Parsing of Context-Free Languages *

Klaas Sikkel

GMD- German National Research Center for Information Technology,
FIT.CSCW, 53757 Schloß Birlinghoven,
53757 Sankt Augustin, Germany
sikkel@gmd.de

Anton Nijholt

University of Twente, Computer Science Department, P.O. Box 217,
7500 AE Enschede, The Netherlands,
anijholt@cs.utwente.nl

Abstract

Parsing is the process of assigning structure to sentences. The structure is obtained from the grammatical description of the language. Both in Computer Science and in Computational Linguistics, context-free grammars and associated parsing algorithms are among the most useful tools. Numerous parsing algorithms have been developed. Special subclasses of the context-free grammars have been introduced in order to allow and induce efficient parsing algorithms. Special superclasses of the context-free grammars have been introduced in order to allow use of variants of efficient parsing methods that had been developed for context-free grammars. At first sight many parsing algorithms seem to be different, but nevertheless related. Some unifying approaches have been attempted in the past, but none survived the changing field. This report introduces a unifying approach at a level between grammars and algorithms, introducing so-called parsing schemata. In the parsing schemata framework the essentials of different parsing algorithms can be compared and it can be shown how to derive an algorithm from another one. The insight that is obtained this way also allows the derivation of new algorithms and it allows less tedious observations about correctness than usual. The framework can also be applied to grammar formalisms beyond the context-free grammars.

Table of Contents

1. Introduction

In computer science, grammars are human-constructed formalisms that are meant to define languages. These can be programming languages or, in theoretical computer science formal languages. This description is often partial. It is not unusual to see a formal description of the syntactic structure of a language, while the semantic part remains ill-defined. Finding the syntactic structure of a program (which is a sentence in the language) is part of the compilation process of a program. The construction of this structure is called parsing. The result of the parsing process is a hierarchical account of the elements that make up the program. This account makes it possible to assign semantics to a program.

Formal syntactic descriptions of languages were first given by the linguist Noam Chomsky. Because the descriptions were formal, the languages were also formal: sequences of symbols that satisfied descriptions based on finite state automata or regular grammars, context-free grammars, or context-sensitive grammars. In Chomsky's view, these descriptions were the starting point for descriptions of the syntax of natural, human spoken, languages. Moreover, these descriptions would allow to assign meaning to sentences. Interestingly, at about the same time Chomsky introduced different classes of grammars and languages, a committee defining a programming language (ALGOL) introduced a programming language description formalism called Backus-Naur-Form (BNF) which turned out to be equivalent to one of Chomsky's grammar classes, the so-called context-free grammars.

In Chomsky's view, human language grammars were not human-constructed formalisms. The rules of the formalism, or, more generally, the principles that determine the rules, are supposed to be innate. This view led to a distinction between competence and performance in human language use. Each language user has a language competence that allows him to construct all kinds of sentences using the rules of a grammar. Constructing sentences can be compared with using rules to compute a multiplication or a division in arithmetic. Language users can construct sentences using rules of syntax. Due to environmental circumstances in normal man-to-man communication, these rules are not always obeyed. Performance differs from competence.

It is much easier, however, to do research on self-chosen rules of sentence construction and analysis than to do research on actual language behaviour. For this obvious reason, grammar formalisms and their parsing methods have drawn so much attention by computer scientists and computational linguists. It should be admitted, on the other hand, that nowadays natural and programming language processing systems can be built on the basis of these formalisms. Whether or not formalisms that are used for natural language processing meet certain linguistic principles in some way or other, or even some principles of human language innateness, is not the main concern of those doing research and development in this area.

1.1 Parsing Algorithms

Parsing algorithms have been defined for all kinds of language descriptions. After the introduction of the well-known Chomsky hierarchy in the late fifties and early sixties, we see a common interest of computer scientists and computational linguists in parsing methods for context-free languages. The quest for efficient parsing methods led to polynomial-time algorithms for general context-free grammars in the middle and late sixties. Among them, the so-called Cocke-Younger-Kasami and the Earley parsing algorithms. In computer science, however, these formalisms were thought to be unnecessarily general for describing the syntactic properties of programming languages, and therefore to be unnecessarily inefficient. Linear-time algorithms like LL and LR were introduced. These are sufficiently general for dealing with the syntactic backbone of programming languages. Interest in general context-free methods diminished, or was left to theoretical computer scientists. In computational linguistics there were other reasons to become critical of the context-free grammar formalism. Its descriptional adequacy, that is, its ability to cover linguistic generalities in a natural way, was considered to be too weak. It was also doubted whether it provides sufficient generative capacity. The LL and LR approaches favoured in computer science were clearly much less suitable, because these do not allow representation of syntactic ambiguities.

It is remarkable that in the late seventies and early eighties we see a growing interest in LR-like methods and context-free grammars in computational linguistics and a growing interest in general context-free grammar descriptions in computer science. How can this be explained? In computational linguistics, first of all, the so-called "determinism hypothesis" attracted a lot of attention. The idea is that, in general, people do not "backtrack" while analysing a sentence. Backtracking becomes necessary only when a started analysis cannot be continued at some point in the sentence. Mitch Marcus introduced an LR-like "wait and see" stack formalism in order to parse sentences "deterministically". Reviewing the literature from that period, one sees lots of misconceptions and confusion among researchers. Apparently these are partly due to lack of knowledge about formal parsing methods such as, for example, Earley's method and how issues like "backtracking", "determinism", and "efficiency" relate to these algorithms. Since then, however, knowledge of formal methods has become more wide-spread. This can also be illustrated with the introduction of formalisms like Lexical Functional Grammar (LFG), Generalized Phrase Structure Grammar (GPSG), Head-Driven Phrase Structure Grammar (HPSG), Unification Formalisms, Definite Clause Grammars and Tree Adjoining Grammars (TAGs) in the early 1980s. It led to a new discussion on the question whether the generative capacity of context-free formalisms would suffice to describe the syntax of natural languages, it led to a systematic comparison of grammar formalisms, yielding the weakly context-sensitive languages as a newly discovered class for which adequate generative capacity was claimed [JVW91], and it led to many less efficient, but nevertheless polynomial variants of general context-free parsing algorithms.

The formalisms mentioned above are certainly much more general than a pure context-free formalism. However, their backbone is context-free or the way the formalisms are defined and used bear very much resemblance to the context-free paradigm.

1.2 Parsing Technology

We have not yet mentioned one of the main influences that caused researchers in computational linguistics and natural language processing to shift their attention to existing formal parsing methods and possible extensions of these methods. That influence was the increasing demand of society, military, and funding organisations to produce research results that could be used to build tools and systems for practical natural language processing applications. Applications like speech understanding systems, natural language interfaces to information systems, machine translation of texts, information retrieval, help systems for complex software and machinery, knowledge extraction from documents, text image processing, and so on. The availability of a comprehensive cognitive and linguistic theory does not seem to be a precondition for applications in the area of natural language understanding. Many applications do not require this comprehensive theory. Moreover, many applications can be built using research results that are not influenced in any way by cognitive, psycho-linguistic or linguistic principles. Generalized LR parsing, introduced in the mid-eighties by Masaru Tomita, is such a method that was introduced as a simple, straightforward and efficient parser for general context-free languages and grammars. Due to its straightforwardness, like the deterministic LR method, it has attracted a lot of attention and it has been used in many natural language processing research projects and in some applications.

In computer science, it was stated above, more and more attention has been devoted to general context-free parsing methods. The just mentioned generalized LR method, for example, has been used in grammar, parser and compiler development environments. In general, software engineering environments may offer their users syntax-dependent tools. The compiler construction level is only one level, and a rather low one, where descriptions based on formal grammar play a role. Furthermore, computer science is a growing science in the sense that borders between so-called "pure" computer science and several application areas are disappearing. Grammars and parsing methods play a role in pattern recognition, they have been used to describe and analyse command and action languages (human interaction with a computer system through key presses, cursor movements, etc.), to describe screen lay-outs, etc. Human factors have become important in computer science. Increasing the accessibility of computer systems through the use of speech and natural language in the man-machine interface is an aim worth pursuing. It is obvious that computer scientists and computational linguists will meet each other here and that they can learn from each other's methods to deal with languages.

Finally, we would like to mention another influence that caused computer scientists to go back to general context-free parsing methods. Parallelism is

the keyword here. The introduction of new types of machine architectures and the possibility to implement algorithms on single chips have led to new research on existing parsing algorithms. Some research has been purely theoretical, in the sense that all kinds of efficiency limits were explored. Some research has been practical, in the sense that all kinds of extensions and variations of existing parsing algorithms have been investigated in order to make them suitable for parallel implementations and in the sense that these implementations have been realized, analysed and evaluated.

1.3 About this report

Parsing schemata are introduced in this report and used to bring some order in the field of context-free language parsing. In order to compare the essentials of parsing algorithms we want to abstract as much as possible from implementation issues, including the data, control and communication structures. The general idea is that all kinds of different parsers are "item" based, in the sense that they start with an initial set of items (constructed from the sentence), compute intermediate sets of items and deliver a final set of items. Items can have different interpretations. In this context it suffices to consider an item as a set of constraints on a (partial or complete) parse tree. This is, for example, a possible interpretation of an item in an LR or Earley parsing algorithm. Recognition or production of items can be interpreted as logical deduction from a set of hypotheses (initial items) to a set of final items (representing completed parse trees) by applying deduction steps. Underlying the parsers we are familiar with are different "deduction" systems using different items and different deduction steps. Relations between parsing algorithms can be found by defining operations on items and deduction steps. A parsing schema is the formal model, on a level of abstraction between grammar and algorithm, in which these ideas are expressed. Having these parsing schemata and being able, as we claim, to use them to understand the relations between parsing algorithms, it becomes also possible to port improvements and optimizations for one algorithm to related algorithms.

In the next section, parsing schemata are introduced by means of some informal examples. These examples are the standard CYK algorithm and the Earley algorithm. A formal introduction to parsing schemata follows in Section 3. Here we also introduce the notion of correctness of a parsing schema. That is, we need a way to say that a parse tree satisfies the constraints expressed in the items. Proving correctness on the level of parsing schemata is a less tedious task than on the level of algorithms since all details about data, control and communication structures are not present.

Section 4 and Section 5 are concerned with generalization and filtering, respectively. Generalizations and filters are relations between parsing schemata. Relations between different parsers can be uncovered when relations between their underlying parsing schemata have been established.

Generalization takes a more fine-grained look on the parsing process. It leads to more items and more steps. Generalization can be decomposed into several "primitive" relationsships. These primitive generalizations and their

combinations allow, for example, the derivation of a simple version of the Earley parser from the CYK parser.

The aim of filtering is to reduce the number of steps and items. The different basic kinds of filtering that are introduced allow, for example, the derivation of the canonical Earley parser from the simple (bottom-up) Earley parser that was obtained as a generalization of the CYK parser. In a similar way, the well known and efficient Graham-Harrison-Ruzzo parser can be filtered from the Earley parser using basic filtering techniques on the underlying parsing schemata. Some observations can be made about the consequences of using these techniques for run-time and compile-time optimization and consequences for parallel implementations.

After having formalized and illustrated the theoretical concepts we can, in Section 6, show more elaborate examples of how filtering and generalization can be used to relate parsing schemata underlying different parsers. Rytter's parallel parsing algorithm and Left-Corner parsing are among the algorithms that will be related to previously mentioned algorithms through comparisons of the underlying parsing schemata.

Section 7 surveys some other well-known approaches to the parsing problem and shows how they relate to the parsing schemata framework that is introduced here. In particular, the relation with LR-like methods is examined. In Section 8 we briefly review some other grammar formalisms in computational linguistics and discuss how they relate to the parsing schemata framework. Conclusions are summarized in the final section.

2. An Informal Introduction

We introduce the general idea of a parsing schema by means of a few informal examples. A more rigorous treatment follows in Section 3. A comprehensive discussion of parsing schemata will appear in [Sik97].

The following conventions apply throughout this report:

A *context-free grammar* is a 4-tuple $G = (N, \Sigma, P, S)$, with N a set of nonterminal symbols, Σ a set of terminal symbols, P a finite set of productions, and $S \in N$ the start symbol. Furthermore, $N \cap \Sigma = \emptyset$. We write V for $N \cup \Sigma$.

We write $A, B, \ldots \in N$ for nonterminals; $a, b, \ldots \in \Sigma$ for terminals; $X, Y, \ldots \in V$ for arbitrary variables; $\alpha, \beta, \ldots \in V^*$ for strings of arbitrary variables; ε for the empty string. The letters $i, j, \ldots$ denote nonnegative integers.

We write $A \rightarrow \alpha$ for a production (A, α) in P. The relation $\Rightarrow$ on $V^* \times V^*$ is defined by $\alpha \Rightarrow \beta$ if there are $\alpha_1, \alpha_2, A, \gamma$ such that $\alpha = \alpha_1 A \alpha_2$, $\beta = \alpha_1 \gamma \alpha_2$ and $A \rightarrow \gamma \in P$.

The class of context-free grammars is denoted by $\mathcal{CFG}$. A subclass of $\mathcal{CFG}$ is $\mathcal{CNF}$, the class of grammars in Chomsky Normal Form. If $G \in \mathcal{CNF}$ then P contains productions of the form $A \rightarrow BC$ and $A \rightarrow \alpha$ only.

A very simple parsing algorithm is the so-called CYK algorithm [Kas65, You67], called after Cocke, Younger and Kasami. It is restricted to grammars in Chomsky Normal Form.

Assume that we have some grammar $G \in \mathcal{CNF}$ and a string $a_1 \ldots a_n$ to be parsed. The CYK algorithm recognizes *items* $[A, i, j]$ that satisfy $A \Rightarrow^* a_{i+1} \ldots a_j$.

The canonical way to implement this is to use a triangular matrix T with cells $T_{i,j}$ for all applicable value pairs of i and j. Recognition of an item $[A, i, j]$ is denoted by adding A to $T_{i,j}$. If we have $a = a_j$ and $A \rightarrow a \in P$ then A can be added to entry $T_{j-1,j}$. If we have $B \in T_{i,k}$, $C \in T_{k,j}$ and $A \rightarrow BC \in P$ then A can be added to $T_{i,j}$. The CYK algorithm gives an obvious control structure to make sure that all items are recognized that can be recognized.

It is worth noting that the output of the algorithm is *not* a parse tree, or a collection of parse trees. The output of the CYK algorithm (abstracting from its canonical data structure) is a *set of items*

$$\{[A, i, j] \mid A \Rightarrow^* a_{i+1} \ldots a_j\}.$$

The string is correct if and only if $[S, 0, n]$ is in this set. Moreover, if the string is correct, a parse forest or a particular (e.g. leftmost) parse can be constructed fairly easy from the items in this set. If we have $[S, 0, n]$ then there must be B, C, and k such that $S \rightarrow BC \in P$ and $[B, 0, k]$ and $[C, k, n]$ have been recognized as well. So, in a strict sense, CYK is not a parser but a recognizer enhanced with information that facilitates parse tree construction. It is common practice to call this a parser as well, and most parsers discussed in the remainder of this report will be of the same nature.

The way in which the CYK algorithm recognizes items for a given grammar $G \in \mathcal{CNF}$ and string $a_1 \ldots a_n$ can be denoted by a logical deduction system, called a *parsing system*.

Example 2.1. (*CYK*)
Firstly, we define a domain of items

$$\mathcal{I}_{\mathrm{CYK}} \quad = \quad \{[A, i, j] \mid A \in N \ \wedge \ 0 \leq i < j\}.$$

One could restrict $\mathcal{I}$ to items with $j \leq n$, of course, but there are some advantages in choosing the domain of items independent of the given sentence. Secondly, we need a set of so-called *hypotheses*[1]

$$H \qquad = \quad \{[a, i-1, i] \mid a = a_i \wedge 1 \leq i \leq n\}$$

that represent the string.
Thirdly, we need inference rules. We specify an inference rule by a *set of deduction steps* that covers all instances of inferences[2]. A set of inference rules,

---

[1] Whether the hypotheses are included in the domain of items or not does not really matter. It will turn out te be more convenient to define a separate set of hypotheses.

[2] This way of specifying rules has been chosen because it allows a certain flexibility. For example, it allows specification of *conditional rules*, to be applied only in certain circumstances, simply by restricting the set of deduction steps.

therefore, can be denoted by the union of corresponding sets of deduction steps. For CYK we define:

$$D^{(1)} = \{[a, i-1, i] \vdash [A, i-1, i] \mid A{\to}a \in P\},$$

$$D^{(2)} = \{[B, i, j], [C, j, k] \vdash [A, i, k] \mid A{\to}BC \in P\},$$

$$D_{\mathrm{CYK}} = D^{(1)} \cup D^{(2)}.$$

As with the domain $\mathcal{I}$, we have not bothered to restrict the deduction steps to items with $j \leq n$. The parsing system $\mathbb{P}_{\mathrm{CYK}}$ for G and $a_1 \ldots a_n$ is defined by the triple $\langle \mathcal{I}, H, D \rangle$.

A *parsing schema* **CYK** is a generalization of $\mathbb{P}_{\mathrm{CYK}}$ to arbitrary strings and arbitrary grammars in $\mathcal{CNF}$. One can see a parsing schema as a function that yields a parsing system for a given grammar and a given string over the alphabet of that grammar.

The CYK algorithm has the disadvantage that it is restricted to grammars in Chomsky Normal Form. A similar algorithm for arbitrary context-free grammars has been discovered by Earley [Ear68, Ear70]. Different variants of Earley's algorithm exist. First we investigate the one that is closest to CYK, the *bottom-up* Earley parser.

Example 2.2. (*bottom-up Earley*)
An Earley item has the form $[A{\to}\alpha{\bullet}\beta, i, j]$, with $A{\to}\alpha\beta \in P$. The bottom-up Earley parser recognizes the item set

$$\{[A{\to}\alpha{\bullet}\beta, i, j] \mid \alpha \Rightarrow^* a_{i+1} \ldots a_j\} \quad \text{for } G \text{ and } a_1 \ldots a_n.$$

A recognized item denotes partial recognition of a production. If $\beta = \varepsilon$, we have recognized a full production – and hence the left-hand side A, corresponding to $[A, i, j]$ in the CYK case. Partially recognized productions can be expanded by "moving the dot rightwards", i.e. recognizing the symbol behind the dot. How to organize this and store the results does not concern us here. We only specify the domain of items, the hypotheses and the deduction steps. For some grammar G and string $a_1 \ldots a_n$ we specify a parsing system $\mathbb{P}_{\mathrm{buE}}$ by[3]

$$\mathcal{I}_{\mathrm{buE}} = \{[A{\to}\alpha{\bullet}\beta, i, j] \mid A{\to}\alpha\beta \in P \ \wedge \ 0 \leq i \leq j\};$$

$$H_{\mathrm{buE}} = \{[a, i-1, i] \mid a = a_i \ \wedge \ 1 \leq i \leq n\};$$

$$D^{\mathrm{Init}} = \{\vdash [A{\to}{\bullet}\gamma, i, i]\},$$

$$D^{\mathrm{Scan}} = \{[A{\to}\alpha{\bullet}a\beta, i, j], [a, j, j+1] \vdash [A{\to}\alpha a{\bullet}\beta, i, j+1]\},$$

$$D^{\mathrm{Compl}} = \{[A{\to}\alpha{\bullet}B\beta, i, j], [B{\to}\gamma{\bullet}, j, k] \vdash [A{\to}\alpha B{\bullet}\beta, i, k]\},$$

---

[3] From the usual set notation $\{\ldots \mid \ldots\}$ we omit the second part if there are no further constraints on the elements that comprise the set. It should be evident (and it will be formally stated in Section 3.1) that only items are used that relate to productions of the grammar G.

$$D_{\mathrm{buE}} \quad = \quad D^{\mathrm{Init}} \cup D^{\mathrm{Scan}} \cup D^{\mathrm{Compl}}.$$

Deduction steps D^{Init} are needed to start the deduction of further valid items, hence these have no antecedents. In the definition of D^{Init} there is no need to state explicitly that $A \to \gamma \in P$ is required, as the deduction steps are only meaningful for items drawn from from $\mathcal{I}$ and H.

D^{Scan} and D^{Compl} conform to the *scan* and *complete* steps of Earley's algorithm. In Figure 2.1 it is sketched how the *complete* step produces an item representing a larger partial parse from two known partial parses.
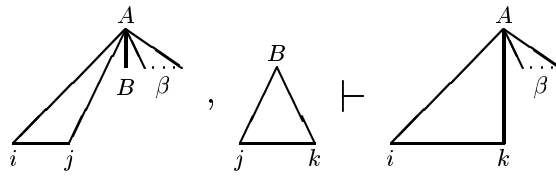

Fig. 2.1. The *complete* step

Earley's original algorithm is more restrictive in the items it recognizes. It makes use of top-down filtering. That is, the recognition of a production is started only if there is a need to do so. Only if we have an item $[A \to \alpha \bullet B\beta, i, j]$ there is a need to start recognizing a nonterminal B that produces $a_{j+1} \dots a_k$ for some k. Top-down filtering reduces the number of recognized items, but also reduces the possibilities for parallel processing. Earley's algorithm is essentially left-to-right. Initial items start at position 0 and a parser has to work its way rightwards, unlike the bottom-up case where one can start recognizing items at all positions in the sentence in parallel.

Example 2.3. (*canonical Earley*)
The parsing system $\mathbb{P}_{\mathrm{Earley}}$ for a given context-free grammar G and string $a_1 \dots a_n$ is defined by $\mathcal{I}$ and H as in $\mathbb{P}_{\mathrm{buE}}$ (cf. Example 2.2) and by D_{Earley} as follows:

$$D^{\mathrm{Init}} \quad = \quad \{ \vdash [S \to \bullet\gamma, 0, 0] \},$$

$$D^{\mathrm{Pred}} \quad = \quad \{ [A \to \alpha \bullet B\beta, i, j] \vdash [B \to \bullet\gamma, j, j] \},$$

$$D^{\mathrm{Scan}} \quad = \quad \{ [A \to \alpha \bullet a\beta, i, j], [a, j, j+1] \vdash [A \to \alpha a \bullet \beta, i, j+1] \},$$

$$D^{\mathrm{Compl}} \quad = \quad \{ [A \to \alpha \bullet B\beta, i, j], [B \to \gamma \bullet, j, k] \vdash [A \to \alpha B \bullet \beta, i, k] \},$$

$$D_{\mathrm{Earley}} \quad = \quad D^{\mathrm{Init}} \cup D^{\mathrm{Scan}} \cup D^{\mathrm{Compl}} \cup D^{\mathrm{Pred}}.$$

The Earley parsing system for G and $a_1 \dots a_n$ yields the following set of recognized items:

$$\{ [A \to \alpha \bullet \beta, i, j] \mid \alpha \Rightarrow^* a_{i+1} \dots a_j \ \wedge \ S \Rightarrow^* a_1 \dots a_i A\gamma \ \text{ for some } \gamma \}.$$

3. Parsing Schemata

Parsing systems and parsing schemata are formally introduced in Section 3.1 and 3.2, respectively. Section 3.3 discusses the nature of items and introduces a concept of parsing schema correctness.

3.1 Parsing Systems

Definition 3.1. (*parsing system*)
A parsing system $\mathbb{P}$ for some grammar G and string $a_1 \ldots a_n$ is a triple $\mathbb{P} = \langle \mathcal{I}, H, D \rangle$, in which

- $\mathcal{I}$ is a set of items[4], called the *domain* or the *item set* of $\mathbb{P}$;
- H is a finite set of items called the *hypotheses* of $\mathbb{P}$;
- $D \subseteq \wp_{fin}(H \cup \mathcal{I}) \times \mathcal{I}$ is a set of deduction steps.

Note that H need not be a subset of $\mathcal{I}$. $\wp_{fin}$ in the above definition denotes the powerset restricted to finite sets. As a more convenient notation for deduction steps, we write $\eta_1, \ldots, \eta_k \vdash \xi$ rather than $(\{\eta_1, \ldots, \eta_k\}, \xi)$. Furthermore if we have $Y = \{\eta_1, \ldots, \eta_k\}$, we may also write $Y \vdash \xi$ as an abbreviation for $\eta_1, \ldots, \eta_k \vdash \xi$.

To be formally correct, however, we make a distinction between the set of deduction steps D and the inference relation $\vdash$ on $\wp_{fin}(H \cup \mathcal{I}) \times \mathcal{I}$. We want the inference relation to have the following conventional property:

if $\eta_1, \ldots, \eta_k \vdash x$ holds, then also $\eta_1, \ldots, \eta_k, \zeta \vdash x$ for any ζ.

Therefore we define $\vdash$ as the closure of D under addition of antecedents to an inference:

Definition 3.2. (*inference relation* $\vdash$)
Let $\mathbb{P} = \langle \mathcal{I}, H, D \rangle$ be a parsing system. The relation $\vdash \subseteq \wp_{fin}(H \cup \mathcal{I}) \times \mathcal{I}$ is defined by

$Y \vdash \xi$ if $(Y', \xi) \in D$ for some $Y' \subseteq Y$.

Before we define the transitive closure of $\vdash$ we introduce the notion of a *deduction sequence* (which will be needed for some definitions and proofs in Section 4.2).

Definition 3.3. (*deduction sequence*)
Let $\mathbb{P} = \langle \mathcal{I}, H, D \rangle$ be a parsing system. We write $\mathcal{I}^+$ for the set of non-empty, finite sequences $\xi_1, \ldots, \xi_j$, with $j \geq 1$ and $\xi_i \in \mathcal{I}$ ($1 \leq i \leq j$).
A deduction sequence in $\mathbb{P}$ is a pair $(Y; \xi_1, \ldots, \xi_j) \in \wp_{fin}(H \cup \mathcal{I}) \times \mathcal{I}^+$, such that $Y \cup \{\xi_1, \ldots, \xi_{i-1}\} \vdash \xi_i$ for $1 \leq i \leq j$.

As a practical informal notation we write $Y \vdash \xi_1 \vdash \ldots \vdash \xi_j$ for a deduction sequence $(Y; \xi_1, \ldots, \xi_j)$.

---

[4] Here we treat 'item' as an undefined basic concept. A discussion about the nature of items follows in Section 3.3.

Definition 3.4. (Δ)

The *set of deduction sequences* $\Delta \subseteq \wp_{fin}(H \cup \mathcal{I}) \times \mathcal{I}^+$ for a parsing system $\mathbb{P} = \langle \mathcal{I}, H, D \rangle$ is defined by

$$\Delta \;=\; \{(Y; \xi_1, \ldots, \xi_j) \in \wp_{fin}(H \cup \mathcal{I}) \times \mathcal{I}^+ \;\mid\; Y \vdash \xi_1 \vdash \ldots \vdash \xi_j\}.$$

Definition 3.5. ($\vdash^*$)

For a parsing system $\mathbb{P} = \langle \mathcal{I}, H, D \rangle$ we define the relation $\vdash^*$ on $\wp_{fin}(H \cup \mathcal{I}) \times \mathcal{I}$ by

$$Y \vdash^* \xi \;\; \text{if} \;\; \xi \in Y \;\; \text{or} \;\; Y \vdash \ldots \vdash \xi.$$

Definition 3.6. (*valid items*)

For a parsing system $\mathbb{P} = \langle \mathcal{I}, H, D \rangle$ the set of valid items is defined by

$$\mathcal{V}(\mathbb{P}) \;=\; \{\xi \in \mathcal{I} \mid H \vdash^* \xi\}.$$

We do not make a distinction between semantic validity (usually denoted $\models \xi$) and syntactic provability (i.e. $H \vdash^* \xi$).

3.2 Parsing Schemata

A parsing system has been defined for a fixed grammar and string. In two steps we will extend this to a parsing schema for arbitrary grammmars and strings.

Definition 3.7. (*uninstantiated parsing system*)

An uninstantiated parsing system for a grammar G is a triple $\langle \mathcal{I}, \mathcal{H}, D \rangle$ with $\mathcal{H}$ a function that assigns a set of hypotheses to each string $a_1 \ldots a_n \in \Sigma^*$, such that $\langle \mathcal{I}, \mathcal{H}(a_1 \ldots a_n), D \rangle$ is a parsing system.

A function $\mathcal{H}$ that will be used throughout the remainder of this report (unless specifically stated otherwise) is

$$\mathcal{H}(a_1 \ldots a_n) = \{[a, i-1, i] \mid a = a_i \;\wedge\; 1 \le i \le n\}.$$

In the sequel we will omit the hypotheses H from the specification of a parsing system when the default $\mathcal{H}(a_1 \ldots a_n)$ applies.

Definition 3.8. (*parsing schema*)

A parsing schema for some (sub)class of context-free grammars $\mathcal{CG} \subseteq \mathcal{CFG}$ is a function that assigns an uninstantiated parsing system to every grammar $G \in \mathcal{CG}$.

Schema 3.9. (**CYK**)

The parsing schema **CYK** is defined for any $G \in \mathcal{CNF}$ and for any $a_1 \ldots a_n \in \Sigma^*$ by $\mathbf{CYK}(G)(a_1 \ldots a_n) = \mathbb{P}_{\text{CYK}}$ as in Example 2.1.

Schema 3.10. (**buE**)

The parsing schema **buE** is defined for any $G \in \mathcal{CFG}$ and for any $a_1 \ldots a_n \in \Sigma^*$ by $\mathbf{buE}(G)(a_1 \ldots a_n) = \mathbb{P}_{\text{buE}}$ as in Example 2.2.

Schema 3.11. (**Earley**)

The parsing schema **Earley** is defined for any $G \in \mathcal{CFG}$ and for any $a_1 \ldots a_n \in \Sigma^*$ by $\mathbf{Earley}(G)(a_1 \ldots a_n) = \mathbb{P}_{\text{Earley}}$ as in Example 2.3.

3.3 Correctness of Parsing Schemata

In order to define a notion of correctness, some understanding of the nature of items is needed. We have seen two kinds of items so far, there are other parsing algorithms that involve different kinds of items. What, exactly *is* an item?

An item lists a set of constraints on a (partial or complete) parse tree. Recognition of an Earley item $[A{\rightarrow}\alpha{\bullet}\beta, i, j]$ means: There is *some* tree that has a root labelled A with children labelled $\alpha\beta$ (concatenated from left to right). Moreover, the nodes labelled α are the roots of sub-trees that yield $a_{i+1} \ldots a_j$ whereas the nodes labelled β are leaves, cf. Figure 3.1.
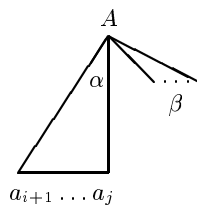

Fig. 3.1. A partially specified tree

One way to interpret an item is to identify it with a *set of trees*, viz., all trees that satisfy the constraints stated in the item. This approach is taken in [Sik93a]. Pursuing this line of thought, an item set is defined by a congruence relation on a set of trees with respect to the deduction relation.

A rather more simple approach is to regard an item as a partial specification of a tree. We assume that there is some general item specification language and that all items used in practical algorithms are (efficient notations for) specific instances of this specification language. We will not further formalize this, because in all practical cases it is abundantly clear what is meant by the various types of items.

Before we define correctness, there are two regularity properties on item sets that have to be stated explicitly.

Firstly, we have tacitly assumed that there is a clear separation between *final items*, denoting completed parse trees, and *intermediate items*, denoting partial, not yet completed trees.

It is possible − but admittedly rather artifical − to contruct *mixed items* that denote a combination of both types. Consider, for example, a grammar in Chomsky Normal Form that has productions $A{\rightarrow}SC$ and $A{\rightarrow}BC$, with S and B not occuring anywhere else in the right-hand side of a production. For the recognition of A, therefore, it is irrelevant whether $[S, i, j]$ or $[B, i, j]$ has been recognized. So we could replace these two items by a single item $[(S, B), i, j]$. But then we have a problem with the item $[(S, B), 0, n]$. If this item is recognized, it is unclear whether it denotes the existence of a parse tree.

Secondly, we assume that for each parse tree of a sentence, this parse tree conforms to the partial specification of some item in $\mathcal{I}$.

14

Definition 3.12. (*semiregularity*)[5]
A parsing system $\mathbb{P} = \langle \mathcal{I}, H, D \rangle$ for a grammar G and string $a_1 \ldots a_n$ is called semiregular if $\mathcal{I}$ does not contain mixed items and each parse tree of $a_1 \ldots a_n$ conforms to the specification of some item in $\mathcal{I}$.
A parsing schema $\mathbf{P}$ for a class of grammars $\mathcal{CG}$ is semiregular if $\mathbf{P}(G)(a_1 \ldots a_n)$ is semiregular for all $G \in \mathcal{CG}$ and all $a_1 \ldots a_n \in \Sigma^*$.

Definition 3.13. (*correct final items*)
We write $\mathcal{F}(\mathbb{P}) \subseteq \mathcal{I}$ for the set of the final items of a parsing system $\mathbb{P}$ for a grammar G and a string $a_1 \ldots a_n$.
A final item is correct if there is a parse tree for $a_1 \ldots a_n$ that conforms to the specification expressed by this item. We write $\mathcal{C}(\mathbb{P}) \subseteq \mathcal{F}(\mathbb{P})$ for the set of correct final items of $\mathbb{P}$.

Example 3.14. (*final and correct final items*)

- $\mathcal{F}(\mathbb{P}_{\mathrm{CYK}}) = \{[S, 0, n]\}$;
- $\mathcal{C}(\mathbb{P}_{\mathrm{CYK}}) = \{[S, 0, n]\}$ if $a_1 \ldots a_n \in L(G)$,
  $\mathcal{C}(\mathbb{P}_{\mathrm{CYK}}) = \emptyset$ if $a_1 \ldots a_n \notin L(G)$;
- $\mathcal{F}(\mathbb{P}_{\mathrm{buE}}) = \mathcal{F}(\mathbb{P}_{\mathrm{Earley}}) = \{[S \rightarrow \alpha \bullet, 0, n] \mid S \rightarrow \alpha \in P\}$;
- $\mathcal{C}(\mathbb{P}_{\mathrm{buE}}) = \mathcal{C}(\mathbb{P}_{\mathrm{Earley}}) = \{[S \rightarrow \alpha \bullet, 0, n] \mid \alpha \Rightarrow^* a_1 \ldots a_n\}$.

Definition 3.15. (*correctness of a parsing schema*)
A semiregular parsing system $\mathbb{P}$ is *sound* if $\mathcal{F}(\mathbb{P}) \cap \mathcal{V}(\mathbb{P}) \subseteq \mathcal{C}(\mathbb{P})$, i.e., all valid final items are correct.
A semiregular parsing system $\mathbb{P}$ is *complete* if $\mathcal{F}(\mathbb{P}) \cap \mathcal{V}(\mathbb{P}) \supseteq \mathcal{C}(\mathbb{P})$, i.e., all correct final items are valid.
A semiregular parsing system is correct if $\mathcal{F}(\mathbb{P}) \cap \mathcal{V}(\mathbb{P}) = \mathcal{C}(\mathbb{P})$, i.e., it is sound and complete.
A semiregular parsing schema $\mathbf{P}$ is sound/complete/correct for a class of grammars $\mathcal{CG}$ if $\mathbf{P}(G)(a_1 \ldots a_n)$ is sound/complete/correct for all $G \in \mathcal{CG}$ and $a_1 \ldots a_n \in \Sigma^*$.

CYK, **buE**, and **Earley** are correct semiregular parsing schemata (and so are the other schemata that will be proposed in the remainder of this report). This is well-known from the literature and we will not further explore the issue of how to prove the correctness of a parsing schema.

4. Generalization

Various kinds of relations between parsing algorithms can be formally established by defining relations between their underlying parsing schemata. In this section we will look at *generalization* of a schema, that can be obtained by *refinement* into a more detailed parsing schema and/or *extension* to a larger class of grammars.

---

[5] The notion *regularity* was introduced in [Sik93a] for parsing systems and schemata that do not contain inconsistent specifications, viz. the empty set of items. We do not need the regularity property in this context.

Adding detail to a schema means more (refined) items, more deduction steps, hence more work to parse a sentence. This is useful if it leads to *qualitative* improvements in the parsing algorithm. The canonical example (that we spell out first as an illustration) is that the bottom-up Earley parser is a generalization of the CYK parser.

4.1 Some Examples

More precisely, but still informally, we distinguish the following basic kinds of generalizations:

- A parsing schema $\mathbf{P}_2$ is an *item refinement* of a schema $\mathbf{P}_1$ if a single item in $\mathbf{P}_1$ is broken down into multiple items in $\mathbf{P}_2$ (and the set of deduction steps adapted accordingly);
- A parsing schema $\mathbf{P}_2$ is a *step refinement* of a schema $\mathbf{P}_1$ if a single deduction step in $\mathbf{P}_1$ is decomposed into a sequence of deduction steps in $\mathbf{P}_2$ (and new items are introduced, when needed, to store the refined intermediate results);
- A schema $\mathbf{P}_2$ is called an *extension* of a schema $\mathbf{P}_1$ if it is defined for a larger class of grammars.

A relation is called a *refinement* if it is a step refinement, an item refinement or a combination of these. A relation is called a *generalization* if it is a refinement, an extension or a combination of these. We write $\overset{\mathbf{ir}}{\Longrightarrow}$ for item refinement, $\overset{\mathbf{sr}}{\Longrightarrow}$ for step refinement, $\overset{\mathbf{ref}}{\Longrightarrow}$ for refinement, $\overset{\mathbf{ext}}{\Longrightarrow}$ for extension, and $\overset{\mathbf{gen}}{\Longrightarrow}$ for generalization.

Example 4.1. (CYK $\overset{\mathbf{gen}}{\Longrightarrow}$ buE)

In order to generalize **CYK** into **buE** we introduce two intermediate parsing systems **CYK**$'$ and **ECYK**, such that

$$\mathbf{CYK} \overset{\mathbf{ir}}{\Longrightarrow} \mathbf{CYK}' \overset{\mathbf{sr}}{\Longrightarrow} \mathbf{ECYK} \overset{\mathbf{ext}}{\Longrightarrow} \mathbf{buE}.$$

The only thing we change in **CYK**$'$ is that CYK items $[A, i, j]$ are replaced by completed Earley items $[A{\to}\alpha\bullet, i, j]$. We define **CYK**$'$ by specifying a parsing system $\mathbb{P}_{\mathrm{CYK'}}$ for an arbitrary grammar in $\mathcal{CNF}$ as follows:

$$
\begin{aligned}
\mathcal{I}_{\mathrm{CYK'}} &= \{[A{\to}\alpha\bullet, i, j] \mid A{\to}\alpha \in P \land 0 \le i \le j\}; \\
D^{(1)} &= \{[a, j-1, j] \vdash [A{\to}a\bullet, j-1, j]\}, \\
D^{(2)} &= \{[B{\to}\beta\bullet, i, j], [C{\to}\gamma\bullet, j, k] \vdash [A{\to}BC\bullet, i, k]\}, \\
D_{\mathrm{CYK'}} &= D^{(1)} \cup D^{(2)}.
\end{aligned}
$$

Note that a single CYK item $[A, i, j]$ may correspond to multiple Earley items $[A{\to}\alpha\bullet, i, j]$, $[A{\to}\beta\bullet, i, j]$, etc., if there are different productions with left-hand side A. That is why this is an item refinement, not merely a change of notation.

In the next step, we refine a single CYK deduction step

$$[B{\to}\beta\bullet, i, j], [C{\to}\gamma\bullet, j, k] \vdash [A{\to}BC\bullet, i, k]$$

into a sequence of deduction steps

$$\vdash [A{\to}\bullet BC, i, i],$$
$$[A{\to}\bullet BC, i, i], [B{\to}\beta\bullet, i, j] \vdash [A{\to}B\bullet C, i, j],$$
$$[A{\to}B\bullet C, i, j], [C{\to}\gamma\bullet, j, k] \vdash [A{\to}BC\bullet, i, k].$$

This is encorporated in the parsing schema **ECYK**, defined by a parsing system $\mathbb{P}_{\text{ECYK}}$ for an arbitrary grammar in $\mathcal{CNF}$:

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

$$D^{\text{Init}} = \{\vdash [A{\to}\bullet\alpha, j, j]\},$$

$$D^{\text{Scan}} = \{[A{\to}\alpha\bullet a\beta, i, j], [a, j, j+1] \vdash [A{\to}\alpha a\bullet\beta, i, j+1]\},$$

$$D^{\text{Compl}} = \{[A{\to}\alpha\bullet B\beta, i, j], [B{\to}\gamma\bullet, j, k] \vdash [A{\to}\alpha B\bullet\beta, i, k]\},$$

$$D_{\text{ECYK}} = D^{\text{Init}} \cup D^{\text{Scan}} \cup D^{\text{Compl}}.$$

ECYK is identical to **buE**, cf. Example 2.2, except for the fact that **ECYK** is defined only for grammars in $\mathcal{CNF}$. Hence, obviously, **buE** is an extension of **ECYK**.

4.2 Formalization

We will now formalize the concepts that have been informally introduced and illustrated above. In the sequel we write $\mathbb{P}_i$ for a parsing system $\mathbb{P}_i = \langle \mathcal{I}_i, H, D_i \rangle$; we write $\vdash_i$ and $\vdash_i^*$ for an inference relation and its closure, based on D_i, cf. Section 3.1.

For the definition of item refinement we make use of an *item mapping* $f : \mathcal{I}_2 {\to} \mathcal{I}_1$ that maps items of $\mathbb{P}_2$ to items of $\mathbb{P}_1$. The function f can be extended to cover sets of items in the usual way: For $Y \subseteq \mathcal{I}_2$ we define

$$f(Y) = \{\xi \in \mathcal{I}_1 \mid \exists \eta \in Y : f(\eta) = \xi\}$$

Moreover, we extend[6] f to a function $f : \mathcal{I}_2 \cup H {\to} \mathcal{I}_1 \cup H$ by letting $f(h) = h$ for $h \in H$. Then we can apply f to deduction steps by letting

$$f(\eta_1, \ldots, \eta_k \vdash \xi) = f(\eta_1), \ldots, f(\eta_k) \vdash f(\xi).$$

In the same fashion we can extend f to deduction sequences[7], to sets of deduction steps and to sets of deduction sequences. The equation

$$\Delta_1 = f(\Delta_2)$$

is a clear and concise notation for: $Y_1 \vdash_1 x_1 \vdash_1 \ldots \vdash_1 x_j$ if and only if there are $Y_2 \in \wp_{fin}(H_2 \cup \mathcal{I}_2)$ and $x_2', \ldots, x_j' \in \mathcal{I}_2$ with $f(Y_2) = Y_1$ and $f(x_i) = x_i'$ for $1 \le i \le j$, such that $Y_2 \vdash_2 x_1' \vdash_1 \ldots \vdash_1 x_j'$.

---

[6] Assuming that $H \cap \mathcal{I} = \emptyset$; otherwise we demand that f restricted to $H \cap \mathcal{I}$ is the identity function.

[7] Note, however, that the image of a deduction sequence is not necessarily a deduction sequence (cf. Definition 3.3), hence $f : \Delta_2 {\to} \Delta_1$ and $f : \wp(\Delta_2) {\to} \wp(\Delta_1)$ are partial functions.

Definition 4.2. (*item refinement*)

The relation $\mathbb{P}_1 \xrightarrow{\mathbf{ir}} \mathbb{P}_2$ holds between parsing systems $\mathbb{P}_1$ and $\mathbb{P}_2$ if there is an item mapping $f : \mathcal{I}_2 \rightarrow \mathcal{I}_1$ such that

(i) $\mathcal{I}_1 = f(\mathcal{I}_2)$,
(ii) $\Delta_1 = f(\Delta_2)$,

Let $\mathbf{P}_1$ and $\mathbf{P}_2$ be parsing schemata for some class of grammars $\mathcal{CG}$. The relation $\mathbf{P}_1 \xRightarrow{\mathbf{ir}} \mathbf{P}_2$ holds if $\mathbf{P}_1(G)(a_1 \ldots a_n) \xrightarrow{\mathbf{ir}} \mathbf{P}_2(G)(a_1 \ldots a_n)$ for all $G \in \mathcal{CG}$ and for all $a_1 \ldots a_n$.

The first condition in Definition 4.2 states that no items are 'lost' in the refinement; the second condition ensures that deduction sequences are carried over into the refined system. Deduction sequences are needed in the definition of refinement, in order to guarantee the transitivity of generalization. A weaker condition

(ii)$'$ $\vdash_1^* = \vdash_2^*$,

which might seem sufficient, will not do. An example of a relation that satisfies (i) and (ii)$'$ but is not a refinement is given by

$$\mathbb{P}_1 = \langle \{\xi, \eta, \zeta\}, \{h\}, \{h \vdash \xi, \ h \vdash \eta, \ \xi \vdash \zeta, \ \eta \vdash \zeta\} \rangle,$$

$$\mathbb{P}_2 = \langle \{\xi_1, \xi_2, \eta, \zeta\}, \{h\}, \{h \vdash \xi_1, \ h \vdash \eta, \ \xi_2 \vdash \zeta, \ \eta \vdash \zeta\} \rangle,$$

with $f(\xi_i) = \xi$, f is the identity function otherwise.

Definition 4.3. (*step refinement*)

The relation $\mathbb{P}_1 \xrightarrow{\mathbf{sr}} \mathbb{P}_2$ holds between parsing systems $\mathbb{P}_1$ and $\mathbb{P}_2$ if

(i) $\mathcal{I}_1 \subseteq \mathcal{I}_2$,
(ii) $\vdash_1^* \subseteq \vdash_2^*$.

Let $\mathbf{P}_1$ and $\mathbf{P}_2$ be parsing schemata for some class of grammars $\mathcal{CG}$. The relation $\mathbf{P}_1 \xRightarrow{\mathbf{sr}} \mathbf{P}_2$ holds if $\mathbf{P}_1(G)(a_1 \ldots a_n) \xrightarrow{\mathbf{sr}} \mathbf{P}_2(G)(a_1 \ldots a_n)$ for all $G \in \mathcal{CG}$ and for all $a_1 \ldots a_n$.

A sufficient condition[8] for (ii) in Definition 4.3 is $D_1 \subseteq \vdash_2^*$, that is, a single deduction step in $\mathbb{P}_1$ is emulated by a sequence of deduction steps in $\mathbb{P}_2$. Furthermore, the domain of $\mathbb{P}_2$ may contain items that did not exist in $\mathbb{P}_1$.

Definition 4.4. (*refinement*)

Let $\mathbf{P}_1$ and $\mathbf{P}_2$ be parsing schemata for a class of grammars $\mathcal{CG}$. The relation $\mathbf{P}_1 \xRightarrow{\mathbf{ref}} \mathbf{P}_2$ holds if there is a parsing schema $\mathbf{P}'$ such that $\mathbf{P}_1 \xRightarrow{\mathbf{ir}} \mathbf{P}' \xRightarrow{\mathbf{sr}} \mathbf{P}_2$.

Definition 4.5. (*extension*)

Let $\mathbf{P}_1$ be a parsing schema for a class of grammars $\mathcal{CG}_1$, $\mathbf{P}_2$ a parsing schema for a class of grammars $\mathcal{CG}_2$ and $\mathcal{CG}_1 \subseteq \mathcal{CG}_2$. The relation $\mathbf{P}_1 \xRightarrow{\mathbf{ext}} \mathbf{P}_2$ holds if $\mathbf{P}_1(G) = \mathbf{P}_2(G)$ for all $G \in \mathcal{CG}_1$.

---

[8] We write $\vdash_1^*$ just for symmetry, because all other relations defined in this section and the next one display the same kind of symmetry.

Definition 4.6. (*generalization*)
Let $\mathbf{P}_1$ be a parsing schema for a class of grammars $\mathcal{CG}_1$, $\mathbf{P}_2$ a parsing schema for a class of grammars $\mathcal{CG}_2$ and $\mathcal{CG}_1 \subseteq \mathcal{CG}_2$. The relation $\mathbf{P}_1 \overset{\textbf{gen}}{\Longrightarrow} \mathbf{P}_2$ holds if there is parsing schema $\mathbf{P}'$ such that $\mathbf{P}_1 \overset{\textbf{ref}}{\Longrightarrow} \mathbf{P}' \overset{\textbf{ext}}{\Longrightarrow} \mathbf{P}_2$.

4.3 Properties of Generalization

Proposition 4.7.
Each of the relations $\overset{\textbf{ir}}{\Longrightarrow}$, $\overset{\textbf{sr}}{\Longrightarrow}$, $\overset{\textbf{ext}}{\Longrightarrow}$ is transitive and reflexive. $\qquad\square$

If xR_1y implies xR_2y for relations R_1, R_2, we write $R_1 \subseteq R_2$ (the set inclusion is in the Cartesian product of the domain of the relations).

Corollary 4.8.

$(a)\quad \overset{\textbf{ir}}{\Longrightarrow} \ \subseteq \ \overset{\textbf{ref}}{\Longrightarrow}$;

$(b)\quad \overset{\textbf{sr}}{\Longrightarrow} \ \subseteq \ \overset{\textbf{ref}}{\Longrightarrow}$;

$(c)\quad \overset{\textbf{ref}}{\Longrightarrow} \ \subseteq \ \overset{\textbf{gen}}{\Longrightarrow}$;

$(d)\quad \overset{\textbf{ext}}{\Longrightarrow} \ \subseteq \ \overset{\textbf{gen}}{\Longrightarrow}$.

Next, we will establish the transitivity of $\overset{\textbf{ref}}{\Longrightarrow}$ and $\overset{\textbf{gen}}{\Longrightarrow}$. The former is not entirely trivial.

Lemma 4.9. (*refinement lemma*)
Let $\mathbb{P}_1$, $\mathbb{P}_2$, $\mathbb{P}_3$ be parsing systems such that $\mathbb{P}_1 \overset{\textbf{sr}}{\longrightarrow} \mathbb{P}_2 \overset{\textbf{ir}}{\longrightarrow} \mathbb{P}_3$. Then there is a parsing system $\mathbb{P}_4$ such that $\mathbb{P}_1 \overset{\textbf{ir}}{\longrightarrow} \mathbb{P}_4 \overset{\textbf{sr}}{\longrightarrow} \mathbb{P}_3$.
Let $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$ be parsing schemata for some class of grammars $\mathcal{CG}$, such that $\mathbf{P}_1 \overset{\textbf{sr}}{\Longrightarrow} \mathbf{P}_2 \overset{\textbf{ir}}{\Longrightarrow} \mathbf{P}_3$. Then there is a parsing schema $\mathbf{P}_4$ for $\mathcal{CG}$ such that $\mathbf{P}_1 \overset{\textbf{ir}}{\Longrightarrow} \mathbf{P}_4 \overset{\textbf{sr}}{\Longrightarrow} \mathbf{P}_3$.

Proof. It suffices to prove the lemma for parsing systems, the generalization to parsing schemata is trivial. Let $f : \mathcal{I}_3 \to \mathcal{I}_2$ be the item mapping from $\mathbb{P}_3$ to $\mathbb{P}_2$. Then we define $\mathbb{P}_4$ by

$$\mathcal{I}_4 \ = \ \{x \in \mathcal{I}_3 \mid f(x) \in \mathcal{I}_1\},$$

$$D_4 \ = \ \{(Y, x) \in \wp_{fin}(H \cup \mathcal{I}_4) \times \mathcal{I}_4 \mid f((Y, x)) \in D_1 \wedge Y \vdash_3^* x\}.$$

We now have to show that $\mathbb{P}_4$ is a parsing schema, that $\mathbb{P}_1 \overset{\textbf{ir}}{\longrightarrow} \mathbb{P}_4$ holds, and that $\mathbb{P}_4 \overset{\textbf{sr}}{\longrightarrow} \mathbb{P}_2$ holds. We prove $\mathbb{P}_1 \overset{\textbf{ir}}{\longrightarrow} \mathbb{P}_4$, as an exemplary case (the only one that needs some care in spelling out the details) and omit the other parts.
From the definition of $\mathbb{P}_4$ it is clear that $\mathcal{I}_1 = f(\mathcal{I}_4)$, hence it remains to be shown that $\Delta_1 = f(\Delta_4)$.

$(i)\quad f(\Delta_4) \subseteq \Delta_1$.
This follows from the definition of $\mathbb{P}_4$ with induction on the length of deduction sequences.

(*ii*) $\Delta_1 \subseteq f(\Delta_4)$.

We use an ad-hoc notation $Y \vdash^* x_1 \vdash^* \ldots \vdash^* x_j \in \Delta$, meaning that there are (possibly empty) sequences $z_{i,1}, \ldots, z_{i,m_i}$ for $1 \le i \le j$ such that $Y \vdash z_{1,1} \vdash \ldots \vdash z_{1,m_1} \vdash x_1 \vdash \ldots \vdash z_{j,1} \vdash \ldots \vdash z_{j,m_j} \vdash x_j \in \Delta$.

Assume now that $Y \vdash_1 x_1 \vdash_1 \ldots \vdash_1 x_j \in \Delta_1$. Because $\mathbb{P}_1 \xrightarrow{\mathbf{sr}} \mathbb{P}_2$ it must hold that $Y \vdash_2^* x_1 \vdash_2^* \ldots \vdash_2^* x_j \in \Delta_2$. Moreover, from $\mathbb{P}_2 \xrightarrow{\mathbf{ir}} \mathbb{P}_3$ we obtain $Y' \in \wp_{fin}(H \cup \mathcal{I}_3) \times \mathcal{I}_3$ with $f(Y') = Y$ and $x'_1, \ldots, x'_j$ with $f(x'_1) = x_1, \ldots, f(x'_j) = x_j$, such that $Y' \vdash_3^* x'_1 \vdash_3^* \ldots \vdash_3^* x'_j \in \Delta_3$. From the definition of $\mathbb{P}_4$ it follows that $Y' \vdash_4 x'_1 \vdash_4 \ldots \vdash_4 x'_j \in \Delta_4$. Thus we have shown $Y \vdash_1 x_1 \vdash_1 \ldots \vdash_1 x_j \in f(\Delta_4)$ which proves (*ii*). $\square$

Lemma 4.10.
Let $\mathbf{P}_1$ be a parsing schema for a class of grammar $\mathcal{CG}_1$ and $\mathbf{P}_2$, $\mathbf{P}_3$ be parsing schemata for a class of grammar $\mathcal{CG}_2$, such that $\mathbf{P}_1 \xrightarrow{\mathbf{ext}} \mathbf{P}_2 \xrightarrow{\mathbf{ref}} \mathbf{P}_3$. Then there is a parsing schema $\mathbf{P}_4$ such that $\mathbf{P}_1 \xrightarrow{\mathbf{ref}} \mathbf{P}_4 \xrightarrow{\mathbf{ext}} \mathbf{P}_3$.

Proof. Define $\mathbf{P}_4(G) = \mathbf{P}_2(G)$ for $G \in \mathcal{CG}_1$. $\square$

Theorem 4.11. The relation $\xrightarrow{\mathbf{gen}}$ is transitive and reflexive.

Proof. Straightforward from Lemmata 4.9 and 4.10. $\square$

Correctness of parsing schemata is, in general, not preserved by generalization. A useful partial result is that completeness is preserved by $\xrightarrow{\mathbf{sr}}$, that is, if $\mathbf{P}_1$ is a complete semiregular parsing schema and $\mathbf{P}_1 \xrightarrow{\mathbf{sr}} \mathbf{P}_2$, then $\mathbf{P}_2$ is a complete semiregular parsing schema.

5. Filtering

Generalization increases the number of steps that have to be performed, but the more fine-grained look on the parsing process may allow qualitative improvements. Filtering is, in a way, the reverse. The purpose is to obtain quantitative improvements in parsing algorithms, by decreasing the number of items and deduction steps. It is often possible to argue that some kinds of items need not be recognized, because they cannot contribute to a valid parse. Discarding those items from the parsing schema means less work for the algorithm that implements the schema, but sometimes a more complicated description of the schema.

We distinguish three kinds of filtering:

— *static filtering*: redundant parts of a parsing schema are simply discarded;
— *dynamic filtering*: the validity of some items can be made dependent on the validity of other items, hence context information can be taken into account;
— *step contraction*: sequences of deduction steps are replaced by single deduction steps.

The theoretical framework is simple and elegant. As in the previous section we assume that a parsing system $\mathbb{P}_i$ is defined as $\langle \mathcal{I}_i, H, D_i \rangle$, with inference relations $\vdash_i$ and $\vdash_i^*$ on $\mathbb{P}_i$ according to Section 3.1.

5.1 Static Filtering

Static filtering can be demonstrated by means of a very simple example (a more exciting example will follow in Section 6.2).

A nonterminal $A \in N$ is called *reduced* if

(*i*) there are $v, w \in \Sigma^*$ such that $S \Rightarrow^* vAw$,
(*ii*) there is some $w \in \Sigma^*$ such that $A \Rightarrow^* w$.

A grammar is called reduced if all its nonterminals are reduced. Let $G \in \mathcal{CFG}$ be an arbitrary context-free grammar. We can define a reduced grammar G' by

$$N' = \{A \in N \mid A \text{ is reduced}\}$$

$$P' = \{A{\rightarrow}\alpha \in P \mid A \in N' \ \wedge \ \alpha \in (N' \cup \Sigma)^*\}$$

$$G' = (N', \Sigma, P', S).$$

If G is reduced, then $G = G'$. Furthermore, it is clear that G and G' yield the same parse trees for any sentence.

Example 5.1. (*reduced* **buE**)
Let $\mathbb{P}_{\mathrm{buE}}$ be a parsing system for some grammar G and string $a_1 \ldots a_n$. We define a parsing system $\mathbb{P}_{\mathrm{buE}'}$ by

$$\mathcal{I}_{\mathrm{buE}'} = \{[A{\rightarrow}\alpha_\bullet\beta, i, j] \mid A{\rightarrow}\alpha\beta \in P'\}$$

and $D_{\mathrm{buE}'}$ as in Example 2.2. Because $D \subseteq \wp_{\mathit{fin}}(H \cup \mathcal{I}) \times \mathcal{I}$ by definition, only deduction steps remain that contain non-reduced nonterminals wherever applicable.
A parsing schema **buE**′ is defined for all $G \in \mathcal{CFG}$ and $a_1 \ldots a_n \in \Sigma^*$ by $\mathbf{buE}'(G)(a_1 \ldots a_n) = \mathbb{P}_{\mathrm{buE}'}$, as usual.

Definition 5.2. (*static filtering*)
The relation $\mathbb{P}_1 \xrightarrow{\mathbf{sf}} \mathbb{P}_2$ holds if

(*i*) $\mathcal{I}_1 \supseteq \mathcal{I}_2$
(*ii*) $D_1 \supseteq D_2$.

Let $\mathbf{P}_1$ and $\mathbf{P}_2$ be parsing schemata for some class of grammars $\mathcal{CG}$.
The relation $\mathbf{P}_1 \xRightarrow{\mathbf{sf}} \mathbf{P}_2$ holds if $\mathbf{P}_1(G)(a_1 \ldots a_n) \xrightarrow{\mathbf{sf}} \mathbf{P}_2(G)(a_1 \ldots a_n)$ for all $G \in \mathcal{CG}$ and for all $a_1 \ldots a_n$.

5.2 Dynamic Filtering

The purpose of dynamic filtering is to take context information into account. If some type of constituent can only occur directly after another type of constituent, we may defer recognizing the former until we have established the latter. The technique to do this is to add antecedents to deduction steps. If we decided that an item ξ is to be valid only if some other item ζ is also valid, we simply replace deduction steps $\eta_1, \ldots, \eta_k \vdash \xi$ by deduction steps $\eta_1, \ldots, \eta_k, \zeta \vdash \xi$.

Definition 5.3. (*dynamic filtering*)
The relation $\mathbb{P}_1 \xrightarrow{\text{df}} \mathbb{P}_2$ holds if

(*i*) $\mathcal{I}_1 \supseteq \mathcal{I}_2$
(*ii*) $\vdash_1 \supseteq \vdash_2$.

Let $\mathbf{P}_1$ and $\mathbf{P}_2$ be parsing schemata for some class of grammars $\mathcal{CG}$.
The relation $\mathbf{P}_1 \xRightarrow{\text{df}} \mathbf{P}_2$ holds if $\mathbf{P}_1(G)(a_1 \ldots a_n) \xrightarrow{\text{df}} \mathbf{P}_2(G)(a_1 \ldots a_n)$ for all $G \in \mathcal{CG}$ and for all $a_1 \ldots a_n$.

Proposition 5.4. buE $\xRightarrow{\text{sf}}$ Earley.

Proof. We consider the parsing systems $\mathbb{P}_{\text{buE}}$ and $\mathbb{P}_{\text{Earley}}$ for some given G and $a_1 \ldots a_n$. By the definition of the two parsing systems (cf. Examples 2.3 and 2.2), $\mathcal{I}_{\text{buE}} = \mathcal{I}_{\text{Earley}}$ holds. In order to prove $\vdash_{\text{buE}} \supseteq \vdash_{\text{Earley}}$, it suffices to show that $\vdash_{\text{buE}} \supseteq D_{\text{Earley}}$. For each deduction step $\mathbb{P}_{\text{Earley}}$ we show that it is an inference in $\mathbb{P}_{\text{buE}}$.
Every *init*, *scan*, and *complete* step in $\mathbb{P}_{\text{Earley}}$ also exists in $\mathbb{P}_{\text{buE}}$. Only the Earley *predict* steps have to be accounted for. Let $[A \rightarrow \alpha \bullet \beta, i, j] \vdash_{\text{Earley}} [B \rightarrow \bullet \gamma, j, j]$ be such a *predict* step. Then $\mathbb{P}_{\text{buE}}$ contains an *init* step $\vdash_{\text{buE}} [B \rightarrow \bullet \gamma, j, j]$, hence, by definition of the inference relation $\vdash$, it holds that $[A \rightarrow \alpha \bullet \beta, i, j] \vdash_{\text{buE}} [B \rightarrow \bullet \gamma, j, j]$. $\qquad\square$

Another example of dynamic filtering is the application of *look-ahead*. Recognition of an item does not need to take place if the next symbol(s) in the string cannot logically follow, given the context of the item. For the sake of convenience, we augment the grammar with an *end-of-sentence marker* $\$$ and a new start symbol S'. Assuming $\{S', \$\} \cap V = \emptyset$, we define

$$N' = N \cup \{S'\},$$

$$\Sigma' = \Sigma \cup \{\$\},$$

$$P' = P \cup \{S' \rightarrow S\$\},$$

$$G' = (N', \Sigma', P', S').$$

There is only a single final item: $[S' \rightarrow S \bullet \$, 0, n]$. Furthermore, we define the function $\text{FOLLOW} : N \rightarrow \wp(\Sigma')$ by

$$\text{FOLLOW}(A) = \{a \mid \exists \alpha, \beta : S' \Rightarrow^* \alpha A a \beta\}.$$

Schema 5.5. (E(1))
The parsing schema **E(1)** is defined by a parsing system $\mathbb{P}_{\mathrm{E}(1)}$ for any $G \in \mathcal{CFG}$ and for any $a_1 \ldots a_n \in \Sigma^*$ by

$$\mathcal{I}^{\mathrm{Compl}} = \{[A{\to}\alpha\bullet\beta, i, j] \mid A{\to}\alpha\beta \in P' \wedge 0 \leq i \leq j\};$$

$$H = \{[a, i-1, i] \mid a = a_i \ \wedge \ 1 \leq i \leq n\} \cup \{[\$, n, n+1]\};$$

$$D^{\mathrm{Init}} = \{ \vdash [S'{\to}\bullet S\$, 0, 0]\},$$

$$D^{\mathrm{Pred}} = \{[A{\to}\alpha\bullet B\beta, i, j] \ \vdash \ [B{\to}\bullet\gamma, j, j]\},$$

$$D^{\mathrm{Scan}} = \{[A{\to}\alpha\bullet a\beta, i, j], [a, j, j+1] \ \vdash \ [A{\to}\alpha a\bullet\beta, j, j+1]\},$$

$$D^{\mathrm{Compl}} = \{[A{\to}\alpha\bullet B\beta, h, i], [B{\to}\gamma\bullet, i, j], [a, j, j+1] \\ \vdash \ [A{\to}\alpha B\bullet\beta, h, j] \ \mid \ a \in \textsc{Follow}(B)\},$$

$$D_{\mathrm{E}(1)} = D^{\mathrm{Init}} \cup D^{\mathrm{Pred}} \cup D^{\mathrm{Scan}} \cup D^{\mathrm{Compl}}.$$

The astute reader may wonder why the look-ahead is restricted to $a \in \textsc{Follow}(B)$ and not extended to, for example, $a \in \textsc{First}(\beta\,\textsc{Follow}(A))$. A similar filter, moreover, could be applied to the *scan* steps. This schema incorporates the look-ahead that is used in the construction of an SLR(1) parsing table. It can be shown that an SLR(1) parser is an implementation of **E(1)**[9]. More examples of dynamic filtering will follow in Sections 6.2

A few important remarks must be made about static and dynamic filtering.

Firstly, dynamic filtering reduces the number of valid items, but at the same time reduces the possibilities for parallel processing. The bottom-up Earley parser has been introduced as a non-filtered version of Earley's algorithm, specifically because it can be carried out in parallel in a straightforward manner.

Secondly, the two types of filters refer to different optimization techniques in parser implementation. Static (i.e. *compile-time*) optimization can take the specific grammar structure into account, but is necessarily unrelated to the sentence. Dynamic optimization is *run-time*, and hence can take into account those parts of the sentence that have been analysed already. It is exactly this difference that is expressed on a higher level of abstraction.

Note that every static filter is also a dynamic filter. This means that any static optimization could also be done run-time, rather than compile-time (but the former is generally less efficient).

5.3 Step Contraction

The last and most powerful type of filtering is step contraction. This is the inverse of step refinement, cf. Definition 4.3.

---

[9] Note, however, that a deterministic SLR(1) parser is defined only for a suitably small subclass of context-free grammars. See also Section 7.

Definition 5.6. (*step contraction*)
The relation $\mathbb{P}_1 \xrightarrow{\mathbf{sc}} \mathbb{P}_2$ holds if

(*i*) $\mathcal{I}_1 \supseteq \mathcal{I}_2$
(*ii*) $\vdash_1^* \supseteq \vdash_2^*$.

Let $\mathbf{P}_1$ and $\mathbf{P}_2$ be parsing schemata for some class of grammars $\mathcal{CG}$.
The relation $\mathbf{P}_1 \xRightarrow{\mathbf{sc}} \mathbf{P}_2$ holds if $\mathbf{P}_1(G)(a_1 \ldots a_n) \xrightarrow{\mathbf{sc}} \mathbf{P}_2(G)(a_1 \ldots a_n)$ for all $G \in \mathcal{CG}$ and for all $a_1 \ldots a_n$.

As a realistic example we give the parsing schema that underlies the improved Earley algorithm of Graham, Harrison and Ruzzo [GHR80]. This is a combination of two different step contractions:

- *nullable symbols* (i.e. symbols that can be rewritten to the empty string) can be skipped when the dot is worked rightwards through a production;
- *chain derivations* (i.e. derivations of the form $A \Rightarrow^+ B$) are reduced to single steps.

Schema 5.7. (GHR)
The parsing schema **GHR** is defined by a parsing system $\mathbb{P}_{\mathrm{GHR}}$ for any $G \in \mathcal{CFG}$ and for any $a_1 \ldots a_n \in \Sigma^*$ by

$$\mathcal{I}_{\mathrm{GHR}} = \{[A{\rightarrow}\alpha{\bullet}\beta, i, j] \mid A{\rightarrow}\alpha\beta \in P \wedge 0 \leq i \leq j\};$$

$$D^{\mathrm{Init}} = \{\vdash [S{\rightarrow}\beta{\bullet}\gamma, 0, 0] \mid \beta \Rightarrow^* \varepsilon\},$$

$$D^{\mathrm{Scan}} = \{[A{\rightarrow}\alpha{\bullet}a\beta\gamma, i, j], [a, j, j+1] \vdash [A{\rightarrow}\alpha a\beta{\bullet}\gamma, i, j+1] \\ \mid \beta \Rightarrow^* \varepsilon\},$$

$$D^{\mathrm{C1}} = \{[A{\rightarrow}\alpha{\bullet}B\beta\gamma, i, j], [B{\rightarrow}\delta{\bullet}, j, k] \vdash [A{\rightarrow}\alpha B\beta{\bullet}\gamma, i, k] \\ \mid i < j < k \wedge \beta \Rightarrow^* \varepsilon\},$$

$$D^{\mathrm{C2}} = \{[A{\rightarrow}\alpha{\bullet}B\beta\gamma, i, i], [C{\rightarrow}\delta{\bullet}, i, j] \vdash [A{\rightarrow}\alpha B\beta{\bullet}\gamma, i, j] \\ \mid i < j \wedge B \Rightarrow^* C \wedge \beta \Rightarrow^* \varepsilon\},$$

$$D^{Pred} = \{[A{\rightarrow}\alpha{\bullet}B\beta, i, j] \vdash [C{\rightarrow}\alpha'{\bullet}\beta', j, j] \mid B \Rightarrow^* C\gamma \wedge \alpha' \Rightarrow^* \varepsilon\},$$

$$D_{\mathrm{GHR}} = D^{\mathrm{Init}} \cup D^{\mathrm{Scan}} \cup D^{\mathrm{C1}} \cup D^{\mathrm{C2}} \cup D^{\mathrm{Pred}}.$$

Proposition 5.8. Earley $\xRightarrow{\mathbf{sc}}$ GHR. □

Other examples of step refinement follow in Section 6.1.

5.4 Properties of Filtering Relations

Unlike similar properties of refinement and generalization, the following are trivial.

Proposition 5.9. $\xRightarrow{\mathbf{sf}} \subseteq \xRightarrow{\mathbf{df}} \subseteq \xRightarrow{\mathbf{sc}}$. □

Proposition 5.10. $\xRightarrow{\mathbf{sf}}$, $\xRightarrow{\mathbf{df}}$, and $\xRightarrow{\mathbf{sc}}$ are transitive and reflexive. □

Proposition 5.11. $\xRightarrow{\mathbf{sf}}$, $\xRightarrow{\mathbf{df}}$, and $\xRightarrow{\mathbf{sc}}$ preserve soundness. □

6. Some Larger Examples

The emphasis in the previous sections was on formalizing the theoretical concepts. In 6.1–6.3 we present some nontrivial examples of parsing schemata and relations between them. In 6.4 we review the value of these exercises.

6.1 Left-Corner Parsing

As a more elaborate example of how filters can be used to relate parsing schemata to one another, we will precisely establish the relation between Earley parsing and Left-Corner parsing. We define a parsing schema **LC** that underlies the (generalized) Left-Corner algorithm that is known from the literature, cf. [M&a83, Ned93] (as opposed to *deterministic* LC parsing [RL70]). Along the way will show that **Earley** $\overset{\text{sc}}{\Longrightarrow}$ **LC**. As a conceptual aid, we will first consider a "bottom-up Left-Corner" parser that is a rather trivial step contraction of bottom-up Earley.

Consider an item of the form $[A{\rightarrow}B{\bullet}\beta, i, j]$ in $\mathbb{P}_{\text{buE}}$. The item is valid if some $[B{\rightarrow}\gamma{\bullet}, i, j]$ is valid, because $[A{\rightarrow}{\bullet}B\beta, i, i]$ is valid by definition. So we can contract the sequence of deduction steps

$$\vdash \quad [A{\rightarrow}{\bullet}B\beta, i, i],$$
$$[A{\rightarrow}{\bullet}B\beta, i, i], [B{\rightarrow}\gamma{\bullet}, i, j] \quad \vdash \quad [A{\rightarrow}B{\bullet}\beta, i, j].$$

to a single deduction step

$$[B{\rightarrow}\gamma{\bullet}, i, j] \quad \vdash \quad [A{\rightarrow}B{\bullet}\beta, i, j].$$

A similar argument applies to items of the form $[A{\rightarrow}a{\bullet}\beta, i, j]$ and the appropriate *scan* step. The (bottom-up) *left-corner* step is illustrated in Figure 6.1. This is incorporated in the following *bottom-up left-corner* parsing schema.
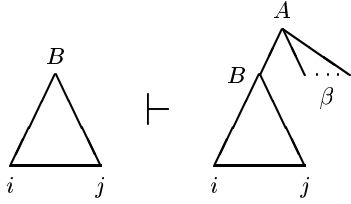

Fig. 6.1. The (bottom-up) *left-corner* step

Schema 6.1. (buLC)
The parsing schema **buLC** is defined by a parsing system $\mathbb{P}_{\text{buLC}}$ for any $G \in \mathcal{CFG}$ and for any $a_1 \ldots a_n \in \Sigma^*$ by

$$\mathcal{I}^{(1)} \quad = \quad \{[A{\rightarrow}X\alpha{\bullet}\beta, i, j] \mid A{\rightarrow}X\alpha\beta \in P \wedge 0 \leq i \leq j\},$$

$$\mathcal{I}^{(2)} \quad = \quad \{[A{\rightarrow}{\bullet}, j, j] \mid A{\rightarrow}\varepsilon \in P \wedge j \geq 0\},$$

$$\mathcal{I}_{\text{buLC}} \quad = \quad \mathcal{I}^{(1)} \cup \mathcal{I}^{(2)};$$

$$D^\varepsilon \quad = \quad \{\vdash [A{\rightarrow}{\bullet}, j, j]\},$$

$$D^{\mathrm{LC}(a)} \;=\; \{[a, j-1, j] \vdash [B{\rightarrow}a{\bullet}\beta, j-1, j]\},$$

$$D^{\mathrm{LC}(A)} \;=\; \{[A{\rightarrow}\alpha{\bullet}, i, j] \vdash [B{\rightarrow}A{\bullet}\beta, i, j]\},$$

$$D^{\mathrm{Scan}} \;=\; \{[A{\rightarrow}\alpha{\bullet}a\beta, i, j], [a, j, j+1] \vdash [A{\rightarrow}\alpha a{\bullet}\beta, i, j+1]\},$$

$$D^{\mathrm{Compl}} \;=\; \{[A{\rightarrow}\alpha{\bullet}B\beta, i, j], [B{\rightarrow}\gamma{\bullet}, j, k] \vdash [A{\rightarrow}\alpha B{\bullet}\beta, i, k]\},$$

$$D_{\mathrm{buLC}} \;=\; D^{\varepsilon} \cup D^{\mathrm{LC}(a)} \cup D^{\mathrm{LC}(A)} \cup D^{\mathrm{Scan}} \cup D^{\mathrm{Compl}}.$$

Proposition 6.2. buE $\stackrel{\mathrm{sc}}{\Longrightarrow}$ buLC. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$

Things get more interesting – and rather more complicated – if we apply the same transformation to **Earley**, rather than **buE**. It is *not* the case that $[A{\rightarrow}{\bullet}B\beta, i, i]$ is always valid. Therefore, the replacement of $[A{\rightarrow}{\bullet}B\beta, i, i]$, $[B{\rightarrow}\gamma{\bullet}, i, j] \;\vdash\; [A{\rightarrow}B{\bullet}\beta, i, j]$ by a deduction $[B{\rightarrow}\gamma{\bullet}, i, j] \;\vdash\; [A{\rightarrow}B{\bullet}\beta, i, j]$ should be allowed *only in those cases* where $[A{\rightarrow}{\bullet}B\beta, i, i]$ is actually valid. Under which conditions is this the case?

The item $[A{\rightarrow}{\bullet}B\beta, i, i]$ is predicted by **Earley** only if there is some valid item of the form $[C{\rightarrow}\alpha{\bullet}A\delta, h, i]$. But if, by chance, $\alpha = \varepsilon$, then this is one of the very items that we seek to eliminate. In that case we continue the search for an item that licences the validity of $[C{\rightarrow}{\bullet}A\delta, i, i]$. This search can end in two ways: either we find some item with the dot not in leftmost position, or (only in case $i = 0$) we may move all the way up to $[S{\rightarrow}{\bullet}\gamma, 0, 0]$.

Definition 6.3. (*left-corner relation*)
The *left corner* of a non-empty production is the leftmost right-hand side symbol (i.e., production $A{\rightarrow}X\alpha$ has left corner X); the left corner of an empty production is ε.
The relation $>_\ell$ on $N \times (N \cup \Sigma \cup \{\varepsilon\})$ is defined by

$$A >_\ell U \text{ if there is } p = A{\rightarrow}\alpha \in P \text{ with } U \text{ the left corner of } p.$$

The transitive and reflexive closure of $>_\ell$ is denoted $>_\ell^*$.

We can now proceed to define a schema **LC** for a left-corner parser. Clearly, $[A{\rightarrow}{\bullet}B\beta, i, i]$ will be recognized by the Earley algorithm if there is some valid item $[C{\rightarrow}\alpha{\bullet}E\delta, h, i]$ with $E >_\ell^* A$. Moreover, there is such an item with $\alpha \neq \varepsilon$, unless, perhaps, $i = 0$ and $C = S$. For this exceptional case we retain items $[S{\rightarrow}{\bullet}\gamma, 0, 0]$ as usual. The discarded *complete* steps are replaced by *left-corner* steps as follows:

$$[C{\rightarrow}\alpha{\bullet}E\delta, h, i], \; [B{\rightarrow}\gamma{\bullet}, i, j] \;\vdash\; [A{\rightarrow}B{\bullet}\beta, i, j] \text{ only if } E >_\ell^* A,$$

see Figure 6.2; similarly for consequents of the form $[A{\rightarrow}a{\bullet}\beta, j-1, j]$.

The general idea of a *left-corner* step involves slightly different details for nonterminal, terminal, and empty left corners. Thus we obtain the following parsing schema.

Schema 6.4. (LC)
The parsing schema **LC** is defined by a parsing system $\mathbb{P}_{\mathrm{LC}}$ for any $G \in \mathcal{CFG}$ and for any $a_1 \dots a_n \in \Sigma^*$ by
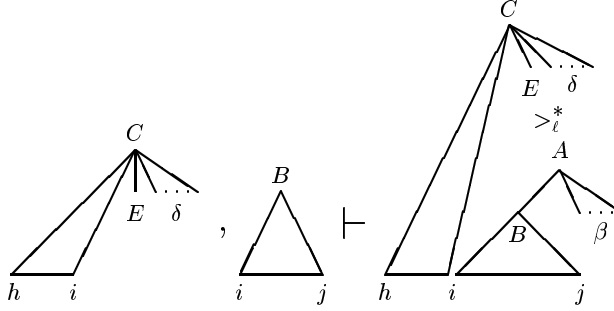
Fig. 6.2. The (*predictive*) *left-corner* step

$$\mathcal{I}^{(1)} \quad = \{[A{\rightarrow}X\alpha{\bullet}\beta, i, j] \mid A{\rightarrow}X\alpha\beta \in P \wedge 0 \le i \le j\},$$

$$\mathcal{I}^{(2)} \quad = \{[A{\rightarrow}{\bullet}, j, j] \mid A{\rightarrow}\varepsilon \in P \wedge j \ge 0\},$$

$$\mathcal{I}^{(3)} \quad = \{[S{\rightarrow}{\bullet}\gamma, 0, 0] \mid S{\rightarrow}\gamma \in P\},$$

$$\mathcal{I}_{\mathrm{LC}} \quad = \mathcal{I}^{(1)} \cup \mathcal{I}^{(2)} \cup \mathcal{I}^{(3)};$$

$$D^{\mathrm{Init}} \quad = \{\vdash [S{\rightarrow}{\bullet}\gamma, 0, 0]\},$$

$$D^{\mathrm{LC}(A)} \quad = \{[C{\rightarrow}\gamma{\bullet}E\delta, h, i], [A{\rightarrow}\alpha{\bullet}, i, j] \vdash [B{\rightarrow}A{\bullet}\beta, i, j] \mid E >_{\ell}^{*} B\},$$

$$D^{\mathrm{LC}(a)} \quad = \{[C{\rightarrow}\gamma{\bullet}E\delta, h, i], [a, i, i+1] \vdash [B{\rightarrow}a{\bullet}\beta, i, i+1] \mid E >_{\ell}^{*} B\},$$

$$D^{\mathrm{LC}(\varepsilon)} \quad = \{[C{\rightarrow}\gamma{\bullet}E\delta, h, i] \vdash [B{\rightarrow}{\bullet}, i, i] \mid E >_{\ell}^{*} B\},$$

$$D^{\mathrm{Scan}} \quad = \{[A{\rightarrow}\alpha{\bullet}a\beta, i, j], [a, j, j+1] \vdash [A{\rightarrow}\alpha a{\bullet}\beta, i, j+1]\},$$

$$D^{\mathrm{Compl}} \quad = \{[A{\rightarrow}\alpha{\bullet}B\beta, i, j], [B{\rightarrow}\gamma{\bullet}, j, k] \vdash [A{\rightarrow}\alpha B{\bullet}\beta, i, k]\},$$

$$D_{\mathrm{LC}} \quad = D^{\mathrm{Init}} \cup D^{\mathrm{LC}(a)} \cup D^{\mathrm{LC}(A)} \cup D^{\mathrm{LC}(\varepsilon)} \cup D^{\mathrm{Scan}} \cup D^{\mathrm{Compl}}.$$

Proposition 6.5. buLC $\overset{\mathrm{df}}{\Longrightarrow}$ LC. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$

Proposition 6.6. Earley $\overset{\mathrm{sf}}{\Longrightarrow}$ LC. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$

When it comes to implementing the schema **LC**, a practical simplification can be made. In order to apply a *left-corner* step we have to look for *some* item of the form $[C{\rightarrow}\alpha{\bullet}E\delta, h, i]$, with arbitrary C, α, β, and h. We can introduce a special *predict item*, denoted $[i, E]$, to indicate that E has been predicted as a feasible constituent at position i. The details are straightforward and need not be spelled out here.

6.2 De Vreught and Honig's Algorithm

We define several variants of a parsing schema for an algorithm defined by de Vreught and Honig [VH89, VH91], primarily intended for parallel processing. Rather than working through a production from left to right, as is done in Earley's algorithm, one could start at an arbitrary position in the right-hand side and from there extend the recognized part in both directions. To this

end, we use *double-dotted items* of the form $[A{\to}\alpha{\bullet}\beta{\bullet}\gamma, i, j]$. Recognition of such an item indicates that $\beta \Rightarrow^* a_{i+1} \ldots a_j$, while α and γ still have to be expanded. An item $[A{\to}{\bullet}\alpha{\bullet}\beta, i, j]$ corresponds to the canonical Earley item.

The algorithm of de Vreught and Honig has two basic steps, called *include* and *concatenate*. The idea of both steps is illustrated in Figure 6.3. The following schema for our first version of the algorithm should be clear.
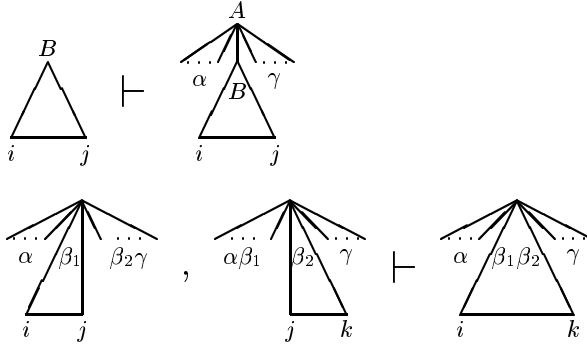

Fig. 6.3. The *include* and *concatenate* steps

Schema 6.7. (dVH1)
The parsing schema **dVH1** is defined by a parsing system $\mathbb{P}_{\mathrm{dVH1}}$ for any $G \in \mathcal{CFG}$ and for any $a_1 \ldots a_n \in \Sigma^*$ by

$$\mathcal{I}_{\mathrm{dVH1}} = \{[A{\to}\alpha{\bullet}\beta{\bullet}\gamma, i, j] \mid A{\to}\alpha\beta\gamma \in P \wedge 0 \le i \le j \\ \wedge\ (\beta \neq \varepsilon \text{ or } \alpha\gamma = \varepsilon)\};$$

$$D^{\mathrm{Init}} = \{[a, j-1, j] \vdash [A{\to}\alpha{\bullet}a{\bullet}\gamma, j-1, j]\},$$

$$D^{\varepsilon} = \{\ \vdash [B{\to}{\bullet}{\bullet}, j, j]\},$$

$$D^{\mathrm{Incl}} = \{[B{\to}{\bullet}\beta{\bullet}, i, j] \vdash [A{\to}\alpha{\bullet}B{\bullet}\gamma, i, j]\},$$

$$D^{\mathrm{Concat}} = \{[A{\to}\alpha{\bullet}\beta_1{\bullet}\beta_2\gamma, i, j], [A{\to}\alpha\beta_1{\bullet}\beta_2{\bullet}\gamma, j, k] \\ \vdash [A{\to}\alpha{\bullet}\beta_1\beta_2{\bullet}\gamma, i, k]\},$$

$$D_{\mathrm{dVH1}} = D^{\mathrm{Init}} \cup D^{\varepsilon} \cup D^{\mathrm{Incl}} \cup D^{\mathrm{Concat}}.$$

Next, we observe that D_{dVH1} is redundant, in the following way. An item $[A{\to}\alpha{\bullet}XYZ{\bullet}\gamma, i, j]$ can be concatenated in two different ways:

$$[A{\to}\alpha{\bullet}X{\bullet}YZ\gamma, i, k], [A{\to}\alpha X{\bullet}YZ{\bullet}\gamma, k, j] \vdash [A{\to}\alpha{\bullet}XYZ{\bullet}\gamma, i, j];$$

$$[A{\to}\alpha{\bullet}XY{\bullet}Z\gamma, i, l], [A{\to}\alpha XY{\bullet}Z{\bullet}\gamma, l, j] \vdash [A{\to}\alpha{\bullet}XYZ{\bullet}\gamma, i, j].$$

Moreover, if $[A{\to}\alpha{\bullet}XYZ{\bullet}\gamma, i, j]$ is valid, then each of the four antecedents is also valid for some value of k and l. Hence, if we delete the former deduction step from D, the set of valid items is not affected. For items with more than 3 symbols between the dots, the redundancy in deduction steps increases accordingly.

Schema 6.8. (dVH2)
In the specification of $\mathbb{P}_{\mathrm{dVH1}}$ in Schema 6.7 we replace D^{Concat} by

$$D^{\mathrm{Concat}} \;=\; \{[A{\rightarrow}\alpha\bullet\beta\bullet X\gamma, i, j], [A{\rightarrow}\alpha\beta\bullet X\bullet\gamma, j, k] \vdash [A{\rightarrow}\alpha\bullet\beta X\bullet\gamma, i, k]\},$$

and leave $\mathcal{I}$, D^{Init}, D^{ε} and D^{Incl} as in Schema 6.7.

Further optimization of **dVH2** is possible. Observe that items of the form $[A{\rightarrow}\alpha\bullet\beta\bullet, i, j]$ with $|\alpha| \geq 1$ and $|\beta| \geq 2$ are useless in $\mathbb{P}_{\mathrm{dVH2}}$, in the sense that they do not occur as an antecedent in any derivation step. Hence, these items can be discarded. Similarly, any item of the form $[A{\rightarrow}\alpha\bullet\beta\bullet\gamma, i, j]$ with $|\alpha| \geq 1$, $|\beta| \geq 2$ and $|\gamma| \geq 1$ can concatenate to the right, but cannot contribute to the recognition of an item of the form $[A{\rightarrow}\bullet\beta\bullet, i, j]$. Hence the whole set

$$\{[A{\rightarrow}\alpha\bullet\beta\bullet\gamma, i, j] \;\mid\; |\alpha| \geq 1 \wedge |\beta| \geq 2\}$$

can be considered useless; these items can only be used to recognize further items in this set, but none of these items can be used to recognize an item outside this set. Hence we delete this set and discard all deduction steps that have one of these items as antecedent or as consequent.

Schema 6.9. (dVH3)
The parsing schema **dVH3** is defined by a parsing system $\mathbb{P}_{\mathrm{dVH3}}$ for any $G \in \mathcal{CFG}$ and for any $a_1 \ldots a_n \in \Sigma^*$ by

$$\mathcal{I}^{(1)} \;=\; \{[A{\rightarrow}\alpha\bullet X\bullet\gamma, i, j] \mid A{\rightarrow}\alpha X\gamma \in P \wedge 0 \leq i \leq j\},$$

$$\mathcal{I}^{(2)} \;=\; \{[A{\rightarrow}\bullet X\beta\bullet\gamma, i, j] \mid A{\rightarrow}X\beta\gamma \in P \wedge 0 \leq i \leq j\},$$

$$\mathcal{I}^{(3)} \;=\; \{[A{\rightarrow}\bullet\bullet, j, j] \mid A{\rightarrow}\varepsilon \in P \wedge j \geq 0\},$$

$$\mathcal{I}_{\mathrm{dVH3}} \;=\; \mathcal{I}^{(1)} \cup \mathcal{I}^{(2)} \cup \mathcal{I}^{(3)};$$

$$D^{\mathrm{Init}} \;=\; \{[a, j-1, j] \vdash [A{\rightarrow}\alpha\bullet a\bullet\gamma, j-1, j]\},$$

$$D^{\varepsilon} \;=\; \{\vdash [B{\rightarrow}\bullet\bullet, j, j]\},$$

$$D^{\mathrm{Incl}} \;=\; \{[B{\rightarrow}\bullet\beta\bullet, i, j] \vdash [A{\rightarrow}\alpha\bullet B\bullet\gamma, i, j]\},$$

$$D^{\mathrm{Concat}} \;=\; \{[A{\rightarrow}\bullet\alpha\bullet X\gamma, i, j], [A{\rightarrow}\alpha\bullet X\bullet\gamma, j, k] \vdash [A{\rightarrow}\bullet\alpha X\bullet\gamma, i, k]\},$$

$$D_{\mathrm{dVH3}} \;=\; D^{\mathrm{Init}} \cup D^{\varepsilon} \cup D^{\mathrm{Incl}} \cup D^{\mathrm{Concat}}.$$

Proposition 6.10. dVH1 $\stackrel{\mathbf{sf}}{\Longrightarrow}$ dVH2 $\stackrel{\mathbf{sf}}{\Longrightarrow}$ dVH3. $\square$

The various parsing schemata for the algorithm of de Vreught and Honig can be dynamically filtered with look-ahead and *look-back*. The original algorithm as proposed in [VH89] is an implementation of **dVH2** with one position look-ahead and look-back.

Proposition 6.11. dVH3 $\stackrel{\mathbf{sc}}{\Longrightarrow}$ buLC.

Proof. In order to show that **dVH3** can be filtered to **buLC**, we have to realize that a single-dotted item $[A{\rightarrow}\alpha{\bullet}\beta, i, j]$ and a double-dotted item $[A{\rightarrow}{\bullet}\alpha{\bullet}\beta, i, j]$ are merely different notations for the same object. Hence, clearly, $\mathcal{I}_{\mathrm{buLC}} \subset \mathcal{I}_{\mathrm{dVH3}}$.

It remains to be shown that $\vdash^*_{\mathrm{buLC}} \subseteq \vdash^*_{\mathrm{dVH3}}$. To this end it suffices to show that for every deduction step $\eta_1 \ldots, \eta_k \vdash \xi \in D_{\mathrm{buLC}}$ it holds that $\eta_1 \ldots, \eta_k \vdash^*_{\mathrm{dVH3}} \xi$.

An arbitrary deduction step in D^{Compl} in $\mathbb{P}_{\mathrm{buLC}}$

$$[A{\rightarrow}{\bullet}\alpha{\bullet}B\beta, i, j], [B{\rightarrow}{\bullet}\gamma{\bullet}, j, k] \vdash [A{\rightarrow}{\bullet}\alpha B{\bullet}\beta, i, k]$$

is emulated in $\mathbb{P}_{\mathrm{dVH3}}$ by

$$[B{\rightarrow}{\bullet}\gamma{\bullet}, j, k] \vdash [A{\rightarrow}\alpha{\bullet}B{\bullet}\beta, j, k],$$
$$[A{\rightarrow}{\bullet}\alpha{\bullet}B\beta, i, j], [A{\rightarrow}\alpha{\bullet}B{\bullet}\beta, j, k] \vdash [A{\rightarrow}{\bullet}\alpha B{\bullet}\beta, i, k];$$

similarly for D^{Scan}.

The other cases are trivial, hence **dVH3** $\overset{\mathbf{sc}}{\Longrightarrow}$ **buLC**. $\qquad\square$

By Propositions 6.5 6.10, and 6.11 we have shown that **dVH1** $\overset{\mathbf{sc}}{\Longrightarrow}$ **LC**. The conclusion should *not* be, however, that de Vreught and Honig's algorithm is a sub-optimal version of (bottom-up) left-corner parsing. A subtle but decisive change took place in the seemingly harmless static filter **dVH1** $\overset{\mathbf{sf}}{\Longrightarrow}$ **dVH2**, where we laid down that the the concatenation of right-hand side elements is from left to right.

A different, more general way to eliminate the redundancy in **dVH1** is to start expanding the right-hand side from the "most interesting" symbol, called the *head* of a production, rather than the leftmost symbol. This leads to so-called *Head Corner* (HC) parsers. A bottom-up head-corner parser that is very similar to the one defined by Satta and Stock[10] [SS89] can be obtained as a step contraction of the Vreught and Honig's algorithm along similar lines; top-down prediction can be added as in [SA96].

A context-free head grammar, in which every production has some right-hand side symbol assigned as head, can be seen as a generalization of a context-free grammar (take the left corner by default if no head has been specified explicitly). A head-corner parsing schema **HC** can be specified that is a generalization of **LC**.

6.3 Rytter's Algorithm

Another example of step refinement is provided by Rytter's algorithm [Ryt85, GR88]. This algorithm has theoretical, rather than practical value. It allows

---

[10] If there are right-hand side symbols both to the left and the right of a head, there is a choice in which direction the item should be expanded first. Satta and Stock leave this choice to the parser but block the other step when a choice has been made. This leads to a nondeterministic set of valid items, which is rather undesirable in this framework. The nondeterminism can be removed by prescribing a choice at the level of the parsing schema.

parallel recognition in logarithmic time, but requires $O(n^6)$ processors to do so. The algorithm is based on CYK – hence only defined for grammars in Chomsky Normal Form – but can generalized to arbitrary context-free grammars just as **CYK** was generalized to **buE**.

In addition to the conventional CYK items we introduce *Rytter items*, denoted $[A, h, k; B, i, j]$. A Rytter item is recognized if

$$A \Rightarrow^* a_{h+1} \ldots a_i B a_{j+1} \ldots a_k,$$

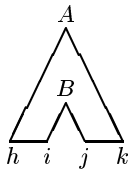that is, the part $B \Rightarrow^* a_{i+1} \ldots a_j$ is still missing, cf. Figure 6.4.

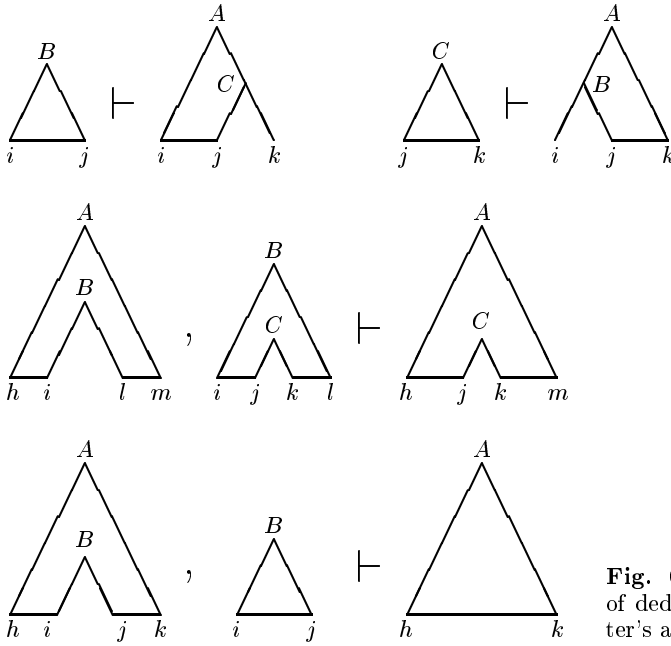**Fig. 6.4.** A Rytter item $[A, h, k; B, i, j]$

Fig. 6.5. Different types of deduction steps in Rytter's algorithm

Another way to interpret a Rytter item is as a *conditional item*: if $[B, i, j]$ is valid, then $[A, i, j]$ is also valid. The different kinds of deduction steps (excluding the initial CYK steps) are shown in Figure 6.5.

A CYK deduction step $[B, i, j], [C, j, k] \vdash [A, i, k]$ can be refined into

$$[B, i, j] \quad \vdash \quad [A, i, k; C, j, k]$$
$$[A, i, k; C, j, k], [C, j, k] \quad \vdash \quad [A, i, k].$$

Note that, given $[B, i, j]$ and $[C, j, k]$, we also could have used $[A, i, k; B, i, j]$ as an intermediate conditional item. In fact, unless $i+1 = j = k-1$, there are many more ways to recognize $[A, i, k]$, by combining conditional items that have been created at various stages. There is a massive redundancy in the different ways in which a single item can be recognized. It is this redundancy, that guarantees the existence of a balanced recognition tree for each item, which – given enough computing resources – allows for parallel parsing in logarithmic time. For a proof, see [GR88] or [Sik93a, Sik97].

A parsing schema for Rytter's algorithm is defined as follows. The operations associated with the sets of deduction steps $D^{(1)}$, $D^{(2)}$, and $D^{(3)}$, are originally called *activate*, *square*, and *pebble*, respectively. In this context the original names do not make much sense and we rather use numbers.

Schema 6.12. (Rytter)
The parsing schema **Rytter** is defined by a parsing system $\mathbb{P}_{\mathrm{Rytter}}$ for any $G \in \mathcal{CNF}$ and for any $a_1 \ldots a_n \in \Sigma^*$ by

$$\mathcal{I}^{(1)} \quad = \quad \{[A, i, j] \mid A \in N \wedge 0 \le i < j\};$$

$$\mathcal{I}^{(2)} \quad = \quad \{[A, h, k; B, i, j] \mid [A, h, k] \in \mathcal{I} \ \wedge \ [B, i, j] \in \mathcal{I}$$
$$\wedge \ h \le i < j \le k \ \wedge \ (h \neq i \text{ or } j \neq k)\}$$

$$\mathcal{I}_{\mathrm{Rytter}} \quad = \quad \mathcal{I}^{(1)} \cup \mathcal{I}^{(2)};$$

$$D^{(0)} \quad = \quad \{[a, i-1, i] \vdash [A, i-1, i] \mid A{\rightarrow}a \in P\},$$

$$D^{(1\mathrm{a})} \quad = \quad \{[B, i, j] \vdash [A, i, k; C, j, k] \mid A{\rightarrow}BC \in P\},$$

$$D^{(1\mathrm{b})} \quad = \quad \{[C, j, k] \vdash [A, i, k; B, i, j] \mid A{\rightarrow}BC \in P\},$$

$$D^{(2)} \quad = \quad \{[A, h, m; B, i, l], [B, i, l; C, j, k] \vdash [A, h, m; C, j, k]\},$$

$$D^{(3)} \quad = \quad \{[A, h, k; B, i, j], [B, i, j] \vdash [A, h, k]\},$$

$$D_{\mathrm{Rytter}} \quad = \quad D^{(0)} \cup D^{(1\mathrm{a})} \cup D^{(1\mathrm{b})} \cup D^{(2)} \cup D^{(3)}.$$

Between CYK and Rytter's algorithm another algorithm is hiding that is not uninteresting. It addresses the problem of parallel *on-line* parsing: The symbols of a string arrive one by one, as the words of a sentence in spoken natural language. If the processing of each word is finished when the next word arrives, parsing can be done in real time. For on-line parsing, "gaps" in items are useful in rightmost position, so that the still missing words can be anticipated with a partial syntactic analysis. But the large number of position markers, which accounts for the excessive resources required by Rytter's algorithm, can be reduced.

These ideas underlie the definition of the following parsing schema **OCYK** (for *on-line* CYK). In addition to $[A, i, j]$ we introduce items $[A, i, j; B]$ to denote $A \Rightarrow^* a_{i+1} \ldots a_j B$. There is no need to specify the size of the "gap" $[B, j, ?]$.

Schema 6.13. (OCYK)
The parsing schema **OCYK** is defined by a parsing system $\mathbb{P}_{\mathrm{OCYK}}$ for any $G \in \mathcal{CNF}$ and for any $a_1 \ldots a_n \in \Sigma^*$ by

$$
\begin{aligned}
\mathcal{I}^{(1)} &= \{[A,i,j] \mid A \in N \wedge 0 \le i < j\}, \\
\mathcal{I}^{(2)} &= \{[A,i,j;B] \mid A,B \in N \wedge 0 \le i < j\}, \\
\mathcal{I}_{\mathrm{OCYK}} &= \mathcal{I}^{(1)} \cup \mathcal{I}^{(2)}; \\
D^{(0)} &= \{[a,j-1,j] \vdash [A,j-1,j] \mid A{\rightarrow}a \in P\}, \\
D^{(1)} &= \{[B,i,j] \vdash [A,i,j;C] \mid A{\rightarrow}BC \in P\}, \\
D^{(2)} &= \{[A,i,j;B],[B,j,k;C] \;\vdash\; [A,i,k;C]\}, \\
D^{(3)} &= \{[A,i,j;B],[B,j,k] \;\vdash\; [A,i,k]\}, \\
D_{\mathrm{OCYK}} &= D^{(0)} \cup D^{(1)} \cup D^{(2)} \cup D^{(3)}.
\end{aligned}
$$

The schema **OCYK** can be implemented on a parallel random access machine with $O(n^2)$ processors such that only constant time per word is needed, cf. [Sik93b].

Proposition 6.14. CYK $\stackrel{\mathbf{sr}}{\Longrightarrow}$ OCYK $\stackrel{\mathbf{sr}}{\Longrightarrow}$ Rytter. □

The schemata **OCYK** and **Rytter** can be generalized from $\mathcal{CNF}$ to $\mathcal{CFG}$ in a way that is similar to the generalization **CYK $\stackrel{\mathbf{gen}}{\Longrightarrow}$ buE** as discussed in Example 4.1.

Another way to obtain a logarithmic-time parallel parsing algorithm with $O(n^6)$ processors is a step refinement of any of the **dVH** schemata, similar to **CYK $\stackrel{\mathbf{sr}}{\Longrightarrow}$ Rytter**. This has been worked out in [VH91].

6.4 Some general remarks

After these extensive examples, some general remarks are due.

We have shown that the parsing schemata framework is able to handle nontrivial algorithms of different flavours. Very general frameworks, offering a good insight at a high level of abstraction, have a risk of becoming unwieldy when confronted with problems that stretch far beyond their canonical example. The fact that we were able to cover these algorithms with sufficient clarity provides circumstantial evidence that the parsing schemata framework makes abstractions that are *right* in some way.

The above examples are taken from the computer science and computational linguistics literature on parsing, but some variants (**dVH3** and **OCYK** in the discussed examples) have been discovered as a *result* of analysing these algorithms by means of parsing schemata. Also, the close relation between Earley parsing and Left-Corner parsing was noticed in [Sik93a] for the first time.

An inevitable weakness of the formalism is a diminished understanding of practical algorithm efficiency. Because of the absence of any kind of data structure it is not immediately clear what the algorithmic complexity of a parsing schema is. Improving the efficiency of parsers, therefore, cannot be done only at the level of schemata. How such an improved schema is to be

realized in a more efficient implementation is a matter that has not been (and need not be) addressed in this context.

But in order to find optimizations of algorithms, one must have a very good insight in the characteristic behaviour of an algorithm. It is such an insight that is offered by the parsing schemata framework.

7. From Schemata to Algorithms

We will not discuss parsing algorithms in detail, but briefly review how some well-known classes of parsing algorithms relate to the framework presented in the preceding sections.

Parsing schemata are a generalization of *chart parsers* [Kay80, Kay82, Win83]. From the view that has been unfolded in the previous sections, we can see a chart parser as the canonical implementation of a parsing schema.

A chart parser employs two data structures: An *agenda*, containing items that will be actively used to search for new items that can be recognized, and a *chart*, storing the items that need no further attention. At each step, one item, say ξ, is taken from the agenda and put on the chart. The chart is searched for all (combinations of) items $\eta_1 \ldots \eta_k$ such that $\xi, \eta_1 \ldots \eta_k \vdash \zeta \in D$. All ζ that are found in the way and were not recognized before are added to the agenda. An Earley chart parser, for example, is initialized with items $[a, i-1, i]$ on the chart and $[S \rightarrow \bullet \gamma, 0, 0]$ on the agenda.

The control structure of the chart parser guarantees that the final chart, which is reached when the agenda is empty, contains $\mathcal{V}(\mathbb{P})$. An issue that has to be addressed (but not in this context) is how to structure the chart and agenda so as to make the parser efficient. For an overview of chart based approaches to CYK and Earley parsing, see [Nij94].

The chart parsing framework offers a generic way to handle *nondeterminism*. Hence it is not surprising that chart parsers has attracted widespread attention in computational linguistics.

Logical deduction as a basis for the description of chart parsers is due to Pereira and Warren [PW83]. But our framework has a rather different emphasis. While the "Parsing as Deduction" approach is primarily interested in connecting the parsing logic with unification-based grammar formalisms, parsing schemata use deduction merely as a convenient notation for describing the essential traits of arbitrary parsing algorithms. We come back to unification grammars in Section 8.

A different parsing paradigm is provided by the pushdown automaton (PDA). A fundamental theorem in formal language theory states that the class of languages accepted by (nondeterministic) PDA's is equal to the class of languages generated by context-free languages. Many parsing algorithms from the field of compiler construction are based on the PDA paradigm. The canonical example is the family of *LR-parsers*, discovered by Knuth [Knu65] and extended to the more practical *SLR* and *LALR* parsers by

DeRemer [DeR69, DeR71]. See [ASU86] for a good introduction and [Nij83] for an extensive bibliography of LR parsing.

While deterministic LR parsers on restricted classes of context-free grammars are particularly efficient, nondeterministic LR parsers (known as *generalized LR* (*GLR*) parsers) have been introduced to cover wider classes of grammars, in particular for use in computational linguistics. A general method to handle nondeterministic PDA's in an efficient manner has been given by Lang [Lan74]. Generalized LR parsing has attracted more attention in the form of Tomita's algorithm [Tom85], based on a graph-structured stack as the data structure to handle the ambiguities that occur during parsing.

Tomita's algorithm cannot handle certain classes of grammars (cyclic grammars and grammars with hidden left-recursion[11]). Rekers has improved Tomita's algorithm to handle these grammars as well [Rek92]. See also [Nij91] for a historic overview and [Ned94] for some optimizations in generalized LR parsing.

The question arises how PDA-based algorithms like LR relate to parsing schemata. LR parsers use items *compile-time* in the construction of the parsing table. The states of an LR parser are in fact sets dotted productions $A{\rightarrow}\alpha{\bullet}\beta$. A state comprises those productions that could apply at the current point in the parsing process. One can partially uncompile the various LR-type algorithms and make these dotted rules visible during parsing. It is easy to add position markers to a dotted rule (viz., the position where a particular production was started and the current position). This yields Earley-type items. An item is recognized when a state that contains the dotted rule, in combination with the appropriate position markers, is pushed onto the stack.

An LR parser, therefore, implements some underlying parsing schema. Let **LR(0)** denote the parsing schema that underlies a generalized LR(0) parser. Then it holds that $\textbf{LR(0)} \overset{\textbf{ext}}{\Longrightarrow} \textbf{Earley}$[12]. In Section 5.2 we defined **E(1)** such that $\textbf{SLR(1)} \overset{\textbf{ext}}{\Longrightarrow} \textbf{E(1)}$.

Having uncovered the close relation between the Algorithms of Earley and Tomita, it is possible to apply many optimizations, extensions and variants of one algorithm to the other as well. An interesting example of such cross-fertilization is a parallel bottom-up Tomita parser [SL92, Sik93a, Sik97] which applies the usual "vertical" parallelization of Earley to the graph-structured stack of Tomita (in contrast to the "horizontal" parallelization as in [TN89, NT90, TDL91] that seems more obvious from the LR point of view), which showed reasonable efficiency in a test implementation.

An ambiguous grammar can be supplied with various kinds of disambiguation rules so as to guarantee that only a single valid parse tree remains. Consider, for example, the grammar

$E{\rightarrow}E + E \mid E * E \mid a$

---

[11] A grammar is called hidden left-recursive if $A \Rightarrow^+ \alpha A\beta$ and $\alpha \Rightarrow^+ \varepsilon$. The term has been coined in [NS93].

[12] GLR parsers typically assume grammars to be reduced. Hence, to be formally correct, GLR applies only to a subset of $\mathcal{CFG}$

for arithmetic expressions. The string $a + a * a * a$ has five parse trees. But by introducing *operator precedence* and *associativity* one can specify that $[a + [[a * a] * a]]$ is the only valid parse.

Rather than constructing all parse trees and discarding the invalid ones, an efficient parser must apply the disambiguation rules locally during the parsing process. For the above mentioned grammar for arithmetic expressions. it is in fact possible to construct a deterministic LR-type parser by disambiguating the states and transitions in the LR parsing table.

A general formal framework for this type of optimization is given by Visser [KV94, Vis95] who applies disambiguation rules[13] at the level of parsing schemata.

Head-Driven parsing [Kay89] does not proceed through the sentence from left to right, but starts with the most informative parts. This introduces some extra administrative burden on the parser (as it jumps up and down the sentence) but may lead to substantial savings if the semantic information captured in the head excludes possibilities that would have been explored without this information available. Bouma and van Noord [BN93] have done several experiments with head-driven parsing and conclude (unsurprisingly) that the efficiency of the algorithm is critically dependent on the discriminative nature of the information captured in the heads.

The parsing schema for Head-Corner parsing presented in [SA96] has been extended with typed structures and implemented by Moll [Mol95]. It is used as a parser in an experimental natural language dialogue system [A&a95].

8. Beyond context-free grammars

The parsing schemata framework has been specified for context-free grammars, but it can easily be extended to other grammar formalisms as well.

Unification-based grammars are the predominant class of grammar formalisms in current computational linguistics. Between 1980 and 1990 a variety of different kinds of unification grammar has been introduced. Some of these, like Definite Clause Grammars [PW80], Functional Unification Grammar [Kay85], and PATR-II [Shi86] have been introduced primarily to offer powerful formalisms for grammar description; others like Lexical-Functional Grammar [KB82] and Head-Driven Phrase Structure Grammar [PS87, PS94] with the aim to provide a theory of linguistic phenomena in natural language.

Unification grammars are related to *attribute grammars* [Knu68, Knu71], which are typically used in the field of compiler construction. There are some fundamental differences in the underlying logic, but these cannot be explained satisfactorily in a few lines. The interested reader is referred to [Shi92] and [Car92] for a thorough treatment of unification logics.

---

[13] called *filters* in the cited publications, but not to be confused with the filters in Section 5.

One of the properties of context-free grammars that was felt constraining for the description of natural languages is the rigid order of right-hand side elements in a production. In Generalized Phrase Structure Grammar [G&a85], the notion of *ID/LPgrammars* was introduced. There are separate specifications for the set of right-hand side elements of a production (*immediate dominance*) and constraints on the order in which these elements may appear (*linear precedence*). A parsing schema for unification-based ID/LP-grammars is given by Morawietz [Mor95].

Unification grammars treat syntactic and semantic information in a uniform manner. One can reduce the role of syntax and consider syntactic category as a feature like any other. Indeed there seems to be a trend that less and less information is stored in the context-free backbone of a grammar – i.e., the `cat` feature in a feature structure – because various syntactic properties can be expressed more elegantly by other kinds of feature constraints. A typical example is *subcategorization* of verbs: all verbs have syntactic category *verb*; constraints on the various kinds of objects that a verb can take are denoted in the `subcat` feature of the particular verb.

Nagata [Nag92] and Maxwell and Kaplan [MK93] have independently pointed out that this is convenient for writing natural language grammars, but that it has repercussions on parsing efficiency. Context-free parsing is much more efficient than feature structure unification. Hence is it not surprising that the experiments reported in [Nag92] and [MK93] show that the efficiency of unification grammar parsing can be increased by retrieving an (implicit) context-free backbone from a unification grammar that covers more than just the `cat` feature and using this context-free part for syntactic analysis.

9. Conclusions

Parsing schemata provide a general framework for description, analysis and comparison of parsing algorithms, both sequential and parallel. Data structures, control structures and (for parallel algorithms) communication structures are abstracted from. This framework constitutes an intermediate, well-defined level of abstraction between grammars (defining what valid parses are) and parsing algorithms (prescribing how to compute these).

At this high level of abstraction, the essential traits of a particular type of parser stand out more clearly. Moreover, it is possible to clarify exactly the relationships between different parsers that have some fundamental principles common – even though the realizations of these parsers may look radically different. The price to be paid for this improved clarity and insight is the loss of some details that are of practical importance. The notion of algorithm complexity is strongly related to the data structures used to store intermediate results.

The prime strength of parsing schemata, therefore, is in the analysis of algorithms known to exist. Clarifying the basic traits of an algorithm may

suggest improvements that have been overlooked so far. Also, showing that different algorithms have closely related parsing schemata improves cross-fertilization of extensions and optimizations to these algorithms.

We have presented many examples of parsing schemata and their usage, in order to show that this framework, rather then being a mere theoretical nicety, constitutes are a valuable contribution to parsing theory.

References

[A&a95]   op den Akker R., ter Doest H., Moll M., Nijholt A. (1995): Parsing in Dialogue Systems Using Typed Feature Structures. *4th International Workshop on Parsing Technologies*, Prague, Czech Republic, 10–11.

[ASU86]   Aho A.V., Sethi R., Ullman J.D. (1986): *Compilers: Principles, Techniques and Tools*. Addison-Wesley, Reading, Mass.

[BN93]   Bouma G., van Noord G. (1993): Head-driven Parsing for Lexicalist Grammars: Experimental Results. *6th Meeting of the European Chapter of the Association of Computational Linguistics*, Utrecht, 71–80.

[Car92]   Carpenter B. (1992): *The Logic of Typed Feature Structures*. Cambridge University Press, Cambridge, UK.

[DeR69]   DeRemer F.L. (1969): Practical Translators for LR(k) Languages. Ph.D. Thesis, MIT, Cambridge, Mass.

[DeR71]   DeRemer F.L. (1971): Simple LR(k) grammars. *Communications of the ACM* **14**, 94–102.

[Ear68]   Earley J. (1986): An Efficient Context-Free Parsing Algorithm. Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, Pa.

[Ear70]   Earley J. (1970): An Efficient Context-Free Parsing Algorithm. *Communications of the ACM* **13**, 94–102.

[G&a85]   Gazdar G., Klein E., Pullum G.K., Sag I.A. (1985): *Generalized Phrase Structure Grammar*. Harvard University Press, Cambridge, Mass.

[GHR80]   Graham S.L., Harrison M.A., Ruzzo W.L. (1980): An Improved Context-Free Recognizer. *ACM Transactions on Programming Languages and Systems* **2**, 415–462.

[GR88]   Gibbons A., Rytter W. (1988): *Efficient Parallel Algorithms*. Cambridge University Press, Cambridge, UK.

[JVW91]   Joshi A., Vijay-Shanker K., Weir D., (1991): The Convergence of Mildly Context-Sensitive Grammar Formalisms. In: Sells P., Shieber S.M., Wasow T. (Eds), *Foundational Issues in Natural Language Processing*, MIT Press, Cambridge, Mass., 31–81.

[Kas65]   Kasami T. (1965): An Efficient Recognition and Syntax Analysis Algorithm for Context-Free Languages. Scientific Report AFCLR-65-758, Air Force Cambridge Research Laboratory, Bedford, Mass.

[Kay80]   Kay M. (1980): Algorithm Schemata and Data Structures in Syntactic Processing. Report CSL-80-12, Xerox PARC, Palo Alto, Ca.

[Kay82]   Kay M. (1982): Algorithm Schemata and Data Structures in Syntactic Processing. In: Grosz B.J., Sparck Jones, K., Webber B.L. (Eds), *Readings in Natural Language Processing*, Morgan Kaufmann, Los Altos, Ca.

[Kay85]   Kay M. (1985): Parsing in Functional Unification Grammar. In: D.R. Dowty, L. Karttunen, and A. Zwicky (Eds.), *Natural Language Parsing*, Cambridge University Press, Cambridge, UK, 251–278.

[Kay89]   Kay M. (1989): Head Driven Parsing. *1st International Workshop on Parsing Technologies*, Pittsburgh, Pa., 52–62.

[KB82]   Kaplan R.M., Bresnan J. (1982): Lexical-Functional Grammar: a formal system for grammatical representation. In: J. Bresnan (Ed.), *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge, Mass., 173–281.

[Knu65]   Knuth D.E. (1965): On the Translation of Languages from Left to Right. *Information and Control* **8**, 607–639.

[Knu68]   Knuth D.E. (1968): Semantics of Context-Free Languages. *Mathematical Systems Theory* **2**, 127–145.

[Knu71]   Knuth D.E. (1971): Semantics of Context-Free Languages, Correction. *Mathematical Systems Theory* **5**, 95–96.

[KV94]   Klint P., Visser E. (1994): Using Filters for the Disambiguation of Context-free Grammars. Proc. ASMICS Workshop on Parsing Theory, Milan, October 1994, Report 126-94, Dept. of Computer Science, University of Milan, Italy.

[Lan74]   Lang B. (1974): Deterministic Techniques for Efficient Non-Deterministic Parsers. *2nd Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 14, Springer-Verlag, Berlin, 255–269.

[M&a83]   Matsumoto Y., Tanaka H., Hirakawa H., Miyoshi H., Yasukawa H. (1983): BUP: a bottom-up parser embedded in Prolog. *New Generation Computing* **1**, 145–158.

[MK93]   Maxwell J.T., Kaplan R.M. (1993): The Interface between Phrasal and Functional Constraints. *Computational Linguistics* **19**, 571–590.

[Mol95]   Moll M. (1995): Head-Corner Parsing using Typed Feature Structures. M.Sc. Thesis, University of Twente, Dept. of Computer Science, Enschede, the Netherlands.

[Mor95]   Morawietz F. (1995): A Unification-based ID/LP Parsing Schema. *4th International Workshop on Parsing Technologies*, Prague, Czech Republic, 162–173.

[Nag92]   Nagata M. (1992): An Empirical Study on Rule Granularity and Unification Interleaving Toward an Efficient Unification-Based Parsing System. *14th International Conference on Computational Linguistics*, Nantes, France, 177–183.

[Ned93]   Nederhof M.J. (1993): Generalized Left-Corner Parsing. *6th Meeting of the European Association of Computational Linguistics*, Utrecht, the Netherlands, 305–314.

[Ned94]   Nederhof M.J. (1994): Linguistic Parsing and Program Transformations. Ph.D. Thesis, University of Nijmegen, the Netherlands.

[Nij83]   Nijholt A. (1983): Deterministic Top-Down and Bottom-Up Parsing: Historical Notes and Bibliographies. Mathematisch Centrum, Amsterdam, the Netherlands.

[Nij91]   Nijholt A. (1991): (Generalized) LR parsing: From Knuth to Tomita. In: *Tomita's Algorithm: Extensions and Applications.* Proceedings Twente Workshop on Language Technology 1 (TWLT1), University of Twente, Dept. of Computer Science, 1–8.

[Nij94]   Nijholt A. (1994): Parallel approaches to context-free language parsing. In: Hahn U., Adriaens G. (Eds.), *Parallel Natural Language Processing.* Ablex Publishing Corporation, Norwood, New Jersey.

[NS93]   Nederhof M.J., Sarbo J.J. (1993): Increasing the Applicability of LR Parsing. *3rd International Workshop on Parsing Technologies* Tilburg and Durbuy, Netherlands/Belgium, 187–201.

[NT90]   Numazaki H., Tananaka H. (1990): A New Parallel Algorithm for Generalized LR Parsing, *13th International Conference on Computational Linguistics*, Helsinki, Vol. 2, 304–310.

[PS87]   Pollard C., Sag I.A. (1987): *An Information-Based Syntax and Semantics, Vol. 1: Fundamentals.* CSLI Lecture Notes 13, Center for the Study of Language and Information, Stanford University, Stanford, Ca.

[PS94]   Pollard C., Sag I.A. (1994): *Head-Driven Phrase Structure Grammar*, University of Chicago Press, Chicago, Ill.

[PW80]   Pereira F.C.N., Warren D.H.D. (1980): Definite Clause Grammars for Language Analysis – A Survey of the Formalism and a Comparison with Augmented transition Networks. *Artificial Intelligence* **13**, 231–278.

[PW83]   Pereira F.C.N., Warren, D.H.D. (1983): Parsing as Deduction. *21th Annual Conference of the Association of Computational Linguistics*, Cambridge, Mass., 137–144.

[Rek92]   Rekers J. (1992): Parser Generation for Interactive Environments. Ph.D. Thesis, University of Amsterdam.

[RL70]   Rosenkrantz D.J., Lewis P.M. (1970): Deterministic Left Corner Parsing. *11th Annual Symposium on Switching and Automata Theory*, 139–152.

[Ryt85]   Rytter W. (1985): On the recognition of context-free languages. *5th Symposium on Fundamentals of Computation Theory*, Lecture notes in Computer Science 208, Springer-Verlag, 315–322.

[SA96]   Sikkel K., op den Akker R. (1996): Predictive Head-Corner Chart Parsing. In: Bunt H., Tomita M. (Eds), *Recent Advances in Parsing Technology*, Kluwer, Boston, Mass., 1996, 171–184.

[Shi86]   Shieber S.M. (1986): *An Introduction to Unification-Based Approaches to Grammar*. CSLI Lecture Notes 4, Center for the Study of Language and Information, Stanford University, Stanford, Ca.

[Shi92]   Shieber S.M. (1992): *Constraint-Based Grammar Formalisms: Parsing and Type Inference for Natural and Computer Languages*. The MIT Press, Cambridge, Mass.

[Sik93a]   Sikkel K. (1993): Parsing schemata. Ph.D. Thesis, University of Twente, Enschede, the Netherlands.

[Sik93b]   Sikkel K. (1993): On-line Parsing in Constant Time per Word. *Theoretical Computer Science* **120**, 303–310.

[Sik97]   Sikkel K.: *Parsing schemata – a framework for specification and analysis of parsing algorithms*. Texts in Theoretical Computer Science - An EATCS Series. Springer-Verlag, Berlin (in preparation).

[SL92]   Sikkel K., Lankhorst M. (1992): A Parallel Bottom-Up Tomita Parser. *1. Konferenz Verarbeitung natürlicher Sprache*, Nürnberg, Germany, 238–247.

[SS89]   Satta G., Stock O. (1989): Head-Driven Bidirectional Parsing: A Tabular Method. *1st International Workshop on Parsing Technologies*, Pittsburgh, Pa., 43–51.

[TDL91]   Thompson H.S., Dixon M., Lamping J. (1991): Compose-Reduce Parsing. *29th Annual Meeting of the Association of Computational Linguistics*, Berkeley, Ca., 87–97,

[TN89]   Tanaka H., Numazaki H. (1989): Parallel Generalized LR Parsing based on Logic Programming. *1st International Workshop on Parsing Technologies*, Pittsburgh, Pa., 329–338.

[Tom85]   Tomita M. (1985): *Efficient Parsing for Natural Language*. Kluwer Academic Publishers, Boston, Mass., 1985.

[VH89]   de Vreught J.P.M., Honig H.J. (1989): A Tabular Bottom-Up Recognizer. Report 90-31, Dept. of Applied Mathematics and Informatics, Delft University of Technology, Delft, the Netherlands.

[VH91]   de Vreught J.P.M., Honig H.J. (1991): Slow and fast parallel recognition. *2nd International Workshop on Parsing Technologies*, Cancun, Mexico, 127–135.

[Vis95]   Visser E. (1995): A Case Study in Optimizing Parsing Schemata by Disambiguation Filters. Report P9507, Dept. of Computer Science, University of Amsterdam, the Netherlands.

[Win83]   Winograd T. (1983): *Language as a Cognitive Process. Vol. I: Syntax*. Addison-Wesley, Reading, Mass.

40

[You67]   Younger D.H. (1967): Recognition of context-free languages in time n^3,
         Information and Control **10**, 189–208.