alpha, \epsilon)$, and assume that the initial state can not be reached from other states.

The non-terminals of the Lang grammar are the start symbol $S$ and four-tuple entities (Lang's 'items') of the form $< q, \alpha, p, \beta >$, where $p$ and $q$ are states, and $\alpha$ and $\beta$ stack symbols. The idea is that *iff* there exists a computation that consumes input symbols $x_i..x_j$, starting at state $p$ with a stack $\beta\zeta_0$ (the leftmost symbol is the top), and ending in state $q$ with stack $\alpha\beta\zeta_0$, and if the stack $\beta\zeta_0$ does not re-occur in intermediate configurations, *then* $< q, \alpha, p, \beta > \rightarrow^* x_i..x_j$. The rewrite rules of the Lang grammar are defined as follows (universal quantification over $p, q, r, s \in Q; \alpha, \beta, \gamma \in \Gamma; x \in \Sigma \cup \epsilon$, $y \in \Sigma$ is understood):

$S \rightarrow < p, \alpha, q_0, \zeta_0 > \equiv p \in F$ (final rules)

$< r, \beta, s, \gamma > \rightarrow < q, \beta, s, \gamma >< p, \alpha, q, \beta > x \equiv$

$\quad (r, \epsilon) \in \delta(p, \alpha, x)$

$< r, \gamma, q, \beta > \rightarrow < p, \alpha, q, \beta > x \equiv$

$\quad ((r, \gamma) \in \delta(p, \alpha, x)) \vee ((r, \epsilon) \in \delta(p, \epsilon, x) \wedge (\alpha = \gamma))$

$< r, \gamma, p, \alpha > \rightarrow y \equiv$

$\quad ((r, \gamma) \in \delta(p, \epsilon, y)) \vee ((r, \gamma\alpha) \in \delta(p, \alpha, y))$

$< q_0, \zeta_0, q_0, \zeta_0 > \rightarrow \epsilon$ (initial rule)

From each NPDA one may deduce context-free grammars that generate the same language [5]. The above construction yields such a grammar in bilinear form. It only works for automata, that have transitions like we use above. Lang grammars are rather big, in the rough form given above. Many of the non-terminals do not occur, however, in the derivation of any sentence. They can be removed by a standard procedure [5]. In addition, during parsing, predict functions can be used to limit the number of possible contents of parse matrix elements. The following initial prediction and predict functions render the restricted C-parser functionally equivalent to Lang's original algorithm, albeit that Lang considered a class of NPDA's which is slightly different from the class we alluded to above:

$\theta^c = \{< q_0, \zeta_0, q_0, \zeta_0 >\}$

$predict_k(L) = \emptyset$ if $k = 0$ else

$predict_k(L) = \{< s, \beta, q, \alpha > | \exists_{r,\gamma} < q, \alpha, r, \gamma > \in L\}$
$\qquad \cup \{S | k = n\}$ ($n$ is sentence length).

The Tomita parser [3] simulates an NPDA, constructed from a context-free grammar via LR-parsing tables. Within our formalism we can implement this idea, and arrive at an Earley-like version of the Tomita parser, which is able to handle general context-free grammars, including cyclic ones.

# 4 Extension to RTN's

In the preceding section we discussed various ways of deriving bilinear covers. Reversely, one may try to discover what kinds of grammars are covered by certain bilinear grammars.

A bilinear grammar $C_E(G)$, generated from a context-free grammar by the Earley prescription, has peculiar properties. In general, the sections $P_i$ defined above constitute regular subgrammars, with the $x_i^j$ as terminals. Alternatively, $P_i$ may be seen as a finite state automaton with states $I_i^j$. Each rule $I_i^{j-1} \rightarrow I_i^j x_i^j$ corresponds to a transition from $I_i^j$ to $I_i^{j-1}$ labeled by $x_i^j$. This correspondence between regular grammars and finite state automata is in fact a special instance of the correspondence between Lang bilinear grammars and NPDA's.

The $P_i$ of the above kind are very restricted finite state automata, generating only one string. It is a natural step to remove this restriction and study covers that are the union of general regular subgrammars. Such a grammar will cover a grammar, consisting of rules of the form $N_i \rightarrow \alpha$, where $\alpha$ is a regular expression of terminals and non-terminals. Such grammars go under the names of RTN grammars [8], or extended context-free grammars [9], or regular right part grammars [10]. Without loss of generality we may restrict the format of the finite state automata, and stipulate that it have one initial state $I_i^{M_i}$ and one final state $I_i^0$, and only the following type of rules:

- final rules $I_i^0 \rightarrow I_i^j$
- rules $I_i^j \rightarrow I_i^k x$, where $x \in \bigcup_m \{I_m^0\} \cup \Sigma$, $k <> 0$ and $j <> M_i$.
- the initial rule $I_i^{M_i} \rightarrow \epsilon$.

For future reference we define define the set $I$ of non-terminals as $I = \bigcup_{ij} \{I_i^j\}$, and its subset $I^0 = \bigcup_i \{I_i^0\}$.

A covering prescription that turns an RTN into a set of such subgrammars, reduces to $C_E$ if applied to normal context-free grammars, and will be referred to by the same name, although in general the above format does not determine the cover uniquely. For some example definitions of items for RTN's (i.e. the $I_i^k$), see [1,9].

# 5 The CNLR Cover

A different cover for RTN grammars may be derived from the one discussed in the previous section. So our starting point is that we have a bilinear grammar $C_E(G)$, consisting of regular subgrammars. We (approximately) follow the idea of Tomita, and construct an NPDA from an $LR(0)$-automaton, whose states are sets of items. In our case, the items are the non-terminals of $C_E(G)$. The full specification of the automaton is extracted from [9] in a straightforward way. Subsequently, the general prescription of chapter 3 yields a bilinear grammar. In this way we arrive at what we would like to call the canonical non-deterministic LR-parser (CNLR parser, for short).

## 5.1 LR(0) states

In order to derive the set $Q$ of LR(0) states, which are subsets of $I$, we first need a few definitions. Let $s$ be an element of $2^I$, then $closure(s)$ is the smallest element of $2^I$, such that

$$s \subset closure(s) \wedge ((I_i^j \in closure(s) \wedge (I_i^k \rightarrow I_i^j I_m^0)) \Rightarrow I_m^{M_m} \in closure(s))$$

Similarly, the sets $goto_1(s,x)$, and $goto_2(s,x)$, where $x \in I^0 \cup \Sigma$, are defined as

$$goto_1(s,x) = closure(\{I_i^k | I_i^j \in s \wedge (I_i^k \rightarrow I_i^j x) \wedge j <> M_i\})$$
$$goto_2(s,x) = closure(\{I_i^k | I_i^{M_i} \in s \wedge (I_i^k \rightarrow I_i^{M_i} x)\})$$

The set $Q$ then is the smallest one that satisfies

$$closure(\{I_0^{M_0}\}) \in Q \wedge (s \in Q \Rightarrow (goto_1(s,x) = \emptyset \vee goto_1(s,x) \in Q) \wedge (goto_2(s,x) = \emptyset \vee goto_2(s,x) \in Q))$$

The automaton we look for can be constructed in terms of the LR(0) states. In addition to the *goto* functions, we will need the predicate *reduce*, defined by

$$reduce(s, I_i^0) \equiv \exists_k((I_i^0 \rightarrow I_i^k) \wedge I_i^k \in s).$$

A point of interest is the possible existence of stacking conflicts[9]. These arise if for some $s, x$ both $goto_1(s,x)$ and $goto_2(s,x)$ are not empty. Stacking conflicts cause an increase of non-determinism that can always be avoided by removing the conflicts. One method for doing this has been detailed in [9], and consists of the splitting in parts of the right hand side of grammar rules that cause conflicts. Here we need not and will not assume anything about the occurrence of stacking conflicts.

Grammars, of which Earley covers do not give rise to stacking conflicts, form a proper subset of the set of extended context-free grammars. It could very well be that natural language grammars, written as RTN's in order to produce 'natural' syntax trees, generally belong to this subset. For an example, see section 6.

## 5.2 The automaton

To determine the automaton we specify, in addition to the set of states $Q$, the set of stack symbols $\Gamma = Q \cup I^0 \cup \{\zeta_0\}$, the initial state $q_0 = closure(\{I_0^{M_0}\})$, the final states $F = \{s | reduce(s, I_0^0)\}$, and the transition function $\delta$:

$$\delta(s,\gamma,y) = \{(t, q\gamma) | \gamma \notin I^0 \wedge ((t = goto_2(s,y) \wedge q = s) \vee (t = goto_1(s,y) \wedge q = \epsilon))\}$$
$$\delta(s,\gamma,\epsilon) = \{(t,q) | \gamma \in I^0 \wedge ((t = goto_1(s,\gamma) \wedge q = \epsilon) \vee ((t = goto_2(s,\gamma) \wedge q = s))\} \cup \{(\gamma, I_i^0) | \gamma \in Q \wedge reduce(s, I_i^0)\}$$

## 5.3 The grammar

From the automaton, which is of the type discussed in section 3.2, we deduce the bilinear grammar

$$S \rightarrow < s, \alpha, q_0, \zeta_0 > \equiv reduce(s, I_0^0)$$
$$< t, r, q, \beta > \rightarrow < s, r, q, \beta > y \equiv t = goto_1(s,y)$$
$$< t, s, s, r > \rightarrow y \equiv t = goto_2(s,y)$$
$$< t, \beta, p, \alpha > \rightarrow < q, \beta, p, \alpha > < s, I_i^0, q, \beta > \equiv t = goto_1(s, I_i^0)$$
$$< t, s, q, \beta > \rightarrow < s, I_i^0, q, \beta > \equiv t = goto_2(s, I_i^0)$$
$$< p, I_i^0, q, \beta > \rightarrow < s, p, q, \beta > \equiv reduce(s, I_i^0)$$
$$< q_0, \zeta_0, q_0, \zeta_0 > \rightarrow \epsilon,$$

where $s,t,q,p \in Q$, $r \in Q \cup \{\zeta_0\}$, $\alpha, \beta \in \Gamma$, $y \in \Sigma$. As was mentioned in section 3.2, this grammar can be reduced by a standard algorithm to contain only useful non-terminals.

### 5.3.1 A reduced form

If the reduction algorithm of [5] is performed, it turns out that the structure of the above grammar is such that useful non-terminals $< p, \alpha, q, \beta >$ satisfy

$\alpha \in Q \Rightarrow \alpha = q$

$\alpha \notin Q \Rightarrow p = q$

Furthermore, two non-terminals that differ only in their fourth tuple-element always derive the same strings of terminals. Hence, the fourth element can safely be discarded, as can the second if it is in $Q$ and the first if the second is not in $Q$. The non-terminals then become pairs $< \alpha, s >$, with $\alpha \in \Gamma$ and $s \in Q$. For such non-terminals, the predict functions, mentioned in section 2, must be changed:

$\theta^c = \{< q_0, q_0 >\}$

$predict_k(L) = \emptyset$ if $k = 0$ else

$predict_k(L) = \{< \alpha, s > | \exists_q < s, q > \in L\} \cup \{S | k = n\}$

The grammar gets the general form

$S \rightarrow < s, q_0 > \equiv reduce(s, I_0^0)$

$< t, q > \rightarrow < s, q > y \equiv t = goto_1(s, y)$

$< t, s > \rightarrow y \equiv t = goto_2(s, y)$

$< t, q > \rightarrow < s, q > < I_i^0, s > \equiv t = goto_1(s, I_i^0)$

$< t, s > \rightarrow < I_i^0, s > \equiv t = goto_2(s, I_i^0)$

$< I_i^0, q > \rightarrow < s, q > \equiv reduce(s, I_i^0)$

Note that the terminal $< q_0, q_0 >$ does not appear in this grammar, but will appear in the parse matrix because of the initial prediction $\theta^c$. Of course, when the automaton is fully specified for a particular language, the corresponding CNLR grammar can be reduced still further, see section 6.4.

### 5.3.2 Final form

Even the grammar in reduced form contains many non-terminals that derive the same set of strings. In particular, all non-terminals that only differ in their second component generate the same language. Thus, the second component only encodes information for the predict functions. The redundancy can be removed by the following means. Define the function $\sigma : \Gamma \rightarrow 2^Q$, such that

$\sigma(\alpha) = \{s | < \alpha, s >$ is a useful non-terminal of the above grammar$\}$.

Then we may simply parse with the 'bare' grammar, the non-terminals of which are the automaton stack symbols $\Gamma$:

$S \rightarrow s \equiv reduce(s, I_0^0)$

$t \rightarrow sy \equiv t = goto_1(s, y)$

$t \rightarrow y \equiv \exists_s(t = goto_2(s, y))$

$t \rightarrow sI_i^0 \equiv t = goto_1(s, I_i^0)$

$t \rightarrow I_i^0 \equiv \exists_s(t = goto_2(s, I_i^0))$

$I_i^0 \rightarrow s \equiv reduce(s, I_i^0)$,

using the predict functions

$\theta^c = \{q_0\}$

$predict_k(L) = \emptyset$ if $k = 0$ else

$predict_k(L) = \{\alpha | \exists_s (s \in L \wedge s \in \sigma(\alpha))\} \cup \{S | k = n\}$.

The function $\sigma$ can also be deduced directly from the bare grammar, see section 7.

## 5.4 Parse trees

Each parse tree $\tau$ according to the original grammar can be obtained from a corresponding parse tree $t$ according to the cover. Each subset of the set of nodes of $t$ is partially ordered by the relation 'is descendant of'. Now consider the set of nodes of $t$ that correspond to non-terminals $I_i^0$. The 'is descendant of' ordering defines a projected tree that contains, apart from the terminals, only these nodes. The desired parse tree $\tau$ is now obtained by replacing in the projected tree, each node $I_i^0$ by a node labeled by $N_i$, the left hand side of grammar rule $i$ of the original grammar.

## 6 Example

The foregoing was rather technical and we will try to repair this by showing, very explicitly, how the formalism works for a small example grammar. In particular, we will for a small RTN grammar, derive the Earley cover of section 4, and the two covers of sections 5.3.1 and 5.3.2.

### 6.1 The grammar

The following is a simple grammar for finite subordinate clauses in Dutch.

$S \rightarrow conj\ NP\ VP$

$VP \rightarrow [NP]\ \{PP\}\ verb\ [S]$

$PP \rightarrow prep\ NP$

$NP \rightarrow det\ noun\ \{PP\}$

So we have four regular expressions defining $N_0 = S$, $N_1 = VP$, $N_2 = PP$, $N_3 = NP$.

### 6.2 The Earley cover

The above grammar is covered by four regular subgrammars:

$I_0^0 \rightarrow I_0^1; I_0^1 \rightarrow I_0^2 I_1^0; I_0^2 \rightarrow I_0^3 I_3^0; I_0^3 \rightarrow I_0^4 conj; I_0^4 \rightarrow \epsilon$

$I_1^0 \rightarrow I_1^1; I_1^0 \rightarrow I_1^2; I_1^1 \rightarrow I_1^2 I_0^0; I_1^2 \rightarrow I_1^3 verb; I_1^3 \rightarrow I_1^4 I_2^0; I_1^3 \rightarrow I_1^3 I_2^0; I_1^2 \rightarrow I_1^4 verb; I_1^4 \rightarrow I_1^5 I_3^0; I_1^3 \rightarrow I_1^5 I_2^0; I_1^2 \rightarrow I_1^5 verb; I_1^5 \rightarrow \epsilon$

$I_2^0 \rightarrow I_2^1; I_2^1 \rightarrow I_2^2 I_3^0; I_2^2 \rightarrow I_2^3 prep; I_2^3 \rightarrow \epsilon$

$I_3^0 \rightarrow I_3^1; I_3^0 \rightarrow I_3^2; I_3^1 \rightarrow I_3^1 I_2^0; I_3^1 \rightarrow I_3^2 I_2^0; I_3^2 \rightarrow I_3^3 noun; I_3^3 \rightarrow I_3^4 det; I_3^4 \rightarrow \epsilon$

Note that the $M_i$ in this case turn out as $M_0 = 4$, $M_1 = 5$, $M_2 = 3$, $M_3 = 4$.

## 6.3 The automaton

The construction of section 5.1 yields the following set of states:

$q_0 = \{I_0^4\}; q_1 = \{I_0^3, I_3^4\}; q_2 = \{I_0^2, I_1^5, I_2^3, I_3^4\};$

$q_3 = \{I_3^3\}; q_4 = \{I_0^1\}; q_5 = \{I_1^3, I_2^3\}; q_6 = \{I_1^4, I_2^3\};$

$q_7 = \{I_0^4, I_1^2\}; q_8 = \{I_2^2, I_3^4\}; q_9 = \{I_2^3, I_3^2\};$

$q_{10} = \{I_1^1\}; q_{11} = \{I_2^1\}; q_{12} = \{I_2^3, I_3^1\}$

The transitions are grouped into two parts. First we list the function $goto_2$:

$goto_2(q_0, conj) = q_1; goto_2(q_1, det) = q_3;$

$goto_2(q_2, I_2^0) = q_5; goto_2(q_2, I_3^0) = q_6;$

$goto_2(q_2, verb) = q_7; goto_2(q_2, prep) = q_8;$

$goto_2(q_2, det) = q_3; goto_2(q_5, prep) = q_8;$

$goto_2(q_6, prep) = q_8; goto_2(q_7, conj) = q_1;$

$goto_2(q_8, det) = q_3; goto_2(q_9, prep) = q_8;$

$goto_2(q_{12}, prep) = q_8$

Likewise, we have the $goto_1$ function, which gives the non-stacking transitions for our grammar:

$goto_1(q_1, I_3^0) = q_2; goto_1(q_2, I_1^0) = q_4;$

$goto_1(q_3, noun) = q_9; goto_1(q_5, I_2^0) = q_5;$

$goto_1(q_5, verb) = q_7; goto_1(q_6, I_2^0) = q_5;$

$goto_1(q_7, I_0^0) = q_{10}; goto_1(q_8, I_3^0) = q_{11};$

$goto_1(q_9, I_2^0) = q_{12}; goto_1(q_{12}, I_2^0) = q_{12}$

The predicate $reduce$ holds for six pairs of states and non-terminals:

$reduce(q_4, I_0^0); reduce(q_{10}, I_1^0); reduce(q_7, I_1^0);$

$reduce(q_{11}, I_2^0); reduce(q_9, I_3^0); reduce(q_{12}, I_3^0)$

## 6.4 CNLR parser

Given the automaton, the CNLR grammar follows according to section 5.3. After removal of the useless non-terminals we arrive at the following grammar, which is of the format of section 5.3.1.

$S \rightarrow < q_4, q_0 >$

$< q_9, q > \rightarrow < q_3, q > noun$, where $q \in [q_1, q_2, q_8]$

$< q_7, q_2 > \rightarrow < q_5, q_2 > verb$

$< q_1, q > \rightarrow conj$, where $q \in [q_0, q_7]$

$< q_3, q > \rightarrow det$, where $q \in [q_1, q_2, q_8]$

$< q_7, q_2 > \rightarrow verb$

$< q_8, q > \rightarrow prep$, where $q \in [q_2, q_5, q_6, q_9, q_{12}]$

$< q_2, q > \rightarrow < q_1, q > < I_3^0, q_1 >$, where $q \in [q_0, q_7]$

$< q_4, q > \rightarrow < q_2, q > < I_1^0, q_2 >$, where $q \in [q_0, q_7]$

$< q_5, q_2 > \rightarrow < q_5, q_2 >< I_2^0, q_5 >$

$< q_5, q_2 > \rightarrow < q_6, q_2 >< I_2^0, q_6 >$

$< q_{10}, q_2 > \rightarrow < q_7, q_2 >< I_0^0, q_7 >$

$< q_{11}, q > \rightarrow < q_8, q > < I_3^0, q_8 >$, where $q \in [q_2, q_5, q_6, q_9, q_{12}]$

$< q_{12}, q > \rightarrow < q_9, q > < I_2^0, q_9 >$, where $q \in [q_1, q_2, q_8]$

$< q_{12}, q > \rightarrow < q_{12}, q > < I_2^0, q_{12} >$, where $q \in [q_1, q_2, q_8]$

$< q_5, q_2 > \rightarrow < I_2^0, q_2 >, < q_6, q_2 > \rightarrow < I_3^0, q_2 >$

$< I_0^0, q_7 > \rightarrow < q_4, q_7 >, < I_1^0, q_2 > \rightarrow < q_{10}, q_2 >$

$< I_1^0, q_2 > \rightarrow < q_7, q_2 >$

$< I_2^0, q > \rightarrow < q_{11}, q >$, where $q \in [q_2, q_5, q_6, q_9, q_{12}]$

$< I_3^0, q > \rightarrow < q_9, q >$, where $q \in [q_1, q_2, q_8]$

$< I_3^0, q > \rightarrow < q_{12}, q >$, where $q \in [q_1, q_2, q_8]$

From this grammar, the function $\sigma$ can be deduced. It is given by

$\sigma(q_1) = \sigma(q_2 = \sigma(q_4) = [q_0, q_7]$

$\sigma(q_3) = \sigma(q_9) = \sigma(q_{12}) = \sigma(I_3^0) = [q_1, q_2, q_8]$

$\sigma(q_5) = \sigma(q_6) = \sigma(q_7) = \sigma(q_{10}) = \sigma(I_1^0) = [q_2]$

$\sigma(q_8) = \sigma(q_{11}) = \sigma(I_2^0) = [q_2, q_5, q_6, q_9, q_{12}]$

$\sigma(I_0^0) = [q_7]$

Either by stripping the above cover, or by directly deducing it from the automaton, the bare cover can be obtained. We list it here for completeness.

$S \rightarrow q_4,\ q_9 \rightarrow q_3 noun,\ q_7 \rightarrow q_5 verb$

$q_1 \rightarrow conj,\ q_3 \rightarrow det,\ q_7 \rightarrow verb$

$q_8 \rightarrow prep,\ q_2 \rightarrow q_1 I_3^0,\ q_4 \rightarrow q_2 I_1^0$

$q_5 \rightarrow q_5 I_2^0,\ q_5 \rightarrow q_6 I_2^0,\ q_{10} \rightarrow q_7 I_0^0$

$q_{11} \rightarrow q_8 I_3^0,\ q_{12} \rightarrow q_9 I_2^0,\ q_{12} \rightarrow q_{12} I_2^0$

$q_5 \rightarrow I_2^0,\ q_6 \rightarrow I_3^0,\ I_0^0 \rightarrow q_4$

$I_1^0 \rightarrow q_{10},\ I_1^0 \rightarrow q_7,\ I_2^0 \rightarrow q_{11}$

$I_3^0 \rightarrow q_9,\ I_3^0 \rightarrow q_{12},$

Together with the predict functions defined in section 5.3.2, this grammar should provide an efficient parser for our example grammar.

## 7 Tadpole Grammars

The function $\sigma$ has been defined, in section 5, via a grammar reduction algorithm. In this section we wish to show that an alternative method exists, and, moreover, that it can be applied to the class of bilinear tadpole grammars. This class consists of all bilinear grammars without epsilon rules, and with no useless symbols, with non-terminals (the head) preceding terminals (the tail) at the right hand side of rules.Thus, rules are of the form

$A \rightarrow \alpha\delta,$

where we use the symbol $\delta$ as a variable over possibly empty sequences of terminals, and $\alpha$ denotes a possibly empty sequence of at most two non-terminals. Capital roman letters are used for non-terminals. Note that a CNLR cover is a member of this class of grammars, as are all grammars that are in Chomsky normal form.

First we change the grammar a little bit by adding $q_0$ to the set of non-terminals of the grammar, assuming that it was not there yet. Next, we create a new grammar, inspired by the grammar of 5.3.1, with pairs $< A, C >$ as non-terminals. The rules of the new grammar are such that (with implicit universal quantification over all variables, as before)

$$< A, C > \rightarrow \delta \equiv A \rightarrow \delta$$

$$< A, C > \rightarrow < B, C > \delta \equiv A \rightarrow B\delta$$

$$< A, C > \rightarrow < B, C > < D, B > \delta \equiv A \rightarrow BD\delta$$

The start symbol of the new grammar, which can be seen as a parametrized version of the tadpole grammar, is defined to be $< S, q_0 >$. A non-terminal $< B, C >$ is a useful one, whence $C \in \sigma(B)$ according to the definition of $\sigma$, if it occurs in a derivation of the parametrized grammar:

$$< S, q_0 > \rightarrow^* \kappa < B, C > \lambda,$$

where $\kappa$ is an arbitrary sequence of non-terminals, and $\lambda$ is a sequence of terminals and non-terminals. Then, we conclude that

$$q_0 \in \sigma(B) \equiv < S, q_0 > \rightarrow^* < B, q_0 > \lambda$$

$$C \in \sigma(B) \wedge C <> q_0 \equiv \exists_{A,D}(< A, C > \rightarrow^* < B, C > \mu$$
$$\wedge < S, q_0 > \rightarrow^* \kappa < C, D > < A, C > \lambda)$$

This definition may be rephrased without reference to the parametrized grammar. Define, for each non-terminal $A$ a set $firstnonts(A)$, such that

$$firstnonts(A) = \{B | A \rightarrow^* B\lambda\}.$$

The predict set $\sigma(A)$ then is obtainable as

$$\sigma(B) = \{C | \exists_{A,D,\delta}(B \in firstnonts(A) \wedge$$
$$D \rightarrow CA\delta)\} \cup \{q_0 | B \in firstnonts(S)\},$$

where $S$ is the start symbol. As in section 5.3.2, the initial prediction is given by $\theta_c = \{q_0\}$.

## 8 An LL/LR-automaton

In order to illustrate the amount of freedom that exists for the construction of automata and associated parsers, we shall construct a non-deterministic LL/LR-automaton and the associated cover, along the lines of section 5.

### 8.1 The automaton

We change the *goto* functions, such that they yield sets of states rather that just one state, as follows:

$$goto_1(s, x) = \{closure(\{I_i^k\}) |$$
$$I_i^j \in s \wedge (I_i^k \rightarrow I_i^j x) \wedge j <> M_i\}$$

$$goto_2(s, x) = \{closure(\{I_i^k\}) | I_i^{M_i} \in s \wedge (I_i^k \rightarrow I_i^{M_i} x)\}$$

The set $Q$ is changed accordingly to be the smallest one that satisfies

$$closure(\{I_0^{M_0}\}) \in Q \wedge (s \in Q \Rightarrow$$
$$(goto_1(s, x) = \emptyset \vee goto_1(s, x) \subset Q) \wedge$$
$$(goto_2(s, x) = \emptyset \vee goto_2(s, x) \subset Q))$$

Every state in this automaton is defined as a set $closure(\{I_i^j\})$ and is, as a consequence, completely characterized by the one non-terminal $I_i^j$. The reason for calling the above an LL/LR-automaton lies in the fact that the states of LR(0) automata for LL(1) grammars have exactly this property. The predicate *reduce* is defined as in section 5.1.

### 8.2 The LL/LR-cover

The cover associated with the LL/LR-automaton just defined, is a simple variant of the cover of section 5.3.2:

$$S \rightarrow s \equiv reduce(s, I_0^0)$$

$$t \rightarrow sy \equiv t \in goto_1(s, y)$$

$$t \rightarrow y \equiv \exists_s(t \in goto_2(s, y))$$

$$t \rightarrow sI_i^0 \equiv t \in goto_1(s, I_i^0)$$

$$t \rightarrow I_i^0 \equiv \exists_s(t \in goto_2(s, I_i^0))$$

$$I_i^0 \rightarrow s \equiv reduce(s, I_i^0),$$

As it is of the tadpole type, the predict mechanism works as explained in section 7.

We just mentioned that each LL/LR-state, and hence each non-terminal of the LL/LR-cover, is completely characterized by one non-terminal, or 'item', of the Earley cover. This correspondence between their non-terminals leads to a tight connection between the two covers. Indeed, the cover we obtained from the LL/LR-automaton can be obtained from the cover of section 4, by eliminating the $\epsilon$-rules $I_i^{M_i} \rightarrow \epsilon$. Of course, the predict functions associated to both covers differ considerably, as it are the non-terminals deriving $\epsilon$, the items beginning with a dot, that are the object of prediction in the Earley algorithm, and they are no longer present in the LL/LR-cover.

## 9 Efficiency

We have discussed a number of bilinear covers now, and we could add many more. In fact, the space of bilinear covers for each context-free grammar, or RTN grammar, is huge. The optimal one would be the one that makes C-parser spend the least time on the average sentence. In general, the least time will be more or less equivalent to the smallest content of the parse matrix. Naively, this content would be proportional to the size of the cover. Under this assumption, the smallest cover would be optimal. Note that the number of non-terminals of the CNLR cover is equal to the number of states of the LR-automaton plus the number of non-terminals of the original grammar. The size of the Earley cover is given by the number of items. In worst case situations the size of the CNLR cover is an exponential function of the size of the original grammar, whereas the size of the Earley cover clearly grows linearly with the size of the original grammar. For many grammars, however, the number of LR(0)-states, may be considerably smaller than the number of items. This seems to be the case for the natural language grammars considered by Tomita[3]. His data even suggest that the number of LR(0) states is a sub-linear function of the original grammar size. Note, however, that predict functions may influence the relation between grammar size and average parse matrix content, as some grammars may allow more restrictive predict functions then others. Summarizing, it seems unlikely, that a single parsing approach would be optimal for all grammars. A viable goal of research would be to find methods for determining the optimal cover for a given grammar. Such research should have a solid experimental back-bone.

The matter gets still more complicated when the original grammar is an attribute grammar. Attribute evaluation may lead to the rejection of certain parse trees that are correct for the grammar without attributes. Then the ease and efficiency of on-the-fly attribute evaluation becomes important, in order to stop wrong parses as soon as possible. In the Rosetta machine translation system [11,12], we use an attributed RTN during the analysis of sentences. The attribute evaluation is bottom-up only, and designed in such a way that the grammar is covered by an attributed Earley cover.

Other points concerning efficiency that we would like to discuss, are issues of precomputation. In the conventional Earley parser, the calculation of the cover is done dynamically, while parsing a sentence. However, it could just as well be done statically, i.e. before parsing, in order to increase parsing performance. For instance, set operations can be implemented more efficiently if the set elements are known non-terminals, rather than unknown items, although this would depend on the choice of programming language. The procedure of generating bilinear covers from LR-automata should always be performed statically, because of the amount of computation involved. Tomita has reported [3], that for a number of grammars, his parsing method turns out to be more efficient than the Earley algorithm. It is not clear, whether his results would still hold if the creation of the cover for the Earley parser were being done statically.

One might be inclined to think that if use is made of precomputed sets of items, as in LR-parsers, one is bound to have a parser that is significantly different from and probably faster than Earley's algorithm, which computes these sets at parse time. The question is much more subtle as we showed in this paper. On the one hand, non-deterministic LR-parsing comes down to the use of certain covers for the grammar at hand, just like the Earley algorithm. Reversely, we showed that the Earley cover can, with minor modifications, be obtained from the LL/LR-automaton, which also uses precomputed sets of items.

## 10 Conclusions

We studied parsing of general context-free languages, by splitting the process into two parts. Firstly, the grammar is turned into bilinear grammar format, and subsequently a general parser for bilinear grammars is applied. Our view on the relation between parsers and covers is similar to the work on covers of Nijholt [7] for grammars that are deterministically parsable.

We established that the Lang algorithm for simulating pushdown automata, hides a prescription for deriving bilinear covers from automata that satisfy certain constraints. Reversely, the LR-parser construction technique has been presented as a way to derive automata from certain bilinear grammars.

We found that the Earley algorithm is intimately related to an automaton that simulates non-deterministic LL-parsing and, furthermore, that non-deterministic LR-automata provide general parsers for context-free grammars, with the same complexity as the Earley algorithm. It should be noted, however, that there are as many parsers with this property, as there are ways to obtain bilinear covers for a given grammar.

## References

1 Earley, J. 1970. An Efficient Context-Free Parsing Algorithm, *Communications ACM* 13(2):94-102.

2 Lang, B. 1974. Deterministic Techniques for Efficient Non-deterministic Parsers, *Springer Lecture Notes in Computer Science* 14:255-269.

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

4 Graham, S.L., M.A. Harrison and W.L. Ruzzo 1980. An improved context-free recognizer, *ACM transactions on Progr. Languages and Systems* 2:415-462.

5 Aho, A.V. and J.D. Ullman 1972. The theory of parsing, translation, and compiling, *Prentice Hall Inc. Englewood Cliffs N.J.*

6 Kruseman Aretz, F.E.J. 1989. A new approach to Earley's parsing algorithm, *Science of Computer Programming* volume 12. .

7 Nijholt, A. 1980. Context-free Grammars: Covers, Normal Forms, and Parsing, *Springer Lecture Notes in Computer Science 93.*

8 Woods, W.A. 1970. Transition network grammars for natural language analysis, *Commun. ACM* 13:591-602.

9 Purdom, P.W. and C.A. Brown 1981. Parsing extended LR(k) grammars, *Acta Informatica* 15:115-127.

10 Nagata, I and M. Sassa 1986. Generation of Efficient LALR Parsers for Regular Right Part Grammars, *Acta Informatica* 23:149-162.

11 Leermakers, R. and J. Rous 1986. The Translation Method of Rosetta, *Computers and Translation* 1:169-183.

12 Appelo L., C Fellinger and J. Landsbergen 1987. Subgrammars, Rule Classes and Control in the Rosetta Translation System, *Proceedings of 3rd Conference ACL, European Chapter, Copenhagen* 118-133.