A CASE STUDY IN OPTIMIZING PARSING SCHEMATA BY DISAMBIGUATION FILTERS

Eelco Visser
Programming Research Group
University of Amsterdam
visser@acm.org

Abstract

Disambiguation methods for context-free grammars enable concise specification of programming languages by ambiguous grammars. A disambiguation filter is a function that selects a subset from a set of parse trees—the possible parse trees for an ambiguous sentence. The framework of filters provides a declarative description of disambiguation methods independent of parsing. Although filters can be implemented straightforwardly as functions that prune the parse forest produced by some generalized parser, this can be too inefficient for practical applications.

In this paper the optimization of parsing schemata, a framework for high-level description of parsing algorithms, by disambiguation filters is considered in order to find efficient parsing algorithms for declaratively specified disambiguation methods. As a case study the optimization of the parsing schema of Earley's parsing algorithm by two filters is investigated. The main result is a technique for generation of efficient LR-like parsers for ambiguous grammars disambiguated by means of priorities.

1 Introduction

The syntax of programming languages is conventionally described by context-free grammars. Although programming languages should be unambiguous, they are often described by ambiguous grammars because these allow a more natural formulation and yield better abstract syntax. For instance, consider the following grammars. The first, ambiguous grammar gives a clearer and more concise description of arithmetic expressions than the second unambiguous one.

```
"a"         -> E
E "+" E     -> E
E "*" E     -> E
"(" E ")"   -> E
```

```
"a"         -> V    T -> E
E "+" T     -> E    V -> T
T "*" V     -> T
"(" E ")"   -> V
```

To obtain an unambiguous specification of a language described by an ambiguous grammar it has to be disambiguated. For example, the first grammar above can be disambiguated by associativity and priority rules that express that `E "*" E -> E` has higher priority than `E "+" E -> E` and that both productions are left associative. In the second grammar these disambiguation rules have been encoded in the grammar itself by means of extra non-terminals.

In Klint and Visser (1994) we have set up a framework for specification and comparison of disambiguation methods. In this framework a disambiguation method is described as a *filter* on sets of parse trees. A disambiguation filter is interpreted by parsing sentences according to the ambiguous context-free grammar with some generalized parsing method, for instance Generalized LR parsing (Tomita, 1985, Rekers, 1992), and then prune the resulting parse forest with the filter. Because this method of specification of disambiguation is independent of parsing, a language definition can be understood without understanding a parsing algorithm and it can be implemented by any generalized parser.

Although filters provide a uniform model for the description of disambiguation, they are too inefficient for several applications because all possible parse trees for a sentence have to be built before the intended ones are selected. (The number of possible parse trees for the first grammar above grows exponentially with the length of strings.) The *optimization problem* for filters is to find an efficient parser for the combination of a context-free grammar and a disambiguation filter. The filter can be used to prevent parse steps that lead to

parse trees that would be removed by the filter after parsing. *Parsing schemata*, introduced by Sikkel (1993, 1994), are high-level descriptions of parsing algorithms that abstract from control- and data-structures and provide a suitable framework for the study of the interaction between filters and parsers.

Since it is not clear how to solve the optimization problem in general, if that is possible at all, an instance of the problem is studied in this paper, i.e., the optimization of the underlying parsing schema of Earley's (1970) parsing algorithm by a filter for disambiguation by priorities. This method, which is the disambiguation method of the formalism SDF (Heering *et al.*, 1989), interprets a priority relation on context-free productions as two consecutive filters. The first selects trees without a priority conflict. The second selects trees that are minimal with respect to a multi-set ordering on trees induced by the priority relation.

The main result of this paper is a parsing schema for parsing with priorities. The schema specifies a complete implementation of parsing modulo priority conflicts and a partial implementation for the multi-set order. The schema can be implemented as an adaptation of any parser generator in the family of LR parser generators. The resulting parsers yield parse trees without priority conflicts.

The method of specifying a disambiguation method by a filter and applying it to optimize the parsing schema of some parsing algorithm appears to be fertile soil for growing new parsing algorithms from old ones.

The rest of the paper is structured as follows. In §2 some preliminary notions are defined. In §3 disambiguation filters are defined. In §4 parsing schemata are informally introduced. In §5 priority rules and the notion of priority conflict are defined and a parsing schema optimized for the priority conflict filter is derived. In §6 the relation between Earley parsing and LR parsing is discussed and it is shown how optimization results can be translated from the former to the latter. Furthermore, the results are extended to SLR(1) parsing. In §7 the multi-set filter induced by a priority declaration is defined and a partial optimization of the Earley schema for this filter is derived. The two optimizations can be combined in a single schema, obtaining an efficient implementation of disambiguation with priorities.

2 Preliminaries

Definition 2.1 (Context-free Grammar) A *context-free grammar* $\mathcal{G}$ is a triple $\langle V_N, V_T, \mathcal{P} \rangle$, where V_N is a finite set of nonterminal symbols, V_T a finite set of terminal symbols, V the set of symbols of $\mathcal{G}$ is $V_N \cup V_T$, and $\mathrm{P}(\mathcal{G}) = \mathcal{P} \subseteq V^* \times V_N$ a finite set of productions. We write $\alpha \to A$ for a production $p = \langle \alpha, A \rangle \in \mathcal{P}$. □

The $\alpha \dashrightarrow A$ notation for productions (instead of the traditional $A \to \alpha$) is a convention of the syntax definition formalism SDF to emphasize the use of productions as mixfix function declarations. The string rewrite relation $\to_{\mathcal{G}}^*$ induced by a context-free grammar is therefore also reversed, from a generation relation to a recognition relation. Repeated application of productions rewrites a string to its syntactic category. The statement $w \to^* A$ means that the string w can be reduced to the symbol A.

Observe that we do not distinguish a start symbol from which sentences are derived. Each nonterminal in V_N generates a set of phrases as is defined in the following definition.

Definition 2.2 (Parse Trees) A context-free grammar $\mathcal{G}$ generates a family of sets of *parse trees* $\mathcal{T}(\mathcal{G}) = (\mathcal{T}(\mathcal{G})(X) \mid X \in V)$, which contains the minimal sets $\mathcal{T}(\mathcal{G})(X)$ such that

$$\frac{X \in V}{X \in \mathcal{T}(\mathcal{G})(X)}$$

$$\frac{A_1 \ldots A_n \to A \in \mathrm{P}(\mathcal{G}),\ t_1 \in \mathcal{T}(\mathcal{G})(A_1),\ \ldots,\ t_n \in \mathcal{T}(\mathcal{G})(A_n)}{[t_1 \ldots t_n \to A] \in \mathcal{T}(\mathcal{G})(A)}$$

We will write t_α for a list $t_1 \ldots t_n$ of trees where α is the list of symbols $X_1 \ldots X_n$ and $t_i \in \mathcal{T}(\mathcal{G})(X_i)$ for $1 \leq i \leq n$. Correspondingly we will denote the set of all lists of trees of type α as $\mathcal{T}(\mathcal{G})(\alpha)$. Using this notation $[t_1 \ldots t_n \to A]$ can be written as $[t_\alpha \dashrightarrow A]$ and the concatenation of two lists of trees t_α and t_β is written as $t_\alpha t_\beta$ and yields a list of trees of type $\alpha\beta$.

The *yield* of a tree is the concatenation of its leaves. The language $L(\mathcal{G})$ defined by a grammar $\mathcal{G}$ is the family of sets of strings $L(\mathcal{G})(A) = \mathrm{yield}(\mathcal{T}(\mathcal{G})(A))$. □

Definition 2.3 (Parsing) A *parser* is a function Π that maps each string $w \in V_T^*$ to a set of parse trees. A parser Π *accepts* a string w if $|\Pi(w)| > 0$. A parser Π is *deterministic* if $|\Pi(w)| \leq 1$ for all strings w. A parser for a context-free grammar $\mathcal{G}$ that accepts exactly the sentences in $L(\mathcal{G})$ is defined by

$$\Pi(\mathcal{G})(w) = \{t \in \mathcal{T}(\mathcal{G})(A) \mid A \in V_N, \text{yield}(t) = w\}$$ □

Example 2.4 As an example consider the ambiguous grammar

```
"a"         -> E
E "+" E     -> E
E "*" E     -> E
"(" E ")"   -> E
```

from the introduction. According to this grammar the string $a + a * a$ has two parses:

$$\begin{aligned}\Pi(\mathcal{G})(a + a * a) = \{&[[[a \to E] + [a \to E] \to E] * [a \to E] \to E] \\ &[[a \to E] + [[a \to E] * [a \to E] \to E] \to E]\}\end{aligned}$$ □

3 Disambiguation Filters

Definition 3.1 (Disambiguation Filter) A *filter* $\mathcal{F}$ for a context-free grammar $\mathcal{G}$ is a function $\mathcal{F} : \wp(\mathcal{T}) \to \wp(\mathcal{T})$ that maps sets of parse trees to sets of parse trees, where $\mathcal{F}(\Phi) \subseteq \Phi$ for any $\Phi \subseteq \mathcal{T}$. The *disambiguation* of a context-free grammar $\mathcal{G}$ by a filter $\mathcal{F}$ is denoted by $\mathcal{G}/\mathcal{F}$. The *language* $L(\mathcal{G}/\mathcal{F})$ generated by $\mathcal{G}/\mathcal{F}$ is the set

$$L(\mathcal{G}/\mathcal{F}) = \{w \in V_T^* \mid \exists \Phi \subseteq \mathcal{T}(\mathcal{G}) : \text{yield}(\Phi) = \{w\} \ \wedge \ \mathcal{F}(\Phi) = \Phi\}$$

The *interpretation* of a string w by $\mathcal{G}/\mathcal{F}$ is the set of trees $\mathcal{F}(\Pi(\mathcal{G})(w))$. A filter $\mathcal{F}_2$ is also applicable to a disambiguated grammar $\mathcal{G}/\mathcal{F}_1$, which is denoted by $(\mathcal{G}/\mathcal{F}_1)/\mathcal{F}_2$ and is equivalent to $\mathcal{G}/(\mathcal{F}_2 \circ \mathcal{F}_1)$. □

Several properties and examples of filters are discussed in Klint and Visser (1994). In §5 and §7 two examples of disambiguation filters will be presented. The optimization problem for disambiguation filters can be formulated as follows.

Definition 3.2 (Optimization by Filter) Given a context-free grammar $\mathcal{G}$ and a filter $\mathcal{F}$, a parser π is an optimization of $\Pi(\mathcal{G})$ if for any string w

$$\mathcal{F}(\Pi(\mathcal{G})(w)) \subseteq \pi(w) \subseteq \Pi(\mathcal{G})(w)$$

We say that π *approximates* $\mathcal{F} \circ \Pi(\mathcal{G})$. π is an optimal approximation if $\pi(w) = \mathcal{F}(\Pi(\mathcal{G})(w))$ for any w. □

4 Parsing Schemata

Parsing schemata (Sikkel, 1993, 1997) abstract from the details of control- and data-structures of full parsing algorithms by only considering the intermediate results of parsing. A parsing system is a deduction system that specifies how from a set of hypotheses (the tokens of a sentence) assertions (the intermediate parser states) can be derived according to a set of deduction rules for some context-free grammar. A parsing schema is a parsing system parameterized with a context-free grammar and a sentence. Below parsing schemata are introduced informally by means of an example. A formal treatment can be found in Sikkel (1993, 1997). A related approach is the deductive parsing method of Shieber *et al.* (1995), where inference rules describing parsing algorithms much like parsing schemata are interpreted as chart parsers in Prolog.

Definition 4.1 defines a parsing schema for Earley's (1970) parsing algorithm. Its specification consists of an implicit definition of the set of hypotheses H, the definition of a set of items $\mathcal{I}$ and the definition of a set of deduction rule schemata. For each string $a_1 \ldots a_n$ the set of hypotheses H is the set containing the items $[a_i, i-1, i]$ for $1 \leq i \leq n$. The set of items $\mathcal{I}$ is the domain of the deduction system, i.e., the items are the subject of deductions. According to this definition, Earley items are of the form $[\alpha \bullet \beta \to A, i, j]$, where $\alpha\beta \to A$

is a production of grammar $\mathcal{G}$. The indices refer to positions in the string $a_1 \dots a_n$. The intention of this definition is that an item $[\alpha \bullet \beta \to A, i, j]$ can be derived if $a_{i+1} \dots a_j \to^*_{\mathcal{G}} \alpha$ and $a_1 \dots a_i A\gamma \to^*_{\mathcal{G}} B$, for some non-terminal B and string of symbols γ. The deduction rules (I) through (C) describe how these items can be derived. Rule (I), the *initialization* rule, specifies that the item $[\bullet\alpha \to A, 0, 0]$ can always be derived. The *predict* rule (P), states that a production $\gamma \to B$ can be predicted at position j, if the item $[\alpha \bullet B\beta \to A, i, j]$ has already been derived. Finally, the rules (S) and (C) finalize the recognition of a predicted and recognized token or nonterminal—witnessed by the second premise—by shifting the $\bullet$ over the predicted symbol.

Definition 4.1 (Earley) Parsing schema for Earley's parsing algorithm (Earley, 1970).

$$\frac{\alpha\beta \to A \in \mathrm{P}(\mathcal{G}),\ 0 \le i \le j}{[\alpha \bullet \beta \to A, i, j] \in \mathcal{I}}$$

$$\frac{}{[\bullet\alpha \to A, 0, 0]} \quad \text{(I)}$$

$$\frac{[\alpha \bullet B\beta \to A, i, j]}{[\bullet\gamma \to B, j, j]} \quad \text{(P)}$$

$$\frac{[\alpha \bullet a\beta \to A, i, j],\ [a, j, j+1]}{[\alpha a \bullet \beta \to A, i, j+1]} \quad \text{(S)}$$

$$\frac{[\alpha \bullet B\beta \to A, h, i],\ [\gamma \bullet \to B, i, j]}{[\alpha B \bullet \beta \to A, h, j]} \quad \text{(C)}$$

□

A derivation according to a parsing schema is a sequence $I_0, \dots, I_m$ of items such that for each i $(0 \le i \le m)$ $I_i \in H$ or there is a $J \subseteq \{I_0, \dots, I_{i-1}\}$ such that $J \vdash I_i$ is (the instantiation of) a deduction rule. (Observe that if J is empty this corresponds to the case of using a rule without premises, such as the initialization rule.) A string $w = a_1 \dots a_n$ is in the language of context-free grammar $\mathcal{G}$ if an item $[\alpha\bullet \to A, 0, n]$ is derivable from the hypotheses corresponding to w in the instantiation of the parsing schema in Definition 4.1 with $\mathcal{G}$. An item of the form $[\alpha\bullet \to A, 0, n]$ is called a *final* item and signifies that the entire string is recognized as an A phrase. The predicate $w \vdash^P_{\mathcal{G}} I$ expresses that there is a derivation $I_0, \dots, I_m = I$ of the item I from the hypotheses generated from string w in the instantiation of parsing schema P with grammar $\mathcal{G}$.

The schema in Example 4.1 only defines how strings can be recognized. Since disambiguation filters are defined on sets of trees and not on items, a way to relate items to trees is needed. Definition Definition 4.3 gives an extension of the schema in Definition 4.1 that describes how trees can be built as a result of the deduction steps. First we need a definition of partial parse tree

Definition 4.2 (Partial Parse Tree) A partial parse tree is a tree expression of the form $[t_\alpha \to A]$ where $t_\alpha \in \mathcal{T}(\mathcal{G})(\alpha)$ and such that the tree can be completed to a normal tree by adding a list of trees t_β, i.e., $[t_\alpha t_\beta \to A] \in \mathcal{T}(\mathcal{G})(A)$. (which requires $\alpha\beta \to A \in \mathrm{P}(\mathcal{G})$.) □

The items in the schema have the form $[\alpha \bullet \beta \to A, i, j] \Rightarrow [t_\alpha \to A]$ and express that from position i to position j a phrase of type α has been recognized and the partial parse tree $[t_\alpha \to A]$ has been built as a result. The set of hypotheses H is changed such that token items are annotated with trees, i.e., for each token a_i in the string $[a_i, i-1, i] \Rightarrow a_i \in H$. Note how the shift and complete rules extend partial parse trees.

Definition 4.3 (Earley with Trees) Parsing schema for Earley's algorithm with construction of parse trees.

$$\frac{\alpha\beta \to A \in \mathrm{P}(\mathcal{G}),\ 0 \le i \le j,\ t_\alpha \in \mathcal{T}(\mathcal{G})(\alpha)}{[\alpha \bullet \beta \to A, i, j] \Rightarrow [t_\alpha \to A] \in \mathcal{I}}$$

$$\frac{}{[\bullet\alpha \to A, 0, 0] \Rightarrow [\to A]} \quad \text{(I)}$$

$$\frac{[\alpha \bullet B\beta \to A, h, i] \Rightarrow [t_\alpha \to A]}{[\bullet\gamma \to B, i, i] \Rightarrow [\to B]} \quad \text{(P)}$$

$$\frac{[\alpha \bullet a\beta \to A, h, i] \Rightarrow [t_\alpha \to A],\ [a, i, i+1] \Rightarrow a}{[\alpha a \bullet \beta \to A, h, i+1] \Rightarrow [t_\alpha a \to A]} \quad \text{(S)}$$

$$\frac{[\alpha \bullet B\beta \to A, h, i] \Rightarrow [t_\alpha \to A],\ [\gamma \bullet \to B, i, j] \Rightarrow t_B}{[\alpha B \bullet \beta \to A, h, j] \Rightarrow [t_\alpha t_B \to A]} \quad \text{(C)}$$

$$
\begin{array}{ll}
[a,0,1] & \Rightarrow a \\
[+,1,2] & \Rightarrow + \\
[a,2,3] & \Rightarrow a \\
[\bullet E + E \rightarrow E,0,0] & \Rightarrow [\rightarrow E] \\
[\bullet a \rightarrow E,0,0] & \Rightarrow [\rightarrow E] \\
[a\bullet \rightarrow E,0,1] & \Rightarrow [a \rightarrow E] \\
[E \bullet + E \rightarrow E,0,1] & \Rightarrow [[a \rightarrow E] \rightarrow E] \\
[E + \bullet E \rightarrow E,0,2] & \Rightarrow [[a \rightarrow E] + \rightarrow E] \\
[\bullet a \rightarrow E,2,2] & \Rightarrow [\rightarrow E] \\
[a\bullet \rightarrow E,2,3] & \Rightarrow [a \rightarrow E] \\
[E + E\bullet \rightarrow E,0,3] & \Rightarrow [[a \rightarrow E] + [a \rightarrow E] \rightarrow E]
\end{array}
$$

Figure 1: Derivation with the parsing schema in Definition 4.3 and the grammar from Example 2.4.

□

Figure 1 shows the derivation of a parse tree for the string $a + a$ with the grammar from Example 2.4. The following theorem states that parsing as defined in Definition 2.3 and derivation with Earley's parsing schema in Definition 4.3 are equivalent.

Theorem 4.4 (Correctness) *Parsing schema Earley with trees derives exactly the trees produced by a parser, i.e.,* $\{t \mid A \in V_N, w \vdash_{\mathcal{G}}^{4.3} [\alpha\bullet \rightarrow A,0,n] \Rightarrow t\} = \Pi(\mathcal{G})(w)$

The following proposition states that the decoration of items with partial parse trees makes no difference to what can be derived. Items in a parsing schema can be annotated with trees as long as they do not affect the deduction.

Proposition 4.5 *Parsing schema Earley with trees preserves the derivations of parsing schema Earley, i.e.,* $w \vdash_{\mathcal{G}}^{4.1} [\alpha \bullet \beta \rightarrow A,i,j] \iff \exists t_\alpha \in \mathcal{T}(\mathcal{G})(\alpha) : w \vdash_{\mathcal{G}}^{4.3} [\alpha \bullet \beta \rightarrow A,i,j] \Rightarrow [t_\alpha \rightarrow A]$

The optimization problem can now be rephrased as:

Definition 4.6 (Optimizing Parsing Schemata) The optimization of a parsing schema P by a disambiguation filter $\mathcal{F}$ constitutes in finding a derived parsing schema P' such that

$$\mathcal{F}(\Pi(\mathcal{G})(w)) \subseteq \{t \mid w \vdash_{\mathcal{G}}^{P'} I \Rightarrow t\} \subseteq \{t \mid w \vdash_{\mathcal{G}}^{P} I \Rightarrow t\}$$

where I is some final item. □

5 Priority Conflicts

We consider the optimization of parsing schema Earley by two disambiguation filters that are used to interpret the priority disambiguation rules of the formalism SDF of Heering *et al.* (1989). This disambiguation method is also used in the generalization of SDF to SDF2 presented in Visser (1997a). The subject of this section is a filter that removes trees with a priority conflict. This filter is similar to the conventional precedence and associativity filter. The declaration of priority rules will also be used in the definition of the multi-set filter in §7.

Definition 5.1 (Priority Declaration) A *priority declaration* $\mathrm{Pr}(\mathcal{G})$ for a context-free grammar $\mathcal{G}$ is a tuple $\langle \mathrm{L}, \mathrm{R}, \mathrm{N}, > \rangle$, where $\oplus \subseteq \mathcal{P} \times \mathcal{P}$ for $\oplus \in \{\mathrm{L}, \mathrm{R}, \mathrm{N}, >\}$, such that L, R and N are symmetric and $>$ is irreflexive and transitive. □

The relations L, R and N declare left-, right- and non-associativity, respectively, between productions. The relation $>$ declares priority between productions. A tree with signature p_1 can not be a child of a tree with signature p_2 if $p_2 > p_1$. The syntax of priority declarations used here is similar to that in Earley (1975). In SDF (Heering *et al.*, 1989) a formalism with the same underlying structure but with a less Spartan and more concise syntax is used. In SDF one writes `left` for L, `right` for R and `non-assoc` for N. We will use both notations.

Definition 5.2 (Priority Conflict) The set conflicts($\mathcal{G}$) generated by the priority declaration of a grammar $\mathcal{G}$ is the smallest set of partial trees of the form $[\alpha[\beta \to B]\gamma \to A]$ defined by the following rules.

$$\frac{\alpha B\gamma \to A > \beta \to B \in \Pr(\mathcal{G})}{[\alpha[\beta \to B]\gamma \to A] \in \text{conflicts}(\mathcal{G})}$$

$$\frac{\gamma \neq \epsilon,\ \beta \to B\ (\texttt{right} \cup \texttt{non-assoc})\ B\gamma \to A \in \Pr(\mathcal{G})}{[[\beta \to B]\gamma \to A] \in \text{conflicts}(\mathcal{G})}$$

$$\frac{\alpha \neq \epsilon,\ \beta \to B\ (\texttt{left} \cup \texttt{non-assoc})\ \alpha B \to A \in \Pr(\mathcal{G})}{[\alpha[\beta \to B] \to A] \in \text{conflicts}(\mathcal{G})}$$

This set defines the patterns of trees with a priority conflict. □

Using the definition of priority conflict we can define a filter on sets of parse trees.

Definition 5.3 (Priority Conflict Filter) A tree t has a *root priority conflict* if its root matches one of the tree patterns in conflicts($\mathcal{G}$). A tree t has a *priority conflict*, if t has a subtree s that has a root priority conflict. The filter $\mathcal{F}^{\Pr}$ is now defined by $\mathcal{F}^{\Pr}(\Phi) = \{t \in \Phi \mid t \text{ has no priority conflict}\}$. The pair $\langle \mathcal{G}, \Pr \rangle$ defines the disambiguated grammar $\mathcal{G}/\mathcal{F}^{\Pr}$. □

Example 5.4 Consider the following grammar with priority declaration

```
syntax
  "a"      -> E
  E "*" E -> E {left}
  E "+" E -> E {left}
priorities
  E "*" E -> E >
  E "+" E -> E
```

Here the attribute `left` of a production p abbreviates the declaration p L p. The tree

$$[[[a \to E] + [a \to E] \to E] * [a \to E] \to E]$$

has a priority conflict over this grammar—it violates the first priority condition since multiplication has higher priority than addition. The tree

$$[[a \to E] + [[a \to E] * [a \to E] \to E] \to E]$$

does not have a conflict. These trees correspond to the (disambiguated) strings $(a + a) * a$ and $a + (a * a)$, respectively. The implication operator in logic is an example of a right associative operator: $a \to a \to a$ should be read as $a \to (a \to a)$. Non-associativity can be used to exclude unbracketed nested use of the equality operator in expressions using the production `E "=" E -> E`. □

The priority conflict filter induced by a priority declaration can be used to optimize the Earley parsing schema. By the following observation a more general optimization problem can be solved.

Definition 5.5 (Subtree Exclusion) A *subtree exclusion filter* based on a set Q of excluded subtrees is defined by

$$\mathcal{F}^{Q}(\Phi) = \{t \in \Phi \mid \neg t \lhd Q\}$$

where $t \lhd Q$ (t is excluded by Q) if t has a subtree that matches one of the patterns in Q. □

The optimized parsing schema should not derive trees that contain a subtree contained in Q. As is shown in definition 4.3 such patterns are constructed in the complete rule and predicted in the predict rule. The construction of trees with priority conflicts can be prevented by adding an extra condition to these rules. This leads to the following adaptation of the Earley parsing schema.

Definition 5.6 (Earley modulo Q) Parsing schema Earley modulo a set Q of parse trees of the form $[\alpha[\gamma \to B]\beta \to A]$, which are excluded as subtrees. The set of items $\mathcal{I}$ and the deduction rules (H), (I) and (S) are copied unchanged from Definition 4.3.

$$\frac{[\alpha \bullet B\beta \to A, h, i] \Rightarrow [t_\alpha \to A],\ [\alpha[\gamma \to B]\beta \to A] \notin Q}{[\bullet\gamma \to B, i, i] \Rightarrow [\to B]} \quad \text{(P)}$$

$$\frac{[\alpha \bullet B\beta \to A, h, i] \Rightarrow [t_\alpha \to A],\ [\gamma\bullet \to B, i, j] \Rightarrow t_B,\ [\alpha[\gamma \to B]\beta \to A] \notin Q}{[\alpha B \bullet \beta \to A, h, j] \Rightarrow [t_\alpha t_B \to A]} \quad \text{(C)}$$

□

The following theorem states that parsing schema in Definition 5.6 is an optimal approximation of the composition of a subtree exclusion filter (with trees of the form $[\alpha[\gamma \to B]\beta \to A]$) and a generalized parser.

Theorem 5.7 (Correctness) *Parsing schema Earley modulo Q derives exactly the trees produced by the composition of a parser and a subtree exclusion filter for Q, i.e.,* $\{t \in \mathcal{T}(\mathcal{G})(A) \mid A \in V_N,\ w \vdash^{5.6}_{\mathcal{G},Q} [A \to \alpha\bullet, 0, n] \Rightarrow t\} = \mathcal{F}^Q(\Pi(\mathcal{G})(w))$

This is proved using two lemmas. The soundness lemma asserts that no intermediate parse tree derived with the deduction rules has an excluded subtree (i.e., a priority conflict). The completeness lemma states that every parse tree without a priority conflict can be derived. The completeness lemma is obtained by reverting the implication of the soundness lemma. In these lemmata we use the notion of a context $t_B[\bullet]$ that represents a tree context of type B with one subtree that is a hole $\bullet$. The instantiation $t_B[t_A]$ of a context $t_B[\bullet]$ is the tree obtained by replacing the $\bullet$ subtree by the tree t_A.

Lemma 5.8 (Soundness) *For all context-free grammars $\mathcal{G}$, strings $w = a_1 \dots a_n \in V_T^*$, symbols $A \in V_N$ and $\alpha, \beta \in V^*$, natural numbers $i \leq j \in \mathbf{N}$, and trees $t_\alpha \in \mathcal{T}(\mathcal{G})(\alpha)$ such that $\alpha\beta \to A \in \mathrm{P}(\mathcal{G})$, and Q a set of parse tree patterns of the form $[\alpha[\gamma \to B]\beta \to A]$ we have that*

$$\frac{w \vdash^{5.6}_{\mathcal{G}} [\alpha \bullet \beta \to A, i, j] \Rightarrow [t_\alpha \to A]}{\begin{array}{l}\mathrm{yield}(t_A = [t_\alpha\beta \to A]) = a_{i+1} \dots a_j\beta, \\ \exists t_B[\bullet] \in \mathcal{T}(\mathcal{G})(B) : \neg t_B[t_A] \lhd Q \wedge \mathrm{yield}(t_B[A]) = a_1 \dots a_i A\delta\end{array}}$$

6 From Earley to LR

There is a close correspondence between Earley's algorithm and LR parsing (Knuth, 1965). In fact, parsing schema Earley in Definition 4.1 can also be considered the underlying parsing schema of an LR(0) parser. The main difference between the algorithms is that in LR parsing the instantiation of the parsing schema with a grammar is compiled into a transition table. Definition 6.1 defines a parsing schema for 'compiled' LR(0) parsing. The intermediate results of an LR parser, the LR states, are sets of LR items closed under prediction, defined by the function *closure*. The function *goto* computes the set of items that results from a state by shifting the dot in the items over a symbol X. The schema defines three deduction rules. Rule (I) generates the initial state consisting of the set of all items $[\bullet\alpha \to A]$ predicting all productions of the grammar. Rule (Sh) obtains a new state from a state by *shifting* a terminal. Rule (Re) reduces a number of states to a new state upon the complete recognition of a production $B_1 \dots B_m \to B$. It is clear that the function *closure* corresponds to the predict rule (P) in Earley, that (Sh) corresponds to (S) and that (Re) corresponds to (C). A goto-graph is a precomputation of the goto function. Figure 2 shows a goto-graph for the grammar of Example 2.4.

Definition 6.1 (LR(0) Parsing) LR items are Earley items without indices. The items used in LR parsing are sets of LR-items with a pair of indices.

$$\mathcal{I}_{LR} = \{[\alpha \bullet \beta \to A] \mid \alpha\beta \to A \in \mathrm{P}(\mathcal{G})\} \quad \mathcal{I} = \{[\Phi, i, j] \mid \Phi \subseteq \mathcal{I}_{LR}\}$$

The closure of a set of items Φ is the smallest set of items containing Φ and closed under prediction, i.e.,

$$\frac{}{\Phi \subseteq \mathrm{closure}(\Phi)}$$

$$\frac{[\alpha \bullet B\beta \to A],\ \gamma \to B \in \mathrm{P}(\mathcal{G})}{[\bullet\gamma \to B] \in \mathrm{closure}(\Phi)}$$

Given a symbol X the goto function maps a set of items to the closure of the set obtained by shifting all items with X.

$$\mathrm{goto}(X, \Phi) = \mathrm{closure}(\{[\alpha X \bullet \beta \to A] \mid [\alpha \bullet X\beta \to A] \in \Phi\})$$

Given these functions an LR parser is defined[1] by the following deduction rules.

$$\frac{}{[\{[\bullet\alpha \to A] \mid \alpha \to A \in \mathcal{G}\}, 0, 0]} \quad \text{(I)}$$

$$\frac{[\Phi, h, i],\ [a, i, i+1]}{[\mathrm{goto}(a, \Phi), h, i+1]} \quad \text{(Sh)}$$

$$\frac{[\Phi^{[\alpha \bullet B\beta \to A]}, h, i],\ [\Phi_1^{[B_1 \bullet \ldots B_k \to B]}, i, i_1],\ \ldots\ [\Phi_k^{[B_1 \ldots B_k \bullet \to B]}, i, i_k]}{[\mathrm{goto}(B, \Phi), h, i_k]} \quad \text{(Re)}$$

□

In the same way that an LR parser is derived from the Earley schema an LR parser can be derived from the optimized parsing schema of Definition 5.6 by adapting the closure and goto functions.

Definition 6.2 (LR(0) parsing modulo Q) Items are only predicted if they do not lead to a conflict.

$$\frac{[\alpha \bullet B\beta \to A],\ \gamma \to B \in \mathrm{P}(\mathcal{G}),\ [\alpha[\gamma \to B]\beta \to A] \notin Q}{[\bullet\gamma \to B] \in \mathrm{closure}(\Phi)}$$

Given a production $\gamma \to B$ the goto function maps a set of items to the closure of the set obtained by shifting all items with $\gamma \to B$ for which that does not lead to a conflict.

$$\mathrm{goto}(\gamma \to B, \Phi) = \mathrm{closure}(\{[\alpha B \bullet \beta \to A] \mid [\alpha \bullet B\beta \to A] \in \Phi \wedge [\alpha[\gamma \to B]\beta \to A] \notin Q\})$$

□

Note that the goto function has to be parameterized with the production that is recognized instead of with just the symbol. (For the (Sh) rule the old goto function is used.) Figure 3 shows the goto-graph for the disambiguated grammar from Example 5.4.

6.1 SLR(1) Parsing

The LR(0) goto graph in Figure 3 contains conflicts that are easy to prevent with the SLR(1) (Simple LR(1)) extension of LR(0) parsing due to DeRemer (1971). The SLR algorithm is based on the observation that a reduction is only useful if the next symbol in the string can follow the symbol that is recognized by the reduction, i.e., the right hand-side of the production that is reduced. This is expressed in the following adaptation of the LR(0) parsing schema of Definition 6.1. The function $\mathrm{first}(\alpha, \Psi)$ yields the set of symbols that can start a phrase derived from a string of symbols α followed by a symbol from the set Ψ. The expression $\mathrm{follow}(B, \Psi)$ denotes the set of symbols that can follow symbol B in a phrase that is followed by a symbol from the set Ψ. The reduce rule now only applies if a production has been recognized *and* the next symbol in the string can follow the right-hand side of the production.

[1] This definition gives the intermediate results of an LR parser, not its exact control flow.

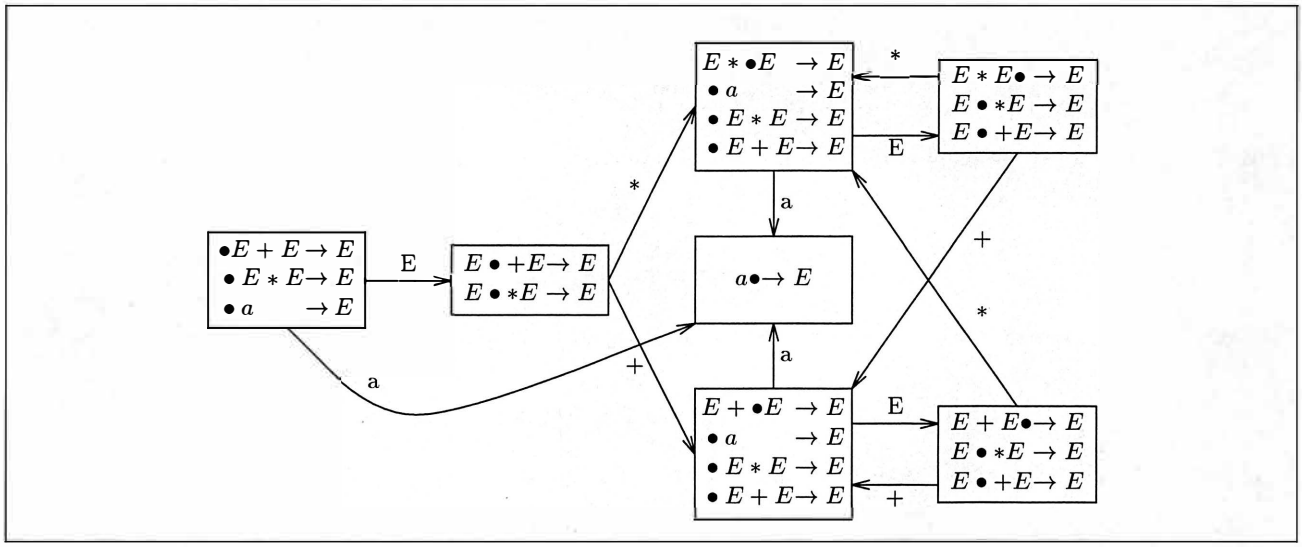

Figure 2: LR(0) goto graph for the grammar of Example 2.4

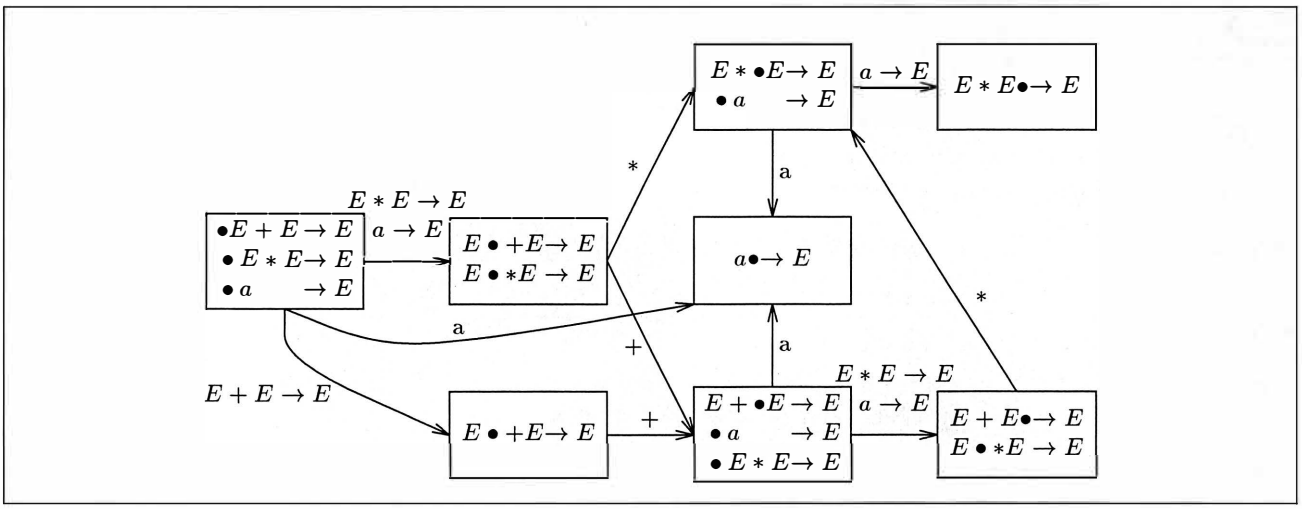

Figure 3: LR(0) goto graph for the grammar of Example 5.4.

Definition 6.3 (SLR(1) Parsing) The set of first symbols of a phrase generated by a string α followed by an element from Ψ, is the smallest set first(α, Ψ) such that

$$\begin{aligned} \text{first}(\epsilon, \Psi) &= \Psi \\ \text{first}(a\alpha, \Psi) &= \{a\} \\ \text{first}(A\alpha, \Psi) &= \bigcup_{\beta \to A \in \mathrm{P}(\mathcal{G})} \text{first}(\beta\alpha, \Psi) \end{aligned}$$

The set of symbols that can follow a symbol B in a phrase generated by $\mathcal{G}$ followed by a symbol from Ψ is the smallest set such that

$$\frac{\alpha B \beta \to A \in \mathrm{P}(\mathcal{G})}{\text{follow}(B, \Psi) \supseteq \text{first}(\beta, \text{follow}(A, \Psi))}$$

state	a	$*$	$+$	\$	1	2	3	4
0	s 1				3	3	4	2
1		r 1	r 1	r 1				
3		s 8	s 5	acc				
4			s 5	acc				
5	s 1				7	7		
7		s 8	r 3	r 3				
8	s 1				9			
9		r 2	r 2	r 2				

(1) $a \rightarrow E$
(2) $E * E \rightarrow E$
(3) $E + E \rightarrow E$
(4) $E\,\$ \rightarrow S$

(2) > (3)
(2) L (2)
(3) L (3)

Figure 4: SLR(1) table for the grammar of example 5.4. s n denotes *shift to state* n, r n denotes reduce with production n, acc denotes *accept*. The right part of the table contains the goto entries for the productions. This parse table corresponds to the goto graph of figure 3.

The reduce rule of the schema in Definition 6.1 is restricted by requiring that the next symbol in the string is an element of the follow set of B.

$$\frac{[\Phi^{[\alpha\bullet B\beta\rightarrow A]}, h, i],\ [\Phi_1^{[B_1\bullet\ldots B_k\rightarrow B]}, i, i_1],\ \ldots,[\Phi_k^{[B_1\ldots B_k\bullet\rightarrow B]}, i, i_k],\ [a, i_k, i_k+1],\ a \in \text{follow}(B, \{\$\})}{[\text{goto}(B, \Phi), h, i_k]}$$

□

The SLR(1) schema can be adapted in the same way as the LR(0) schema to account for priority conflicts (or subtree exclusion). However, the definition of follow above is too weak for this extended schema. For instance, in the grammar of Example 5.4, the token $*$ is an element of the follow set of E. However, $*$ can not follow an E if it is a $E + E \rightarrow E$, i.e., if a reduction is done with $E + E \rightarrow E$, no action for $*$ is possible. The following parsing schema optimizes the SLR(1) parsing schema by defining the follow set for a production instead of for a symbol and adapting the reduce rule accordingly. Figure 4 shows the SLR(1) table for the grammar of Example 5.4.

Definition 6.4 (SLR(1) Parsing Modulo Q) This schema defines SLR(1) parsing modulo a set Q of parse trees of the form $[\alpha[\beta \rightarrow B]\gamma \rightarrow A]$ using the definition of the closure and goto functions from the parsing schema in Definition 6.2 and the definition of first from Definition 6.3.

$$\frac{\alpha B\gamma \rightarrow A \in \text{P}(\mathcal{G}),\ [\alpha[\beta \rightarrow B]\gamma \rightarrow A] \notin Q}{\text{follow}(\beta \rightarrow B, \Psi) \supseteq \text{first}(\beta, \text{follow}(\alpha B\gamma \rightarrow A, \Psi))}$$

The reduce rule is adapted to the new definition of follow.

$$\frac{[\Phi^{[\alpha\bullet B\beta\rightarrow A]}, h, i],\ [\Phi_1^{[B_1\bullet\ldots B_k\rightarrow B]}, i, i_1],\ \ldots,[\Phi_k^{[B_1\ldots B_k\bullet\rightarrow B]}, i, i_k],\ [a, i_k, i_k+1],\ a \in \text{follow}(B_1\ldots B_k \rightarrow B, \{\$\})}{[\text{goto}(B_1\ldots B_k \rightarrow B, \Phi), h, i_k]} \quad \text{(Re)}$$

□

6.2 Discussion

Conventional methods for disambiguating grammars that apply to LR parsing disambiguate the grammar by solving conflicts in an existing LR table. The classical method of Aho *et al.* (1975) uses associativity and precedence information of a limited form—a linear chain of binary operators that have non-overlapping operator syntax—to solve shift/reduce conflicts in LR tables. The method is based on observations on how such conflicts should be solved given precedence information, without a real understanding of the cause of the conflicts. Aasa (1991, 1992) describes filtering of sets of parse trees by means of precedences. Thorup (1994a) describes a

method that tries to find a consistent solution for all conflicts in an LR table starting from, and producing a set of excluded subtrees.

All these methods fail on grammars that are inherently non-LR(k), i.e., for which there is no complete solution of all conflicts in any LR table for the grammar. An example is the grammar

```
syntax
                -> L
  L [\ \t\n]    -> L
  "a"           -> E
  E L "*" L E -> E {left}
  E L "+" L E -> E {left}
priorities
  E L "*" L E -> E >
  E L "+" L E -> E
```

that models arithmetic expressions with layout. The tokens of expressions can be separated by any number of spaces, tabs or newlines, which requires unbounded lookahead. Such grammars are the result of integrating the lexical syntax and context-free syntax of a language into a single grammar as is proposed in Visser (1997b). Parsers for such grammars are called scannerless parsers because the tokens they read are the characters from the input file. This grammar is disambiguated completely (it has no ambiguous sentences) with priorities, resulting in an LR table that contains some LR-conflicts, but that does not produce trees with priority conflicts. In combination with a nondeterministic interpreter, e.g., Tomita's generalized LR algorithm (Tomita, 1985), of the parse tables this gives an efficient disambiguation method for languages on the border of determinism.

Thorup (1994b) describes a transformation on grammars based on a set of excluded subtrees to disambiguate a grammar. This method could be used to generate conflict free parse tables as far as possible. Because such a transformation introduces new grammar symbols, more states and transitions are needed in the parse table than for the original grammar. Since the method defined above also introduces some extra states, it would be interesting to compare the LR tables produced by both methods.

7 Multi-set Filter

The multi-set ordering on parse trees induced by a priority declaration solves ambiguities not solvable by priority conflicts. A certain class of ambiguities solved by the multi-set order does not need the full power of multi-sets, only a small part of both trees are actually compared. Based on this observation an optimization of the Earley schema that partially implements the multi-set filters can be defined.

Definition 7.1 (Multi-sets) A *multi-set* is a function $M : \mathrm{P}(\mathcal{G}) \to \mathbf{N}$ that maps productions to the number of their occurrences in the set. The union $M \uplus N$ of two multi-sets M and N is defined as $(M \uplus N)(p) = M(p) + N(p)$. The empty multi-set is denoted by $\emptyset$, i.e., $\emptyset(p) = 0$ for any p. We write $p \in M$ for $M(p) > 0$. A multi-set with a finite number of elements with a finite number of occurrences can be written as $M = \{p_1, p_1, \ldots, p_2, \ldots\}$, where $M(p)$ is the number of occurrences of p in the list. A parse tree t is interpreted as a multi-set of productions by counting the number of times a production acts as the signature of a subtree of t, where $\alpha \to A$ is the signature of $[t_\alpha \to A]$. □

The following definition due to Jouannaud and Lescanne (1982) defines an ordering on multi-sets.

Definition 7.2 (Multi-set Order) Given some priority declaration $\mathrm{Pr}(\mathcal{G})$, the order $\prec^{\mathrm{Pr}(\mathcal{G})}$ on multi-sets is defined as

$$M \prec^{\mathrm{Pr}(\mathcal{G})} N \iff$$
$$M \neq N \wedge \forall y \in M : M(y) > N(y) \Rightarrow \exists x \in N : y >^{\mathrm{Pr}(\mathcal{G})} x \wedge M(x) < N(x)$$

□

Definition 7.3 (Multi-set Filter) Given a priority relation $\mathrm{Pr}(\mathcal{G})$, the multi-set filter $\mathcal{F}^{\prec^{\mathrm{Pr}(\mathcal{G})}}$ is defined by

$$\mathcal{F}^{\prec^{\mathrm{Pr}(\mathcal{G})}}(\Phi) = \{t \in \Phi \mid \neg\exists s \in \Phi : s \prec^{\mathrm{Pr}(\mathcal{G})} t\}$$ □

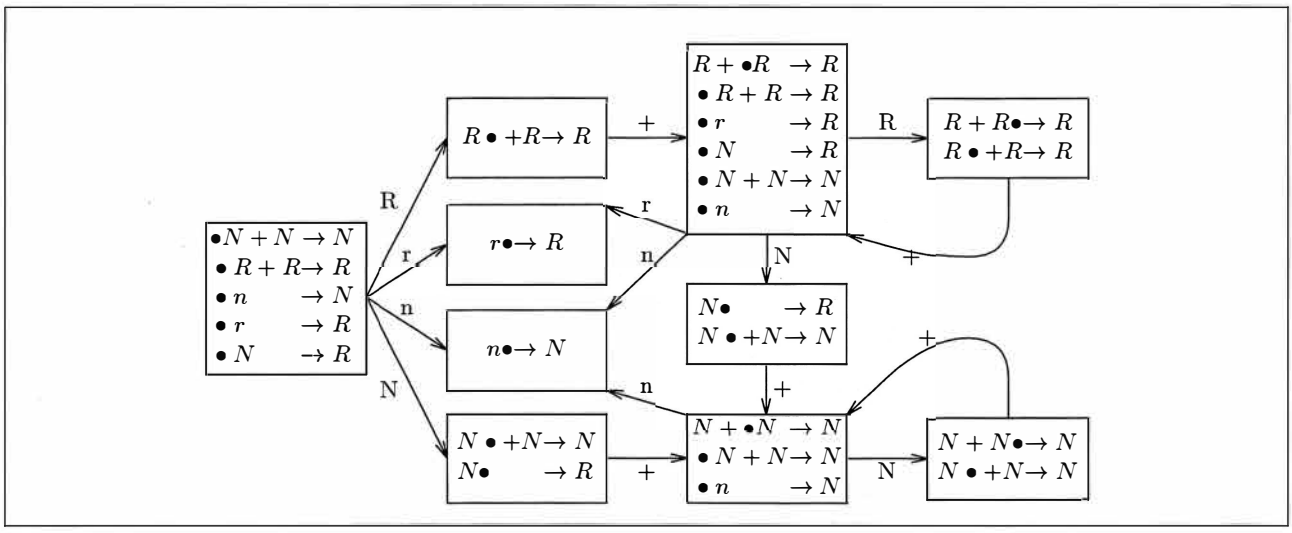

Figure 5: Goto graph for the grammar of Example 7.4

The motivation for this filter is that it prefers parse trees that are constructed with the smallest possible number of productions of the highest possible priority.

Example 7.4 Consider the grammar

```
syntax
  "n"      -> N
  N "+" N -> N
  N        -> R
  "r"      -> R
  R "+" R -> R
```

that describes the language of 'naturals' and 'reals' with an overloaded addition operator. The sentence $n + n$ can be parsed as $[[n \to N] + [n \to N] \to N]$ and as $[[[n \to N] \to R] + [[n \to N] \to R] \to R]$. This ambiguity can be solved, choosing either the first or the second tree, by declaring one of the priority rules

```
N "+" N -> N > R "+" R -> R
```

or

```
R "+" R -> R > N "+" N -> N
```

Note that with the second priority rule, the production $N + N \to N$ is only used as a parse tree in a context where no R is allowed. Therefore, the first priority rule is assumed in further examples. □

The multi-set order is too strong for this kind of disambiguation. To solve the ambiguity there is no need to compare the complete trees, as the multi-set order does. Comparing the patterns $[[N + N \to N] \to R]$ and $[[N \to R] + [N \to R] \to R]$ is sufficient. The goto graph corresponding to the Earley parser for the example grammar (Figure 5) shows that the partial phrase $n+$ causes a conflict (in the left-most state at the bottom row) after completing the production $[n \to N]$. The parser can either shift with + or complete with the chain rule of $N \to R$. However, only after having seen what follows the + a decision can be made. In the following adaptation of the Earley parsing schema the cause of these early decision problems is solved by not predicting and completing chain production, but instead storing them in items.

Definition 7.5 (Earley modulo Chain Rules) Let V_C be the set containing all chain symbols $[C \to B]$, where B and C are nonterminals in context-free grammar $\mathcal{G}$ and $B = C$ or $C \to_\mathcal{G} B_1 \to_\mathcal{G} \cdots \to_\mathcal{G} B_m \to_\mathcal{G} B$, $(m \geq 0)$. Symbols $[A \to A]$ and A are identified. A production with chain symbols $[C \to B]$ in its left-hand side is identifiable (member of grammar, priority relation) with a production where the chain symbols are replaced

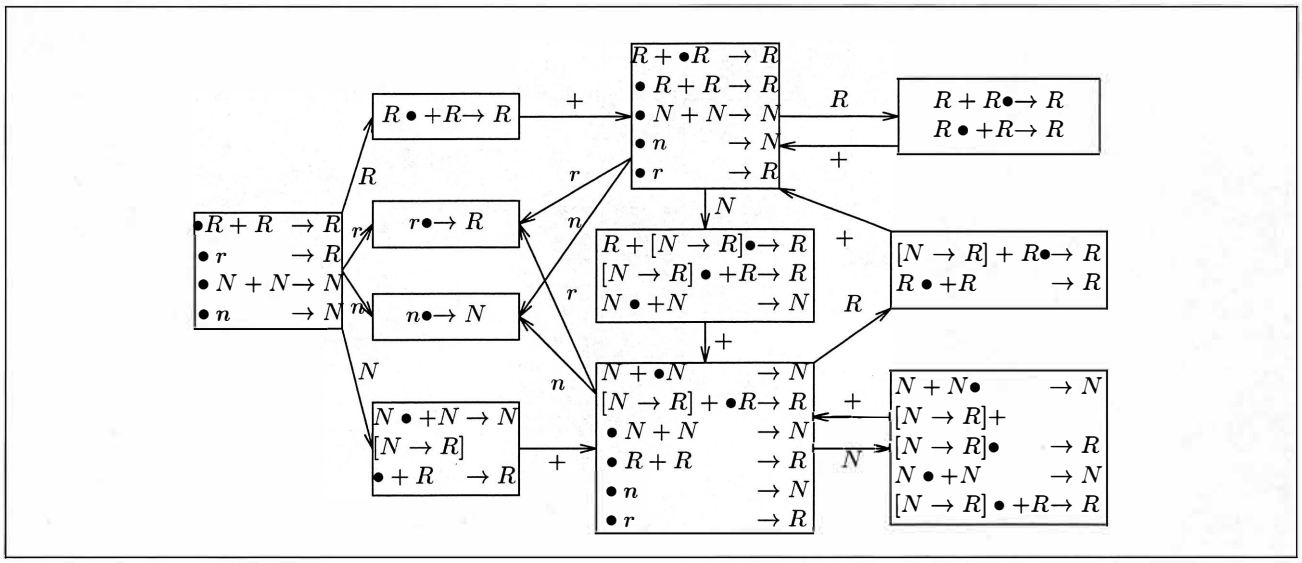

Figure 6: Goto graph for grammar of Example 7.4 corresponding to parsing schema in Definition 7.5. The item $[[N \to R] + [N \to R]\bullet \to R]$ is present if the negative premise of rule (C2) is absent.

with their heads B. The (I) and (S) rules are as usual.

$$\frac{\alpha\beta \to A \in \mathcal{G},\ |\alpha\beta| \neq 1 \vee \alpha\beta = a \in V_T,\ 0 \leq i \leq j}{[\alpha \bullet \beta \to A, i, j] \in \mathcal{I}}$$

$$\frac{[\alpha \bullet B\beta \to A, h, i],\ [C \to B] \in V_C}{[\bullet\gamma \to C, i, i]} \tag{P}$$

$$\frac{[\alpha \bullet B\beta \to A, h, i],\ [\gamma\bullet \to C, i, j],\ |\beta| > 0}{[\alpha[C \to B] \bullet \beta \to A, h, j]} \tag{C1}$$

$$\frac{[\alpha \bullet B \to A, h, i], [\gamma\bullet \to C, i, j], \neg[\alpha'\bullet \to A', h, j],\ \alpha' \to A' > \alpha B \to A}{[\alpha[C \to B]\bullet \to A, h, j]} \tag{C2}$$

□

The negative premise $\neg[\alpha\bullet \to A, i, j]$ in combination with the condition $A' \to \alpha' > A \to \alpha B$ is used in rule (C2) to express that an item $[\alpha[C \to B]\bullet \to A, h, j]$ can be derived from $[\alpha \bullet B \to A, h, i]$ and $[\gamma\bullet \to C, i, j]$ only if no item $[\alpha'\bullet \to A', h, j]$ can be derived such that $A' \to \alpha'$ has higher priority than $A \to \alpha B$.

With the introduction of negative premises we leave the domain of parsing schemata as defined in Sikkel (1993) and this deserves a more thorough investigation than is possible in the scope of this paper. However, two points about this feature can be observed: (1) As used here the notion has a straightforward implementation in an LR-like compilation scheme: first construct the complete set of items and then choose the maximal items from it. (2) The priority relation $>$ on productions is irreflexive by definition, which entails that rule (C2) has no instantiation of the form $I_1, I_2, \neg I_3 \vdash I_3$ that would make the schema inconsistent.

Example 7.6 Figure 6 shows the goto graph for the grammar of Example 7.4 according to the parsing schema in Definition 7.5. The shift/reduce conflict between the items $[N \bullet +N \to N]$ and $[N\bullet \to R]$ is changed into a reduce/reduce conflict between the items $[N + N\bullet \to N]$ and $[[N \to R] + [N \to R]\bullet \to R]$. If the negative premise of rule (C2) is taken into account the item $[[N \to R] + [N \to R]\bullet \to R]$ can not be derived, and is not present in the goto-graph. The conflict is solved. □

The method does not help for grammars where the ambiguity is not caused by chain rules, for instance consider the following example due to Kamperman (1992)

```
syntax
  E E     -> E
```

```
    "-" E   -> E
    E "-" E -> E
  priorities
    E E     -> E >
    "-" E   -> E >
    E "-" E -> E
```

It defines expressions formed by concatenation, prefix minus and infix minus.

The methods developed in this paper can be combined into a parsing schema that handles both priority conflicts and the partial implementation of multi-set filters by adding the subset exclusion conditions to the (P), (C1) and (C2) rules of the parsing schema in Definition 7.5. As a bonus this combined parsing schema handles priority conflicts modulo chain rules.

8 Conclusions

In this paper two disambiguation methods specified as a filter on sets of parse trees were considered. These filters were used to optimize parsers for context-free grammars by adapting their underlying parsing schema.

The first optimization uses priority conflicts to prevent ambiguities. The resulting Earley parsers modulo priority conflicts are guaranteed not to produce trees with priority conflicts, even for grammars with overlapping operators, layout in productions or other problems that need unbounded lookahead. In combination with a GLR interpreter of the parse tables this gives an efficient disambiguation method for languages with unbounded lookahead. The second optimization covers a subset of the ambiguities solved by multi-set filters. Together these optimizations can be used in the generation of efficient parsers for a large class of ambiguous context-free grammars disambiguated by means of priorities.

Parsing schemata provide a high-level description of parsing algorithms that is suitable for the derivation of new algorithms. The introduction of negative items was needed to express the optimization for the multiset filter and needs more research. This first experiment in implemenation of disambiguation methods from formal specifications encourages research into a fuller optimization of multiset filters and application of this approach to other disambiguation methods.

The deductive parsing approach of Shieber *et al.* (1995) and its implementation in Prolog could be used to prototype such optimized schemata. Deductive parsing consists in computing the closure of a set of axiom items under the inference rules of a schema, resulting in all items derivable for a sentence. Compiling the inference rule of a schema into a parse table for a specific grammar increases the efficiency of an algorithm, since work is shifted from the parser into the parser generator. It seems feasible to generalize the compilation of Earley rules into LR tables to other schemata, thus obtaining a very declarative method for creating new parser generators.

Acknowledgements I thank Mark van den Brand and several referees for their comments on this paper. This research was supported by the Netherlands Computer Science Research Foundation (SION) with financial support from the Netherlands Organisation for Scientific Research (NWO). Project 612-317-420: Incremental parser generation and context-dependent disambiguation, a multi-disciplinary perspective.

References

Aasa, A. (1991). Precedences in specifications and implementations of programming languages. In J. Maluzynski and M. Wirsing, editors, *Programming Language Implementation and Logic Programming*, volume 528 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag.

Aasa, A. (1992). *User Defined Syntax*. Ph.D. thesis, Department of Computer Sciences, Chalmers University of Technology and University of Göteborg, S-412 96 Göteborg, Sweden.

Aho, A. V., Johnson, S. C., and Ullman, J. D. (1975). Deterministic parsing of ambiguous grammars. *Communications of the ACM*, **18**(8), 441–452.

DeRemer, F. L. (1971). Simple LR(k) grammars. *Communications of the ACM*, **14**, 453–460.

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

Earley, J. (1975). Ambiguity and precedence in syntax description. *Acta Informatica*, **4**(1), 183–192.

Heering, J., Hendriks, P. R. H., Klint, P., and Rekers, J. (1989). The syntax definition formalism SDF – reference manual. *SIGPLAN Notices*, **24**(11), 43–75.

Jouannaud, J.-P. and Lescanne, P. (1982). On multiset orderings. *Information Processing Letters*, **15**(2), 57–63.

Kamperman, J. (1992). A try at improving the second disambiguation phase in SDF. technical note.

Klint, P. and Visser, E. (1994). Using filters for the disambiguation of context-free grammars. In G. Pighizzini and P. San Pietro, editors, *Proc. ASMICS Workshop on Parsing Theory*, pages 1–20, Milano, Italy. Tech. Rep. 126–1994, Dipartimento di Scienze dell'Informazione, Università di Milano.

Knuth, D. E. (1965). On the translation of languages from left to right. *Information and Control*, **8**, 607–639.

Rekers, J. (1992). *Parser Generation for Interactive Environments*. Ph.D. thesis, University of Amsterdam. ftp://ftp.cwi.nl/pub/gipe/reports/Rek92.ps.Z.

Shieber, S. M., Schabes, Y., and Pereira, F. C. N. (1995). Principles and implementation of deductive parsing. *The Journal of Logic Programming*, pages 3–36.

Sikkel, K. (1993). *Parsing Schemata*. Ph.D. thesis, Universiteit Twente, Enschede.

Sikkel, K. (1994). How to compare the structure of parsing algorithms. In G. Pighizzini and P. San Pietro, editors, *Proc. ASMICS Workshop on Parsing Theory*, pages 21–39, Milano, Italy. Tech. Rep. 126–1994, Dipartimento di Scienze dell'Informazione, Università di Milano.

Sikkel, K. (1997). *Parsing Schemata. A Framework for Specification and Analysis of Parsing Algorithms*, volume XVI of *Texts in Theoretical Computer Science. An EATCS Series*. Springer-Verlag, Berlin / Heidelberg / New York.

Thorup, M. (1994a). Controlled grammatic ambiguity. *ACM Transactions on Programming Languages and Systems*, **16**(3), 1024–1050.

Thorup, M. (1994b). Disambiguating grammars by exclusion of sub-parse trees. Technical Report 94/11, Dept. of Computer Science, University of Copenhagen, Denmark.

Tomita, M. (1985). *Efficient Parsing for Natural Languages. A Fast Algorithm for Practical Systems*. Kluwer Academic Publishers.

Visser, E. (1997a). A family of syntax definition formalisms. Technical Report P9706, Programming Research Group, University of Amsterdam.

Visser, E. (1997b). Scannerless generalized-LR parsing. Technical Report P9707, Programming Research Group, University of Amsterdam.