

Parsing Schemata and Correctness of Parsing Algorithms

Klaas Sikkel

GMD – Forschungszentrum Informationstechnik,
FIT.CSCW, Schloß Birlinghoven,
53757 Sankt Augustin, Germany,
sikkel@gmd.de

ABSTRACT

Parsing schemata provide a high-level formal description of parsers. These can be used, among others, as an intermediate level of abstraction for deriving the formal correctness of a parser. A parser is correct if it duely implements a parsing schema that is known to be correct.

In this paper it is discussed how the correctness of a parsing schema can be proven and how parsing schemata relate to some well-known classes of parsers, viz. chart parsers and LR-type parsers.

1 INTRODUCTION

Parsing schemata were introduced in [Sik93] as a framework for high-level description of parsing algorithms, both parallel and sequential. A parsing schema abstracts from implementation details of an algorithm like data structures and control structures. A prime application of this framework is the analysis of relations between different parsing algorithms by studying formal relations between their underlying parsing schemata. For a concise introduction, see [Sik94].

Here we concentrate on correctness, an aspect of parsing schemata that not been treated very extensively so far. Formal correctness proofs are easier for schemata than for algorithms, simply because there is much less to prove. A general proof method for parsing schemata will be introduced and illustrated with examples.

We do not dwell on implementation details of parsing algorithms, but discuss some general relations between parsing schemata and some well-known classes of algorithms.

Parsing schemata are informally introduced in Section 2 and formalized in Section 3. While this includes a notion of correctness of a schema, it is

not clear how to proceed when one wants to prove a given schema correct. This subject is dealt with in Section 4.

How chart parsers relate to parsing schemata is the subject of Section 5. Parsers based on push-down automata are treated in Section 6.

In Section 7 we briefly review the relation between context-free parsing and unification grammars, which have become the predominant grammar formalism in natural language processing. Conclusions follow in Section 8.

2 PARSING SCHEMATA

We introduce the general idea of a parsing schema by means of a few informal examples. A more rigorous treatment follows in Section 3.

The following conventions apply throughout this article:

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 $A, B, \dots \in N$ for nonterminals; $a, b, \dots \in \Sigma$ for terminals; $X, Y, \dots \in V$ for arbitrary variables; $\alpha, \beta, \dots \in V^*$ for strings of arbitrary variables; ε for the empty string. The letters i, j, \dots 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} . An 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 $a_1 \dots a_n$ to be parsed. The CYK algorithm recognizes *items* $[A, i, j]$ that satisfy $A \Rightarrow^* a_{i+1} \dots a_j$.

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*

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

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 article 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 $a_1 \dots a_n$ can be denoted by a logical deduction system, called a *parsing system*.

Example 2.1 (CYK)

Firstly, we define a domain of items

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

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

¹ Whether the hypotheses are included in the domain

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

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

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

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

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

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}_{CYK} for G and $a_1 \dots a_n$ is defined by the triple $\langle \mathcal{I}, H, D \rangle$.

A *parsing schema* **CYK** is a generalization of \mathbb{P}_{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 $[A \rightarrow \alpha \bullet \beta, i, j]$, with $A \rightarrow \alpha \beta \in P$. The bottom-up Earley parser recognizes the item set

$$\{[A \rightarrow \alpha \bullet \beta, i, j] \mid \alpha \Rightarrow^* a_{i+1} \dots a_j\}$$

for some $G \in \mathcal{CFG}$ and $a_1 \dots a_n \in \Sigma^*$. 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

of items or not does not really matter. It will turn out to be more convenient to define a separate set of hypotheses.

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 $a_1 \dots a_n$ we specify a parsing system \mathbb{P}_{buE} by²

$$\begin{aligned} \mathcal{I}_{\text{buE}} &= \{[A \rightarrow \alpha \bullet \beta, i, j] \mid A \rightarrow \alpha \beta \in P \\ &\quad \wedge 0 \leq i \leq j\}; \\ H_{\text{buE}} &= \{[a, i-1, i] \mid a = a_i \wedge 1 \leq i \leq n\}; \\ D^{\text{Init}} &= \{\vdash [A \rightarrow \bullet \gamma, i, i]\}, \\ D^{\text{Scan}} &= \{[A \rightarrow \alpha \bullet a \beta, i, j], [a, j, j+1] \\ &\quad \vdash [A \rightarrow \alpha a \bullet \beta, i, j+1]\}, \\ D^{\text{Compl}} &= \{[A \rightarrow \alpha \bullet B \beta, i, j], [B \rightarrow \gamma \bullet, j, k] \\ &\quad \vdash [A \rightarrow \alpha B \bullet \beta, i, k]\}, \\ D_{\text{buE}} &= D^{\text{Init}} \cup D^{\text{Scan}} \cup D^{\text{Compl}}. \end{aligned}$$

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 $A \rightarrow \gamma \in P$ is required, as the deduction steps are only meaningful for items drawn from \mathcal{I} and H . D^{Scan} and D^{Compl} conform to the *scan* and *complete* steps of Earley's algorithm. In Figure 1 it is sketched how the *complete* step produces an item representing a larger partial parse from two known partial parses.

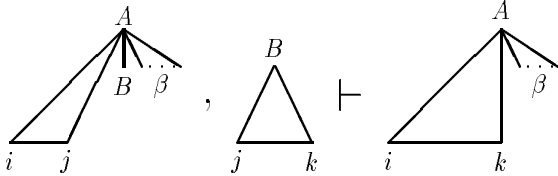


Figure 1: The *complete* step

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 $[A \rightarrow \alpha \bullet B \beta, i, j]$ 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.

²From the usual set notation $\{\dots | \dots\}$ 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 .

Example 2.3 (canonical Earley)

The parsing system $\mathbb{P}_{\text{Earley}}$ 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}_{buE} (cf. Example 2.2) and by D_{Earley} as follows:

$$\begin{aligned} D^{\text{Init}} &= \{\vdash [S \rightarrow \bullet \gamma, 0, 0]\}, \\ D^{\text{Pred}} &= \{[A \rightarrow \alpha \bullet B \beta, i, j] \vdash [B \rightarrow \bullet \gamma, j, j]\}, \\ D^{\text{Scan}} &= \{[A \rightarrow \alpha \bullet a \beta, i, j], [a, j, j+1] \\ &\quad \vdash [A \rightarrow \alpha a \bullet \beta, i, j+1]\}, \\ D^{\text{Compl}} &= \{[A \rightarrow \alpha \bullet B \beta, i, j], [B \rightarrow \gamma \bullet, j, k] \\ &\quad \vdash [A \rightarrow \alpha B \bullet \beta, i, k]\}, \\ D_{\text{Earley}} &= D^{\text{Init}} \cup D^{\text{Scan}} \cup D^{\text{Compl}} \cup D^{\text{Pred}}. \end{aligned}$$

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

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

3 A FORMAL TREATMENT

Parsing systems and parsing schemata are formally introduced in Section 3.1 and 3.2, respectively. Various types of relations between parsing schemata are discussed in 3.3. Section 3.4 reviews 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 $a_1 \dots a_n$ is a triple $\mathbb{P} = \langle \mathcal{I}, H, D \rangle$, in which

- \mathcal{I} is a set of items³, called the *domain* or the *item set* of \mathbb{P} ;
- H is a finite set of items called the *hypotheses* of \mathbb{P} ;
- $D \subseteq \wp_{\text{fin}}(H \cup \mathcal{I}) \times \mathcal{I}$ 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 $\eta_1, \dots, \eta_k \vdash \xi$ rather than $(\{\eta_1, \dots, \eta_k\}, \xi)$. Furthermore if we

³Here we treat 'item' as an undefined basic concept. A discussion about the nature of items follows in Section 3.4.

have $Y = \{\eta_1, \dots, \eta_k\}$, we may also write $Y \vdash \xi$ as an abbreviation for $\eta_1, \dots, \eta_k \vdash \xi$.

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 $\eta_1, \dots, \eta_k \vdash x$ holds, then also
 $\eta_1, \dots, \eta_k, \zeta \vdash x$ 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 $Y' \subseteq Y$.

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 ξ_1, \dots, ξ_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 $(Y; \xi_1, \dots, \xi_j) \in \wp_{fin}(H \cup \mathcal{I}) \times \mathcal{I}^+$, such that $Y \cup \{\xi_1, \dots, \xi_{i-1}\} \vdash \xi_i$ for $1 \leq i \leq j$.

As a practical informal notation we write

$Y \vdash \xi_1 \vdash \dots \vdash \xi_j$

for a deduction sequence $(Y; \xi_1, \dots, \xi_j)$.

Definition 3.4 (\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

$Y \vdash^* \xi$ if $\xi \in Y$ or $Y \vdash \dots \vdash \xi$.

Definition 3.5 (*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 grammars and strings.

Definition 3.6 (*uninstantiated parsing system*)

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

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

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

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

Definition 3.7 (*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 3.8 (**CHK**)

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

Schema 3.9 (**buE**)

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

Schema 3.10 (**Earley**)

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

3.3 RELATIONS BETWEEN PARSING SCHEMATA

Various types of relations between parsing schemata can be defined.

A parsing schema \mathbf{P}_2 is a *refinement* of a schema \mathbf{P}_1 if

- single deduction steps in \mathbf{P}_1 correspond to sequences of deduction steps in \mathbf{P}_2 (and, most likely, \mathbf{P}_2 contains additional items for the additional intermediate results);
- single items in \mathbf{P}_1 correspond to multiple items in \mathbf{P}_2 .

A parsing 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 *generalization* is a combination of refinement and extension (in which either relation can be the identity relation).

We will give a simple, informal example. For formal definitions and a proof of the transitivity of these relations (nontrivial for refinement), see [Sik93, SN96].

Example 3.11

buE is a refinement of **CYK**. This relation is established as follows:

- Firstly, the CYK items $[A, i, j]$ are replaced by final Earley items $[A \rightarrow \alpha \bullet, i, j]$. Note that a single CYK item is replaced, in general, by a set of Earley items, as there can be different productions with left-hand side A . $D^{(2)}$ does now contain deduction steps of the form $[B \rightarrow \beta \bullet, i, j], [C \rightarrow \gamma \bullet, j, k] \vdash [A \rightarrow BC \bullet, i, k]$.
- Secondly, we introduce the remaining Earley items, split each CYK deduction step into the appropriate set of *scan* and *complete* steps, and add *init* steps as in **buE**.
- With the above two simple transformations we have obtained a parsing system \mathbb{P}_{buE} for each grammar in CNF and each string. As a trivial third step, we extend the schema to the entire class of context-free grammars.

Refinement means that more deduction steps have to be performed, in order to find all valid items. This is useful when it leads to a *qualitative* change in the algorithm. **buE** is more useful than **CYK** because it handles a larger class of grammars within the same complexity bounds. For a more mundane *quantitative* improvement, one can try to reduce the number of deduction steps needed to obtain all valid items. This is called *filtering*. One can differentiate between

- *static filtering*: deleting irrelevant items and deduction steps;
- *dynamic filtering*: adding antecedents (= extra conditions) to deduction steps;
- *step contraction*: replacing sequences of steps by single steps.

Again, see [Sik93, SN96] for formal definitions.

Example 3.12

Earley is obtained from **buE** by applying a

(dynamic) filter, as follows. Deduction steps $\vdash [B \rightarrow \bullet \beta, i, i]$ are replaced by deduction steps $[A \rightarrow \alpha \bullet B \gamma, h, i] \vdash [B \rightarrow \bullet \beta, i, i]$, except for the remaining *init* steps in $\mathbb{P}_{\text{Earley}}$.

3.4 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 $a_{i+1} \dots a_j$ whereas the nodes labelled β are leaves, cf. Figure 2.

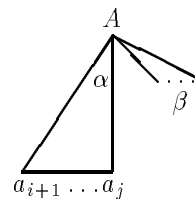


Figure 2: A partially specified tree

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 [Sik93]. 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 artificial—to construct *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 occurring 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} .

Definition 3.13 (*semiregularity*)⁴

A parsing system $\mathbb{P} = \langle \mathcal{I}, H, D \rangle$ for a grammar G and string $a_1 \dots a_n$ is called semiregular if \mathcal{I} does not contain mixed items and each parse tree of $a_1 \dots a_n$ 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 $\mathbf{P}(G)(a_1 \dots a_n)$ is semiregular for all $G \in \mathcal{CG}$ and all $a_1 \dots a_n \in \Sigma^*$.

Definition 3.14 (*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 $a_1 \dots a_n$.

A final item is correct if there is a parse tree for $a_1 \dots a_n$ 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 3.15 (*final and correct final items*)

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

⁴The notion *regularity* was introduced in [Sik93] 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.

Definition 3.16 (*correctness*)

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 grammar \mathcal{CG} if $\mathbf{P}(G)(a_1 \dots a_n)$ is sound/complete/correct for all $G \in \mathcal{CG}$ and $a_1 \dots a_n \in \Sigma^*$.

How to show that **CYK**, **buE**, and **Earley** are indeed correct semiregular parsing schemata will be discussed in the next section.

4 CORRECTNESS REVISITED

We have introduced a formal notion of correctness, following [Sik93] in a simplified form. We must have such a notion, so as to be able to state that certain useful schemata are formally correct. But in order to prove correctness, however, the notions introduced in Section 3.4 are not of much help. For a given parsing system \mathbb{P} , it is generally trivial to identify $\mathcal{F}(\mathbb{P})$ and to establish semiregularity of \mathbb{P} . Given $\mathcal{V}(\mathbb{P})$, furthermore, it is generally trivial to establish $\mathcal{F}(\mathbb{P}) \cap \mathcal{V}(\mathbb{P}) = \mathcal{C}(\mathbb{P})$.

The real problem is to establish $\mathcal{V}(\mathbb{P})$. In the previous sections we have dodged the issue and tacitly assumed that $\mathcal{V}(\mathbb{P})$ is known somehow. For the given examples this was not entirely untrue, because the algorithms for which we gave schemata are well known from the parsing literature. We will now repair this omission and discuss a general method to establish the contents of $\mathcal{V}(\mathbb{P})$. It should be noted that this method is not suitable for automation, because the essential steps require some ingenuity: the criteria for a successful solution are clear, but how to find a successful solution is not stated.

4.1 A PROOF METHOD

Usually, we start with some educated guess which items are valid and which items are not. The task is to turn this educated guess into a firm proof. This is accomplished as follows.

- Define a set of *viable items* $\mathcal{W} \subseteq \mathcal{I}$ for a parsing system \mathbb{P} for some arbitrary $G \in \mathcal{CG}$ and $a_1 \dots a_n \in \Sigma^*$. There is only one right choice: the set $\mathcal{V}(\mathbb{P})$ itself. But at this point we only

guess what $\mathcal{V}(\mathbb{P})$ is. It has to be proven that $\mathcal{V}(\mathbb{P}) = \mathcal{W}$.

- Show the soundness⁵ of all deduction steps, i.e., $\mathcal{V}(\mathbb{P}) \subseteq \mathcal{W}$. To that end, it suffices to show that for all $\eta_1 \dots \eta_k \vdash \xi \in D$ with $\eta_i \in H \cup \mathcal{W}$, $1 \leq i \leq k$, it holds that $\xi \in \mathcal{W}$.
- Construct a so-called *deduction length function* (*dlf*) on \mathcal{W} . *Dlf*'s will be defined shortly. Let d be such a *dlf*. Then the completeness, i.e., $\mathcal{W} \subseteq \mathcal{V}(\mathbb{P})$, is proven by induction on $d(\xi)$. From the assumption that items η with $d(\eta) < m$ are valid, it has to be proven that all ξ with $d(\xi) = m$ are valid.
- $\mathcal{V}(\mathbb{P}) = \mathcal{W}$ follows from the two previous steps. Generalization from parsing systems to parsing schemata is straightforward as usual.

The difficulty is the choice of an appropriate *dlf*. A function d that is guaranteed to be right is $d(\xi) = m$ if there is a deduction sequence $H \vdash^m \xi$ (hence the name “deduction length function”). But it is not at all clear a priori how to do this *without* using induction in the definition of d that would lead to a circularity in the proof.

We define a *dlf* in such a way that completeness follows automatically.

Definition 4.1 (*dlf*)

Let \mathbb{P} be a parsing schema, $\mathcal{W} \subseteq \mathcal{I}$ a set of items. A function $d : H \cup \mathcal{W} \rightarrow \mathbb{N}$ is a *dlf* if

- (i) $d(h) = 0$ for $h \in H$
- (ii) for each $\xi \in \mathcal{W}$ there is some $\eta_1 \dots, \eta_k \vdash \xi \in D$ such that $\{\eta_1 \dots \eta_k\} \subseteq \mathcal{W}$ and $d(\eta_i) < d(\xi)$ for $1 \leq i \leq k$.

Proposition 4.2

Let \mathbb{P} be a parsing schema, $\mathcal{W} \subseteq \mathcal{I}$ a set of items, $d : H \cup \mathcal{W} \rightarrow \mathbb{N}$ a *dlf*. Then $\mathcal{W} \subseteq \mathcal{V}(\mathbb{P})$. \square

Example 4.3 ($\mathcal{V}(\mathbb{P}_{\text{CYK}})$)

For a parsing system \mathbb{P}_{CYK} for arbitrary $G \in \mathcal{CNF}$ and $a_1 \dots a_n \in \Sigma^*$ we define

$$\mathcal{W} = \{[A, i, j] \mid A \Rightarrow^* a_{i+1} \dots a_j\}$$

and a function d by

$$d([A, i, j]) = j - i$$

⁵Soundness means, in this section, soundness of a parsing system with respect to the postulated set of valid items \mathcal{W} . This is stronger than then soundness with respect to final items, as stated in Definition 3.16. The same holds for completeness.

Then d is a *dlf*. The soundness of \mathbb{P}_{CYK} with respect to \mathcal{W} is trivial, hence $\mathcal{V}(\mathbb{P}_{\text{CYK}}) = \mathcal{W}$.

Example 4.4 ($\mathcal{V}(\mathbb{P}_{\text{buE}})$)

For a parsing system \mathbb{P}_{buE} for arbitrary $G \in \mathcal{CFG}$ and $a_1 \dots a_n \in \Sigma^*$ we define

$$\mathcal{W} = \{[A \rightarrow \alpha \bullet \beta, i, j] \mid \alpha \Rightarrow^* a_{i+1} \dots a_j\}$$

and a function d by

$$d([A \rightarrow \alpha \bullet \beta, i, j]) = \min\{\mu + j - i \mid \alpha \Rightarrow^\mu a_{i+1} \dots a_j\}.$$

Then d is a *dlf*. Note that we take the minimum in case there are different ways to recognize an item. Again, the soundness is trivial, hence $\mathcal{V}(\mathbb{P}_{\text{buE}}) = \mathcal{W}$.

4.2 Earley IS CORRECT

For the canonical Earley algorithm, represented by the parsing schema $\mathbb{P}_{\text{Earley}}$ for an arbitrary grammar G and string $a_1 \dots a_n$, it is not immediately clear how to define a *dlf*. Therefore we examine in some detail how the Earley algorithm proceeds through a sentence.

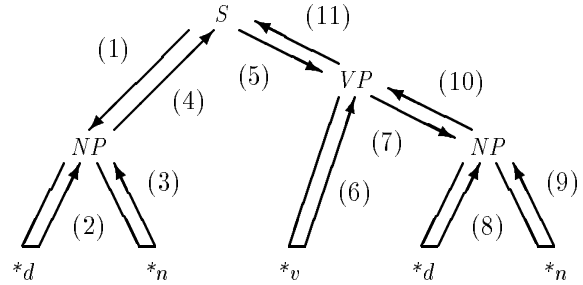


Figure 3: The Earley tree walk

As an exemplary case, we consider a grammar that produced only a single sentence with a single parse tree:

$$S \rightarrow NPVP, \quad NP \rightarrow *d *n, \quad VP \rightarrow *vNP.$$

In Figure 3 it is shown that the algorithm performs a *tree walk* through the parse tree. Every arrow corresponds to (the recognition of) a single item. Predict steps correspond to downward arrows, e.g.

$$[S \rightarrow \bullet NPVP, 0, 0] \vdash [NP \rightarrow \bullet *d *n, 0, 0] \quad (1)$$

Scan steps only move the dot over a terminal, as in

$$[NP \rightarrow \bullet *d *n, 0, 0], [*d, 0, 1] \\ \vdash [NP \rightarrow *d \bullet *n, 0, 1], \quad (2)$$

$$[NP \rightarrow *d \bullet *n, 0, 1], [*n, 1, 2] \\ \vdash [NP \rightarrow *d *n \bullet, 0, 2]. \quad (3)$$

Complete steps correspond to upward arrows:

$$[S \rightarrow \bullet NP VP, 0, 0], [NP \rightarrow *d *n \bullet, 0, 2] \\ \vdash [S \rightarrow NP \bullet VP, 0, 2] \quad (4)$$

and so on.

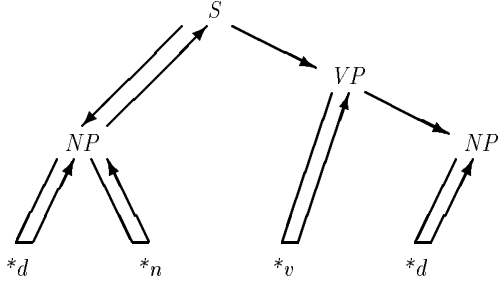


Figure 4: Recognition of $[NP \rightarrow *d \bullet *n, 3, 4]$

Consider now the item $[NP \rightarrow *d \bullet *n, 3, 4]$ that is the result of

$$[NP \rightarrow \bullet *d *n, 3, 3], [*d, 3, 4] \\ \vdash [NP \rightarrow *d \bullet *n, 3, 4]. \quad (8)$$

For clarity, the arrows that have been traversed in order to obtain this item are shown in Figure 4. Expansion of productions needed to predict this NP causes *single* edge traversals; expansion of productions needed in already recognized parts of the sentence causes *double* edge traversals. The notion of predict steps is expressed in terms of derivations as follows. Let $S \Rightarrow^\lambda a_1 \dots a_i A \gamma$. Then there are δ and π such that $S \Rightarrow^\pi \delta A \gamma$ and $\delta \Rightarrow^{\lambda-\pi} a_1 \dots a_i$. In the above example: $S \Rightarrow^2 NP *v NP$ and $NP *v \Rightarrow^1 *d *n *v$.

It should be stressed, perhaps, that the analysis of a single example does not play a role in the proof. It serves to make an educated guess for an appropriate *dlf*; if we guess right the proof that we did so is generally straightforward. The above case should provide sufficient intuition to complete the Earley case.

Example 4.5 ($\mathcal{V}(\mathbb{P}_{\text{Earley}})$)

For a parsing system $\mathbb{P}_{\text{Earley}}$ for arbitrary $G \in \mathcal{CFG}$ and $a_1 \dots a_n \in \Sigma^*$ we define

$$\mathcal{W} = \{[A \rightarrow \alpha \bullet \beta, i, j] \mid \alpha \Rightarrow^* a_{i+1} \dots a_j \\ \wedge S \Rightarrow^* a_1 \dots a_i A \gamma\}.$$

The soundness of $\mathbb{P}_{\text{Earley}}$ with respect to \mathcal{W} is trivial as usual. For the completeness we define a function $d : \mathcal{W} \rightarrow \mathbb{N}$ by

$$d([A \rightarrow \alpha \bullet \beta, i, j]) = \\ \min\{\pi + 2\lambda + 2\mu + j \mid S \Rightarrow^\pi \delta A \gamma \wedge \\ \delta \Rightarrow^\lambda a_1 \dots a_i \wedge \\ \alpha \Rightarrow^\mu a_{i+1} \dots a_j\}.$$

Note that d is indeed properly defined on \mathcal{W} . It remains to be checked that condition (ii) in Definition 4.1 holds for all $\xi \in \mathcal{W}$. We distinguish the following cases:

- $\xi = [A \rightarrow \alpha \bullet \beta, i, j + 1]$:
Then $\eta = [A \rightarrow \alpha \bullet a \beta, i, j] \in \mathcal{W}$ and $\zeta = [a, j, j + 1] \in H$. Moreover, $d(\zeta) = 0$ and $d(\eta) = d(\xi) - 1$.
- $\xi = [A \rightarrow \alpha B \bullet \beta, i, k]$:
Let $\alpha \Rightarrow^\mu a_{i+1} \dots a_j$ and $B \Rightarrow \gamma \Rightarrow^\varrho a_{j+1} \dots a_k$ with minimal $\mu + \varrho$. Then $\eta = [A \rightarrow \alpha \bullet B \beta, i, j] \in \mathcal{W}$, $\zeta = [B \rightarrow \gamma \bullet, j, k] \in \mathcal{W}$, and $d(\eta) < d(\zeta) = d(\xi) - 1$.
- $\xi = [B \rightarrow \bullet \gamma, j, j]$:
Let $S \Rightarrow^* a_1 \dots a_j B \gamma'$, and let $\delta, \gamma'', \pi, \lambda, \mu$ such that $S \Rightarrow^\pi \delta A \gamma''$, $\delta \Rightarrow^\lambda a_1 \dots a_i$, $A \Rightarrow \alpha B \beta$, $\alpha \Rightarrow^\mu a_{i+1} \dots a_j$ with $\pi + 2\lambda + 2\mu$ minimal. Then $\eta = [A \rightarrow \alpha \bullet B \beta] \in \mathcal{W}$, $\eta \vdash \xi$, and $d(\eta) = d(\xi) - 1$.

Hence we conclude

$$\mathcal{V}(\mathbb{P}_{\text{Earley}}) = \{[A \rightarrow \alpha \bullet \beta, i, j] \mid \\ \alpha \Rightarrow^* a_{i+1} \dots a_j \wedge S \Rightarrow^* a_1 \dots a_i A \gamma\}. \quad \square$$

4.3 SLR(1) IS CORRECT

There is a close relation between Earley-type algorithms and LR parsers. This will be discussed in Section 6. An LR(0) parser is in fact an implementation of the parsing schema **Earley**⁶. We will now introduce a schema that employs *look-ahead*. This constitutes another kind of filtering.

Recognition of an item does not need to take place if the next symbol 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' .

⁶Note, however, that LR-type parsers—also generalized LR-parsers like Tomita's algorithm—make some restrictions on the class of grammars that can be used.

Definition 4.6 (*augmented grammar*)

For each grammar $G \in \mathcal{CFG}$ we define an augmented grammar G' by

$$\begin{aligned} N' &= N \cup \{S'\}, \\ \Sigma' &= \Sigma \cup \{\$, \}, \\ P' &= P \cup \{S' \rightarrow S\$, \}, \\ G' &= (N', \Sigma', P', S') \end{aligned}$$

with $\{S', \$\} \cap V = \emptyset$.

For a string $a_1 \dots a_n$ an augmented grammar has only a single final item $[S' \rightarrow S.\$, 0, n]$.

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 4.7 (SLR(1))

The parsing schema **SLR(1)** is defined by a parsing system $\mathbb{P}_{\text{SLR}(1)}$ for any $G \in \mathcal{CFG}$ and for any $a_1 \dots a_n \in \Sigma^*$ by

$$\begin{aligned} \mathcal{I}^{\text{Compl}} &= \{[A \rightarrow \alpha.\beta, i, j] \mid A \rightarrow \alpha\beta \in P' \\ &\quad \wedge 0 \leq i \leq j\}; \\ H &= \{[a, i-1, i] \mid a = a_i \wedge 1 \leq i \leq n\} \\ &\quad \cup \{[\$, n, n+1]\}; \\ D^{\text{Init}} &= \{\vdash [S' \rightarrow .S\$, 0, 0]\}, \\ D^{\text{Pred}} &= \{[A \rightarrow \alpha.B\beta, i, j] \vdash [B \rightarrow \cdot\gamma, j, j]\}, \\ D^{\text{Scan}} &= \{[A \rightarrow \alpha.a\beta, i, j], [a, j, j+1] \\ &\quad \vdash [A \rightarrow \alpha a.\beta, j, j+1]\}, \\ D^{\text{Compl}} &= \{[A \rightarrow \alpha.B\beta, i, j], [B \rightarrow \gamma.\cdot, j, k], \\ &\quad [a, k, k+1] \\ &\quad \vdash [A \rightarrow \alpha B.\beta, i, k] \mid a \in \text{FOLLOW}(B)\}, \\ D_{\text{SLR}(1)} &= D^{\text{Init}} \cup D^{\text{Pred}} \cup D^{\text{Scan}} \cup D^{\text{Compl}}. \end{aligned}$$

Note that it is possible to exploit the look-ahead more efficiently, for example by using $a \in \text{FIRST}(\beta \text{ FOLLOW}(A))$, rather than $a \in \text{FOLLOW}(B)$ to filter irrelevant items. Also, one could apply a filter to the *scan* steps. But the schema has been defined such that incorporates exactly the same look-ahead that is used in the construction of an SLR(1) parsing table.

Next, we will establish $\mathcal{V}(\mathbb{P}_{\text{SLR}(1)})$ with the method introduced above. We define a set

$$\mathcal{W} = \mathcal{W}^{\text{Scan}} \cup \mathcal{W}^{\text{Compl}} \cup \mathcal{W}^{\text{Pred}} \cup \mathcal{W}^{\text{Init}}$$

by

- $[A \rightarrow \alpha a.\beta, i, j+1] \in \mathcal{W}^{\text{Scan}}$ if
 - (i) $S' \Rightarrow^* a_1 \dots a_i A \gamma \$$,
 - (ii) $\alpha \Rightarrow^* a_{i+1} \dots a_j$;
- $[A \rightarrow \alpha B.\beta, i, k] \in \mathcal{W}^{\text{Compl}}$ if there is some $j < k$ such that
 - (i) $S' \Rightarrow^* a_1 \dots a_i A \gamma \$$,
 - (ii) $\alpha \Rightarrow^* a_{i+1} \dots a_j$,
 - (iii) $B \Rightarrow^* a_{j+1} \dots a_k$,
 - (iv) $a_{k+1} \in \text{FOLLOW}(B)$;
- $[A \rightarrow \alpha B.\beta, i, j] \in \mathcal{W}^{\text{Compl}}$ if
 - (i) $S' \Rightarrow^* a_1 \dots a_i A \gamma \$$,
 - (ii) $\alpha \Rightarrow^* a_{i+1} \dots a_j$,
 - (iii) $B \Rightarrow^* \varepsilon$,
 - (iv) $[A \rightarrow \alpha.B\beta, i, j] \in \mathcal{W}$,
 - (v) $a_{j+1} \in \text{FOLLOW}(B)$;
- $\mathcal{W}^{\text{Init}} = \{[S' \rightarrow .S\$, 0, 0]\}$;
- $[C \rightarrow \cdot\gamma, t, j] \in \mathcal{W}^{\text{Pred}}$ if there is some
 $[A \rightarrow \alpha.B\beta, i, j] \in \mathcal{W}^{\text{Init}} \cup \mathcal{W}^{\text{Scan}} \cup \mathcal{W}^{\text{Compl}}$
 such that $B \Rightarrow_{rm}^+ C \delta$ for some δ
 (where \Rightarrow_{rm}^+ denotes *rightmost* derivation).

Note that the recursion in the definition of \mathcal{W} only relates to *nullable* symbols (i.e., B such that $B \Rightarrow^* \varepsilon$). These have to be taken proper care of. Consider, for example a grammar G defined by productions $S \rightarrow ABA$, $S \rightarrow CBC$, $A \rightarrow a$, $B \rightarrow \varepsilon$ $C \rightarrow c$ and an input string ac . Then $[S \rightarrow AB.A, 0, 1] \notin \mathcal{W}$. There is a deduction step $[S \rightarrow A.BA, 0, 1], [B \rightarrow \cdot, 1, 1], [c, 1, 2] \vdash [S \rightarrow AB.A, 0, 1] \in D$, but this is never activated because $[S \rightarrow A.BA, 0, 1] \notin \mathcal{W}$. Completion (or, in LR-terminology, *reduction*) of the first A is blocked by the look-ahead c .

The function d on \mathcal{W} is similar to the one defined on $\mathbb{P}_{\text{Earley}}$:

$$\begin{aligned} d([A \rightarrow \alpha.\beta, i, j]) &= \\ &\min\{\pi + 2\lambda + 2\mu + j \mid S \Rightarrow^\pi \delta A \gamma \wedge \\ &\quad \delta \Rightarrow^\lambda a_1 \dots a_i \wedge \\ &\quad \alpha \Rightarrow^\mu a_{i+1} \dots a_j\}. \end{aligned}$$

In order to prove that $\mathcal{V}(\mathbb{P}_{\text{SLR}(1)}) = \mathcal{W}$ we have to prove:

- (i) for all $\eta_1 \dots \eta_k \vdash \xi \in D$ with $\eta_i \in H \cup \mathcal{W}$, $1 \leq i \leq k$, it holds that $\xi \in \mathcal{W}$ (soundness);

(ii) for each $\xi \in \mathcal{W}$ there is some $\eta_1, \dots, \eta_k \vdash \xi \in D$ such that $d(\eta_i) < d(\xi)$ for $1 \leq i \leq k$. (*dlf*).

(i) can be verified straightforwardly. For a proof of (ii) we do not have to worry about the d -values of the items this time. All deduction sequences in $\mathbb{P}_{\text{SLR}(1)}$ do exist in $\mathbb{P}_{\text{Earley}}$ as well. The difference is that less items are recognized. The point that must be clarified here, is whether for any $\xi \in \mathcal{W}$ a deduction step $\eta_1, \dots, \eta_k \vdash \xi$ can be found with all $\eta_i \in \mathcal{W}$. The reader may verify that this follows straightforwardly in all cases. Note that the *rightmost* condition in $B \Rightarrow_{r,m}^+ C\delta$ is essential: it prevents derivations of the type $B \Rightarrow^+ \alpha C\delta \Rightarrow^+ C\delta$ in which α rewrites to ε .

Hence, we have shown that $\mathcal{V}(\mathbb{P}_{\text{SLR}(1)}) = \mathcal{W}$ as defined above. \square

4.4 OTHER APPLICATIONS

Establishing the correctness of **CYK** and **buE** was trivial, but **Earley** and **SLR(1)** required some ingenuity. This raises the question how the proposed proof method “scales up” to more complicated schemata that are not generally known to be correct from the parsing literature. Rather more involved examples are given in [SA92, Sik93, SA96], where correctness of (parsing schemata for) Left-Corner (LC) and Head-Corner (HC) parsers is established using the same technique. A more detailed treatment of schemata for LC and HC parsers would require a lot of space, hence the interested reader is referred to the cited publications. *Finding* a deduction length function is not more difficult than the Earley case (for LC and HC it is easier, in fact). An analysis of how a single parse tree is processed by the algorithm suggests an appropriate function d . The schemata **LC** and **HC** have different kinds of items and more kinds of deduction steps, hence the proof that d is indeed a *dlf* requires checking quite a large number of different cases. But each of these cases is straightforward as in the above examples. In sum, more elaborate parsing schemata require hardly more complicated proofs. We claim that the predictive Head-Corner parser proposed [SA92, Sik93, SA96] is the only HC parser that has ever been formally proven correct.

Head-Corner parsers do not process a sentence from left to right but start with the “most interesting” parts. In each production some right-hand side element is designated as head. Evaluation of a production always starts with evaluation of the head. Head-first evaluation can

speed up the parser when the information known from the head avoids the evaluation of dead ends that would have been explored otherwise. But the fact that the processed part of a sentence is not contiguous burdens the parser with a rather more complicated administration. Other work on Head-Corner parsing can be found in [Kay89, SS89, BN93, Mol95, A&a95].

5 CHART PARSERS

So far we have only discussed parsing schemata. Although we will not dwell on specific parsing algorithms in detail, we discuss some general classes of parsing algorithms in this section and the next one.

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. The general chart parsing algorithm is shown in Figure 5. 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 $\mathcal{V}(\mathbb{P})$. It needs no further elaboration that if a parsing schema is correct, then also the chart parser for this schema is correct.

In the most general form a chart parser is not particularly efficient. In order to speed up parsing, the chart and agenda have to be enhanced with data structures that allow efficient searching and storing of relevant items. How this can be done for various kinds of chart parsers is beyond the scope of this article.

6 PUSHDOWN AUTOMATA

An important class of parsing algorithms is based on the notion of a 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.

```

program chart parser
begin
  create initial chart and agenda;
  while agenda is not empty
  do
    delete some (arbitrarily chosen)
    current item from agenda;
    for each item that can be recognized
      by current in combination with
      other items in chart
    do
      if item is neither in chart
        nor in agenda
      then add item to agenda
      fi
    od
  od
end.

```

Figure 5: The chart parser algorithm

6.1 (G)LR PARSERS

Many parsing algorithms in the field of compiler construction are based on the PDA paradigm. The canonical example is the family of *LR-parsers*, which was 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.

Certain classes of grammars cannot be handled by Tomita's algorithm in its original form: cyclic grammars and grammars with hidden left-recursion⁷. Rekers has improved Tomita's algorithm to handle these grammars as well [Rek92].

⁷A grammar is called hidden left-recursive if $A \Rightarrow^+ \alpha A \beta$ and $\alpha \Rightarrow^+ \varepsilon$. The term has been coined in [NS93].

6.2 FROM SCHEMATA TO PDA'S

The question arises how PDA-based algorithms like LR and parsing schemata are related to each other. To that end, we will transform the schema **Earley** (or, to be precise, an uninstantiated parsing system for some grammar G) to a PDA, and argue that its correctness (in the sense of Section 3.4) is preserved.

We use a somewhat opportunistic definition of a PDA, that is tuned towards the description of parsers. This is not unusual, however. See, e.g., [Oud93] and [Ned94] for similar definitions of PDA's designed to model parsing algorithms. For the sake of brevity we will only consider recognition and not dwell on how the PDA can be augmented to a *push-down transducer* that yields a parse tree as a side result of recognizing a string.

A PDA is defined by means of a deduction relation on *instantaneous descriptions*, also called *configurations*. A configuration consists of a stack and the remainder of the input. It is a "snapshot" of a PDA at work. From a given configuration, the PDA may move to another configuration, as laid down in rules that take into account the (top part of the) stack and the (beginning of the) remaining input. We write $\varphi, \psi, \dots \in \mathcal{I}^*$ for stacks and parts of stacks. Configurations are denoted as pairs $(\varphi, w) \in (\mathcal{I}^* \times \Sigma^*)$.

Deduction rules for configurations are usually defined by means of a *finite* transition table. We will not demand this in the definition; our first PDA actually requires an infinite transition table, but this inconvenience will be eliminated in the second PDA.

Definition 6.1 (PDA)

Let G be a context-free grammar and G' its augmented grammar according to Definition 4.6.

A pushdown automaton Π for G is a quadruple $\langle \mathcal{I}, \xi_0, \mathcal{F}, D \rangle$ in which

- \mathcal{I} is a set of items;
- $\xi_0 \in \mathcal{I}$ a *start item*;
- $\mathcal{F} \subseteq \mathcal{I}$ a set of *final items*;
- $D \subseteq (\mathcal{I}^* \times \Sigma^*) \times (\mathcal{I}^* \times \Sigma^*)$ a set of deduction steps.

Definition 6.2 (acceptance)

Let Π be a PDA for some grammar $G \in \mathcal{CFG}$. A string $a_1 \dots a_n \in \Sigma^*$ is accepted by Π if

$$(\xi_0, a_1 \dots a_n \$) \vdash^* (\varphi \xi_F, \$)$$

for some $\varphi \in \mathcal{I}^*$ and $\xi_F \in \mathcal{F}$.

Definition 6.3 (Π_{Earley})

The PDA Π_{Earley} is defined for a grammar $G \in \mathcal{CFG}$ by

$$\begin{aligned} \mathcal{I}_{\text{Earley}} &= \{[A \rightarrow \alpha \bullet \beta, i, j] \mid A \rightarrow \alpha \beta \in P'\}, \\ \xi_0 &= [S' \rightarrow \bullet S \$, 0, 0], \\ \mathcal{F} &= \{[S' \rightarrow S \bullet \$, 0, n] \mid n \geq 0\}, \\ D^{\text{Pred}} &= \{(\varphi[A \rightarrow \alpha \bullet B \beta, i, j], w) \vdash \\ &\quad (\varphi[A \rightarrow \alpha \bullet B \beta, i, j][B \rightarrow \bullet \gamma, j, j], w)\}, \\ D^{\text{Sh}} &= \{(\varphi[A \rightarrow \alpha \bullet a \beta, i, j], aw) \vdash \\ &\quad (\varphi[A \rightarrow \alpha \bullet a \beta, i, j]A \rightarrow \alpha a \bullet \beta, i, j + 1], w)\}, \\ D^{\text{Re}} &= \\ &\quad \{(\varphi[A \rightarrow \alpha \bullet B \beta, i, j][B \rightarrow \bullet \gamma, j, j] \dots [B \rightarrow \gamma \bullet, j, k], \\ &\quad w) \vdash (\varphi[A \rightarrow \alpha \bullet B \beta, i, j][A \rightarrow \alpha B \bullet \beta, i, k], w)\}, \\ D_{\text{Earley}} &= D^{\text{Pred}} \cup D^{\text{Sh}} \cup D^{\text{Re}}. \end{aligned}$$

We have replace the terms *scan* and *complete* by *shift* and *reduce*, respectively, the latter ones being more usual to describe LR-type parsers. It should be clear how these operations relate to each other. The PDA Π_{Earley} recognizes the same items for $a_1 \dots a_n$ as the parsing schema $\mathbb{P}_{\text{Earley}}$.

One of the advantages of restricting the notion of correctness to *final* item in Section 3.4 is that we can relate correctness of parsing schemata and pushdown automata straightforwardly.

Proposition 6.4 (*equivalence of Earley(G) and Π_{Earley}*)

Let Π_{Earley} the PDA for some grammar $G \in \mathcal{CFG}$ and $\mathbb{P}_{\text{Earley}}$ the parsing system for G and some string $a_1 \dots a_n \in \Sigma^*$. Then Π_{Earley} recognizes $a_1 \dots a_n$ if and only if $\mathcal{C}(\mathbb{P}_{\text{Earley}}) \neq \emptyset$.

Proof. Straightforward. \square

Next, we observe that the position markers in the items in Π_{Earley} do not have much relevance. Position markers were necessary in a parsing schema, in order to know which part of the sentence an item relates to. But with all the previous items stacked, and the remainder of the sentence given, this information can be discarded.

So we can simplify the PDA. The next PDA, surprisingly, defines an LL(0) parser (this is elaborated further in, e.g., [OS92, Oud93]).

Definition 6.5 ($\Pi_{\text{LL}(0)}$)

The PDA $\Pi_{\text{LL}(0)}$ is defined for a grammar $G \in \mathcal{CFG}$ by

$$\begin{aligned} \mathcal{I}_{\text{LL}(0)} &= \{[A \rightarrow \alpha \bullet \beta] \mid A \rightarrow \alpha \beta \in P'\}, \\ \xi_0 &= [S' \rightarrow \bullet S \$], \\ \mathcal{F} &= \{[S' \rightarrow S \bullet \$]\}, \\ D^{\text{Pred}} &= \{(\varphi[A \rightarrow \alpha \bullet B \beta], w) \vdash \\ &\quad (\varphi[A \rightarrow \alpha \bullet B \beta][B \rightarrow \bullet \gamma], w)\}, \\ D^{\text{Sh}} &= \{(\varphi[A \rightarrow \alpha \bullet a \beta], aw) \vdash \\ &\quad (\varphi[A \rightarrow \alpha \bullet a \beta][A \rightarrow \alpha a \bullet \beta], w)\}, \\ D^{\text{Re}} &= \\ &\quad \{(\varphi[A \rightarrow \alpha \bullet B \beta][B \rightarrow \bullet \gamma] \dots [B \rightarrow \gamma \bullet], w) \\ &\quad \vdash (\varphi[A \rightarrow \alpha \bullet B \beta][A \rightarrow \alpha B \bullet \beta], w)\}, \\ D_{\text{LL}(0)} &= D^{\text{Pred}} \cup D^{\text{Sh}} \cup D^{\text{Re}}. \end{aligned}$$

Proposition 6.6 (*equivalence of Π_{Earley} and $\Pi_{\text{LL}(0)}$*)

Let Π_{Earley} and $\Pi_{\text{LL}(0)}$ be PDAs for some grammar $G \in \mathcal{CFG}$. Then $\Pi_{\text{LL}(0)}$ recognizes a string $a_1 \dots a_n \in \Sigma^*$ if and only if Π_{Earley} recognizes $a_1 \dots a_n$.

Proof. Trivial. \square

The transformation from LL(0) to LR(0) need not be spelled out. See, e.g., [OS92, Oud93] for a more detailed treatment. Two further transformations are needed:

- Items are extended from dotted productions to sets of dotted productions. For each item, its *closure* is computed by inserting all the dotted rules that can be obtained with *predict* steps. Hence the *predict* rule can be eliminated from D . Moreover, one item less is popped from the stack in a *reduce* step.
- A full-fledged LR(0) PDA is obtained by combining items. If the dotted productions $[A \rightarrow \alpha \bullet X \beta]$, $[A' \rightarrow \alpha' \bullet X \beta']$, \dots are contained in a single item, then the item set contains another item

$$\text{closure}(\{[A \rightarrow \alpha X \bullet \beta], [A' \rightarrow \alpha' X \bullet \beta'], \dots\}).$$

The algorithm for the construction of the set of LR(0) items can be found in any textbook on compiler construction, e.g. [ASU86].

There is much more to say about the intricacies of constructing LR-type parsers but this is not the place to do so. An issue that has to be avoided is nontermination. On the—still abstract—level of PDA's this issue does not exist. In an implementation, however, it has to be avoided that a parser gets stuck in an infinite loop. Chart parsers are more robust, because by definition (cf. Figure 5) each item is recognized only once.

6.3 CHART PARSERS VS. PDA'S

We have sketched how the uninstantiated parsing system **Earley**(G) can be transformed into an LR(0) PDA. The transformation is bidirectional. The parsing schema **SLR**(1) has in fact been derived by an analogous transformation in reverse direction. For other LR-type algorithms, an underlying parsing schema can be derived in similar fashion.

Recalling that chart parsers are in fact canonical implementations of parsing schemata, we have sketched a general relation between these two seemingly different parsing paradigms. This does not come as a surprise, though, the close relation between Earley's algorithm and Tomita's algorithm has been known for considerable time and was investigated in detail in [Sik91].

7 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. The interested reader is referred to [Shi86] for a simple introduction and [Shi92] and [Car92] for a thorough treatment of unification logics. Parsing schemata for a simple kind unification grammar have been defined in [Sik93]. A parsing schema for a unification-based ID/LP grammar is described in [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.

Independently of each other, Nagata [Nag92] and Maxwell and Kaplan [MK93] have 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.

Therefore, context-free parsing remains to be of importance to natural language analysis.

8 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).

Correctness proofs are easier at this level of abstraction, simply because there is less to prove. The correctness of an algorithm can be derived by showing that is a correct implementation of a schema that is known to be correct.

A general method to prove the correctness of a parsing schema has been introduced, and illustrated with various examples. Also, we have shown how parsing schemata are related to two important classes of parsing algorithms, viz., chart parsers and pushdown automata. A chart parser can be regarded as the canonical implementation of some parsing schema. A PDA can be obtained from a parsing schema—and reversed—with a straightforward transformation that preserves the correctness.

REFERENCES

- [ASU86] Aho A.V., Sethi R., Ullman J.D. (1986): *Compilers: Principles, Techniques and Tools*. Addison-Wesley, Reading, Mass.
- [A&a95] op den Akker R., ter Doest H., Moll M., Nijholt A. (1995): Parsing in dialogue systems using typed feature structures. *Memo-randa Informatica 95-25*, Department of Computer Science, University of Twente, Enschede, the Netherlands.
<http://www.cs.utwente.nl/~moll/opdenakker.parsing-dialogues.ps.Z>
- [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.
- [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.
- [Knu65] Knuth D.E. (1965): On the Translation of Languages from Left to Right. *Information and Control* **8**, 607–639.
- [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.
- [Nij83] Nijholt A. (1983): Deterministic Top-Down and Bottom-Up Parsing: Historical Notes and Bibliographies. Mathematisch Centrum, Amsterdam, the Netherlands.
- [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.
<http://www.cs.utwente.nl/~moll/moll.MScThesis.ps.Z>
- [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.
- [Ned94] Nederhof M.J. (1994): Linguistic Parsing and Program Transformations. Ph.D. Thesis, University of Nijmegen, the Netherlands.
- [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.
- [OS92] Oude Luttighuis P., Sikkel K. (1992): Attribute Evaluation during Generalized Parsing. *Memo-randa Informatica 92-85*, Computer Science Department, University of Twente, Enschede, the Netherlands.
- [Oud93] Oude Luttighuis P. (1993): Parallel algorithms for parsing and attribute evaluation. Ph.D. Thesis, University of Twente.
- [Rek92] Rekers J. (1992): Parser Generation for Interactive Environments. Ph.D. Thesis, University of Amsterdam.
- [SA92] Sikkel K., op den Akker R. (1992): Left-Corner and Head-Corner Chart Parsing. *Memo-randa Informatica 95-22*, University of Twente, Enschede, The Netherlands.
- [SA96] Sikkel K., op den Akker R. (1996): Predictive Head-Corner Chart Parsing. In: Tomita M., Bunt H. (Eds), *Current Issues in Parsing Technology, Vol. 2*, Kluwer Academic Publishers, New York. (to appear)
<http://orgwis.gmd.de/~sikkel/papers/TomitaBunt.ps>

- [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.
- [Sik91] Sikkel K. (1991): Cross-Fertilization of Earley and Tomita. In: van der Wouden T., Sijtsma W. (Eds.), *Computational Linguistics in The Netherlands, Papers from the first CLIN meeting, October 1990*, OTS, Rijksuniversiteit Utrecht, the Netherlands.
- [Sik93] Sikkel K. (1993): *Parsing Schemata*. Ph.D. Thesis, University of Twente, Enschede, the Netherlands.
<http://orgwis.gmd.de/~sikkel/papers/PhD.html>
- [Sik94] Sikkel K. (1994): How to compare the structure of parsing algorithms. In: Pighizzini G., San Pietro P. (Eds.), *Proc. ASMICS Workshop on Parsing Theory, Milano, Italy, October 1994*. Technical Report 126-94, Dipartimento di Scienze dell'Informazione, Università di Milano, 21-39.
<http://orgwis.gmd.de/~sikkel/papers/asmics94.ps>
- [SN96] Sikkel K., Nijholt A. (1996): Parsing of Context-Free Languages. In: Rozenberg G., Salomaa A. (Eds), *Handbook of Formal Languages, Vol. II*, Springer Verlag (to appear).
<http://orgwis.gmd.de/~sikkel/papers/handbook.ps>
- [SS89] Satta G., Stock O. (1989): Head-Driven Bidirectional Parsing: A Tabular Method. *1st International Workshop on Parsing Technologies*, Pittsburg, Pa., 43-51.
- [Tom85] Tomita M. (1985): *Efficient Parsing for Natural Language*. Kluwer Academic Publishers, Boston, Mass., 1985.
- [Win83] Winograd T. (1983): *Language as a Cognitive Process. Vol. I: Syntax*. Addison-Wesley, Reading, Mass.
- [You67] Younger D.H. (1967): Recognition of context-free languages in time n^3 , *Information and Control* **10**, 189-208.