ELSEVIER

Theoretical Computer Science 199 (1998) 87–103

Theoretical Computer Science

Parsing schemata and correctness of parsing algorithms

Klaas Sikkel *

Faculty of Computer Science, University of Twente, P.O. Box 217, 7500 AE Enschede, Netherlands

Abstract

Parsing schemata give 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 duly implements a parsing schema that is known to be correct.

We discuss 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. © 1998 — Elsevier Science B.V. All rights reserved

Keywords: Parsing schemata; Chart parsers; Correctness proof

1. Introduction

Parsing schemata were introduced in [25] 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. See [29] for a concise overview and [27] for a comprehensive treatment.

This article is concerned with correctness, an aspect of parsing schemata that has not been treated extensively. A general proof method for parsing schemata correctness is introduced and illustrated with examples. A parsing schema is easier to prove correct than a parsing algorithm, because there is less to prove.

Parsing schemata are introduced in Section 2. How to prove the correctness of a schema is elaborated in Section 3. In Section 4 it is discussed how correct schemata can be refined into correct parsers, focussing on chart parsers and LR-type parsers. Conclusions are summarized in Section 5.

2. Parsing schemata

We introduce the general idea of a parsing schema by means of a few informal examples in Section 2.1 and then formalize this in Sections 2.2–2.4.

* E-mail: sikkel@cs.utwente.nl.

PII S0304-3975(97)00269-7

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 $A,B,\ldots\in N$ 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 α_1, α_2, A, γ such that $\alpha=\alpha_1A\alpha_2$, $\beta=\alpha_1\gamma\alpha_2$ and $A\rightarrow\gamma\in P$.

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

2.1. Some informal examples

A very simple parsing algorithm is the so-called CYK algorithm [11, 32], called after Cocke, Younger and Kasami. It is restricted to grammars in Chomsky Normal Form. For some grammar $G\in CNF$ and a string $a_1\ldots a_n$, 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 $a=a_j$ and $A\rightarrow a\in P$ then A can be added to entry $T_{j-1,j}$. If $B\in T_{i,k}$, $C\in T_{k,j}$ and $A\rightarrow BC\in P$ then A can be added to $T_{i,j}$. The CYK algorithm employs 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 straightforwardly from the items in this set. If $[S,0,n]$ has been recognized, 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. Hence, strictly speaking, 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 $G\in CNF$ and string $a_1\ldots a_n$ can be denoted by a logical deduction system, called a *parsing system*.

Example 2.1 (*CYK*). We define a *domain of items* $\mathscr{I}_{\mathrm{CYK}}$ and represent the string with a *set of hypotheses* H:

$$\mathscr{I}_{\mathrm{CYK}} = \{[A,i,j] \mid A \in N \wedge 0 \leqslant i < j\},$$
$$H = \{[a,i-1,i] \mid a = a_i \wedge 1 \leqslant i \leqslant n\}.$$

Whether the hypotheses are included in the domain of items or not does not really matter. It has pragmatic advantages to define these separately.

We specify an inference rule by a *set of deduction steps* that covers all instances of inferences. A set of inference rules, then, can be denoted by the union of corresponding sets of deduction steps. For CYK we define:

$$D^{(1)} = \{[a,i-1,i] \vdash [A,i-1,i] \mid A \to a \in P\},$$
$$D^{(2)} = \{[B,i,j],[C,j,k] \vdash [A,i,k] \mid A \to BC \in P\},$$
$$D_{\mathrm{CYK}} = D^{(1)} \cup D^{(2)}.$$

The parsing system $\mathbb{P}_{\mathrm{CYK}}$ for G and $a_1 \ldots a_n$ is defined by the triple $\langle \mathscr{I}, H, D \rangle$.

Note that we could have restricted $\mathscr{I}$ to items with $j \leqslant n$, and D accordingly, but it has certain advantages to define $\mathscr{I}$ and D independent of (the length of) a particular sentence.

The CYK algorithm is restricted to grammars in Chomsky Normal Form. A more general algorithm for arbitrary context-free grammars is the bottom-up variant of Earley's algorithm [8, 9].

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

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

for some $G \in CFG$ and $a_1 \ldots a_n \in \Sigma^*$. 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. The right-hand side of the production is recognized step by step by 'moving the dot rightwards', i.e. recognizing the symbol behind the dot. A sentence $a_1 \ldots a_n$ is correct if and only if an item of the form $[S \to \gamma \bullet, 0, n]$ can be recognized.

A parsing system $\mathbb{P}_{\mathrm{buE}}$ for G and $a_1 \ldots a_n$ is specified by defining appropriate $\mathscr{I}$, H, and D. It should be evident (and it will be formally stated in Section 2.2) that deduction steps are meaningful only for items drawn from $\mathscr{I}$ order H. For the sake of brevity, this is omitted in the specification. In those cases where the second part of the usual set notation $\{\ldots \mid \ldots\}$ becomes empty, we discard the vertical bar as well. Hence, we obtain the following concise specification of $\mathbb{P}_{\mathrm{buE}}$:

$$\mathscr{I}_{\mathrm{buE}} = \{[A \to \alpha \bullet \beta, i, j] \mid A \to \alpha\beta \in P \wedge 0 \leqslant i \leqslant j\},$$
$$H = \{[a,i-1,i] \mid a = a_i \wedge 1 \leqslant i \leqslant n\},$$

$$D^{\text{Init}} = \{\vdash [A \to \bullet\gamma, i, i]\},$$
$$D^{\text{Scan}} = \{[A \to \alpha\bullet a\beta, i, j], [a, j, j+1] \vdash [A \to \alpha a\bullet\beta, i, j+1]\},$$
$$D^{\text{Compl}} = \{[A \to \alpha\bullet B\beta, i, j], [B \to \gamma\bullet, j, k] \vdash [A \to \alpha B\bullet\beta, i, k]\},$$
$$D_{\text{buE}} = D^{\text{Init}} \cup D^{\text{Scan}} \cup D^{\text{Compl}}.$$

Deduction steps D^{Init} are needed to start the deduction of further valid items, hence these have no antecedents. D^{Scan} and D^{Compl} conform to the *scan* and *complete* steps of Earley's algorithm.

The orginal, 'canonical' Earley algorithm is more restrictive in the items it recognizes. Unlike bottom-up Earley, which recognizes all items of the form $[B \to \bullet\gamma, j, j]$, canonical Earley recognizes these only if there is a 'need' to do so: only if $[A \to \alpha\bullet B\beta, i, j]$ has been recognized, then it should be attempted to recognize a B starting at position j in the string. This policy reduces the number of recognized items, but also reduces the possibilities for parallel processing. Earley's algorithm is essentially left-to-right.

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

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

The Earley parsing system for G and $a_1 \ldots a_n$ recognizes the set of items

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

A sentence $a_1 \ldots a_n$ is correct, like in the bottom-up case, if and only if an item of the form $[S \to \gamma\bullet, 0, n]$ can be recognized.

2.2. Parsing systems

Definition 2.4 (*parsing system*). A parsing system $\mathbb{P}$ for some grammar G and string $a_1 \ldots a_n$ is a triple $\mathbb{P} = \langle \mathscr{I}, H, D \rangle$, in which

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

We treat 'item' as an undefined basic concept here. A discussion about the nature of items follows in Section 2.4.

Note that H need not be a subset of $\mathscr{I}$. $\wp_{\text{fin}}$ 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 $\wp_{\text{fin}}(H \cup \mathscr{I}) \times \mathscr{I}$. A property that an inference relation should have – but which is not implied by the definition of D – is closure under addition of antecedents:

Definition 2.5 (*inference relation* $\vdash$). Let $\mathbb{P}=\langle \mathscr{I},H,D\rangle$ be a parsing system. The relation $\vdash \subseteq \wp_{\text{fin}}(H \cup \mathscr{I}) \times \mathscr{I}$ is defined by

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

Definition 2.6 (*deduction sequence*). Let $\mathbb{P}=\langle \mathscr{I},H,D\rangle$ be a parsing system. We write $\mathscr{I}^+$ for the set of nonempty, finite sequences $\xi_1,\ldots,\xi_j$, with $j \geqslant 1$ and $\xi_i \in \mathscr{I}$ $(1 \leqslant i \leqslant j)$.

A deduction sequence in $\mathbb{P}$ is a pair $(Y;\xi_1,\ldots,\xi_j) \in \wp_{\text{fin}}(H \cup \mathscr{I}) \times \mathscr{I}^+$ such that $Y \cup \{\xi_1,\ldots,\xi_{i-1}\} \vdash \xi_i$ for $1 \leqslant i \leqslant j$.

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

Definition 2.7 ($\vdash^*$). For a parsing system $\mathbb{P}=\langle \mathscr{I},H,D\rangle$ we define the relation $\vdash^*$ on $\wp_{\text{fin}}(H \cup \mathscr{I}) \times \mathscr{I}$ by

$$Y \vdash^* \xi \quad \text{if} \quad \xi \in Y \quad \text{or} \quad Y \vdash \cdots \vdash \xi.$$

Definition 2.8 (*valid items*). For a parsing system $\mathbb{P}=\langle \mathscr{I},H,D\rangle$ the set of valid items is defined by

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

2.3. Parsing schemata

A parsing system has been defined for a fixed grammar and string. In two steps this is extended to a parsing schema for arbitrary grammars and strings.

Definition 2.9 (*uninstantiated parsing system*). An uninstantiated parsing system for a grammar G is triple $\langle \mathscr{I},\mathscr{H},D\rangle$ with $\mathscr{H}$ a function that assigns a set of hypotheses to each string $a_1 \ldots a_n \in \Sigma^*$, such that $\langle \mathscr{I},\mathscr{H}(a_1 \ldots a_n),D\rangle$ is a parsing system.

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

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

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

Definition 2.10 (*parsing schema*). A parsing schema for a (sub)class of context-free grammars $CG \subseteq CFG$ is a function that assigns an uninstantiated parsing system to every grammar $G \in CG$.

Example 2.11. The parsing schema **CYK** is defined for any $G \in CNF$ and for any $a_1 \ldots a_n \in \Sigma^*$ by $\mathbf{CYK}(G)(a_1 \ldots a_n) = \mathbb{P}_{\mathrm{CYK}}$ as in Example 2.1.

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

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

One of the most powerful applications of the parsing schemata framework is the possibility to formalize the relation between different parsers by relating their underlying parsing schemata. This allows variants, extensions and optimizations to be exchanged across algorithms.

Qualitative improvements can be obtained in some cases by *generalization*, i.e., *refining* deduction steps into smaller steps (and adapting the items accordingly) and/or *extending* a schema to a larger class of grammars. The **buE** schema is a generalization of **CYK**.

Quantitative optimizations can be obtained by *filtering* a schema, i.e., discarding items and deduction steps or contracting sequences of deduction steps. The **Earley** schema is a filtered version of **buE**.

For formal definitions and examples, see [27, 29].

2.4. 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 $[A \to \alpha \bullet \beta, i, j]$ 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.

In [25], an item is defined as a congruence class of trees in the appropriate domain. We may simplify things by leaving an item to be a partial specification of a tree – assuming that a general item specification language exists and that all items used in practical algorithms are (efficient notations for) items expressible in this specification language.

Two regularity properties on item sets have to be stated explicitly.

Firstly, we have tacitly assumed that there is a clear separation between *final items*, (partially) specifying full parse trees, and *intermediate items*, (partially) specifying partial parse trees. It is possible – but admittedly rather artificial – to construct *mixed items* that denote a combination of both types. Consider, e.g., an item $[S \to \gamma \bullet, 0, *]$

denoting derivations $S \Rightarrow^* a_1 \ldots a_k$ for arbitrary k. This includes partial parse trees ($k < n$) as well as full parse trees ($k = n$). Such items do not clarify whether a parse tree exists, so they have to be ruled out.

Secondly, we demand that each parse tree of a sentence satisfies the specification provided by some item in $\mathscr{I}$.

These properties are captured by the notion 'semiregularity.' ('Regularity' was introduced in [25] for parsing systems and schemata that do not contain inconsistent specifications as items. We do not need the regularity property in this context.)

Definition 2.12 (*semiregularity*). A parsing system $\mathbb{P} = \langle \mathscr{I}, H, D \rangle$ for a grammar G and string $a_1 \ldots a_n$ is called semiregular if $\mathscr{I}$ does not contain mixed items and each parse tree of $a_1 \ldots a_n$ conforms to the specification of some item in $\mathscr{I}$.

A parsing schema $\mathbf{P}$ for a class of grammars CG is semiregular if $\mathbf{P}(G)(a_1 \ldots a_n)$ is semiregular for all $G \in CG$ and all $a_1 \ldots a_n \in \Sigma^*$.

Definition 2.13 (*correct final items*). We write $\mathscr{F}(\mathbb{P}) \subseteq \mathscr{I}$ 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 $\mathscr{C}(\mathbb{P}) \subseteq \mathscr{F}(\mathbb{P})$ for the set of correct final items of $\mathbb{P}$.

Example 2.14 (*final and correct final items*).

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

Definition 2.15 (*correctness*). A semiregular parsing system $\mathbb{P}$ is *sound* if $\mathscr{F}(\mathbb{P}) \cap \mathscr{V}(\mathbb{P}) \subseteq \mathscr{C}(\mathbb{P})$, i.e., all valid final items are correct.

A semiregular parsing system $\mathbb{P}$ is *complete* if $\mathscr{F}(\mathbb{P}) \cap \mathscr{V}(\mathbb{P}) \supseteq \mathscr{C}(\mathbb{P})$, i.e., all correct final items are valid.

A semiregular parsing system is *correct* if $\mathscr{F}(\mathbb{P}) \cap \mathscr{V}(\mathbb{P}) = \mathscr{C}(\mathbb{P})$, i.e., it is sound and complete.

A semiregular parsing schema $\mathbf{P}$ is sound/complete/correct for a class of grammars CG if $\mathbf{P}(G)(a_1 \ldots a_n)$ is sound/complete/correct for all $G \in CG$ and $a_1 \ldots a_n \in \Sigma^*$.

3. An operational notion of correctness

We have defined correctness, following [25] in simplified form. For proving the correctness of a given parsing schema, however, the notions introduced in Section 2.4 do not provide much help. Given a parsing system $\mathbb{P}$ with valid items $\mathscr{V}(\mathbb{P})$, it is generally trivial to establish semiregularity and to show that $\mathscr{F}(\mathbb{P}) \cap \mathscr{V}(\mathbb{P}) = \mathscr{C}(\mathbb{P})$.

The real issue is how to establish the set of valid items. We propose a general method to prove that $\mathscr{V}(\mathbb{P})$ is what we expect it to be.

3.1. A proof method

Usually, we have an 'educated guess' of the set of valid items of a parsing schema. This can be proven to be correct as follows.

- Define a set of *viable items* $\mathscr{W} \subseteq \mathscr{I}$ that should be recognized by a parsing system $\mathbb{P}$. It has to be proven that indeed $\mathscr{V}(\mathbb{P}) = \mathscr{W}$.
- Show $\mathscr{V}(\mathbb{P}) \subseteq \mathscr{W}$, i.e., soundness, by induction on $\vdash$.
- For $\mathscr{W} \subseteq \mathscr{V}(\mathbb{P})$, i.e., completeness, we need a different basis of induction. To that end we construct a *derivation length function* (*dlf*) d on $\mathscr{W}$, that allows to prove completeness by induction on $d(\xi)$.
- Having obtained $\mathscr{V}(\mathbb{P}) = \mathscr{W}$, generalization from parsing systems to parsing schemata is straightforward as usual.

Note that soundness (completeness, correctness) in this section relates to *all* items, with respect to the postulated set of valid items $\mathscr{W}$. Hence it is stronger than soundness (completeness, correctness) as stated in Definition 2.15, which only addresses *final* items.

The choice of an appropriate dlf is, again, a matter of educated guessing. A good guess may turn out to have the property that $H \vdash^m \xi$ if $d(\xi) = m$. But $\vdash$ cannot be used in the definition of d because, in effect, we have to prove that $H \vdash^* \xi$ for any $\xi \in \mathscr{W}$. Typically we define d in terms of the length of grammatical derivations (hence its name).

Definition 3.1 is stated such that completeness follows automatically from the existence of a dlf.

Definition 3.1 (*dlf*). Let $\mathbb{P}$ be a parsing system, $\mathscr{W} \subseteq \mathscr{I}$ a set of items. A function $d : H \cup \mathscr{W} \to \mathbb{N}$ is a dlf if

(i) $d(h) = 0$ for $h \in H$,

(ii) for each $\xi \in \mathscr{W}$ there is some $\eta_1, \ldots, \eta_k \vdash \xi \in D$ such that $\{\eta_1, \ldots, \eta_k\} \subseteq \mathscr{W}$ and $d(\eta_i) < d(\xi)$ for $1 \leqslant i \leqslant k$.

Proposition 3.2. *Let $\mathbb{P}$ be a parsing system, $\mathscr{W} \subseteq \mathscr{I}$ a set of items.*

(a) *If for all $\eta_1, \ldots, \eta_k \vdash \xi \in D$ with $\eta_i \in H \cup \mathscr{W}$, $1 \leqslant i \leqslant k$, it holds that $\xi \in \mathscr{W}$ then $\mathscr{V}(\mathbb{P}) \subseteq \mathscr{W}$.*

(b) *If a* dlf *$d : H \cup \mathscr{W} \to \mathbb{N}$ exists, then $\mathscr{W} \subseteq \mathscr{V}(\mathbb{P})$.*

Example 3.3 ($\mathscr{V}(\mathbb{P}_{\mathrm{CYK}})$). For a parsing system $\mathbb{P}_{\mathrm{CYK}}$ for arbitrary $G \in CNF$ and $a_1 \ldots a_n \in \Sigma^*$ we define $\mathscr{W} = \{[A, i, j] \mid A \Rightarrow^* a_{i+1} \ldots a_j\}$ as expected and a function d by $d([a, i-1, i]) = 0$ and

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

Then d is a dlf. The soundness of $\mathbb{P}_{\mathrm{CYK}}$ with respect to $\mathscr{W}$ is straightforward, hence $\mathscr{V}(\mathbb{P}_{\mathrm{CYK}}) = \mathscr{W}$.

Example 3.4 ($\mathscr{V}(\mathbb{P}_{\text{buE}})$). For a parsing system $\mathbb{P}_{\text{buE}}$ for arbitrary $G \in CFG$ and $a_1 \ldots a_n \in \Sigma^*$ we define $\mathscr{W} = \{[A \to \alpha \bullet \beta, i, j] \mid \alpha \Rightarrow^* a_{i+1} \ldots a_j\}$ as expected and a function d by $d([a, i-1, i]) = 0$ and

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

The motivation behind this definition is the following:

- $d([A \to \bullet \alpha\beta, i, i]) = 0$ by definition;
- $j - i$ counts the number of *scan* steps that is performed in order to obtain $[A \to \alpha \bullet \beta, i, j]$ from $[A \to \bullet \alpha\beta, i, i]$, $[a_{i+1}, i, i+1], \ldots, [a_j, j-1, j]$;
- Every step in the derivation $\alpha \Rightarrow^* a_{i+1} \ldots a_j$ requires a *complete* step, hence μ counts the number of *complete* steps required to obtain $[A \to \alpha \bullet \beta, i, j]$ (the minimum number in case of different derivations).

Thus we have defined d such that $H \vdash^m \xi$ if $d(\xi) = m$. It is left to the reader to verify that d is a dlf.

As in the previous example, soundness is trivial, hence $\mathscr{V}(\mathbb{P}_{\text{buE}}) = \mathscr{W}$.

3.2. Earley is correct

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

Consider the grammar

$$S \to NP\ VP, \quad NP \to {}^*d\ {}^*n, \quad VP \to {}^*v\ NP$$

that produces a single parse tree for a single string ${}^*d\,{}^*n\,{}^*v\,{}^*d\,{}^*n$. As an example, we have a look at the recognition of item $[NP \to {}^*d \bullet {}^*n, 3, 4]$, which is depicted in Fig. 1. A *predict* step, e.g. (1), corresponds to a downward edge traversal; a *complete* step, e.g. (4), corresponds to an upward edge traversal; a *scan* step, e.g. (8), corresponds to an edge traversal in both directions. Hence, we stipulate that fully recognized derivations count double for a dlf, predicted derivations count single, and scanning over terminals counts single as well.

Let us see how we can use this insight to obtain an 'educated guess' for a dlf d for Earley.

(i) Suppose we have $[A \to \bullet \alpha\beta, i, i]$ with $d([A \to \bullet \alpha\beta, i, i]) = m$.
Let $\alpha \Rightarrow^{\mu} a_{i+1} \ldots a_j$ a derivation of minimum length. In addition to bottom–up Earley, every *complete* step is preceded by a *predict* step, so we count 2μ rather than μ, yielding $d([A \to \alpha \bullet \beta, i, j]) = m + 2\mu + j - i$.
(For $[NP \to {}^*d \bullet {}^*n, 3, 4]$ we have $\mu = 0$ and $j - i = 1$.)

(ii) The more complicated part: How many steps are needed to predict an item $[A \to \bullet \alpha\beta, i, i]$?
Let $S \Rightarrow^{\varrho} a_1 \ldots a_i A\gamma$. We can split this in a derivation $S \Rightarrow^{\pi} \delta A\gamma$ that covers the *predict* steps and $\delta \Rightarrow^{\lambda} a_1 \ldots a_i$ that covers the remaining recognition of a valid prefix. For predicting of $[A \to \bullet \alpha\beta, i, i]$ we count $\pi + 2\lambda + i$: the predict part

$$
\begin{aligned}
[S\to\bullet NP\ VP,0,0] &\vdash [NP\to\bullet\,{}^*d\ {}^*n,0,0] && (1)\\
[NP\to\bullet\,{}^*d\ {}^*n,0,0],\ [{}^*d,0,1] &\vdash [NP\to {}^*d\bullet\,{}^*n,0,1] && (2)\\
[NP\to {}^*d\bullet\,{}^*n,0,1],\ [{}^*n,1,2] &\vdash [NP\to {}^*d\ {}^*n\bullet,0,2] && (3)\\
[S\to\bullet NP\ VP,0,0],\ [NP\to {}^*d\ {}^*n\bullet,0,2] &\vdash [S\to NP\bullet VP,0,2] && (4)\\
[S\to NP\bullet VP,0,2] &\vdash [VP\to\bullet\,{}^*v\ NP,2,2] && (5)\\
[VP\to\bullet\,{}^*v\ NP,2,2],\ [{}^*v,2,3] &\vdash [VP\to {}^*v\bullet NP,2,3] && (6)\\
[VP\to {}^*v\bullet NP,2,3] &\vdash [NP\to\bullet\,{}^*d\ {}^*n,3,3] && (7)\\
[NP\to\bullet\,{}^*d\ {}^*n,3,3],\ [{}^*d,3,4] &\vdash [NP\to {}^*d\bullet\,{}^*n,3,4] && (8)
\end{aligned}
$$

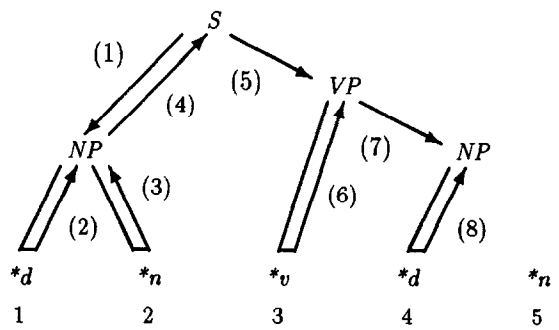

Fig. 1. Recognition of $[NP\to {}^*d\bullet{}^*n,3,4]$.

single, the recognition part double, and, in addition, the *scan* steps involved in the recognition part single as usual.

In the above example, for the item $[NP\to\bullet\,{}^*d\ {}^*n,3,3]$ we find $\pi=2$, by

$$S\Rightarrow NP\ VP\Rightarrow NP\ {}^*v\ NP,$$

represented by the single arrows labelled

(5): $[S\to NP\bullet VP,0,2]\vdash[VP\to\bullet\,{}^*v\ NP,2,2]$ and

(7): $[VP\to {}^*v\bullet NP,2,3]\vdash[NP\to\bullet\,{}^*d\ {}^*n,3,3]$.

Furthermore we find $\lambda=1$, by

$$NP\ {}^*v\Rightarrow {}^*d\ {}^*n\ {}^*v$$

represented by the double arrow labelled

(1): $[S\to\bullet NP\ VP,0,0]\vdash[NP\to\bullet\,{}^*d\ {}^*n,0,0]$ plus

(4): $[S\to\bullet NP\ VP,0,0],[NP\to {}^*d\ {}^*n\bullet,0,2]\vdash[S\to NP\bullet VP,0,2]$.

Summing up (i) and (ii) we obtain

$$\pi+2\lambda+i+2\mu+j-i=\pi+2\lambda+2\mu+j.$$

(For our example $[NP\to {}^*d\bullet{}^*n,3,4]$ this yields: $2+2*1+2*0+4=8$.)

Example 3.5 ($\mathscr{V}(\mathbb{P}_{\text{Earley}})$). For a parsing system $\mathbb{P}_{\text{Earley}}$ for arbitrary $G\in CFG$ and $a_1\ldots a_n\in\Sigma^*$ we define

$$\mathscr{W}=\{[A\to\alpha\bullet\beta,i,j]\mid\alpha\Rightarrow^* a_{i+1}\ldots a_j\wedge S\Rightarrow^* a_1\ldots a_iA\gamma\}.$$

The soundness of $\mathbb{P}_{\text{Earley}}$ with respect to $\mathscr{W}$ is trivial as usual.

We define a function $d : H \cup \mathscr{W} \to \mathbb{N}$ by $d([a, i-1, i]) = 0$ and

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

Note that d is indeed properly defined. To prove that d is a dlf it remains to be checked that condition (ii) in Definition 3.1 holds for all $\xi \in \mathscr{W}$. We distinguish the following cases:

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

  $$d(\eta) = \pi + 2\lambda + 2\mu + j.$$

  For ζ we have $S \Rightarrow^{\pi+1} \delta\alpha B\beta\gamma'$, $\delta\alpha \Rightarrow^{\lambda+\mu} a_1 \ldots a_j$, and $\gamma \Rightarrow^{\varrho} a_{j+1} \ldots a_k$, hence

  $$d(\zeta) \leqslant (\pi + 1) + 2(\lambda + \mu) + \varrho + k$$

  (note: $\leqslant$, rather than $=$, because it is conceivable that δ' and γ'' exist such that $S \Rightarrow^{\pi'} \delta' B\gamma''$ with $\pi' < \pi + 1$).
  For ξ we have $S \Rightarrow^{\pi} \delta A\gamma'$, $\delta \Rightarrow^{\lambda} a_1 \ldots a_i$, and $\alpha B \Rightarrow^{\mu+\varrho+1} a_{i+1} \ldots a_k$, hence

  $$d(\xi) = \pi + 2\lambda + 2(\mu + \varrho + 1) + k \geqslant d(\zeta) + 1,$$

  $$d(\xi) \geqslant d(\eta) + 2.$$

- $\xi = [B \to \bullet \gamma, j, j]$:
  Let $S \Rightarrow^{*} a_1 \ldots a_j B\gamma'$, and let $\delta, A, \gamma'', \alpha, \beta, \pi, \lambda, \mu$ such that $S \Rightarrow^{\pi} \delta A\gamma''$, $\delta \Rightarrow^{\lambda} a_1 \ldots a_i$, $A \to \alpha B\beta \in P$, $\alpha \Rightarrow^{\mu} a_{i+1} \ldots a_j$ with $\pi + 2\lambda + 2\mu$ minimal.
  Let $\eta = [A \to \alpha \bullet B\beta]$. Then $\eta \in \mathscr{W}$, $\eta \vdash \xi$, and

  $$d(\eta) = \pi + 2\lambda + 2\mu + j.$$

  For ξ we have $S \Rightarrow^{\pi+1} \delta\alpha B\beta\gamma$, $\delta\alpha \Rightarrow^{\lambda+\mu} a_1 \ldots a_j$, and the empty string before the dot. Thus

  $$d(\xi) = (\pi + 1) + 2(\lambda + \mu) + 0 + j = d(\eta) + 1$$

  which is minimal due to the minimality assumption above.

Hence we conclude that $\mathscr{V}(\mathbb{P}_{\text{Earley}}) = \mathscr{W}$ as defined above.

3.3. *Beyond CYK and Earley*

Establishing the correctness of **CYK** and **buE** was straightforward, but the correctness of **Earley** required some ingenuity. This raises the question how the proposed proof method 'scales up' to more complicated schemata.

There is a close relation between Earley-type algorithms and LR parsers, that will be further explored Section 4.3. An LR(0) parser is in fact an implementation of the parsing schema **Earley**. (Note, however, that LR-type parsers – including generalized LR-parsers like Tomita's algorithm – make some restrictions on the class of grammars that can be used.)

LR(k), SLR(k) and LALR(k) parsers are further filterings of the schema **Earley**. In [26] a parsing schema **SLR(1)** is defined and proven correct in detail. The dlf constructed for Earley applies to LR-type schemata as well.

Rather more involved examples are given in [25, 28], where correctness of (parsing schemata for) Left-Corner (LC) and Head-Corner (HC) parsers is established using the same technique. Head-Corner parsers do not process a sentence from left to right but start with the 'most interesting' part of each production. Other work on Head-Corner parsing can be found in [15, 22, 3, 2, 33]. We claim that the predictive Head-Corner parser proposed in [25, 28] is the only HC parser that has ever been formally proven correct.

Finding a derivation length function for LC and HC parsers is not more difficult (in fact easier) than for Earley. The schemata **LC** and **HC** have different kinds of items and more kinds of deduction steps, hence the proof that the suggested function 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 proofs which are hardly more complicated.

While parsing schemata are defined at the level of context-free grammars, they can also be applied to other formalisms, like the unification-based grammars used in natural language processing [4, 23, 24]. Parsing schemata have played an essential role in the specification of the unification grammar parser reported in [2]. A parsing schema for unification-based ID/LP grammars is described in [19].

4. From schemata to parsing algorithms

We discuss two important classes of parsing algorithms, viz. chart parsers and LR-type parsers, but do not dwell on details of specific algorithms.

4.1. Chart parsers

Parsing schemata are a generalization of *chart parsers* [12, 13, 31]. 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 to be used for searching new items that can be recognized, and a *chart*, eventually storing all recognized items. The generic chart parsing algorithm is shown in Fig. 2. 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,

```
program chart parser
begin
    create initial chart and agenda;
    while agenda is not empty
    do
        delete some (arbitrarily chosen) current item from agenda
        add current to chart;
        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.
```

Fig. 2. The chart parser algorithm.

obtained 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 can be enhanced with data structures that allow efficient searching and storing of relevant items.

4.2. *(G)LR parsers*

Push-down automata (PDAs) and context-free grammars generate the same set of languages [5, 10]. Many parsing algorithms in the field of compiler construction are based on the PDA paradigm. Most well-known is the family of *LR-parsers*, discovered by Knuth [16] and extended to the more practical *SLR* and *LALR* parsers by DeRemer [6, 7]. See [1] for a good introduction and [18] 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 PDAs in an efficient manner has been given by Lang [17]. Generalized LR parsing has attracted more attention in the form of Tomita's algorithm [30], based on a graph-structured stack as the data structure to handle the ambiguities that occur during parsing.

4.3. *From schemata to PDAs*

The question arises how parsing schemata and PDA-based algorithms like LR 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 2.4) is preserved.

Following LR conventions, we *augment* a grammar with a fresh start symbol S' and an end-of-sentence marker \$.

Definition 4.1 (*augmented grammar*). For each grammar $G \in CFG$ we define an augmented $G' = (N', \Sigma', P', S')$ by $N' = N \cup \{S'\}$, $\Sigma' = \Sigma \cup \{\$\}$, and $P' = P \cup \{S' \to S\$\}$ with $\{S', \$\} \cap V = \emptyset$.

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., [21, 20] for similar definitions of PDAs designed to model parsing algorithms. For the sake of brevity we only consider recognition and do 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 $\varphi, \psi, \ldots \in \mathscr{I}^*$ for stacks and parts of stacks. Configurations are denoted as pairs $(\varphi, w) \in (\mathscr{I}^* \times \Sigma'^*)$.

Definition 4.2 (*PDA*). A push-down automaton Π for a grammar $G \in CFG$ is a quadruple $\langle \mathscr{I}, \xi_0, \mathscr{F}, D \rangle$ in which

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

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

Definition 4.3 (*acceptance*). Let Π be a PDA for some grammar $G \in CFG$. A string $a_1 \ldots a_n \in \Sigma^*$ is *accepted* by Π if $(\xi_0, a_1 \ldots a_n\$) \vdash^* (\varphi \xi_F, \$)$ for some $\varphi \in \mathscr{I}^*$ and $\xi_F \in \mathscr{F}$.

Example 4.4 (Π_{Earley}). The PDA Π_{Earley} is defined for a grammar $G \in CFG$ by

$$\mathscr{I}_{\text{Earley}} = \{[A \to \alpha \bullet \beta, i, j] \mid A \to \alpha\beta \in P' \wedge 0 \leqslant i \leqslant j\},$$

$$\xi_0 = [S' \to \bullet S\$, 0, 0],$$

$$\mathscr{F} = \{[S' \to S \bullet \$, 0, n] \mid n \geqslant 0\},$$

$$D^{\text{Pred}} = \{(\varphi[A \to \alpha \bullet B\beta, i, j], w) \vdash (\varphi[A \to \alpha \bullet B\beta, i, j][B \to \bullet\gamma, j, j], w)\},$$

$$D^{\text{Sh}} = \{(\varphi[A \to \alpha \bullet a\beta, i, j], aw) \vdash (\varphi[A \to \alpha \bullet a\beta, i, j][A \to \alpha a \bullet \beta, i, j+1], w)\},$$

$$D^{\text{Re}} = \{(\varphi[A \to \alpha \bullet B\beta, i, j][B \to \bullet\gamma, j, j] \ldots [B \to \gamma \bullet, j, k], w) \vdash (\varphi[A \to \alpha \bullet B\beta, i, j][A \to \alpha B \bullet \beta, i, k], w)\},$$

$$D_{\text{Earley}} = D^{\text{Pred}} \cup D^{\text{Sh}} \cup D^{\text{Re}}.$$

We have replaced the Earley terms *scan* and *complete* by their LR equivalents *shift* and *reduce*. The PDA Π_{Earley} recognizes the same items for $a_1 \ldots a_n$ as the parsing

schema $\mathbb{P}_{\text{Earley}}$. Correctness of parsing schemata (cf. Definition 2.15) and push-down automata is related as follows.

Proposition 4.5 (equivalence of **Earley**(G) and Π_{Earley}). *Let Π_{Earley} be the PDA for some grammar $G \in CFG$ and $\mathbb{P}_{\text{Earley}}$ the parsing system for G and some string $a_1 \ldots a_n \in \Sigma^*$. Then Π_{Earley} recognizes $a_1 \ldots a_n$ if and only if $\mathscr{C}(\mathbb{P}_{\text{Earley}}) \neq \emptyset$.*

Proof. Straightforward. □

Next, we observe that the position markers in the items in Π_{Earley} can be discarded. In a parsing schema position markers are needed to relate partial derivations to parts of the sentence. In the context provided by the stack and the remainder of the sentence this information is redundant. This leads to a simplified PDA which, perhaps surprisingly, defines an LL(0) parser.

Example 4.6 ($\Pi_{\text{LL}(0)}$). The PDA $\Pi_{\text{LL}(0)}$ is defined for a grammar $G \in CFG$ by

$$\mathscr{I}_{\text{LL}(0)} = \{[A \to \alpha \bullet \beta] \mid A \to \alpha\beta \in P'\},$$

$$\xi_0 = [S' \to \bullet S\$],$$

$$\mathscr{F} = \{[S' \to S \bullet \$]\},$$

$$D^{\text{Pred}} = \{(\varphi[A \to \alpha \bullet B\beta], w) \vdash (\varphi[A \to \alpha \bullet B\beta][B \to \bullet\gamma], w)\},$$

$$D^{\text{Sh}} = \{(\varphi[A \to \alpha \bullet a\beta], aw) \vdash (\varphi[A \to \alpha \bullet a\beta][A \to \alpha a \bullet \beta], w)\},$$

$$D^{\text{Re}} = \{(\varphi[A \to \alpha \bullet B\beta][B \to \bullet\gamma] \ldots [B \to \gamma\bullet], w) \vdash (\varphi[A \to \alpha \bullet B\beta][A \to \alpha B \bullet \beta], w)\},$$

$$D_{\text{LL}(0)} = D^{\text{Pred}} \cup D^{\text{Sh}} \cup D^{\text{Re}}.$$

Proposition 4.7 (equivalence of Π_{Earley} and $\Pi_{\text{LL}(0)}$). *Let Π_{Earley} and $\Pi_{\text{LL}(0)}$ be PDAs for some grammar $G \in CFG$. Then $\Pi_{\text{LL}(0)}$ recognizes a string $a_1 \ldots a_n \in \Sigma^*$ if and only if Π_{Earley} recognizes $a_1 \ldots a_n$.*

Proof. Trivial. □

Two further transformations are needed to obtain an LR(0) PDA from the LL(0) PDA:

- 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.
- A full-fledged LR(0) PDA is obtained by combining items. If the dotted productions $[A \to \alpha \bullet X\beta]$, $[A' \to \alpha' \bullet X\beta'], \ldots$ are contained in a single item, then the item set contains another item *closure* $(\{[A \to \alpha X \bullet \beta], [A' \to \alpha' X \bullet \beta'], \ldots\})$. The algorithm for the construction of the set of LR(0) items can be found in any textbook on compiler construction, e.g. [1].

The details need not be spelled out here (see, e.g., [21, Ch. 8]).

There is more to say about the intricacies of constructing LR-type parsers but this is not the place to do so. Nontermination, for example, an issue to be considered in any implementation, does not worry us at the abstract level of PDAs.

4.4. Chart parsers vs. PDAs

We have sketched how the uninstantiated parsing system **Earley**(G) can be transformed into an LR(0) PDA. The transformation is bidirectional. The schema **SLR(1)** in [26] has been obtained 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 canonical implementations of parsing schemata, we have sketched a general relation between these two superficially different parsing paradigms.

5. Conclusions

Parsing schemata provide a general framework for description, analysis and comparison of parsing algorithms, both sequential and parallel. They abstract from data structures, control structures and (for parallel algorithms) communication structures. 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). Parsing schemata are defined at the level of context-free grammars but extend easily to other more involved formalisms like unification grammars.

Correctness proofs are easier at the more abstract level of schemata, because there is less to prove. The correctness of an algorithm can be derived by showing that it 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 push-down 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

[1] A.V. Aho, R. Sethi, J.D. Ullman, Compilers: Principles, Techniques and Tools, Addison-Wesley, Reading, MA, 1986.

[2] R. op den Akker, H. ter Doest, M. Moll, A. Nijholt, Parsing in dialogue systems using typed feature structures, Memoranda Informatica 95-25, Department of Computer Science, University of Twente, Enschede, Netherlands, 1995.

[3] G. Bouma, G. van Noord, Head-driven parsing for lexicalist grammars: experimental results, Proc. 6th Meeting of the European Chapter of the Association of Computational Linguistics, Utrecht, 1993, pp. 71–80.

[4] B. Carpenter, The Logic of Typed Feature Structures, Cambridge University Press, Cambridge, UK, 1992.

[5] N. Chomsky, Context-free grammars and pushdown storage, Quarterly Prog. Rept. 65, MIT Res. Lab. Elect., Cambridge, MA, 1962, pp. 187–194.
[6] F.L. DeRemer, Practical Translators for LR(k) Languages, Ph.D. Thesis, MIT, Cambridge, MA, 1969.
[7] F.L. DeRemer, Simple LR(k) grammars, Comm. ACM 14 (1971) 94–102.
[8] J. Earley, An efficient context-free parsing algorithm, Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1968.
[9] J. Earley, An efficient context-free parsing algorithm, Comm. ACM 13 (1970) 94–102.
[10] J. Evey, Application of pushdown store machines, Proc. 1963 Fall Joint Computer Conf., AFIPS Press, Montvale, NJ, 1963, pp. 215–227.
[11] T. Kasami, An efficient recognition and syntax analysis algorithm for context-free languages, Scientific Report AFCLR-65-758, Air Force Cambridge Research Laboratory, Bedford, MA, 1965.
[12] M. Kay, Algorithm schemata and data structures in syntactic processing, Report CSL-80-12, Xerox PARC, Palo Alto, CA, 1980.
[13] M. Kay, Algorithm schemata and data structures in syntactic processing, in: B.J. Grosz, K. Sparck Jones, B.L. Webber (Eds.), Readings in Natural Language Processing, Morgan Kaufmann, Los Altos, CA, 1982.
[14] M. Kay, Parsing in functional unification grammar, in: D.R. Dowty, L. Karttunen, A. Zwicky (Eds.), Natural Language Parsing, Cambridge University Press, Cambridge, UK, 1985, pp. 251–278.
[15] M. Kay, Head driven parsing, Proc. 1st Internat. Workshop on Parsing Technologies, Pittsburgh, PA, 1989, pp. 52–62.
[16] D.E. Knuth, On the translation of languages from left to right, Inform. Control 8 (1965) 607–639.
[17] B. Lang, Deterministic techniques for efficient non-deterministic parsers, Proc. 2nd Coll. on Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 14, Springer, Berlin, 1974, pp. 255–269.
[18] A. Nijholt, Deterministic top–down and bottom–up parsing: historical notes and bibliographies, Mathematisch Centrum, Amsterdam, Netherlands, 1983.
[19] F. Morawietz, A unification-based ID/LP Parsing Schema, Proc. 4th Internat. Workshop on Parsing Technologies, Prague, Czech Republic, 1995, pp. 162–173.
[20] M.J. Nederhof, Linguistic parsing and program transformations, Ph.D. Thesis, University of Nijmegen, Netherlands, 1994.
[21] P. Oude Luttighuis, Parallel algorithms for parsing and attribute evaluation, Ph.D. Thesis, University of Twente, Enschede, Netherlands, 1993.
[22] G. Satta, O. Stock, Head-driven bidirectional parsing: a tabular method, Proc. 1st Internat. Workshop on Parsing Technologies, Pittsburgh, PA, 1989, pp. 43–51.
[23] S.M. Shieber, An introduction to unification-based approaches to grammar, CSLI Lecture Notes, vol. 4, Center for the Study of Language and Information, Stanford University, Stanford, CA, 1986.
[24] S.M. Shieber, Constraint-Based Grammar Formalisms: Parsing and Type Inference for Natural and Computer Languages, The MIT Press, Cambridge, MA, 1992.
[25] K. Sikkel, Parsing schemata, Ph.D. Thesis, University of Twente, Enschede, Netherlands, 1993.
[26] K. Sikkel, Parsing schemata and correctness of parsing algorithms, Proc. TWLT/AMAST Workshop on Algebraic Methods in Language Processing, Enschede, Netherlands, 1995, pp. 83–97.
[27] K. Sikkel, Parsing Schemata – a framework for specification and analysis of parsing algorithms, Texts in Theoretical Computer Science – An EATCS Series, Springer, Berlin, 1997.
[28] K. Sikkel, R. op den Akker, Predictive head-corner chart parsing, in: H. Bunt, M. Tomita (Eds.), Recent Advances in Parsing Technology, Kluwer Academic Publishers, Boston, 1996, pp. 171–184.
[29] K. Sikkel, A. Nijholt, Parsing of context-free languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. 2: Linear Modeling: Background and Application, Springer, Berlin, 1997, pp. 61–100.
[30] M. Tomita, Efficient Parsing for Natural Language, Kluwer Academic Publishers, Boston, MA, 1985.
[31] T. Winograd, Language as a Cognitive Process, vol. I: Syntax Addison-Wesley, Reading, MA, 1983.
[32] D.H. Younger, Recognition of context-free languages in time n^3, Inform. Control 10 (1967) 189–208.
[33] G. van Noord, An efficient implementation of the Head-Corner Parser, Computational Linguistics 23 (1997) 425–456.