# A Case Study in Optimizing Parsing Schemata by Disambiguation Filters

Eelco Visser

**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 modulo 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, the grammar

$$E \to E + E; E \to E * E; E \to a$$

gives a clearer description of arithmetic expressions than the grammar

$$E \to E + T; E \to T; T \to T * a; T \to a$$

To obtain an unambiguous specification of a language described by an ambiguous grammar it has to be disambiguated. For example, the grammar above can be disambiguated by associativity and priority rules that express that $E \to E * E$ has higher priority than $E \to E + E$ and that both productions are left associative.

Technical Report P9507, Programming Research Group, University of Amsterdam, July 1995.
Available as: ftp://ftp.fwi.uva.nl/pub/programming-research/reports/1995/P9507.ps.Z
This is an extended version of [Vis95].

In [KV94] a framework for specification and comparison of disambiguation methods is set up. 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 [Tom85, Rek92], 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 parsing 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 many 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* [Sik93, Sik94], high-level descriptions of parsing algorithms that abstract from control- and data-structures, 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: The optimization of the underlying parsing schema of Earley's parsing algorithm [Ear70] by a filter for disambiguation by priorities. This method, the disambiguation method of the formalism SDF [HHKR92], 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 multiset 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 multiset order. The schema can easily 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 section 2 some preliminary notions are defined. In section 3 disambiguation filters are defined. In section 4 parsing schemata are informally introduced. In section 5 priority rules and the notion of priority conflict are defined and a parsing schema optimized for the priority conflict filter is derived. In section 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. In section 7 the multiset 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

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

**2.2** DEFINITION. (parse trees) A CFG $\mathcal{G}$ generates a family of sets of *parse trees* $\mathcal{T}^{\mathcal{G}} = (\mathcal{T}^{\mathcal{G}}_\alpha \mid \alpha \in V^*)$, which contains the minimal sets $\mathcal{T}^{\mathcal{G}}_\alpha$ such that

$$
\begin{array}{ll}
X \in \mathcal{T}^{\mathcal{G}}_X & \Leftarrow X \in V \\
[A \to t_\alpha] \in \mathcal{T}^{\mathcal{G}}_A & \Leftarrow A \to \alpha \in \mathcal{P} \wedge t_\alpha \in \mathcal{T}^{\mathcal{G}}_\alpha \\
t_1 \ldots t_n \in \mathcal{T}^{\mathcal{G}}_{X_1 \ldots X_n} & \Leftarrow \bigwedge_{i=1}^n t_i \in \mathcal{T}^{\mathcal{G}}_{X_i}
\end{array}
$$

The *signature* of a tree is the production used to construct the root of a tree: $sign([A \to t_\alpha]) = A \to \alpha$. The *yield* of a tree is the concatenation of its leaves. $\Box$

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

$$\Pi(\mathcal{G})(w) = \{t \in \mathcal{T}^{\mathcal{G}}_S \mid yield(t) = w\}$$ $\Box$

**2.4** EXAMPLE. As an example consider the ambiguous grammar $\mathcal{G} = E \to E + E; E \to E * E; E \to a$ from the introduction. According to this grammar the string $a + a * a$ has two parses:

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

# 3 Disambiguation Filters

**3.1** DEFINITION. (disambiguation filter) A *filter* $\mathcal{F}$ for a CFG $\mathcal{G}$ 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 CFG $\mathcal{G}$ 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

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

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 $\mathcal{G}/(\mathcal{F}_2 \circ \mathcal{F}_1)$. $\Box$

Several properties and examples of filters are discussed in [KV94]. In sections 5 and 7 two examples of disambiguation filters will be introduced. Now the optimization problem for disambiguation filters can be formulated as:

**3.2** DEFINITION. (optimization by filter) Given a CFG $\mathcal{G}$ and a filter $\mathcal{F}$, a parser $\pi$ 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 $\pi$ *approximates* $\mathcal{F} \circ \Pi(\mathcal{G})$. $\pi$ is an optimal approximation if $\pi(w) = \mathcal{F}(\Pi(\mathcal{G})(w))$ for any $w$. □

# 4 Parsing Schemata

Parsing schemata [Sik93, Sik94] 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 described informally by an example. A formal treatment can be found in [Sik93].

Definition 4.1 defines a parsing schema for Earley's parsing algorithm [Ear70]. 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 $D$. 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 $[A \rightarrow \alpha \bullet \beta, i, j]$, where $A \rightarrow \alpha\beta$ is a production of grammar $\mathcal{G}$. The indices refer to positions in the string $a_1 \ldots a_n$. The intention of this definition is that an item $[A \rightarrow \alpha \bullet \beta, i, j]$ can be derived if $\alpha \rightarrow^*_{\mathcal{G}} a_{i+1} \ldots a_j$ and $S \rightarrow^*_{\mathcal{G}} a_1 \ldots a_i A \gamma$. The deduction rules $D^I$ through $D^C$ describe how these items can be derived. Rule $D^I$, the *initialization* rule, specifies that the item $[S \rightarrow \bullet\gamma, 0, 0]$, where $S$ is the start symbol of the grammar, can always be derived. The *predict* rule $D^P$, states that a production $B \rightarrow \gamma$ can be predicted at position $j$, if the item $[A \rightarrow \alpha \bullet B\beta, i, j]$ has already been derived. Finally, the rules $D^S$ and $D^C$ finalize the recognition of a predicted and recognized token or nonterminal—witnessed by the second premise—by shifting the $\bullet$ over the predicted symbol.

**4.1** DEFINITION. (Earley) Parsing schema for Earley's parsing algorithm [Ear70].

$$\begin{array}{lcl}
\mathcal{I} & = & \{[A \rightarrow \alpha \bullet \beta, i, j] \mid A \rightarrow \alpha\beta \in \mathcal{G} \wedge 0 \leq i \leq j\}, \\
D^I & = & \{\vdash [S \rightarrow \bullet\gamma, 0, 0]\}, \\
D^P & = & \{[A \rightarrow \alpha \bullet B\beta, i, j] \vdash [B \rightarrow \bullet\gamma, j, j]\}, \\
D^S & = & \{[A \rightarrow \alpha \bullet a\beta, i, j], [a, j, j+1] \vdash [A \rightarrow \alpha a \bullet \beta, i, j+1]\}, \\
D^C & = & \{[A \rightarrow \alpha \bullet B\beta, h, i], [B \rightarrow \gamma\bullet, i, j] \vdash [A \rightarrow \alpha B \bullet \beta, h, j]\}, \\
D & = & D^I \cup D^P \cup D^S \cup D^C
\end{array}$$

□

A derivation according to a parsing schema is a sequence $I_0, \ldots, I_m$ of items such that for each $i$ $(0 \leq i \leq m)$ $I_i \in H$ or there is a $J \subseteq \{I_0, \ldots, I_{i-1}\}$ such that $J \vdash I_i$ is (the instantiation of) a rule in $D$. A string $w = a_1 \ldots a_n$ is in the language of context-free grammar $\mathcal{G}$ if the item $[S \to \gamma\bullet, 0, n]$ is derivable from the hypotheses corresponding to $w$ in the instantiation of parsing schema 4.1 with $\mathcal{G}$. The expression $w \vdash^P_{\mathcal{G}} I$ denotes: there is a derivation $I_0, \ldots, I_m = I$ 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. The following definition gives an extension of schema 4.1 that describes how trees can be build as a result of the deduction steps. The items in this schema have the form $[A \to \alpha \bullet \beta, i, j] \Rightarrow [A \to t_\alpha]$ and express the fact 'from position $i$ to position $j$ the string $\alpha$ has been recognized and the partial parse tree $[A \to t_\alpha]$ has been built as a result'. Note how the shift and complete rules extend partial parse trees.

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

$$\mathcal{I} = \{[A \to \alpha \bullet \beta, i, j] \Rightarrow [A \to t_\alpha] \mid A \to \alpha\beta \in \mathcal{G} \wedge 0 \leq i \leq j \wedge t_\alpha \in \mathcal{T}^{\mathcal{G}}_\alpha\}$$

$$\frac{a_i}{[a_i, i-1, i] \Rightarrow a_i}(H) \qquad \frac{}{[S \to \bullet\gamma, 0, 0] \Rightarrow [S \to]}(I)$$

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

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

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

**4.3** Example. Figure 1 shows the derivation of a parse tree for the string $a + a$ with the grammar from example 2.4. $\Box$

The following theorem states that parsing as defined in definition 2.3 and derivation with parsing schema 4.2 are equivalent.

**4.4.** Theorem. (correctness) $\{t \in \mathcal{T}^{\mathcal{G}}_S \mid w \vdash^{4.2}_{\mathcal{G}} [S \to \gamma\bullet, 0, n] \Rightarrow t\} = \Pi(\mathcal{G})(w)$

Proof. See Appendix A. $\Box$

The following proposition states that the decoration of items with partial parse trees in parsing schema Earley 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.

| | |
|---|---|
| $[a, 0, 1]$ | $\Rightarrow a$ |
| $[+, 1, 2]$ | $\Rightarrow +$ |
| $[a, 2, 3]$ | $\Rightarrow a$ |
| $[E \rightarrow \bullet E + E, 0, 0]$ | $\Rightarrow [E \rightarrow]$ |
| $[E \rightarrow \bullet a, 0, 0]$ | $\Rightarrow [E \rightarrow]$ |
| $[E \rightarrow a \bullet, 0, 1]$ | $\Rightarrow [E \rightarrow a]$ |
| $[E \rightarrow E \bullet + E, 0, 1]$ | $\Rightarrow [E \rightarrow [E \rightarrow a]]$ |
| $[E \rightarrow E + \bullet E, 0, 2]$ | $\Rightarrow [E \rightarrow [E \rightarrow a] +]$ |
| $[E \rightarrow \bullet a, 2, 2]$ | $\Rightarrow [E \rightarrow]$ |
| $[E \rightarrow a \bullet, 2, 3]$ | $\Rightarrow [E \rightarrow a]$ |
| $[E \rightarrow E + E \bullet, 0, 3]$ | $\Rightarrow [E \rightarrow [E \rightarrow a] + [E \rightarrow a]]$ |

Figure 1: Derivation with parsing schema 4.2 and the grammar from example 2.4.

**4.5.** PROPOSITION.

$$w \vdash_{\mathcal{G}}^{4.1} [A \rightarrow \alpha \bullet \beta, i, j] \iff \exists t_\alpha \in \mathcal{T}_\alpha^{\mathcal{G}} : w \vdash_{\mathcal{G}}^{4.2} [A \rightarrow \alpha \bullet \beta, i, j] \Rightarrow [A \rightarrow t_\alpha] \quad \square$$

The optimization problem can now be rephrased as:

**4.6** DEFINITION. (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 Optimization 1: Priority Conflicts

In this section and the next the optimization of parsing schema Earley by two disambiguation filters that are used to interpret the priority disambiguation rules of the formalism SDF [HHKR92], will be considered. 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 the priority rules will also be used in the definition of the multiset filter in section 7.

**5.1** DEFINITION. (priority declaration) A *priority declaration* $\mathcal{R}$ for a CFG $\mathcal{G}$ 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 [Ear75]. In SDF [HHKR92] a formalism with the same underlying structure but with a less Spartan and more compact syntax is used.

**5.2** DEFINITION. (priority conflict filter) A tree $t$ has a *root priority conflict* $\mathcal{C}^{\mathcal{R}}(t)$ if it violates a right- or non-associativity rule

$$\mathcal{C}^{\mathcal{R}}([A \to [B \to t_\beta]\ s_\alpha]) \ \Leftarrow\ (A \to B\alpha\ \mathrm{R}^{\mathcal{R}}\ B \to \beta) \vee (A \to B\alpha\ \mathrm{N}^{\mathcal{R}}\ B \to \beta) \quad (1)$$

or violates a left- or non-associativity rule

$$\mathcal{C}^{\mathcal{R}}([A \to s_\alpha\ [B \to t_\beta]]) \ \Leftarrow\ (A \to \alpha B\ \mathrm{L}^{\mathcal{R}}\ B \to \beta) \vee (A \to \alpha B\ \mathrm{N}^{\mathcal{R}}\ B \to \beta) \quad (2)$$

or violates a priority rule

$$\mathcal{C}^{\mathcal{R}}([A \to s_\alpha\ [B \to t_\beta]\ s_\gamma] \ \Leftarrow\ A \to \alpha B \gamma >^{\mathcal{R}} B \to \beta \quad (3)$$

A tree $t$ has a *priority conflict*, $\overline{\mathcal{C}}^{\mathcal{R}}(t)$, if $t$ has a subtree $s$ that has a root priority conflict $\mathcal{C}^{\mathcal{R}}(s)$. The filter $\mathcal{F}^{\overline{\mathcal{C}}^{\mathcal{R}}}$ is now defined by $\mathcal{F}^{\overline{\mathcal{C}}^{\mathcal{R}}}(\Phi) = \{t \in \Phi \mid \neg\overline{\mathcal{C}}^{\mathcal{R}}(t)\}$. The pair $\langle \mathcal{G}, \mathcal{R} \rangle$ defines the disambiguated CFG $\mathcal{G}/\mathcal{F}^{\overline{\mathcal{C}}^{\mathcal{R}}}$. □

**5.3** EXAMPLE. Take grammar $\mathcal{G}$ from example 2.4 and priority declaration $\mathcal{R} = E \to E * E\{left\} > E \to E + E\{left\}$ where $p\{left\}$ is a shortcut for $p\,\mathrm{L}\,p$. The tree

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

has a priority conflict in $\langle \mathcal{G}, \mathcal{R} \rangle$—it violates the priority condition (3) because of $E \to E * E > E \to E + E$, but

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

is conflict free. 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 $a \to (a \to a)$ □

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.

**5.4** DEFINITION. (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 patters in $Q$. □

**5.5.** PROPOSITION. *A tree has a root priority conflict if and only if it matches a tree in the set $Q^{\mathcal{R}}$ of* excluded subtrees *that contains all tree patterns $t = [A \to \alpha[B \to \gamma]\beta]$ such that $\mathcal{C}^{\mathcal{R}}(t)$.* □

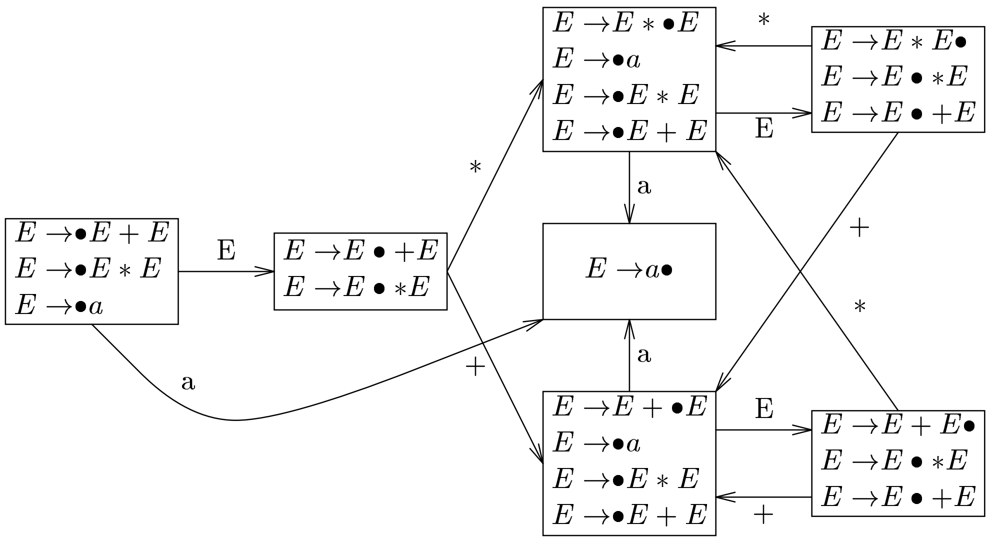


Figure 2: LR(0) goto graph for the grammar of example 2.4

The optimized parsing schema should not derive trees that contain a subtree contained in $Q$. As is shown in definition 4.2 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.

**5.6** DEFINITION. (Earley modulo $Q$) Parsing schema Earley modulo $Q$, where $Q$ is a set of parse trees of the form $[A \to \alpha[B \to \gamma]\beta]$, which are excluded as subtrees. The set of items $\mathcal{I}$ and the deduction rules $(H)$, $(I)$ and $(S)$ are copied unchanged from definition 4.2.

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

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

□

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

**5.7.** THEOREM. $\{t \in \mathcal{T}_S^{\mathcal{G}} \mid w \vdash^{5.6}_{\mathcal{G},Q} [S \to \gamma\bullet, 0, n] \Rightarrow t\} = \mathcal{F}^Q(\Pi(\mathcal{G})(w))$

PROOF. See Appendix A. □

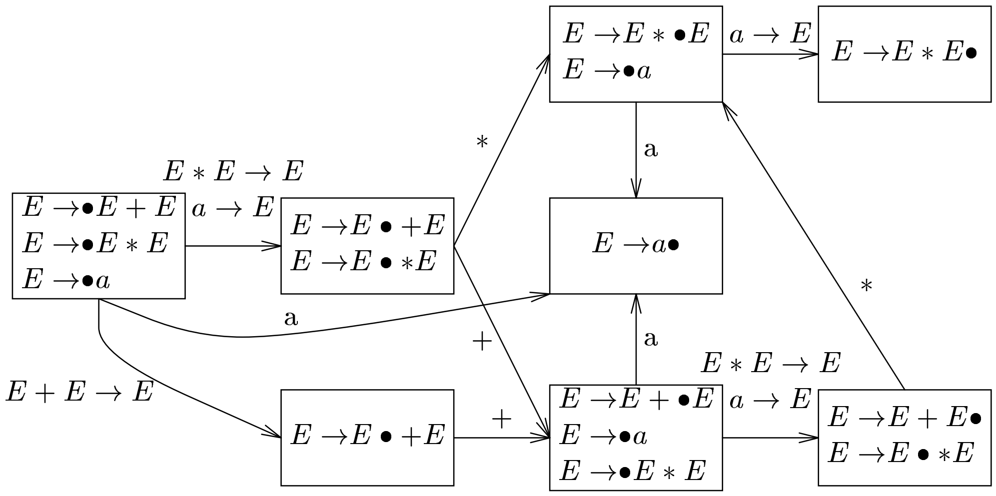


Figure 3: LR(0) goto graph for the grammar of example 5.3.

# 6 Intermezzo: From Earley to LR

There is a close correspondence between Earley's algorithm and LR parsing [Knu65]. In fact, parsing schema 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 two deduction rules. 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 $B \to B_1 \ldots B_m$. 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.

**6.1** Definition. (LR(0) parsing)

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

$$closure(\Phi) = \{I \mid I \in \Phi \vee I = [B \to \bullet\gamma] \wedge [A \to \alpha \bullet B\beta] \in closure(\Phi)\}$$

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

$$[\Phi, h, i], [a, i, i+1] \vdash [goto(a, \Phi), h, i+1] \qquad (Sh)$$

$$\frac{[\Phi^{[A \to \alpha \bullet B\beta]}, h, i], [\Phi_0^{[B \to \bullet B_1 \ldots B_m]}, i, i], \ldots [\Phi_m^{[B \to B_1 \ldots B_m \bullet]}, i, i_m]}{[goto(B, \Phi), h, i_m]} \qquad (Re)$$

$\square$

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.

**6.2** Definition. (LR(0) parsing modulo $Q$)

$$\begin{aligned} closure(\Phi) = \{I \mid I \in \Phi \vee I = [B \to \bullet\gamma] \wedge [A \to \alpha \bullet B\beta] \in closure(\Phi) \\ \wedge [A \to \alpha[B \to \gamma]\beta] \notin Q\} \end{aligned}$$

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

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 grammar from example 5.3 disambiguated with an appropriate priority declaration. In appendix B the extension of LR(0) to SLR(1) and its optimization with the priority conflict filter are described.

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 [AJU75] 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. In a recent paper Thorup [Tho94a] 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.

Both 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

$$L \to \#; L \to; L \to LL; E \to EL + LE; E \to EL * LE; E \to a$$

that models arithmetic expressions with layout; the tokens of expressions can be separated by any number of spaces (#), which requires unbounded lookahead. This grammar can be 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 [Tom85]) of the parse tables this gives an efficient disambiguation method for languages on the border of determinism.

In [Tho94b] Thorup 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 Optimization 2: Multiset Filter

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

**7.1** Definition. (multisets) A *multiset* is a function $M : \mathcal{P} \to \mathbf{N}$ that maps productions to the number of their occurrences in the set. The union $M \uplus N$ of two multisets $M$ and $N$ is defined as $(M \uplus N)(p) = M(p) + N(p)$. The empty multiset is denoted by $\emptyset$, i.e., $\emptyset(p) = 0$ for any $p$. We write $p \in M$ for $M(p) > 0$. A multiset 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 multiset of productions by counting the number of times a production acts as the signature of a subtree of $t$. $\square$

**7.2** Definition. (multiset order [JL82]) Given some priority declaration $\mathcal{R}$, the order $\prec^{\mathcal{R}}$ on multisets is defined as

$$\begin{array}{rcl} M \prec^{\mathcal{R}} N & \iff & M \neq N \\ & \wedge & \forall y \in M : M(y) > N(y) \Rightarrow \exists x \in N : y >^{\mathcal{R}} x \wedge M(x) < N(x) \end{array} \quad \square$$

**7.3** Definition. (multiset filter) Given a priority relation $\mathcal{R}$, the multiset filter $\mathcal{F}^{\prec^{\mathcal{R}}}$ is defined by

$$\mathcal{F}^{\prec^{\mathcal{R}}}(\Phi) = \{t \in \Phi \mid \neg\exists s \in \Phi : s \prec^{\mathcal{R}} t\} \quad \square$$

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.

**7.4** Example. Consider the grammar

$$R \to R + R; N \to N + N; R \to r; N \to n; R \to N$$

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

The multiset order is too strong for this kind of disambiguation. To solve the ambiguity there is no need to compare the complete trees, as the multiset order does; comparing the patterns $[R \to [N \to N + N]]$ and $[R \to [R \to N] + [R \to N]]$ is sufficient. The goto graph corresponding to the Earley parser for the example grammar (Figure 4) shows that the string $n+$ causes a conflict after completing the

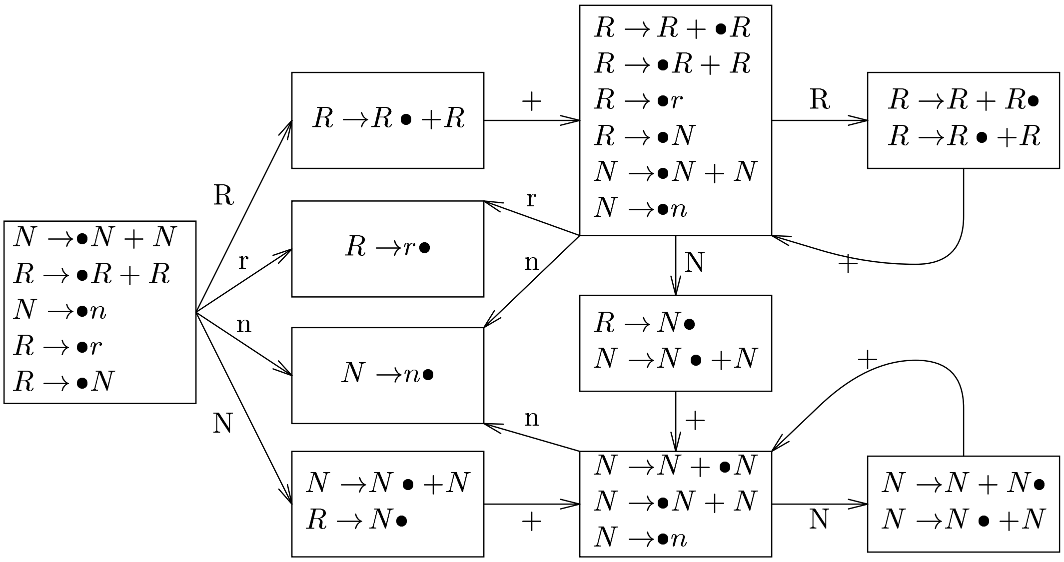


Figure 4: Goto graph for the grammar of example 7.4

production $[N \to n]$. The parser can either shift with $+$ or complete with the chain rule of $R \to N$. 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.

**7.5** Definition. (Earley modulo chain rules) The set $V_C$ contains all chain symbols $[B \to C]$, where $B$ and $C$ are nonterminals in CFG $\mathcal{G}$ and $B = C$ or $B \to_{\mathcal{G}} B_1 \to_{\mathcal{G}} \cdots \to_{\mathcal{G}} B_m \to_{\mathcal{G}} C$, $(m \geq 0)$. Symbols $[A \to A]$ and $A$ are identified. A production with chain symbols $[B \to C]$ in in its right-hand side is identifiable (member of grammar, priority relation) with a production where the chain symbols are replaced with their heads $B$. The $(I)$ and $(S)$ rules are as usual.

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

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

$$\frac{[A \to \alpha \bullet B\beta, h, i], [C \to \gamma\bullet, i, j]}{[A \to \alpha[B \to C] \bullet \beta, h, j]} \ |\beta| > 0 \qquad (C_1)$$

$$\frac{[A \to \alpha \bullet B, h, i], [C \to \gamma\bullet, i, j], \neg[A' \to \alpha'\bullet, h, j]}{[A \to \alpha[B \to C]\bullet, h, j]} \ (A' \to \alpha' > A \to \alpha B) \qquad (C_2)$$

The negative premise $\neg[A \to \alpha\bullet, i, j]$ in combination with the condition $A' \to \alpha' > A \to \alpha B$ is used in rule (C2) to express that: an item $[A \to \alpha[B \to C]\bullet, h, j]$

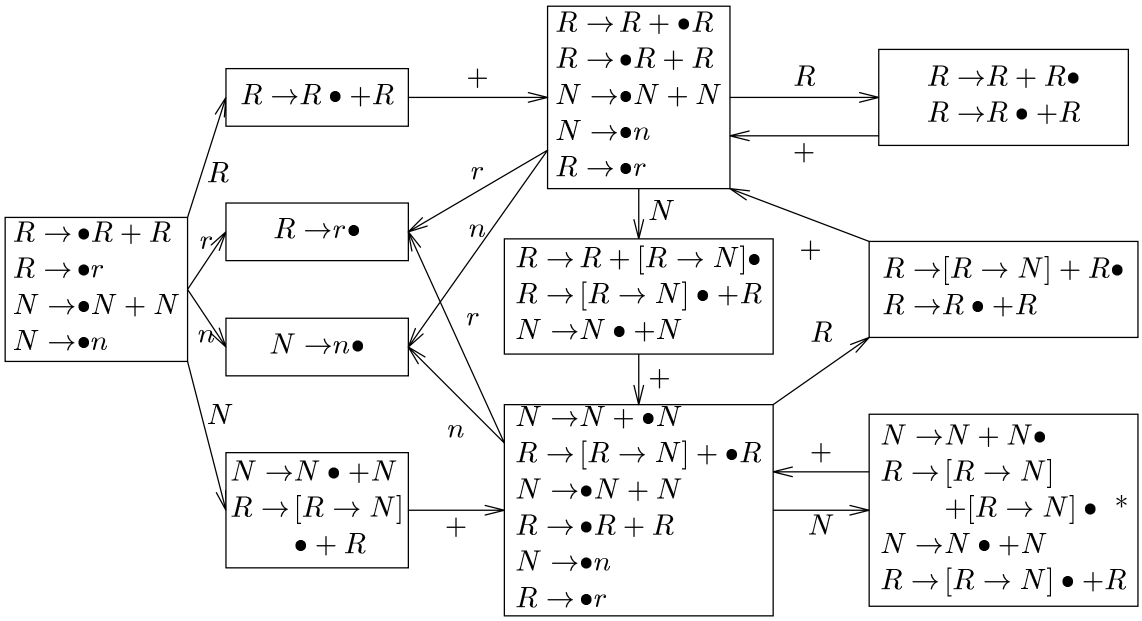


Figure 5: Goto graph for grammar of example 7.4 corresponding to parsing schema 7.5. The item marked with a $*$ is present if the negative premise of rule $(C2)$ is absent.

can be derived from $[A \to \alpha \bullet B, h, i]$ and $[C \to \gamma \bullet, i, j]$ only if no item $[A' \to \alpha' \bullet, h, j]$ 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 [Sik93] 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 $(C_2)$ has no instantiation of the form $I_1, I_2, \neg I_3 \vdash I_3$ that would make the schema inconsistent.

**7.6** Example. Figure 5 shows the goto graph for the grammar of example 7.4 according to parsing schema 7.5. The shift/reduce conflict between the items $[N \to N \bullet +N]$ and $[R \to N \bullet]$ is changed into a reduce/reduce conflict between the items $[N \to N + N \bullet]$ and $[R \to [R \to N] + [R \to N] \bullet]$. If the negative premise of rule $(C_2)$ is taken into account the item marked with a $*$ can not be derived, and is not present in the goto-graph; conflict solved. $\square$

The method does not help for grammars where the ambiguity is not caused by chain rules, for instance consider the grammar (example from [Kam92]) $E \to E\ E; E \to E - E; E \to -E$ that can be disambiguated by taking $E \to E\ E > E \to -E > E \to E - E$.

The methods can be combined into a parsing schema that handles both priority conflicts and the partial implementation of multiset filters by adding the subset exclusion conditions to the $(P)$, and $(C_i)$ rules of parsing schema 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 on the border of determinism. A subset of the ambiguities solved by multiset filters is solved by our second optimization. Together these optimizations can be used in the generation of efficient parsers for a large class of ambiguous context-free grammars.

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.

**Acknowledgements** I thank Mark van den Brand and an anonymous referee for their comments on this paper and Fien McColl for pointing out some stylistic deficiencies. Support for this research has been received from the Dutch Organization for Scientific Research (NWO) under grant 612-317-420: Incremental parser generation and context-dependent disambiguation, a multi-disciplinary perspective.

**Author's Address** Eelco Visser, Programming Research Group, University of Amsterdam, Kruislaan 403, NL-1098 SJ Amsterdam, The Netherlands,
email: visser@fwi.uva.nl, http://adam.fwi.uva.nl/~visser/

# References

[AJU75] A. V. Aho, S. C. Johnson, and J. D. Ullman. Deterministic parsing of ambiguous grammars. *Communications of the ACM*, 18(8):441–452, 1975.

[DeR71] F. L. DeRemer. Simple LR(k) grammars. *Communications of the ACM*, 14:453–460, 1971.

[Ear70] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

[Ear75] J. Earley. Ambiguity and precedence in syntax description. *Acta Informatica*, 4(1):183–192, 1975.

[HHKR92] J. Heering, P. R. H. Hendriks, P. Klint, and J. Rekers. *The syntax definition formalism SDF — Reference Manual*, 1992. Version 6 December 1992. Earlier version in *SIGPLAN Notices*, 24(11):43-75, 1989. Available as ftp://ftp.cwi.nl /pub/gipe/reports/ SDFManual.ps.Z.

[JL82] J.-P. Jouannaud and P. Lescanne. On multiset orderings. *Information Processing Letters*, 15(2):57–63, 1982.

[Kam92] Jasper Kamperman. A try at improving the second disambiguation phase in SDF. technical note, January 8 1992.

[Knu65] Donald E. Knuth. On the translation of languages from left to right. *Information and Control*, 8:607–639, 1965.

[KV94] Paul Klint and Eelco Visser. 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, October 1994. Tech. Rep. 126–1994, Dipartimento di Scienze dell'Informazione, Università di Milano. Also as TR P9426, Programming Research Group, University of Amsterdam, ftp://ftp.fwi.uva.nl /pub/programming-research/ reports/1994/ P9426.ps.Z.

[Lam93] Leslie Lamport. How to write a proof. Technical Report 94, DEC Systems Research Center, Palo Alto, California, February 14 1993.

[Rek92] J. Rekers. *Parser Generation for Interactive Environments.* PhD thesis, University of Amsterdam, 1992. Available by *ftp* from ftp.cwi.nl:/pub/gipe as Rek92.ps.Z.

[Sik93] Klaas Sikkel. *Parsing Schemata.* PhD thesis, Universiteit Twente, Enschede, December 1993.

[Sik94] Klaas Sikkel. 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, October 1994. Tech. Rep. 126–1994, Dipartimento di Scienze dell'Informazione, Università di Milano.

[Tho94a] Mikkel Thorup. Controlled grammatic ambiguity. *ACM Transactions on Programming Languages and Systems*, 16(3):1024–1050, May 1994.

[Tho94b] Mikkel Thorup. Disambiguating grammars by exclusion of sub-parse trees. Technical Report 94/11, Dept. of Computer Science, University of Copenhagen, Denmark, 1994.

[Tom85] Masura Tomita. *Efficient Parsing for Natural Languages. A Fast Algorithm for Practical Systems.* Kluwer Academic Publishers, 1985.

[Vis95] Eelco Visser. A case study in optimizing parsing schemata by disambiguation filters. In *Proceedings of Accolade95*, Amsterdam, 1995. The Dutch Graduate School in Logic.

# A Proofs

This appendix contains the proofs for theorems 4.4 and 5.7. The proofs are written in the style of [Lam93]. Since the proofs of Lemma's A.1 and A.2 are included in the proofs of Lemma's A.4 and A.5, respectively, these proofs are only shown as proof outlines. The proofs of the latter are shown in detail.

**A.1.** Lemma. (soundness of parsing schema 4.2) *For all context-free grammars* $\mathcal{G}$, $w = a_1 \dots a_n \in V_T^*$, $A \in V_N$, $\alpha, \beta \in V^*$, $i \leq j \in N$, $t_\alpha \in \mathcal{T}_\alpha^{\mathcal{G}}$ *such that* $A \to \alpha\beta \in \mathcal{G}$ *we have that*

$$w \vdash_{\mathcal{G}}^{4.2} [A \to \alpha \bullet \beta, i, j] \Rightarrow [A \to t_\alpha] \Rightarrow \wedge\ \mathit{yield}([A \to t_\alpha \beta]) = a_{i+1} \dots a_j \beta \\ \wedge\ \exists t_S[\bullet] \in \mathcal{T}_S^{\mathcal{G}} : \mathit{yield}(t_S[A]) = a_1 \dots a_i A \delta$$

Proof sketch: By induction on derivations. We have to prove for the last step in the derivation that it satisfies the condition above, assuming that this has been proven for all previous steps in the derivation. The last item derived is derived by one of the rules of the parsing schema. Therefore, it suffices to check the condition for the rules of the schema.

$\langle 1 \rangle 1.$ Case: $\dfrac{}{[S \to \bullet\gamma, 0, 0] \Rightarrow [S \to]}(I)$

$\langle 1 \rangle 2.$ Case: 1. $\dfrac{[A \to \alpha \bullet B\beta, h, i] \Rightarrow [A \to t_\alpha]}{[B \to \bullet\gamma, i, i] \Rightarrow [B \to]}(P)$
2. $\mathit{yield}(t_A = [A \to t_\alpha B \beta]) = a_{h+1} \dots a_i B \beta$
3. $\mathit{yield}(t_S[A]) = a_1 \dots a_h A \delta$

$\langle 1 \rangle 3.$ Case: 1. $\dfrac{[A \to \alpha \bullet a_{i+1}\beta, h, i] \Rightarrow [A \to t_\alpha], [a_{i+1}, i, i+1] \Rightarrow a_{i+1}}{[A \to \alpha a_{i+1} \bullet \beta, h, i+1] \Rightarrow [A \to t_\alpha a_{i+1}]}(S)$
2. $\mathit{yield}(t_A = [A \to t_\alpha a_{i+1} \beta]) = a_{h+1} \dots a_i a_{i+1} \beta$
3. $\mathit{yield}(t_S[A]) = a_1 \dots a_h A \delta$

$\langle 1 \rangle 4.$ Case: 1. $\dfrac{[A \to \alpha \bullet B\beta, h, i] \Rightarrow [A \to t_\alpha], [B \to \gamma\bullet, i, j] \Rightarrow t_B}{[A \to \alpha B \bullet \beta, h, j] \Rightarrow [A \to t_\alpha t_B]}(C)$
2. $\mathit{yield}(t_A = [A \to t_\alpha B \beta]) = a_{h+1} \dots a_i B \beta$
3. $\mathit{yield}(t_S[A]) = a_1 \dots a_h A \delta$
4. $\mathit{yield}(t_B) = a_{i+1} \dots a_j$
5. $\mathit{yield}(t'_S[B]) = a_1 \dots a_i B \delta'$

$\langle 1 \rangle 5.$ Q.E.D.

**A.2.** Lemma. (completeness of parsing schema 4.2) *For all context-free grammars* $\mathcal{G}$, $w = a_1 \dots a_n \in V_T^*$, $A \in V_N$, $\alpha, \beta \in V^*$, $i \leq j \in N$, $t_\alpha \in \mathcal{T}_\alpha^{\mathcal{G}}$ *such that* $A \to \alpha\beta \in \mathcal{G}$ *we have that*

$$w \vdash_{\mathcal{G}}^{4.2} [A \to \alpha \bullet \beta, i, j] \Rightarrow [A \to t_\alpha] \Leftarrow \wedge\ \mathit{yield}([A \to t_\alpha \beta]) = a_{i+1} \dots a_j \beta \\ \wedge\ \exists t_S[\bullet] \in \mathcal{T}_S^{\mathcal{G}} : \mathit{yield}(t_S[A]) = a_1 \dots a_i A \delta$$

Proof sketch: By simultaneous induction on $\alpha$ and $t_S$. We have to check all combinations of $\alpha$ and $t_S$ and construct the corresponding derivation.

$\langle 1 \rangle 1.$ Case: 1. $\mathit{yield}([B \to \gamma]) = \gamma$
2. $\mathit{yield}(t_S[B]) = a_1 \dots a_i B \delta$

3. $t_S[\bullet] \equiv \bullet$

$\langle 1\rangle 2.$ CASE: 1. $\mathit{yield}([B \to \gamma]) = \gamma$
2. $\mathit{yield}(t_S[B]) = a_1 \ldots a_i B\delta$
3. $t_S[\bullet] \equiv t'_S[[A \to t_\alpha \bullet \beta]]$ for some $A$, $\alpha$ and $\beta$

$\langle 1\rangle 3.$ CASE: 1. $\mathit{yield}([A \to t_\alpha a_{j+1}\beta]) = a_{i+1} \ldots a_j a_{j+1}\beta$
2. $\mathit{yield}(t_S[A]) = a_1 \ldots a_i A\delta$

$\langle 1\rangle 4.$ CASE: 1. $\mathit{yield}([A \to t_\alpha[B \to t_\gamma]\beta]) = a_{h+1} \ldots a_j\beta$
2. $\mathit{yield}(t_S[A]) = a_1 \ldots a_h A\delta$

$\langle 1\rangle 5.$ Q.E.D.

**A.3.** THEOREM. *For all context-free grammars $\mathcal{G}$ and $w = a_1 \ldots a_n \in V_T^*$ we have that*

$$\{t \in \mathcal{T}_S^{\mathcal{G}} \mid w \vdash_{\mathcal{G}}^{4.2} [S \to \gamma\bullet, 0, n] \Rightarrow t\} = \Pi(\mathcal{G})(w)$$

$\langle 1\rangle 1.$ $a_1 \ldots a_n \vdash [S \to \gamma\bullet, 0, n] \Rightarrow [S \to t_\gamma] \iff \mathit{yield}([S \to t_\gamma]) = a_1 \ldots a_n$
PROOF: by Lemma's A.1 and A.2. □

$\langle 1\rangle 2.$ $t \in \Pi(\mathcal{G})(a_1 \ldots a_n) \iff t \in \mathcal{T}_S \wedge \mathit{yield}(t) = a_1 \ldots a_n$
PROOF: by definition of $\Pi$ (Def. 2.3). □

$\langle 1\rangle 3.$ Q.E.D.
PROOF: by $\langle 1\rangle 1$ and $\langle 1\rangle 2$. □

**A.4.** LEMMA. (soundness of parsing schema 5.6) *For all context-free grammars $\mathcal{G}$, $w = a_1 \ldots a_n \in V_T^*$, $A \in V_N$, $\alpha, \beta \in V^*$, $i \leq j \in N$, $t_\alpha \in \mathcal{T}_\alpha^{\mathcal{G}}$ such that $A \to \alpha\beta \in \mathcal{G}$, and set $Q$ of parse tree patterns of the form $[A \to \alpha[B \to \gamma]\beta]$ we have that*

$$\begin{aligned} w \vdash_{\mathcal{G},Q}^{5.6} [A \to \alpha \bullet \beta, i, j] \Rightarrow [A \to t_\alpha] \Rightarrow & \wedge\ \mathit{yield}(t_A = [A \to t_\alpha\beta]) = a_{i+1} \ldots a_j\beta \\ & \wedge\ \exists t_S[\bullet] \in \mathcal{T}_S^{\mathcal{G}} : \mathit{yield}(t_S[A]) = a_1 \ldots a_i A\delta \\ & \wedge\ \neg t_S[t_A] \lhd Q \end{aligned}$$

PROOF SKETCH: By induction on derivations. We have to prove for the last step in the derivation that it satisfies the condition above, assuming that this has been proven for all previous steps in the derivation. The last item derived is derived by one of the rules of the parsing schema. Therefore, it suffices to check the condition for the rules of the schema.

$\langle 1\rangle 1.$ CASE: $\dfrac{}{[S \to \bullet\gamma, 0, 0] \Rightarrow [S \to]}(I)$

$\langle 2\rangle 1.$ $\mathit{yield}([S \to \gamma]) = \gamma$
PROOF: by definition of *yield*. □

$\langle 2\rangle 2.$ $\exists t_S[\bullet] \in \mathcal{T}_S^{\mathcal{G}} : \mathit{yield}(t_S[S]) = a_1 \ldots a_0 S\delta$
PROOF: take $t_S[\bullet] = \bullet$. □

$\langle 2\rangle 3.$ $\neg[S \to \gamma] \lhd Q$
PROOF: $[S \to \gamma]$ is not of the form $[A \to \alpha[B \to \gamma]\beta]$. □

$\langle 2\rangle 4.$ Q.E.D.
PROOF: by $\langle 2\rangle 1$, $\langle 2\rangle 2$ and $\langle 2\rangle 3$. □

$\langle 1\rangle 2.$ CASE: 1. $\dfrac{[A \to \alpha \bullet B\beta, h, i] \Rightarrow [A \to t_\alpha]}{[B \to \bullet\gamma, i, i] \Rightarrow [B \to]}(P)$
2. $\mathit{yield}(t_A = [A \to t_\alpha B\beta]) = a_{h+1} \ldots a_i B\beta$
3. $\mathit{yield}(t_S[A]) = a_1 \ldots a_h A\delta$

4. $\neg t_S[t_A] \lhd Q$

$\langle 2\rangle 1.$ $yield([B \to \gamma]) = \gamma$

PROOF: by definition of *yield*. □

$\langle 2\rangle 2.$ $\exists t'_S[\bullet] \in \mathcal{T}_S : yield(t'_S[A]) = a_1 \ldots a_i A\delta$

PROOF: take $t'_S[\bullet] = t_S[A \to t_\alpha \bullet \beta]$ then

$$\begin{aligned} yield(t_S[A \to t_\alpha B\beta]) &= a_1 \ldots a_h yield([A \to t_\alpha B\beta])\delta && [\text{by } \langle 1\rangle 2{:}3] \\ &= a_1 \ldots a_i B\beta\delta && [\text{by } \langle 1\rangle 2{:}2] \ \square \end{aligned}$$

$\langle 2\rangle 3.$ $\neg t_S[[A \to t_\alpha[B \to \gamma]\beta]] \lhd Q$

PROOF: by $\langle 1\rangle 2{:}4$ and the restriction on rule (P). □

$\langle 2\rangle 4.$ Q.E.D.

PROOF: by $\langle 2\rangle 1$, $\langle 2\rangle 2$ and $\langle 2\rangle 3$. □

$\langle 1\rangle 3.$ CASE: 1. $\dfrac{[A \to \alpha \bullet a_{i+1}\beta, h, i] \Rightarrow [A \to t_\alpha], [a_{i+1}, i, i+1] \Rightarrow a_{i+1}}{[A \to \alpha a_{i+1} \bullet \beta, h, i+1] \Rightarrow [A \to t_\alpha a_{i+1}]}(S)$

2. $yield(t_A = [A \to t_\alpha a_{i+1}\beta]) = a_{h+1} \ldots a_i a_{i+1}\beta$

3. $yield(t_S[A]) = a_1 \ldots a_h A\delta$

4. $\neg t_S[t_A] \lhd Q$

$\langle 2\rangle 1.$ Q.E.D.

PROOF: by $\langle 1\rangle 3{:}2{,}3{,}4$. □

$\langle 1\rangle 4.$ CASE: 1. $\dfrac{[A \to \alpha \bullet B\beta, h, i] \Rightarrow [A \to t_\alpha], [B \to \gamma\bullet, i, j] \Rightarrow t_B}{[A \to \alpha B \bullet \beta, h, j] \Rightarrow [A \to t_\alpha t_B]}(C)$

2. $yield(t_A = [A \to t_\alpha B\beta]) = a_{h+1} \ldots a_i B\beta$

3. $yield(t_S[A]) = a_1 \ldots a_h A\delta$

4. $yield(t_B) = a_{i+1} \ldots a_j$

5. $yield(t'_S[B]) = a_1 \ldots a_i B\delta'$

6. $\neg t_S[t_A] \lhd Q$

7. $\neg t'_S[t_B] \lhd Q$

$\langle 2\rangle 1.$ $yield([A \to t_\alpha t_B\beta]) = a_{h+1} \ldots a_j\beta$

PROOF: $$\begin{aligned} yield([A \to t_\alpha t_B\beta]) &= yield(t_\alpha)yield(t_B)\beta && [\text{by def. } yield] \\ &= a_{h+1} \ldots a_i a_{i+1} \ldots a_j\beta && [\text{by } \langle 1\rangle 4{:}2{,}4] \ \square \end{aligned}$$

$\langle 2\rangle 2.$ $\neg t_S[[A \to t_\alpha t_B\beta]] \lhd Q$

PROOF: by $\langle 1\rangle 4{:}6{,}7$ and the condition on rule $(C)$. □

$\langle 2\rangle 3.$ Q.E.D.

PROOF: by $\langle 2\rangle 1$ and $\langle 1\rangle 4{:}3$. □

$\langle 1\rangle 5.$ Q.E.D.

PROOF: by cases $\langle 1\rangle 1$, $\langle 1\rangle 2$, $\langle 1\rangle 3$, $\langle 1\rangle 4$ and induction on derivations. □

**A.5.** LEMMA. (completeness of parsing schema 5.6) *For all context-free grammars* $\mathcal{G}$, $w = a_1 \ldots a_n \in V_T^*$, $A \in V_N$, $\alpha, \beta \in V^*$, $i \leq j \in N$, $t_\alpha \in \mathcal{T}_\alpha^{\mathcal{G}}$ *such that* $A \to \alpha\beta \in \mathcal{G}$ *we have that*

$$\begin{aligned} w \vdash_{\mathcal{G}}^{5.6} [A \to \alpha \bullet \beta, i, j] \Rightarrow [A \to t_\alpha] \Leftarrow\ &\wedge\ yield(t_A = [A \to t_\alpha\beta]) = a_{i+1} \ldots a_j\beta \\ &\wedge\ \exists t_S[\bullet] \in \mathcal{T}_S^{\mathcal{G}} : yield(t_S[A]) = a_1 \ldots a_i A\delta \\ &\wedge\ \neg t_S[t_A] \lhd Q \end{aligned}$$

PROOF SKETCH: By simultaneous induction on $\alpha$ and $t_S$. We have to check all combinations of $\alpha$ and $t_S$ and construct the corresponding derivation.

⟨1⟩1. CASE: 1. $yield([B \to \gamma]) = \gamma$
2. $yield(t_S[B]) = a_1 \dots a_i B\delta$
3. $t_S[\bullet] \equiv \bullet$

⟨2⟩1. $B \equiv S$
PROOF: by ⟨1⟩1:3. □

⟨2⟩2. $i = j = 0$
PROOF: $i = j$ because $a_{i+1} \dots a_j\gamma = \gamma = yield([B \to \gamma])$ by ⟨1⟩1:1 and $i = 0$ because $a_1 \dots a_i S\delta = S = yield(t_S[S])$. □

⟨2⟩3. $w \vdash [S \to \bullet\gamma, 0, 0] \Rightarrow [S \to]$
PROOF: by rule $(I)$. □

⟨2⟩4. Q.E.D.
PROOF: by ⟨2⟩1, ⟨2⟩2 and ⟨1⟩1. □

⟨1⟩2. CASE: 1. $yield([B \to \gamma]) = \gamma$
2. $yield(t_S[B]) = a_1 \dots a_i B\delta$
3. $t_S[\bullet] \equiv t'_S[[A \to t_\alpha \bullet \beta]]$ for some $A$, $\alpha$ and $\beta$
4. $\neg t_S[[B \to \gamma]] \lhd Q$

⟨2⟩1. There is some $1 \leq h \leq i$ such that
1. $yield([A \to t_\alpha B\beta]) = a_{h+1} \dots a_i B\beta$
2. $yield(t'_S[A]) = a_1 \dots a_h A\delta'$

⟨3⟩1. $yield(t'_S[[A \to t_\alpha B\beta]]) = a_1 \dots a_i B\delta$
PROOF: by ⟨1⟩2:2,3. □

⟨3⟩2. $a_1 \dots a_i B\delta = a_1 \dots a_h yield([A \to t_\alpha B\beta])\delta'$
PROOF: by ⟨3⟩1 and the definition of *yield*. □

⟨3⟩3. Q.E.D.
PROOF: by ⟨3⟩2 □

⟨2⟩2. 1. $\neg t'_S[[A \to t_\alpha B\beta]] \lhd Q$
2. $[A \to \alpha[B \to \gamma]\beta] \notin Q$
PROOF: by ⟨1⟩2:4 and the definition of $\lhd$. □

⟨2⟩3. $w \vdash [A \to \alpha \bullet B\beta, h, i] \Rightarrow [A \to t_\alpha]$
PROOF: by ⟨2⟩1.1,2, ⟨2⟩2.1 and ⟨0⟩. □

⟨2⟩4. Q.E.D.
PROOF: $w \vdash [B \to \bullet\gamma, i, i] \Rightarrow [B \to]$ by ⟨2⟩3, ⟨2⟩2.2 and rule $(P)$. □

⟨1⟩3. CASE: 1. $yield([A \to t_\alpha a_{j+1}\beta]) = a_{i+1} \dots a_j a_{j+1}\beta$
2. $yield(t_S[A]) = a_1 \dots a_i A\delta$
3. $\neg t_S[a_{j+1}] \lhd Q$

⟨2⟩1. $w \vdash [A \to \alpha \bullet a_{j+1}\beta, i, j] \Rightarrow [A \to t_\alpha]$
PROOF: by ⟨1⟩3:1,2 and ⟨0⟩. □

⟨2⟩2. $w \vdash [a_{j+1}, j, j+1] \Rightarrow a_{j+1}$
PROOF: by rule $(H)$ □

⟨2⟩3. Q.E.D.
PROOF: $w \vdash [A \to \alpha a_{j+1} \bullet \beta, i, j+1] \Rightarrow [A \to t_\alpha]$ by ⟨2⟩1, ⟨2⟩2 and rule $(S)$. □

⟨1⟩4. CASE: 1. $yield(t_A = [A \to t_\alpha[B \to t_\gamma]\beta]) = a_{h+1} \dots a_j\beta$
2. $yield(t_S[A]) = a_1 \dots a_h A\delta$

3. $\neg t_S[t_A] \lhd Q$

$\langle 2\rangle 1.$ There is some $h+1 \leq i \leq j$ such that
1. $\mathit{yield}([A \to t_\alpha B\beta]) = a_{h+1} \dots a_i B\beta$
2. $\mathit{yield}([B \to t_\gamma]) = a_{i+1} \dots a_j$

PROOF:

$$\begin{aligned}\mathit{yield}([A \to t_\alpha[B \to t_\gamma]\beta]) &= \mathit{yield}(t_\alpha)\mathit{yield}([B \to t_\gamma])\beta && \text{[by def. } \mathit{yield}\text{]}\\ &= a_{h+1} \dots a_i a_{i+1} \dots a_j \beta && \text{[by } \langle 1\rangle 4{:}1\text{]} \ \Box\end{aligned}$$

$\langle 2\rangle 2.$ $\mathit{yield}(t_S[[A \to t_\alpha B\beta]]) = a_1 \dots a_i B\beta\delta$

PROOF:

$$\begin{aligned}\mathit{yield}(t_S[[A \to t_\alpha B\beta]]) &= a_1 \dots a_h \mathit{yield}([A \to t_\alpha B\beta])\delta && \text{[by } \langle 1\rangle 4{:}2\text{]}\\ &= a_1 \dots a_h a_{h+1} \dots a_i B\beta\delta && \text{[by } \langle 2\rangle 1.1\text{]} \ \Box\end{aligned}$$

$\langle 2\rangle 3.$ 1. $\neg(t''_S[[A \to t_\alpha B\beta]] = t_S[[A \to t_\alpha B\beta]]) \lhd Q$
2. $\neg(t'_S[B] = t_S[[A \to t_\alpha B\beta]]) \lhd Q$
3. $[A \to \alpha[B \to \gamma]\beta] \notin Q$

PROOF: by $\langle 1\rangle 4$:1,3 and the definition of $\lhd$. $\Box$

$\langle 2\rangle 4.$ $w \vdash [A \to \alpha \bullet B\beta, h, i] \Rightarrow [A \to t_\alpha]$

PROOF: by $\langle 2\rangle 1.1$, $\langle 1\rangle 4$:2, $\langle 2\rangle 3.1$ and $\langle 0\rangle$. $\Box$

$\langle 2\rangle 5.$ $w \vdash [B \to \gamma\bullet, i, j] \Rightarrow [B \to t_\gamma]$

PROOF: by $\langle 2\rangle 1.2$, $\langle 2\rangle 2$, $\langle 2\rangle 3.2$ and $\langle 0\rangle$ $\Box$

$\langle 2\rangle 6.$ Q.E.D.

PROOF: $w \vdash [A \to \alpha B \bullet \beta, h, i] \Rightarrow [A \to t_\alpha t_\beta]$ by $\langle 2\rangle 4$, $\langle 2\rangle 5$, $\langle 2\rangle 3.3$ and rule $(C)$. $\Box$

$\langle 1\rangle 5.$ Q.E.D.

PROOF: by $\langle 1\rangle 1$, $\langle 1\rangle 2$, $\langle 1\rangle 3$ and $\langle 1\rangle 4$ and mutual induction on $\alpha$ and $t_S$. (The cases cover all combinations of $\alpha$ and $t_S$. Each case uses the induction hypothesis with a smaller $\alpha$ or $t_S$.) $\Box$

**A.6.** THEOREM. (correctness of parsing schema 5.6) *For all context-free grammars $\mathcal{G}$ and $w = a_1 \dots a_n \in V_T^*$ and set $Q$ of parse tree patterns of the form $[A \to \alpha[B \to \gamma]\beta]$ we have that*

$$\{t \in \mathcal{T}_S^{\mathcal{G}} \mid w \vdash^{5.6}_{\mathcal{G},Q} [S \to \gamma\bullet, 0, n] \Rightarrow t\} = \mathcal{F}^Q(\Pi(\mathcal{G})(w))$$

$\langle 1\rangle 1.$ $a_1 \dots a_n \vdash [S \to \gamma\bullet, 0, n] \Rightarrow [S \to t_\gamma]$ iff $\mathit{yield}([S \to t_\gamma]) = a_1 \dots a_n$ and $\neg[S \to t_\gamma] \lhd Q$

PROOF: by Lemma's A.4 and A.5. $\Box$

$\langle 1\rangle 2.$ $t \in \mathcal{F}^Q(\Pi(\mathcal{G})(a_1 \dots a_n))$ iff $t \in \mathcal{T}_S \wedge \mathit{yield}(t) = a_1 \dots a_n \wedge \neg[S \to t_\gamma] \lhd Q$

PROOF: by definition of $\Pi$ and $\mathcal{F}^Q$ (Defs. 2.3 and 5.4). $\Box$

$\langle 1\rangle 3.$ Q.E.D.

PROOF: by $\langle 1\rangle 1$ and $\langle 1\rangle 2$. $\Box$

# B SLR(1) Parsing

The LR(0) goto graphs in figures 2 and 3 from section 6 contain shift/reduce conflicts that are easy to prevent with the SLR(1) (Simple LR(1)) extension of LR(0) parsing due to [DeR71]. 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 left 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 First$(\alpha, \Psi)$ yields the set of symbols that can start a sentence derived from a string of symbols $\alpha$ followed by a symbol from the set $\Psi$. The expression Follow$(B, \Psi)$ denotes the set of symbols that can follow symbol $B$ in a sentence that is followed by a symbol from the set $\Psi$. The reduce rule now only applies if a production has been recognized *and* the next symbol in the string can follow the left-hand side of the production.

**B.1** Definition. (SLR(1) parsing) This schema adapts the reduce rule of schema 6.1.

$$\begin{array}{rcl} \mathrm{First}(\epsilon, \Psi) & = & \Psi \\ \mathrm{First}(a\alpha, \Psi) & = & \{a\} \\ \mathrm{First}(A\alpha, \Psi) & = & \bigcup_{A \to \beta \in \mathcal{G}} \mathrm{First}(\beta\alpha, \Psi) \end{array}$$

$$\mathrm{Follow}(B, \Psi) = \{a \mid A \to \alpha B \beta \in \mathcal{G} \wedge a \in \mathrm{First}(\beta, \mathrm{Follow}(A, \Psi))\}$$

$$\frac{\begin{array}{l}[\Phi^{[A \to \alpha \bullet B\beta]}, h, i],\ [\Phi_0^{[B \to \bullet B_1 \ldots B_m]}, i, i], \\ \qquad \ldots, [\Phi_m^{[B \to B_1 \ldots B_m \bullet]}, i, i_m], [a, i_m, i_m + 1]\end{array}}{[goto(B, \Phi), h, i_m]}\ a \in \mathrm{Follow}(B, \{\$\})$$

□

The SLR(1) schema can be adapted in the same way as the LR(0) schema to account for priority conflicts (or $Q$ subtree exclusion). However, the definition of Follow above is too weak for this extended schema. For instance, in the grammar of example 5.3 $*$ is in the Follow set of $E$. However, $*$ can not follow an $E$ if it is a $E \to E + E$, i.e., if a reduction is done with $E \to E + E$, 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 6 shows the SLR(1) table for the grammar of example 5.3.

**B.2** Definition. (SLR(1) parsing modulo $Q$) This schema defines SLR(1) parsing modulo $Q$ using the definition of the closure and goto functions from parsing schema 6.2 and the definition of First from B.1.

$$\begin{array}{rl} \mathrm{Follow}(B \to \gamma, \Psi) = \{a \mid & A \to \alpha B \beta \in \mathcal{G} \\ & \wedge\ a \in \mathrm{First}(\beta, \mathrm{Follow}(A \to \alpha B \beta, \Psi)) \\ & \wedge\ [A \to \alpha[B \to \gamma]\beta] \notin Q\} \end{array}$$

$$\frac{\begin{array}{l}[\Phi^{[A \to \alpha \bullet B\beta]}, h, i],\ [\Phi_0^{[B \to \bullet B_1 \ldots B_m]}, i, i], \\ \qquad \ldots, [\Phi_m^{[B \to B_1 \ldots B_m \bullet]}, i, i_m], [a, i_m, i_m + 1]\end{array}}{[goto(B \to B_1 \ldots B_m, \Phi), h, i_m]}\ a \in \mathrm{Follow}(B \to B_1 \ldots B_m, \{\$\})$$

□

| 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) $E \to a$
(2) $E \to E * E$
(3) $E \to E + E$
(4) $S \to E\ \$$

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

Figure 6: SLR(1) table for the grammar of example 5.3. 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.