Generalized LR Parsing for Extensions of Context-Free Grammars

Miguel A. Alonso, David Cabrero & Manuel Vilares

Universidade da Coruña

Abstract

We present a Generalized LR parsing algorithm for extensions of context-free grammars. It differs from previous approaches in the use of dynamic programming techniques to cope with non determinism, instead of a graph-structured stack. The steps for deriving the algorithm from the classical Earley's parsing algorithm are shown.

1 Introduction

LR parsing strategies can analyze LR grammars, which are deterministic. If we consider LR parsing tables in which each entry can contain several actions, we obtain non-deterministic LR parsing, often known as *generalized LR parsing* (GLR), which can analyze non-deterministic context-free grammars. In this context, some mechanism is needed in order to represent the non-deterministic evolution of the stack. Tomita (1996) has proposed an algorithm based on a graph-structured stack but it has problems with cyclic and hidden left recursive constructions. Rekers (1992) has modified the original algorithm to overcome these limitations. Space and time bounds can be reduced transforming the form of the grammar (Sheil 1976). Some approaches also use transformations in the construction of the LR automaton and in these the treatment of cyclicity is even more complex and so is often avoided (Nederhof & Sarbo 1996).

Generalized LR parsing can be extended to deal with grammatical formalisms which have a context-free backbone, such as unification-based grammars, lexical-functional grammars or definite clause grammars. As has been shown by De la Clergerie (1993), there is a straightforward extension of automata-based context-free parsing techniques to Horn-clause analysis. The most common grammatical framework based on Horn-clauses is Definite Clause Grammars (Pereira & Warren 1980), which is the formalism chosen to work with in this article.

1.1 *Notation*

A *context-free grammar* (CFG) is a tuple (V_N, V_T, P, S), where V_N is a finite set of non-terminal symbols, V_T is a finite set of terminal symbols, P is a finite set of productions $A \rightarrow \alpha$ and $S \in V_N$ is the start symbol or axiom of the grammar. We will write $A, B \ldots$ for elements in V_N, $a, b \ldots$ for elements in V_T, $X, Y \ldots$ for elements in $V = V_N \cup V_T$ and $\alpha, \beta \ldots$ for elements in V^*. The relation $\Rightarrow$ on $V^* \times V^*$ is defined by $\alpha \Rightarrow \beta$ if there are $\alpha', \alpha'', A, \gamma$ such that $\alpha = \alpha' A \alpha''$, $\beta = \alpha' \gamma \alpha''$ and $A \rightarrow \gamma \in P$ exists. We can suffix each element in a production r, thus $A_{r,0} \rightarrow A_{r,1} A_{r,2} \ldots A_{r,n_r}$.

We can think of a *definite clause grammar* (DCG) as a CFG skeleton with attributes associated to grammatical symbols. A DCG is defined by a tuple $(V_N, V_T, \mathcal{P}, S, \mathcal{V}, F)$, where $\mathcal{V}$ is a finite set of variables, F is a finite set of functors and $\mathcal{P}$ is a set of definite clauses

$$A_{r,0}(t^1_{r,0}, \ldots, t^{m_0}_{r,0}) \rightarrow A_{r,1}(t^1_{r,1}, \ldots, t^{m_1}_{r,1}) \ldots A_{r,n_r}(t^1_{r,n_r}, \ldots, t^{m_{n_r}}_{r,n_r})$$

where $A_{r,0} \in V_N$, $A_{r,i} \in (V_N \cup V_T)^*$ for $1 \geq i \geq n_r$ and $t^{m_j}_{r,i}$ are terms, which are inductively defined as being either a constant functor of arity 0, a variable or a compound term $f(t_1, \ldots, t_l)$, where f is a functor of arity l and $t_1, \ldots, t_l$ are terms. If $A_{r,i} \in V_T$, the terms $t^1_{r,i}, \ldots, t^{m_i}_{r,i}$ are related to the lexical entry $A_{r,i}$ and therefore the elements in V_T are preterminals rather than actual terminals. For each clause γ_r we define the vector $\overrightarrow{T_r}$ of variables appearing in γ_r. When possible, definite clauses will be written in short as $\mathbf{A_{r,0}} \rightarrow \mathbf{A_{r,1}} \ldots \mathbf{A_{r,n_r}}$.

A *substitution* σ is a finite mapping $\{R_1/t_1, \ldots, R_n/t_n\}$ from variables R_i to terms t_i. The application of σ to a term t is denoted $t\sigma$ and it is achieved by replacing in t any occurrence of R_i by t_i. A term t generalizes or subsumes t' if there exists a substitution σ such that $t' = t\sigma$ and it is denoted $t \preceq t'$. The *unification* of two terms t and t' without common variables returns the most general substitution $\sigma = \mathrm{mgu}(t, t')$, unique to the renaming of variables, so that $t\sigma = t'\sigma$.

Parsing algorithms will be described using *Parsing Schemata*, a framework for high-level description of parsing algorithms (Sikkel 1997). A *parsing system* for a grammar $\mathcal{G}$ and string $a_1 \ldots a_n$ is a triple $\langle \mathcal{I}, \mathcal{H}, \mathcal{D} \rangle$, with $\mathcal{I}$ a set of *items* which represent intermediate parse results, $\mathcal{H}$ an initial set of items that encodes the sentence to be parsed, and $\mathcal{D}$ a set of *deduction steps* that allow new items to be derived from already known items. Deduction steps are of the form $\eta_1, \ldots, \eta_k \vdash \xi$, meaning that if all antecedents η_i of a deduction step are present, then the consequent ξ should be generated by the parser. A set $\mathcal{F} \subseteq \mathcal{I}$ of *final items* represent the recognizing of a sentence. A *parsing schema* is a parsing system parameterized by a context-free grammar and a sentence.

A parsing schema can be generalized from another one using the following transformations (Sikkel 1997):

- *Item refinement*, breaking single items into multiple items.
- *Step refinement*, decomposing a single deduction step in a sequence of steps.
- *Extension* of a schema by considering a larger class of grammars.

In order to decrease the number of items and deduction steps in a parsing schema, we can apply the following kinds of filtering:

- *Static filtering*, in which redundant parts are simply discarded.
- *Dynamic filtering*, using context information to determine the validity of items.
- *Step contraction*, in which a sequence of deduction steps is replaced by a single one.

The set of items in a parsing system $\mathbb{P}_{\text{Alg}}$ corresponding to the parsing schemata **Alg** describing a given parsing algorithm *Alg* is denoted $\mathcal{I}_{\text{Alg}}$, the set of hypotheses $\mathcal{H}_{\text{Alg}}$, the set of final items $\mathcal{F}_{\text{Alg}}$ and the set of deduction steps is denoted $\mathcal{D}_{\text{Alg}}$.

2 Relating Earley and LR parsing algorithms

Given a context-free grammar $\mathcal{G}$ and a input string $a_1 \ldots a_n$, an Earley parser (Earley 1970) for $\mathcal{G}$ constructs a sequence of $n+1$ sets of items. Each item $[A \rightarrow \alpha \bullet \beta, i, j]$ indicates that $\alpha \overset{*}{\Rightarrow} a_{i+1} \ldots a_j$, with $0 \leq i \leq j$, and that symbol A has been predicted in the item set i. Parsing begins with a set of initial items $[S \rightarrow \bullet\alpha, 0, 0]$, where $S \rightarrow \alpha \in P$, and successively applies three operations until new items can not be generated:

scanner operation can be applied if there exists an item $[A \rightarrow \alpha \bullet a\beta, i, j]$ and $a_{j+1} = a$, yielding the item $[A \rightarrow \alpha a \bullet \beta, i, j+1]$.

predictor operation, which represents the predictive descendent phase of the algorithm, can be applied when an item $[A \rightarrow \alpha \bullet B\beta, i, j]$ exists, generating an item $[B \rightarrow \bullet\gamma, j, j]$ for each $B \rightarrow \gamma \in P$.

completer operation can be applied if two items $[A \rightarrow \alpha \bullet B\beta, i, k]$ and $[B \rightarrow \gamma\bullet, k, j]$ exist, and produces a new item $[A \rightarrow \alpha B \bullet \beta, i, j]$, which represents that $a_{k+1} \ldots a_j$ can be reduced to B and therefore, as we know that α reduces the input $a_{i+1} \ldots a_k$, we can ensure that αB reduces $a_{i+1} \ldots a_j$.

The input sentence belongs to the language described by the grammar if the final item $[S \rightarrow \alpha\bullet, 0, n]$ is generated.

The parsing system $\mathbb{P}_{\text{Earley}}$ corresponding to the parsing schemata **Earley** describing the Earley's algorithm is the following (Sikkel 1997:76):

$$\mathcal{I}_{\text{Earley}} = \{[A \rightarrow \alpha \bullet \beta, i, j]\} \mid A \rightarrow \alpha\beta \in P,\ 0 \leq i \leq j\}$$

$$\mathcal{H}_{\text{Earley}} = \{[a, i, i+1] \mid a = a_i\}$$

$$\mathcal{D}^{\text{Init}}_{\text{Earley}} = \{\vdash [S \rightarrow \bullet\alpha, 0, 0]\}$$

$$\mathcal{D}^{\text{Scan}}_{\text{Earley}} = \left\{\ [A \rightarrow \alpha \bullet a\beta, i, j], [a, j, j+1] \vdash [A \rightarrow \alpha a \bullet \beta, i, j+1]\ \right\}$$

$$\mathcal{D}^{\text{Pred}}_{\text{Earley}} = \{[A \rightarrow \alpha \bullet B\beta, i, j] \vdash [B \rightarrow \bullet\gamma, j, j]\}$$

$$\mathcal{D}^{\text{Comp}}_{\text{Earley}} = \left\{\ [A \rightarrow \alpha \bullet B\beta, i, k], [B \rightarrow \gamma\bullet, k, j] \vdash [A \rightarrow \alpha B \bullet \beta, i, j]\ \right\}$$

$$\mathcal{D}_{\text{Earley}} = \mathcal{D}^{\text{Init}}_{\text{Earley}} \cup \mathcal{D}^{\text{Scan}}_{\text{Earley}} \cup \mathcal{D}^{\text{Pred}}_{\text{Earley}} \cup \mathcal{D}^{\text{Comp}}_{\text{Earley}}$$

$$\mathcal{F}_{\text{Earley}} = \{[S \rightarrow \alpha\bullet, 0, n]\}$$

The previous parsing system corresponds to "uncompiling" an LR(0) parser: while LR(0) parsers *compile* the $\mathcal{D}^{\text{Pred}}_{\text{Earley}}$ steps into a finite state control, an Earley parser uses *run-time* items. What $\mathcal{D}^{\text{Pred}}_{\text{Earley}}$ really does is compute the closure function in run-time.

In order to obtain a version closer to LR stack computations, we can make several minor changes in the Scan and Comp steps, involving a slightly different use of indices: the components i and j of an item $[A \rightarrow \alpha \bullet \beta, i, j]$ will now represent the part of the input string recognized by the element in α just before the dot. If $\alpha = \varepsilon$ then $i = j$. As a consequence, the completer step must now have m elements as antecedents, where m is the length of the right hand side of the production to be reduced. In the new $LR(0)$ deduction steps, the Scan and Comp steps are called Shift and Reduce because they show a close relation to shift and reduce operations of LR parsers:

$$\mathcal{D}^{\text{Shift}}_{\text{LR(0)}} = \left\{ \ [A \to \alpha \bullet a\beta, i, j], [a, j, j+1] \vdash [A \to \alpha a \bullet \beta, j, j+1] \ \right\}$$

$$\mathcal{D}^{\text{Reduce}}_{\text{LR(0)}} = \left\{ \begin{array}{l} [B \to X_1 X_2 \cdots X_m \bullet, j_{m-1}, j_m], \\ \vdots \\ [B \to X_1 \bullet X_2 \cdots X_m, j_0, j_1], \\ [A \to \alpha \bullet B\beta, i, j_0] \\ \vdash [A \to \alpha B \bullet \beta, j_0, j_m] \end{array} \right\}$$

$$\mathcal{D}_{\text{LR(0)}} = \mathcal{D}^{\text{Init}}_{\text{Earley}} \cup \mathcal{D}^{\text{Shift}}_{\text{LR(0)}} \cup \mathcal{D}^{\text{Pred}}_{\text{Earley}} \cup \mathcal{D}^{\text{Reduce}}_{\text{LR(0)}}$$

3 Using lookahead: SLR(1) and LR(1)

We can add lookahead to $\mathbb{P}_{\text{LR(0)}}$ in order to mimic the behavior of SLR(1) parsers. For this purpose, we introduce a dynamic filter in the Reduce step, using the function 'follow', which is defined in relation to a function 'first'.

Given a context-free grammar, an element $a \in V_T$ is in first(X), where $X \in V$ if $X = a$ or $X \to \varepsilon \in P$ and $a = \varepsilon$ or $X \to Y_1 \cdots Y_i \cdots Y_m \in P$ and $a \in \text{first}(Y_i)$ and $\forall_{j=1}^{i-1}\ \varepsilon \in \text{first}(Y_j)$. The extension to first$(\alpha)$, where $\alpha = X_1 \cdots X_i \cdots X_n \in V$, is straightforward: $a \in \text{first}(\alpha)$ if $a \in \text{first}(X_1) \cup \cdots \cup \text{first}(X_i)$ and $\varepsilon \notin \text{first}(X_i)$ and $\forall_{j=1}^{i-1}\ \varepsilon \in \text{first}(X_j)$. If $\alpha \overset{*}{\Rightarrow} \varepsilon$ then $\varepsilon \in \text{first}(\alpha)$.

An element $a \in V_T \cup \{\$\}$ is in follow(A), where \$ is a special end-of-input marker not in V_T and $A \in V_N$, if $a = \$$ and A is the start symbol or $A' \to \alpha A \beta \in P$ and $a \in (\text{first}(\beta) - \{\varepsilon\})$ or $A' \to \alpha A \beta \in P$ and $\varepsilon \in \text{first}(\beta)$ and $a \in \text{follow}(A')$.

Checking of lookahead is performed by the condition $\exists [a, j_m, j_m + 1] \in \mathcal{H}_{\text{SLR}},\ a \in \text{follow}(B)$ introduced in the Reduce deduction step for SLR(1):

$$\mathcal{D}^{\text{Reduce}}_{\text{SLR}} = \left\{ \begin{array}{l} [B \to X_1 X_2 \cdots X_m \bullet, j_{m-1}, j_m], \\ \vdots \\ [B \to X_1 \bullet X_2 \cdots X_m, j_0, j_1], \\ [A \to \alpha \bullet B\beta, i, j_0] \\ \vdash [A \to \alpha B \bullet \beta, j_0, j_m] \mid \\ \exists [a, j_m, j_m + 1] \in \mathcal{H}_{\text{SLR}}, \\ a \in \text{follow}(B) \end{array} \right\}$$

$$\mathcal{D}_{\text{SLR}} = \mathcal{D}^{\text{Init}}_{\text{Earley}} \cup \mathcal{D}^{\text{Shift}}_{\text{LR(0)}} \cup \mathcal{D}^{\text{Pred}}_{\text{Earley}} \cup \mathcal{D}^{\text{Reduce}}_{\text{SLR}}$$

In order to obtain a parsing system for LR(1), we must introduce the notion of lookahead into items, as is done in the classical construction of finite state control for LR(1) parsers (Aho & Ullman 1972). The items in $\mathbb{P}_{\text{LR}}$ can be obtained by item refinement of SLR items if we consider that in $\mathbb{P}_{\text{Earley}}$ and $\mathbb{P}_{\text{SLR}}$ each item represents a set of items having the same dotted production and indices but probably different lookahead:

$$\mathcal{I}_{\text{LR}} = \{[A \to \alpha \bullet \beta, b, i, j] \mid A \to \alpha\beta \in P,\ b \in V_T,\ 0 \le i \le j\}$$

where b represents the lookahead. The parsing system $\mathbb{P}_{\text{LR}}$, in which lookahead is set in $\mathcal{D}^{\text{Pred}}_{\text{LR}}$ and checked in $\mathcal{D}^{\text{Reduce}}_{\text{LR}}$, is the following:

$$\mathcal{H}_{\text{LR}} = \mathcal{H}_{\text{Earley}}$$

$$\mathcal{D}^{\text{Init}}_{\text{LR}} = \{\vdash [S \to \bullet\alpha, \$, 0, 0]\}$$

$$\mathcal{D}^{\text{Shift}}_{\text{LR}} = \left\{ \ [A \to \alpha \bullet a\beta, b, i, j], [a, j, j+1] \vdash [A \to \alpha a \bullet \beta, b, j, j+1] \ \right\}$$

$$\mathcal{D}^{\text{Pred}}_{\text{LR}} = \left\{ \ [A \to \alpha \bullet B\beta, b, i, j] \vdash [B \to \bullet\gamma, b', j, j] \mid b' = \text{first}(\beta b) \ \right\}$$

$$\mathcal{D}^{\text{Reduce}}_{\text{LR}} = \left\{ \begin{array}{l} [B \to X_1 X_2 \cdots X_m \bullet, b', j_{m-1}, j_m] \\ \vdots \\ [B \to X_1 \bullet X_2 \cdots X_m, b', j_0, j_1], \\ [A \to \alpha \bullet B\beta, b, i, j_0] \\ \vdash [A \to \alpha B \bullet \beta, b, j_0, j_m] \mid \\ b' \in \text{first}(\beta b), \\ \exists [a, j_m, j_m + 1] \in \mathcal{H}_{\text{LR}}, \\ a = b' \end{array} \right\}$$

$$\mathcal{D}_{\text{LR}} = \mathcal{D}^{\text{Init}}_{\text{LR}} \cup \mathcal{D}^{\text{Shift}}_{\text{LR}} \cup \mathcal{D}^{\text{Pred}}_{\text{LR}} \cup \mathcal{D}^{\text{Reduce}}_{\text{LR}}$$

$$\mathcal{F}_{\text{LR}} = \{[S \to \alpha\bullet, \$, 0, n]\}$$

4 Compiling predictive information into tables

A more compact and efficient algorithm can be obtained by avoiding the computation of Pred steps in run-time. Instead, they are compiled, resulting in the construction of an LR automaton as in classical LR algorithms (Aho & Ullman 1972). The items in the new parsing system $\mathbb{P}_{\mathrm{LR}^c}$ are equivalent to those of $\mathbb{P}_{\mathrm{LR}}$ because items $[A \rightarrow \alpha \bullet \beta, b, i, j]$ are simply replaced by $[st, i, j]$, where st is the precomputed state of the LR automaton which contains the element $[A \rightarrow \alpha \bullet \beta, b]$. The parsing system $\mathbb{P}_{\mathrm{LR}^c}$ is the following:

$$\mathcal{I}_{\mathrm{LR}^c} = \{[st, i, j] \mid st \in \text{set of states of LR automaton},\ 0 \leq i \leq j\}$$

$$\mathcal{H}_{\mathrm{LR}^c} = \mathcal{H}_{\mathrm{Earley}}$$

$$\mathcal{D}^{\mathrm{Init}}_{\mathrm{LR}^c} = \{\vdash [st_0, 0, 0]\}$$

$$\mathcal{D}^{\mathrm{Shift}}_{\mathrm{LR}^c} = \left\{\ [st, i, j], [a, j, j+1] \vdash [st', j, j+1] \mid \mathrm{shift}_{st'} \in \mathrm{action}(st, a)\ \right\}$$

$$\mathcal{D}^{\mathrm{Reduce}}_{\mathrm{LR}^c} = \left\{ \begin{array}{l} [st^m, j_{m-1}, j_m], \ldots, [st^1, j_0, j_1], [st^0, i, j_0] \vdash [st, j_0, j_m] \mid \\ \exists [a, j_m, j_m + 1] \in \mathcal{H}_{\mathrm{LR}^c},\ \ \mathrm{reduce}_r \in \mathrm{action}(st^m, a), \\ st^i \in \mathrm{reveal}(st^{i+1}),\ \ st \in \mathrm{goto}(st^0, A_{r,0}), \\ m = \mathrm{length}(\mathrm{rhs}(r)) \end{array} \right\}$$

$$\mathcal{D}_{\mathrm{LR}^c} = \mathcal{D}^{\mathrm{Init}}_{\mathrm{LR}^c} \cup \mathcal{D}^{\mathrm{Shift}}_{\mathrm{LR}^c} \cup \mathcal{D}^{\mathrm{Reduce}}_{\mathrm{LR}^c}$$

where $st^i \in \mathrm{reveal}(st^{i+1})$ is equivalent to $st^{i+1} \in \mathrm{goto}(st^i, X)$ if $X \in V_N$ and is equivalent to $\mathrm{shift}_{st^{i+1}} \in \mathrm{action}(st^i, X)$ if $X \in V_T$. More intuitively, we can say that *reveal* function traverses the finite state control of the automaton backwards.

$$\mathcal{F}_{\mathrm{LR}^c} = \{[st_f, 0, n]\}$$

where st_f is a final state of the LR automaton.

In the preceding, ‘action’ and ‘goto’ refer to the tables that code the behavior of the LR automaton:

action table determines what action should be taken for a given state and lookahead. In the case of shift actions, it determines the resulting new state and in the case of reduce actions, the production which is to be applied for the reduction.

goto table determines what will be the state after performing a reduce action. Each entry is accessed using the current state and the non-terminal in the left-hand side of the production to be applied for reduction.

A significant advantage with respect to previous parsing systems is that we can now differentiate between LR(1) and LALR(1) algorithms simply by choosing the appropriate compiling methods for the finite state control.

5 GLR parsing for DCG

Context-free grammars have a finite number of grammar symbols. When these grammars are enriched with terms or feature-structures, the number of grammar symbols can not be guaranteed to be finite. Thus the construction of the parsing tables may be a non terminating process. A solution to this problem is to use positive restrictors (Shieber 1985) or negative restrictors (Trujillo 1994) to define a finite number of equivalence classes into which the infinite number of nonterminals may be sorted. The restrictor must be applied to obtain the compiled information: the first and follow functions, the closure of the LR automaton and the action and goto tables.

Several restrictors exist for each grammar but termination is not guaranteed for all of them and furthermore, the best restrictor can not be chosen automatically since it depends on the amount of grammatical information that is to be preserved. In practice, a good balance is obtained if only the underlying context-free grammar is considered during the compilation phase (Vilares et al. 1998).

Deduction steps need to access the terms associated to grammar symbols. As that information is not contained in the states of the automaton, it must be included in items. Applying *item refinement* to $\mathbb{P}_{\mathrm{LR}^c}$ items, we obtain the new form of the items $[A(t^1, \ldots, t^m), st, i, j]$, with a new element that represents an element of a clause. With respect to deduction steps, Shift steps must be refined into:

InitShift, which is applied when the symbol to be shifted is the first symbol in the right hand side of a clause.

Shift, which is applied in the shift of the other symbols in a clause.

The new schema $\mathbf{LR(DCG)}$ is the extension of the schema $\mathbf{LR^c}$ to the case of definite clause grammars and it is defined by the following parsing system $\mathbb{P}_{\mathrm{LR(DCG)}}$:

$$\mathcal{I}_{\mathrm{LR(DCG)}} = \{[\mathbf{A}, st, i, j]\}$$

$$\mathcal{H}_{\mathrm{LR(DCG)}} = \mathcal{H}_{\mathrm{Earley}}$$

$$\mathcal{D}^{\mathrm{Init}}_{\mathrm{LR(DCG)}} = \{\vdash [-, st_0, 0, 0]\}$$

$$\mathcal{D}^{\mathrm{InitShift}}_{\mathrm{LR(DCG)}} = \left\{ \begin{array}{l} [\mathbf{A}, st, i, j], [a, j, j+1] \vdash [\mathbf{A_{r,1}}, st', j, j+1] \mid \\ \mathrm{shift}_{st'} \in \mathrm{action}(st, a),\ \ \mathbf{A_{r,1}} = a(t^1_{r,1}, \ldots, t^m_{r,1}) \end{array} \right\}$$

$$\mathcal{D}^{\mathrm{Shift}}_{\mathrm{LR(DCG)}} = \left\{ \begin{array}{l} [\mathbf{A_{r,s}}, st, i, j], [a, j, j+1] \vdash [\mathbf{A_{r,s+1}}, st', j, j+1] \mid \\ \mathrm{shift}_{st'} \in \mathrm{action}(st, a),\ \ \mathbf{A_{r,s+1}} = a(t^1_{r,s+1}, \ldots, t^m_{r,s+1}) \end{array} \right\}$$

$$\mathcal{D}^{\mathrm{Reduce}}_{\mathrm{LR(DCG)}} = \left\{ \begin{array}{l} [\mathbf{A_{r,m}}, st^m, j_{m-1}, j_m], \ldots, [\mathbf{A_{r,1}}, st^1, j_0, j_1], [\mathbf{A}, st^0, i, j_0] \\ \vdash [\mathbf{A_{r,0}}\sigma, st, j_0, j_m] \mid \\ \exists [a, j_m, j_m+1] \in \mathcal{H}_{\mathrm{LR(DCG)}},\ \ \mathrm{reduce}_r \in \mathrm{action}(st^m, a), \\ st^i \in \mathrm{reveal}(st^{i+1}),\ \ st \in \mathrm{goto}(st^0, A_{r,0}), \\ m = \mathrm{length}(\mathrm{rhs}(r)),\ \ \sigma = \mathrm{mgu}(\mathbf{A_{r,m}}, \ldots, \mathbf{A_{r,1}}) \end{array} \right\}$$

$$\mathcal{D}_{\mathrm{LR(DCG)}} = \mathcal{D}^{\mathrm{Init}}_{\mathrm{LR(DCG)}} \cup \mathcal{D}^{\mathrm{InitShift}}_{\mathrm{LR(DCG)}} \cup \mathcal{D}^{\mathrm{Shift}}_{\mathrm{LR(DCG)}} \cup \mathcal{D}^{\mathrm{Reduce}}_{\mathrm{LR(DCG)}}$$

$$\mathcal{F}_{\mathrm{LR(DCG)}} = \{[\mathbf{S}, st_f, 0, n]\}$$

6 GLR Logic Push-Down Automata and dynamic programming

The common framework for parsing described by Lang (1991) is based on dynamic programming interpretation of *logic push-down automata*, which are push-down automata storing logic atoms in the stack instead of symbols in $V_T \cup V_N$. In that framework, weakly predictive automata, such as LR, can be interpreted in dynamic programming using items containing only the top element of the stack. The resulting automata are very close to inference systems (De la Clergerie 1993:173-175). In this context, $\mathbb{P}_{\text{LR(DCG)}}$ can be seen as a dynamic programming interpretation of LR(1) or LALR(1) parsing algorithms using an inference system based on that kind of items. It can be easily transformed into a set of logic push-down transitions, applying implicit binarization of clauses. Thus, the reduction of a clause

$$\mathbf{A_{r,0}} \rightarrow \mathbf{A_{r,1}} \dots \mathbf{A_{r,n_r}}$$

can be equivalently performed as the reduction of the following n_r+1 clauses with at most 2 elements on their right-hand side:

$$\begin{array}{l} \mathbf{A_{r,0}} \rightarrow \nabla_{r,0}(\overrightarrow{T_r}) \\ \nabla_{r,0}(\overrightarrow{T_r}) \rightarrow \mathbf{A_{r,1}}\ \nabla_{r,1}(\overrightarrow{T_r}) \\ \vdots \\ \nabla_{r,n_r-1}(\overrightarrow{T_r}) \rightarrow \mathbf{A_{r,n_r}}\ \nabla_{r,n_r}(\overrightarrow{T_r}) \\ \nabla_{r,n_r}(\overrightarrow{T_r}) \rightarrow \varepsilon \end{array}$$

Applying *item refinement* to $\mathbb{P}_{\text{LR(DCG)}}$ items, we obtain the new form of items, in which the element that previously represented a grammar symbol will now represent a symbol in a clause or a $\nabla_{r,i}$ meaning that elements $\mathbf{A_{r,i+1}} \dots \mathbf{A_{r,n_r}}$ have been reduced. Therefore, $\nabla_{r,i}$ is like a dotted clause $A_{r,0} \rightarrow \alpha \bullet \beta$ where $\alpha = \mathbf{A_{r,1}} \dots \mathbf{A_{r,i}}$ and $\beta = \mathbf{A_{r,i+1}} \dots \mathbf{A_{r,n_r}}$. With respect to deduction steps, Reduce steps must be refined into:
Sel, which select the clause to be reduced.
Red, which reduce each implicit binary clause.
Head, which recognize the left-hand symbol of the clause reduced.

Considering that Head deduction steps correspond to SWAP transitions, Init, InitShift, Shift and Sel steps correspond to PUSH transitions and that Red steps correspond to POP transitions, we can transform the deductive steps of the $\mathbb{P}_{\text{LR(DCG)}}$ system into a set of logic push-down transitions, with the advantage that we do not need to deal explicitly with unifications because the operational mechanism of the LPDA is in charge of them. The set of logic push-down transitions is the following:

$$\mathcal{D}^{\text{Init}}_{\text{LR}^{\text{S1}}\text{(DCG)}} = \{\vdash [-, st_0, 0, 0]\}$$

$$\mathcal{D}^{\text{InitShift}}_{\text{LR}^{\text{S1}}\text{(DCG)}} = \left\{ \begin{array}{l} [\mathbf{A}, st, i, j] \vdash [\mathbf{A_{r,1}}, st', j, j+1] \quad [\mathbf{A}, st, i, j] \mid \\ \exists [a, j, j+1] \in \mathcal{H}_{\text{LR(DCG)}},\ \text{shift}_{st'} \in \text{action}(st, a), \\ \mathbf{A_{r,1}} = a(t^1_{r,1}, \dots, t^{m_1}_{r,1}) \end{array} \right\}$$

$$\mathcal{D}^{\text{Shift}}_{\text{LR}^{\text{S1}}(\text{DCG})} = \left\{ \begin{array}{l} [\mathbf{A_{r,s}}, st, i, j] \vdash [\mathbf{A_{r,s+1}}, st', i, j+1] \quad [\mathbf{A_{r,s}}, st, i, j] \mid \\ \exists [a, j, j+1] \in \mathcal{H}_{\text{LR(DCG)}},\ \text{shift}_{st'} \in \text{action}(st, a), \\ \mathbf{A_{r,s+1}} = a(t^1_{r,s+1}, \ldots, t^{m_s+1}_{r,s+1}) \end{array} \right\}$$

$$\mathcal{D}^{\text{Sel}}_{\text{LR}^{\text{S1}}(\text{DCG})} = \left\{ \begin{array}{l} [\mathbf{A_{r,n_r}}, st, i, j] \vdash [\nabla_{r,n_r}(\overrightarrow{T_r}), st, j, j] \quad [\mathbf{A_{r,n_r}}, st, i, j] \mid \\ \exists [a, j, j+1] \in \mathcal{H}_{\text{LR(DCG)}},\ \text{reduce}_r \in \text{action}(st, a) \end{array} \right\}$$

$$\mathcal{D}^{\text{Red}}_{\text{LR}^{\text{S1}}(\text{DCG})} = \left\{ \begin{array}{l} [\nabla_{r,s}(\overrightarrow{T_r}), st, i, k][\mathbf{A_{r,s}}, st, k, j] \vdash [\nabla_{r,s-1}(\overrightarrow{T_r}), st', i, j] \mid \\ st' \in \text{reveal}(st) \end{array} \right\}$$

$$\mathcal{D}^{\text{Head}}_{\text{LR}^{\text{S1}}(\text{DCG})} = \left\{ \ [\nabla_{r,0}(\overrightarrow{T_r}), st, i, j] \vdash [\mathbf{A_{r,0}}, st', i, j] \mid st' \in \text{goto}(st, A_{r,0}) \ \right\}$$

$$\begin{aligned} \mathcal{D}_{\text{LR}^{\text{S1}}(\text{DCG})} = \ & \mathcal{D}^{\text{Init}}_{\text{LR}^{\text{S1}}(\text{DCG})} \cup \mathcal{D}^{\text{InitShift}}_{\text{LR}^{\text{S1}}(\text{DCG})} \cup \mathcal{D}^{\text{Shift}}_{\text{LR}^{\text{S1}}(\text{DCG})} \cup \\ & \mathcal{D}^{\text{Sel}}_{\text{LR}^{\text{S1}}(\text{DCG})} \cup \mathcal{D}^{\text{Red}}_{\text{LR}^{\text{S1}}(\text{DCG})} \cup \mathcal{D}^{\text{Head}}_{\text{LR}^{\text{S1}}(\text{DCG})} \end{aligned}$$

7 Conclusion

We have considered Earley's algorithm as a starting point for deriving other well known parsing algorithms, such us Generalized LR. Several intermediate parsing systems have been used in the path from Earley to LR, applying simple and intuitive transformations in each step. The resulting algorithm has been extended to deal with definite clause grammars and it has been integrated in the common framework for parsing in dynamic programming proposed by Lang. An implementation of a parser for DCGs based on our specification of LALR(1) parsers has been described in (Vilares & Alonso forthcoming).

Acknowledgements. This work has been partially supported by the FEDER of the European Union (1FD97-0047-C04-02), Government of Spain (HF97-223) and Xunta de Galicia (XUGA10505B96 and XUGA20402B97).

REFERENCES

Aho, Alfred V. & Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling.* Englewood Cliffs: Prentice Hall.

De la Clergerie, Eric. 1993. *Automates à Piles et Programmation Dynamique. DyALog : Une Application à la Programmation en Logique.* Ph.D. dissertation. Université Paris 7. Paris, France.

Earley, J. 1970. "An Efficient Context-Free Parsing Algorithm". *Communications of the Associattion for Computing Machinery (CACM)* 13:2.94-102.

Lang, Bernard. 1991. "Towards a Uniform Formal Framework for Parsing". *Current Issues in Parsing Technology* ed. by Masaru Tomita, 153-171. Norwell: Kluwer Academic.

Nederhof, Mark-Jan & J. J. Sarbo. 1996. "Increasing the Applicability of LR Parsing". *Recent Advances in Parsing Technology* ed. by H. Bunt & M. Tomita, 35-57. Dordrecht: Kluwer Academic.

Pereira, Fernando C. N. & David H. D. Warren. 1980. "Definite Clause Grammars for Language Analysis: A Survey of the Formalism and a Comparison with Augmented Transition Networks". *Artificial Intelligence*, 13.231-278.

Rekers, Jan. 1992. *Parsing Generation for Interactive Environments.* Ph.D. dissertation, Amsterdam: University of Amsterdam.

Sheil, B. A. 1976. "Observations on Context-Free Grammars". *Statistical Methods in Linguistics*, 71-109, Stockholm, Sweden.

Shieber, Stuart. 1985. "Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms". *Proceedings of the 23th Annual Meeting of the Association for Computational Linguistics (ACL'85)*, 145-152.

Sikkel, Klaas. 1997. *Parsing Schemata: A Framework for Specification and Analysis of Parsing Algorithms.* Heidelberg: Springer-Verlag.

Tomita, Masaru. 1986. *Efficient Parsing for Natural Language.* Boston: Kluwer Academic.

Trujillo, Arturo. 1994. "Computing First and Follow Functions for Feature-Theoretic Grammars". *Proceedings of the Fifteenth International Conference on Computational Linguistics (COLING'94).* Kyoto, Japan.

Vilares, Manuel & Miguel A. Alonso. Forthcoming. "An LALR Extension for DCGs in Dynamic Programming". To appear in *Mathematical Linguistics II* ed. by Carlos Martín Vide. Amsterdam: John Benjamins.

——, David Cabrero & Miguel A. Alonso. 1998. "A Comparison for Unification-Based Parsers". *Proceedings of APPIA-GULP-PRODE 1998 Joint Conference on Declarative Programming (AGP'98)* ed. by Moreno Falaschi, José L. Freire & Manuel Vilares, 113–123. La Coruña, Spain.