How to compare the structure of parsing algorithms*

Klaas Sikkel

Gesellschaft für Mathematik und Datenverarbeitung,
GMD-FIT, Schloß Birlinghoven,
D-53757 Sankt Augustin, Germany
sikkel@gmd.de

Abstract

Parsing schemata are defined as an intermediate level of abstraction between context-free grammars and parsers. Clear, concise specifications of radically different parsing algorithms can be expressed as parsing schemata. Moreover, because of the uniformity of these specifications, relations between different parsing algorithms can be formally established. This article gives an introduction to the parsing schemata framework.

1 Introduction

A wide variety of parsing algorithms can be found in the Computer Science and Computational Linguistics literature. Algorithms differ a lot with respect to languages in which they are expressed, data structures used, degree of formality, class of grammars that can be handled, etc. Things get worse when we consider parallel, rather than sequential algorithms, because many different architectures can be exploited.

One can compare parsers by evaluating run-time performance, but in order to get a deeper insight into the relative merits and deficiencies of different algorithms, one needs a common framework in which the structure of the algorithms can be described. In this paper we introduce a framework that is both sufficiently general to cover a large range of wildly different parsers, and sufficiently practical to allow legible specifications of nontrivial algorithms. A *parsing schema* describes the basic traits of an algorithm, but abstracts from the technical details needed to make the algoritm work. A schema specifies

- the type of intermediate results used by a parser, i.e., some representation of partial parses,
- steps that allow computation of new intermediate results from given ones.

A parsing schema does *not* specify

- *data structures* for storage and retrieval of such results,
- *control structures* enforcing that all the relevant steps are performed,

*This research was carried out within the *Parlevink* project at the Department of Computer Science, University of Twente, Enschede, the Netherlands; (cf. http://hydra.cs.utwente.nl/parlevink/parlevink.html).

- (only for parallel algorithms:) *communication structures* between different processors.

One can see parsing schemata as a separate, well-defined level of abstraction in between grammars and parsing algorithms.

For the sake of brevity, we consider only context-free parsing; in section 6 but a few words are spent on parsing of unification grammars. A more complete treatment can be found in [Sik93].

Definitions and some examples of parsing schemata given in Section 2. In Sections 3 and 4 we introduce various types of relations between parsing schemata, viz. generalizations and filters. These can be used to formally establish similarities and differences between parsing algorithms. A few more nontrivial examples to illustrate the power of the formalism are presented in Section 5. Related work and various applications of the framework are briefly discussed in 6, conclusions briefly summarized in 7.

2 Parsing systems and parsing schemata

A parsing schema is defined for a class of grammars, and for all strings over the alphabet of any particular grammar. But in order to simplify things, we will first restrict ourselves to a single given grammar and a single given sentence. For this particular grammar and sentence we define a *parsing system*, which is in fact logical deduction system. Parsing systems are defined in 2.1, while some formal details are deferred to 2.2. Parsing schemata follow as a straightforward generalization in 2.3.

The usual notational conventions apply. A grammar is a quadruple $G = (N, \Sigma, P, S)$. We write $A, B, \ldots \in N$ for nonterminals; $a, b, \ldots \in \Sigma$ for terminals; $X, Y, \ldots \in (N \cup \Sigma)$ for arbitrary variables; $\alpha, \beta, \ldots \in (N \cup \Sigma)^*$ for strings of arbitrary variables; ε for the empty string. The letters $i, j, \ldots$ denote nonnegative integers.

2.1 Parsing systems

Definition 1 (*parsing system*)
A parsing system $\mathbb{P}$ for some grammar G and string $a_1 \ldots a_n$ is a triple $\mathbb{P} = \langle \mathcal{I}, H, D \rangle$, in which

- $\mathcal{I}$ is a set of items, called the called the *domain* or *item set* of $\mathbb{P}$;
- H is a finite set of items (not necessarily a subset of $\mathcal{I}$), the *hypotheses* of $\mathbb{P}$;
- D is a set of so-called *deduction steps* of the form

  $$\eta_1, \ldots, \eta_k \vdash \xi,$$

  with $\eta_i \in \mathcal{I} \cup H$ for $0 \leq i \leq k$ and $\xi \in \mathcal{I}$. The items $\eta_1, \ldots, \eta_k$ are called the *antecedents*, ξ the *consequent* of the deduction step. □

Different types of items can be used for parsing systems and schemata underlying different algorithms. Most examples here use the well-known Earley items. An item $[A \rightarrow \alpha \bullet \beta, i, j]$ denotes that we are looking for a nonterminal A that rewrites to $\alpha\beta$ and, furthermore, it is known that $\alpha \Rightarrow^* a_{i+1} \ldots a_j$.

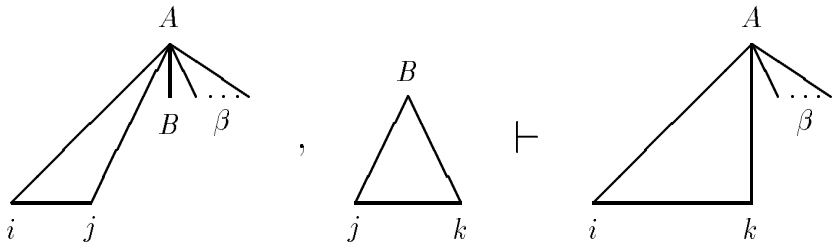

Figure 1: The *complete* step

Example 2 (a bottom-up version of Earley's algorithm)
We give a parsing system for Earley's algorithm [Ear70] without top-down prediction. The parsing system $\mathbb{P}_{buE}$ for some context-free grammar G and string $a_1 \ldots a_n$ is defined by:

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

□

A few more remarks about this example need be made.

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; the definition of a parsing system imposes that only items from $\mathcal{I}$ and H can be used in the definition of D.

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.

The definition of the domain does *not* restrict the position markers in items to the case $i, j \leq n$. This has been done on purpose. It has several advantages that the definitions of $\mathcal{I}$ and D do not depend on the string $a_1 \ldots a_n$. The price for this modularity is infinite, rather than finite sets $\mathcal{I}$ and D. But at this level of abstraction that does not pose any problem — we do not *do* anything, we only define possible algorithmic steps — and in any implementation it will be evident that only those items have to be considered, that have position markers within the appropriate range.

Implicitly specified by a parsing system $\mathbb{P}$ is the set of *valid items* $\mathcal{V}(\mathbb{P})$, that contains all items of $\mathcal{I}$ (and only those) that can be inferred from H, using deduction steps from D. It should be intuitively clear what is meant, but a formal definition of $\mathcal{V}(\mathbb{P})$ will be given in Section 2.2. It is easy to verify that

$$\mathcal{V}(\mathbb{P}_{buE}) = \{[A{\rightarrow}\alpha{\bullet}\beta, i, j] \mid \alpha \Rightarrow^* a_{i+1} \ldots a_j\}.$$

Example 3 (the canonical Earley algorithm)
The parsing system $\mathbb{P}_{Earley}$ for a given context-free grammar G and string $a_1 \ldots a_n$ is defined by $\mathcal{I}$ and H as in $\mathbb{P}_{buE}$ (cf. Example 2) and by D_{Earley} as follows:

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

□

Items of the form $[A \rightarrow \bullet\gamma, i, i]$ can be deduced only if there is a "need" to do so, i.e., an item $[B \rightarrow \alpha \bullet A\beta, h, i]$ has been found to be valid. The set of valid items is given by

$$\mathcal{V}(\mathbb{P}_{Earley}) = \{[A \rightarrow \alpha \bullet \beta, i, j] \in \mathcal{V}(\mathbb{P}_{buE}) \mid S \Rightarrow^* a_1 \ldots a_i A\gamma \text{ for some } \gamma\}.$$

Top-down filtering reduces the number of items, but also reduces the possibilities for parallel processing. Earley's algorithm is essentially left-to-right.

2.2 Some formal details

In 2.1 we have introduced parsing systems with more attention for examples and understanding than for formal details. In this section we make a few remarks on the notation and introduce some new notations. This is not of immediate importance for the understanding of parsing schemata, and the reader is advised to skip this section on first reading and move to 2.3.

A parsing system is just a particular kind of deduction system. Conventionally, one would assume that $H \subset \mathcal{I}$, i.e., the hypothesis are contained in the domain. We have specifically left open the option that H and $\mathcal{I}$ are disjunct; this has pragmatic advantages for the definition of parsing schemata.

We have to make a distinction between the set of deduction steps D and an inference relation $\vdash$. The relation D is a subset of $\wp_{fin}(H \cup \mathcal{I}) \times \mathcal{I}$, where $\wp_{fin}$ denotes the powerset restricted to finite sets. Elements of D, in a formal notation, are in fact pairs of the form $(\{\eta_1, \ldots, \eta_k\}, \xi)$. It is only for convenient reading that we denote such a pair by means of an induction $\eta_1, \ldots, \eta_k \vdash \xi$. If, furthermore, we have $Y = \{\eta_1, \ldots, \eta_k\}$, we may also write $Y \vdash \xi$ as an abbreviation for $\eta_1, \ldots, \eta_k \vdash \xi$.

For an induction relation $\vdash$ we want the conventional property that if $\eta_1, \ldots, \eta_k \vdash x$ holds, then also $\eta_1, \ldots, \eta_k, \zeta \vdash x$ for any ζ. So wo define an inference relation $\vdash$ that is the closure of D under addition of antecedents to an inference:

Definition 4 (*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

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

□

Before we define the transitive closure of $\vdash$ we introduce the notion of a deduction sequence:

Definition 5 (*deduction sequences*)
We write $\mathcal{I}^+$ for the set of non-empty, finite sequences $\xi_1, \ldots, \xi_j$, with $j \geq 1$ and $\xi_i \in \mathcal{I}$ $(1 \leq i \leq j)$. Let $\mathbb{P} = \langle \mathcal{I}, H, D \rangle$, be a parsing system.
A *deduction sequence* in $\mathbb{P}$ is a pair $(Y; \xi_1, \ldots, \xi_j) \in \wp_{fin}(H \cup \mathcal{I}) \times \mathcal{I}^+$, such that

$$Y \cup \{\xi_1, \ldots, \xi_{i-1}\} \vdash \xi_i \quad \text{for } 1 \leq i \leq j.$$

As a practical informal notation we write

$$Y \vdash \xi_1 \vdash \ldots \vdash \xi_j$$

for a deduction sequence $(Y; \xi_1, \ldots, \xi_j)$.
The *set of deduction sequences* $\Delta(\mathbb{P}) \subseteq \wp_{fin}(H \cup \mathcal{I}) \times \mathcal{I}^+$ for $\mathbb{P}$ is defined by

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

Definition 6 (*transitive and reflexive inference relation* $\vdash^*$)
For a parsing system $\mathbb{P} = \langle \mathcal{I}, H, D \rangle$ we define the relations $\vdash^0$, $\vdash^+$ and $\vdash^*$ on $\wp_{fin}(H \cup \mathcal{I}) \times \mathcal{I}$ as follows.

$$Y \vdash^0 \xi \text{ if } \xi \in Y,$$

$$Y \vdash^+ \xi \quad \text{if } Y \vdash \ldots \vdash \xi,$$

$$Y \vdash^* \xi \text{ if } Y \vdash^0 \xi \text{ or } Y \vdash^+ \xi. \qquad \square$$

Definition 7 (*validity*)
For a parsing system $\mathbb{P} = \langle \mathcal{I}, H, D \rangle$ the set of *valid items* is given by

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

We do not make a distinction between semantic validity (usually denoted $\models \xi$) and syntactic provability (i.e. $H \vdash^* \xi$). A more adequate notion of validity for parsing systems can be introduced along the following lines. One can distinguish so-called "final items" that denote the existence of a parse (e.g. an item $[S \rightarrow \gamma \bullet, 0, n]$). A parsing system is valid if the "right" final items are recognized, no more and no less. This is worked out in detail in [Sik93]. It involves the introduction of some additional conceptual machinery and complication of some of the definitions. But this is somewhat beside the point of this article and we can save quite a bit of space and effort by ignoring the issue of formal parser correctness.

2.3 Parsing schemata

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

Definition 8 (*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 $a_1 \ldots a_n \in \Sigma^*$, such that $\langle \mathcal{I}, \mathcal{H}(a_1 \ldots a_n), D \rangle$ is a parsing system. $\square$

In all examples it holds that

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

in the sequel we will no longer explicitly mention the hypotheses.

Definition 9 (*parsing schema*)
A parsing schema for some (sub)class of context-free grammars $\mathcal{CG}$ is a function that assigns an uninstantiated parsing system to every grammar $G \in \mathcal{CG}$. □

Example 10 (**buE**, **Earley**)
The parsing schema **buE** is defined by $\mathbf{buE}(G)(a_1 \ldots a_n) = \mathbb{P}_{buE}$ as in Example 2;
the parsing schema **Earley** is defined by $\mathbf{Earley}(G)(a_1 \ldots a_n) = \mathbb{P}_{Earley}$ as in Example 3. □

Both parsing schemata in Example 10 are defined for the entire class of context-free grammars. In the following example, the schema **CYK** (after the algorithm of Cocke, Younger and Kasami [Kas65, You67]) is defined only for grammars in Chomsky Normal Form (CNF). That is, all productions are of the form $A \rightarrow BC$ or $A \rightarrow a$.

Example 11 (**CYK**)
We assign a parsing system $\mathbb{P}_{CYK} = \mathbf{CYK}(G)(a_1 \ldots a_n)$ for arbitrary G in CNF, with $\mathcal{I}$ and D defined by:

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

□

3 Generalization

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

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

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

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

- $\mathbf{P}_2$ is called an *extension* of $\mathbf{P}_1$ if it is defined for a larger class of grammars.

We write $\mathbf{P}_1 \overset{\mathbf{ir}}{\Longrightarrow} \mathbf{P}_2$, $\mathbf{P}_1 \overset{\mathbf{sr}}{\Longrightarrow} \mathbf{P}_2$, and $\mathbf{P}_1 \overset{\mathbf{ext}}{\Longrightarrow} \mathbf{P}_2$ to denote that $\mathbf{P}_2$ is respectively an item refinement, step refinement or extension of $\mathbf{P}_1$. A generalization, denoted $\overset{\mathbf{gen}}{\Longrightarrow}$, is an arbitrary combination of different types of refinements.

Example 12 (CYK $\overset{\mathbf{gen}}{\Longrightarrow}$ buE)
In order to generalize **CYK** into **buE** we introduce two intermediate parsing systems **CYK'** and **ECYK**, such that

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

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

$$\begin{aligned}
\mathcal{I}_{CYK'} &= \{[A{\rightarrow}\alpha_\bullet,i,j] \mid A{\rightarrow}\alpha \in P \wedge 0 \leq i \leq j\};\\
D^{(1)} &= \{[a,j-1,j] \vdash [A{\rightarrow}a_\bullet,j-1,j]\},\\
D^{(2)} &= \{[B{\rightarrow}\beta_\bullet,i,j],[C{\rightarrow}\gamma_\bullet,j,k] \vdash [A{\rightarrow}BC_\bullet,i,k]\},\\
D_{CYK'} &= D^{(1)} \cup D^{(2)}.
\end{aligned}$$

Note that a single item $[A,i,j] \in \mathcal{I}_{CYK}$ corresponds to multiple items $[A{\rightarrow}\alpha_\bullet,i,j]$, $[A{\rightarrow}\beta_\bullet,i,j]$, etc., if there are different productions $A{\rightarrow}\alpha$, $A{\rightarrow}\beta$, etc.

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

$$[B{\rightarrow}\beta_\bullet,i,j],[C{\rightarrow}\gamma_\bullet,j,k] \;\vdash\; [A{\rightarrow}BC_\bullet,i,k]$$

into a sequence of deduction steps

$$\begin{aligned}
&\vdash\; [A{\rightarrow}{\bullet}BC,i,i],\\
[A{\rightarrow}{\bullet}BC,i,i],[B{\rightarrow}\beta_\bullet,i,j] &\vdash\; [A{\rightarrow}B{\bullet}C,i,j],\\
[A{\rightarrow}B{\bullet}C,i,j],[C{\rightarrow}\gamma_\bullet,j,k] &\vdash\; [A{\rightarrow}BC_\bullet,i,k].
\end{aligned}$$

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

$$\begin{aligned}
\mathcal{I}_{ECYK} &= \{[A{\rightarrow}\alpha_\bullet\beta,i,j] \mid A{\rightarrow}\alpha\beta \in P \;\wedge\; 0 \leq i \leq j\};\\
D^{Init} &= \{\vdash [A{\rightarrow}{\bullet}\alpha,j,j]\},\\
D^{Scan} &= \{[A{\rightarrow}\alpha_\bullet a\beta,i,j],[a,j,j+1] \vdash [A{\rightarrow}\alpha a_\bullet\beta,i,j+1]\},\\
D^{Compl} &= \{[A{\rightarrow}\alpha_\bullet B\beta,i,j],[B{\rightarrow}\gamma_\bullet,j,k] \vdash [A{\rightarrow}\alpha B_\bullet\beta,i,k]\},\\
D_{ECYK} &= D^{Init} \cup D^{Scan} \cup D^{Compl}.
\end{aligned}$$

ECYK is identical to **buE**, cf. Example 2, except for the fact that **ECYK** is defined only for grammars in CNF. Hence, obviously, **buE** is an extension of **ECYK**. □

We will now formalize the concepts that have been informally introduced and illustrated above. In the sequel we write $\mathbb{P}_i$ for a parsing system $\mathbb{P}_i = \langle \mathcal{I}_i, H, D_i \rangle$; we write $\vdash_i$ and $\vdash_i^*$ for inference relation and its closure, based on D_i, cf. Section 2.2.

For the definition of item refinement we make use of an *item mapping* $f : \mathcal{I}_2 \rightarrow \mathcal{I}_1$ that maps items of $\mathbb{P}_2$ to items of $\mathbb{P}_1$. Such a function f is extended (in the conventional way) to cover *sets of* items: Let $Y \subseteq \mathcal{I}_2$ then $f(Y)$ is the subset of $\mathcal{I}_1$ that contains those items that are the image of some item in Y. Moreover, we extend[1] f to a function $f : \mathcal{I}_2 \cup H \rightarrow \mathcal{I}_1 \cup H$ by letting $f(h) = h$ for $h \in H$. Then we can apply f to deduction steps by letting

$$f(\eta_1, \ldots, \eta_k \vdash \xi) \stackrel{\text{def}}{=} f(\eta_1), \ldots, f(\eta_k) \vdash f(\xi).$$

In the same fashion we can extend f to deduction sequences[2] and to sets of deduction steps and sets of deduction sequences.

Definition 13 (*item refinement*)
The relation $\mathbb{P}_1 \xrightarrow{\mathbf{ir}} \mathbb{P}_2$ holds between parsing systems $\mathbb{P}_1$ and $\mathbb{P}_2$ if there is an item mapping $f : \mathcal{I}_2 \rightarrow \mathcal{I}_1$ such that

(i) $\quad \mathcal{I}_1 = f(\mathcal{I}_2)$,

(ii) $\quad \Delta(\mathbb{P}_1) = f(\Delta(\mathbb{P}_2))$,

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

The first condition states the surjectivity of f, no items are 'lost' in the refinement; the second condition ensures that deduction sequences are carried over into the refined system[3].

The item refinement **CYK** $\stackrel{\mathbf{ir}}{\Longrightarrow}$ **CYK'**, is based on the item mapping $f([A \rightarrow \alpha\bullet, i, j]) = [A, i, j]$.

Definition 14 (*step refinement*)
The relation $\mathbb{P}_1 \xrightarrow{\mathbf{sr}} \mathbb{P}_2$ holds if

(i) $\quad \mathcal{I}_1 \subseteq \mathcal{I}_2$,

(ii) $\quad \vdash_1^* \subseteq \vdash_2^*$.

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

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

[2] Note, however, that the image of a deduction sequence is not necessarily a deduction sequence according to Definition 5.

[3] The notion of a deduction sequence has been introduced in Section 2.2 only to be able to state this property. A weaker condition like $(ii)'$: $\vdash_1^* = \vdash_2^*$ will not do, because the transitivity of generalization would be lost. An example of a relation that satisfies (i) and $(ii)'$ but is not a refinement is

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

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

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

A sufficient condition[4] for (ii) is $D_1 \subseteq \vdash_2^*$; that is, a single deduction step in $\mathbb{P}_1$ is emulated by some composite inference in $\mathbb{P}_2$. Furthermore, the domain of $\mathbb{P}_2$ may contain items that did not exist in $\mathbb{P}_1$. It is left to the reader to verify that **CYK'** $\overset{\mathbf{sr}}{\Longrightarrow}$ **ECYK**.

Definition 15 (*extension*)
Let $\mathbf{P}_1$ be a parsing schema for a class of grammars $\mathcal{CG}_1$, $\mathbf{P}_2$ a parsing schema for a class of grammars $\mathcal{CG}_2$ and $\mathcal{CG}_1 \subseteq \mathcal{CG}_2$. Then $\mathbf{P}_1 \overset{\mathbf{ext}}{\Longrightarrow} \mathbf{P}_2$ holds if $\mathbf{P}_1(G)(a_1 \ldots a_n) = \mathbf{P}_2(G)(a_1 \ldots a_n)$ for each grammar G in $\mathcal{CG}_1$ and each $a_1 \ldots a_n$. □

It has been argued above that **ECYK** $\overset{\mathbf{ext}}{\Longrightarrow}$ **buE**.

Definition 16 (*generalization*)
The relation $\mathbf{P}_1 \overset{\mathbf{gen}}{\Longrightarrow} \mathbf{P}_2$ holds between parsing schemata $\mathbf{P}_1$ and $\mathbf{P}_2$ if there are $\mathbf{P}'$ and $\mathbf{P}''$ such that $\mathbf{P}_1 \overset{\mathbf{ir}}{\Longrightarrow} \mathbf{P}' \overset{\mathbf{sr}}{\Longrightarrow} \mathbf{P}'' \overset{\mathbf{ext}}{\Longrightarrow} \mathbf{P}_2$. □

Theorem 17 (*properties of generalization*)
The relation $\overset{\mathbf{gen}}{\Longrightarrow}$ is transitive and reflexive. □

The proof is straightforward (but lengthy when done properly); see [Sik93] for the details.

4 Filtering

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

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

The theoretical framework is simple and elegant. As in the previous section we assume that a parsing system $\mathbb{P}_i$ is defined as $\langle \mathcal{I}_i, H, D_i \rangle$, with inference relations $\vdash_i$ and $\vdash_i^*$ on $\mathbb{P}_i$ according to Section 2.2.

Example 18 (a simple static filter)
As an example of static filtering we narrow the domain of the **buE** schema by discarding some items that are irrelevant for finding a parse. The definition of **buE** allows items of the form $[A \rightarrow \bullet\beta, i, j]$ with $i < j$, which, obviously cannot be valid. But some valid items could be

[4] We write $\vdash_1^*$ just for symmetry, as all other relations defined here and in Section 4 display the same kind of symmetry.

discarded as well. Consider a *non-reduced* grammar with a nonterminal $B \in N$ that cannot be derived from the start symbol. There is no need, then, to recognize items with B as left-hand side even though these may be contained in $\mathcal{V}(\mathbb{P}_{buE})$. **buE'** is defined by a parsing system $\mathbb{P}_{buE'}$, as usual.

$$\mathcal{I}_{buE'} = \{[A{\rightarrow}\alpha{\bullet}\beta, i, j] \mid A{\rightarrow}\alpha\beta \in P \wedge \exists b_1 \ldots b_j \in \Sigma^*, \gamma \in (\Sigma \cup N)^* : (S \Rightarrow^* b_1 \ldots b_i A\gamma \wedge \alpha \Rightarrow^* b_{i+1} \ldots b_j)\}$$

and $D_{buE'}$ as D_{buE} in Example 2. Note that, in general, $D_{buE'}$ is a proper subset of D_{buE} because it applies only to items within the domain. □

A more exiting example of static filtering will be presented in Section 5.2.

Definition 19 (*static filtering*)
The relation $\mathbb{P}_1 \xrightarrow{\mathbf{sf}} \mathbb{P}_2$ holds if

(i) $\mathcal{I}_1 \supseteq \mathcal{I}_2$

(ii) $D_1 \supseteq D_2$.

$\mathbf{P}_1 \overset{\mathbf{sf}}{\Longrightarrow} \mathbf{P}_2$ holds if $\mathbf{P}_1(G)(a_1 \ldots a_n) \xrightarrow{\mathbf{sf}} \mathbf{P}_2(G)(a_1 \ldots a_n)$ for all G and $a_1 \ldots a_n$. □

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

Definition 20 (*dynamic filtering*)
The relation $\mathbb{P}_1 \xrightarrow{\mathbf{df}} \mathbb{P}_2$ holds if

(i) $\mathcal{I}_1 \supseteq \mathcal{I}_2$

(ii) $\vdash_1 \supseteq \vdash_2$.

$\mathbf{P}_1 \overset{\mathbf{df}}{\Longrightarrow} \mathbf{P}_2$ holds if $\mathbf{P}_1(G)(a_1 \ldots a_n) \xrightarrow{\mathbf{df}} \mathbf{P}_2(G)(a_1 \ldots a_n)$ for all G and $a_1 \ldots a_n$. □

Example 21 (**buE** $\overset{\mathbf{sf}}{\Longrightarrow}$ **Earley**)
We consider parsing systems $\mathbb{P}_{buE}$ and $\mathbb{P}_{Earley}$ for some given G and $a_1 \ldots a_n$. Then clearly $\mathcal{I}_{buE} = \mathcal{I}_{Earley}$. Every *init*, *scan*, and *complete* step in $\mathbb{P}_{Earley}$ also exists in $\mathbb{P}_{buE}$ (and the same applies to deductions in $\vdash_{Earley}$ that can be obtained by adding arbitrary antecedents to these steps). Only the Earley *predict* steps have to be accounted for. Let

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

be such a *predict* step. Then $\mathbb{P}_{buE}$ contains an *init* step $\vdash_{buE} [B{\rightarrow}{\bullet}\gamma, j, j]$, hence, by definition of $\vdash$, it holds that

$$[A{\rightarrow}\alpha{\bullet}\beta, i, j] \vdash_{buE} [B{\rightarrow}{\bullet}\gamma, j, j].$$ □

A few important remarks must be made about the above types of filtering.

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

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

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

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

Definition 22 (*step contraction*)
The relation $\mathbb{P}_1 \xrightarrow{\mathbf{sc}} \mathbb{P}_2$ holds if

(i) $\quad \mathcal{I}_1 \supseteq \mathcal{I}_2$

(ii) $\quad \vdash_1^* \supseteq \vdash_2^*$.

$\mathbf{P}_1 \overset{\mathbf{sc}}{\Longrightarrow} \mathbf{P}_2$ holds if $\mathbf{P}_1(G)(a_1 \ldots a_n) \xrightarrow{\mathbf{sc}} \mathbf{P}_2(G)(a_1 \ldots a_n)$. for each G and $a_1 \ldots a_n$. □

Note that every dynamic filter is also a step contraction. It is evident that each kind of filtering relation is transitive and reflexive.

Example 23 (**GHR**)
As a realistic example we give the parsing schema that underlies the improved Earley algorithm of Graham, Harrison and Ruzzo [GHR80]. This is a combination of two different step contractions:

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

For an arbitrary grammar G and string $a_1 \ldots a_n$ we define a parsing system $\mathbb{P}_{GHR}$ as follows.

$$
\begin{aligned}
\mathcal{I}_{GHR} &= \{[A \to \alpha \bullet \beta, i, j] \mid A \to \alpha\beta \in P \wedge 0 \leq i \leq j\};\\
D^{Init} &= \{\vdash [S \to \beta \bullet \gamma, 0, 0] \mid \beta \Rightarrow^* \varepsilon\},\\
D^{Scan} &= \{[A \to \alpha \bullet a\beta\gamma, i, j], [a, j, j+1] \vdash [A \to \alpha a \beta \bullet \gamma, i, j+1] \mid \beta \Rightarrow^* \varepsilon\},\\
D^{C1} &= \{[A \to \alpha \bullet B\beta\gamma, i, j], [B \to \delta \bullet, j, k] \vdash [A \to \alpha B \beta \bullet \gamma, i, k] \mid i < j < k \wedge \beta \Rightarrow^* \varepsilon\},\\
D^{C2} &= \{[A \to \alpha \bullet B\beta\gamma, i, i], [C \to \delta \bullet, i, j] \vdash [A \to \alpha B \beta \bullet \gamma, i, j]\\
&\qquad\qquad \mid i < j \wedge B \Rightarrow^* C \wedge \beta \Rightarrow^* \varepsilon\},\\
D^{Pred} &= \{[A \to \alpha \bullet B\beta, i, j] \vdash [C \to \alpha' \bullet \beta', j, j] \mid B \Rightarrow^* C\gamma \wedge \alpha' \Rightarrow^* \varepsilon\},\\
D_{GHR} &= D^{Init} \cup D^{Scan} \cup D^{C1} \cup D^{C2} \cup D^{Pred}.
\end{aligned}
$$

It is left to the reader to verify that **Earley** $\overset{\mathbf{sc}}{\Longrightarrow}$ **GHR**. □

5 Some more examples

In the previous sections we have deliberately used well-known parsing algorithms, so that the introduced framework could be illustrated with familiar examples. Here we give some more exotic examples of parsing schemata so as to illustrate the power of the formalism.

5.1 Left-Corner parsing

We will precisely establish the relation between Earley parsing and Left-Corner parsing. We define a parsing schema **LC** that underlies the (generalized) Left-Corner algorithm that is known from the literature, cf. [MT+83, Ned93] (as opposed to *deterministic* LC parsing [RL70]) and argue that **Earley** $\stackrel{\mathbf{sc}}{\Longrightarrow}$ **LC**. As a conceptual aid, we will first consider a "bottom-up Left-Corner" parser that is a rather trivial step contraction of bottom-up Earley.

Example 24 (buLC)
Consider an item of the form $[A \rightarrow B \bullet \beta, i, j]$ in **buE**. The item is valid if some $[B \rightarrow \gamma \bullet, i, j]$ is valid, because $[A \rightarrow \bullet B\beta, i, i]$, is valid by definition. So we can contract the sequence of deduction steps

$$\begin{array}{rcl} & \vdash & [A \rightarrow \bullet B\beta, i, i], \\ [A \rightarrow \bullet B\beta, i, i], [B \rightarrow \gamma \bullet, i, j] & \vdash & [A \rightarrow B \bullet \beta, i, j]. \end{array}$$

to a single deduction step

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

A similar argument applies to items of the form $[A \rightarrow a \bullet \beta, i, j]$ and the appropriate *scan* step. This is incorporated in the *bottom-up left-corner* parsing schem **buLC**, defined as usual by means of a parsing system $\mathbb{P}_{buLC}$ for some arbitrary context-free grammar G (and the standard hypothesis for string $a_1 \ldots a_n$).

$$\begin{array}{lcl} \mathcal{I}^{(1)} & = & \{[A \rightarrow X\alpha \bullet \beta, i, j] \mid A \rightarrow X\alpha\beta \in P \wedge 0 \leq i \leq j\}, \\ \mathcal{I}^{(2)} & = & \{[A \rightarrow \bullet, j, j] \mid A \rightarrow \varepsilon \in P \wedge j \geq 0\}, \\ \mathcal{I}_{buLC} & = & \mathcal{I}^{(1)} \cup \mathcal{I}^{(2)}; \\ D^{\varepsilon} & = & \{\vdash [A \rightarrow \bullet, j, j]\}, \\ D^{LC(a)} & = & \{[a, j-1, j] \vdash [B \rightarrow a \bullet \beta, j-1, j]\}, \\ D^{LC(A)} & = & \{[A \rightarrow \alpha \bullet, i, j] \vdash [B \rightarrow A \bullet \beta, i, j]\}, \\ D^{Scan} & = & \{[A \rightarrow \alpha \bullet a\beta, i, j], [a, j, j+1] \vdash [A \rightarrow \alpha a \bullet \beta, i, j+1]\}, \\ D^{Compl} & = & \{[A \rightarrow \alpha \bullet B\beta, i, j], [B \rightarrow \gamma \bullet, j, k] \vdash [A \rightarrow \alpha B \bullet \beta, i, k]\}, \\ D_{buLC} & = & D^{\varepsilon} \cup D^{LC(a)} \cup D^{LC(A)} \cup D^{Scan} \cup D^{Compl}. \end{array}$$

It follows easily that **buE** $\stackrel{\mathbf{sc}}{\Longrightarrow}$ **buLC**. □

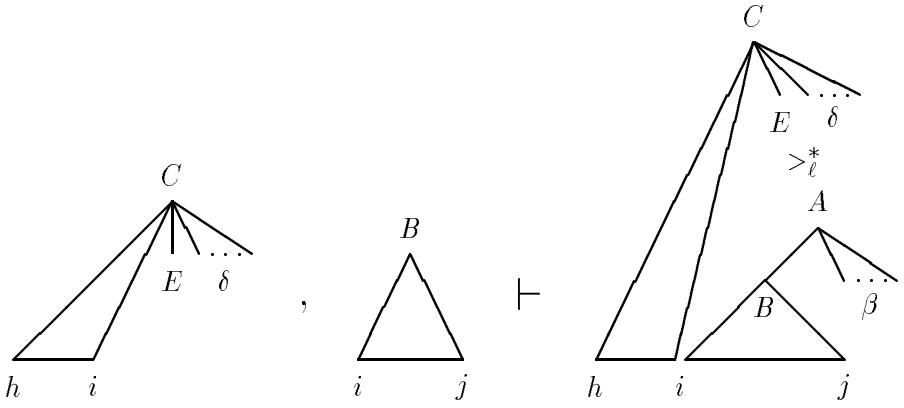

Figure 2: The (*predictive*) *left-corner* step

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

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

Definition 25 (*left corner relations*)
The *left corner* of a non-empty production is the leftmost right-hand side symbol (i.e., production $A \rightarrow X\alpha$ has left corner X); the left corner of an empty production is ε.
The relation $>_\ell$ on $N \times (N \cup \Sigma \cup \{\varepsilon\})$ is defined by

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

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

Example 26 (**LC**)
We can now proceed to define a schema **LC** for a left-corner parser. Clearly, $[A \rightarrow \bullet B\beta, i, i]$ will be recognized by the Earley algorithm if there is some valid item $[C \rightarrow \alpha \bullet E\delta, h, i]$ with $E >_\ell^* A$. Moreover, there is such an item with $\alpha \neq \varepsilon$, unless, perhaps, $i = 0$ and $C = S$. For this exceptional case we retain items $[S \rightarrow \bullet \gamma, 0, 0]$ as usual. The discarded *complete* steps are replaced by *left-corner* steps as follows:

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

see Figure 2; similarly for consequents of the form $[A \rightarrow a \bullet \beta, j-1, j]$.
The general idea of a *left-corner* step involves slightly different details for nonterminal, terminal, and empty left corners. Thus we obtain the following parsing schema **LC**, as usual by defining a parsing system $\mathbb{P}_{LC}$ for an arbitrary grammar.

$$
\begin{aligned}
\mathcal{I}^{(1)} &= \{[A{\rightarrow}X\alpha{\bullet}\beta, i, j] \mid A{\rightarrow}X\alpha\beta \in P \wedge 0 \leq i \leq j\},\\
\mathcal{I}^{(2)} &= \{[A{\rightarrow}{\bullet}, j, j] \mid A{\rightarrow}\varepsilon \in P \wedge j \geq 0\},\\
\mathcal{I}^{(3)} &= \{[S{\rightarrow}{\bullet}\gamma, 0, 0] \mid S{\rightarrow}\gamma \in P\},\\
\mathcal{I}_{LC} &= \mathcal{I}^{(1)} \cup \mathcal{I}^{(2)} \cup \mathcal{I}^{(3)};\\
D^{Init} &= \{\vdash [S{\rightarrow}{\bullet}\gamma, 0, 0]\},\\
D^{LC(A)} &= \{[C{\rightarrow}\gamma{\bullet}E\delta, h, i], [A{\rightarrow}\alpha{\bullet}, i, j] \vdash [B{\rightarrow}A{\bullet}\beta, i, j] \mid E >_\ell^* B\},\\
D^{LC(a)} &= \{[C{\rightarrow}\gamma{\bullet}E\delta, h, i], [a, i, i+1] \vdash [B{\rightarrow}a{\bullet}\beta, i, i+1] \mid E >_\ell^* B\},\\
D^{LC(\varepsilon)} &= \{[C{\rightarrow}\gamma{\bullet}E\delta, h, i], \vdash [B{\rightarrow}{\bullet}, i, i] \mid E >_\ell^* B\},\\
D^{Scan} &= \{[A{\rightarrow}\alpha{\bullet}a\beta, i, j], [a, j, j+1] \vdash [A{\rightarrow}\alpha a{\bullet}\beta, i, j+1]\},\\
D^{Compl} &= \{[A{\rightarrow}\alpha{\bullet}B\beta, i, j], [B{\rightarrow}\gamma{\bullet}, j, k] \vdash [A{\rightarrow}\alpha B{\bullet}\beta, i, k]\},\\
D_{LC} &= D^{Init} \cup D^{LC(a)} \cup D^{LC(A)} \cup D^{LC(\varepsilon)} \cup D^{Scan} \cup D^{Compl}.
\end{aligned}
$$
□

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

It is left to the reader to verify that **buLC** $\stackrel{\text{df}}{\Longrightarrow}$ **LC** and, moreover, **Earley** $\stackrel{\text{sf}}{\Longrightarrow}$ **LC**.

5.2 De Vreught and Honig's algorithm

We define several variants of a parsing schema for an algorithm defined by de Vreught and Honig [dVH89, dVH91], primarily intended for parallel processing. Rather than working to a production from left to right, as is done in Earley's algorithm, one could start at an arbitrary position in the right-hand side and from there extend the recognized part in both directions. To this end, we use *double dotted items* of the form $[A{\rightarrow}\alpha{\bullet}\beta{\bullet}\gamma, i, j]$, where β has been recognized, and α and γ still have to be searched for.

The algorithm of de Vreught and Honig has two basic steps, called *include* and *concatenate*. The idea of both steps is illustrated in Figure 3. The following schema for our first version of the algorithm should be clear.

Example 27 (dVH1)
For an arbitrary context-free grammar a parsing system $\mathbb{P}_{dVH1}$ is defined by

$$
\begin{aligned}
\mathcal{I}_{dVH1} &= \{[A{\rightarrow}\alpha{\bullet}\beta{\bullet}\gamma, i, j] \mid A{\rightarrow}\alpha\beta\gamma \in P \wedge 0 \leq i \leq j\};\\
D^{Init} &= \{[a, j-1, j] \vdash [A{\rightarrow}\alpha{\bullet}a{\bullet}\gamma, j-1, j]\},\\
D^{\varepsilon} &= \{\ \vdash [B{\rightarrow}{\bullet}{\bullet}, j, j]\},
\end{aligned}
$$

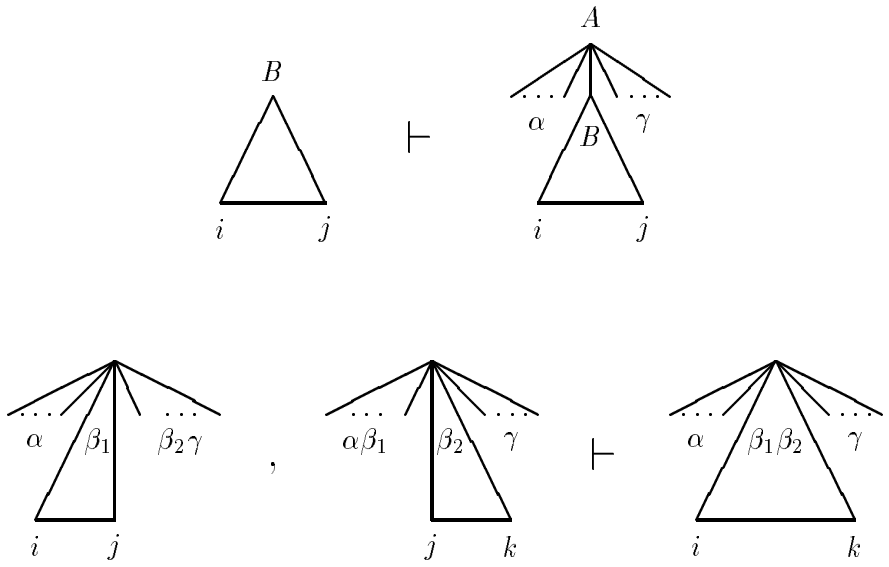

Figure 3: The *include* and *concatenate* steps

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

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

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

Example 28 (dVH2)
We observe that D_{dVH1} is redundant, in the following way.
An item $[A{\rightarrow}\alpha\bullet XYZ\bullet\gamma, i, j]$ can be concatenated in two different ways:

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

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

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

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

and leave $\mathcal{I}$, D^{Init}, D^{ε} and D^{Incl} as in **dVH1**. □

Example 29 (dVH3)
A further optimization of **dVH2** into **dVH3** is carried out as follows. Observe that items of the form $[A{\rightarrow}\alpha\bullet\beta\bullet, i, j]$ with $|\alpha| \geq 1$ and $|\beta| \geq 2$ are useless in $\mathbb{P}_{dVH2}$, in the sense that they do not occur as an antecedent in any derivation step. Hence, these items can be discarded. Similarly, any item of the form $[A{\rightarrow}\alpha\bullet\beta\bullet\gamma, i, j]$ with $|\alpha| \geq 1$, $|\beta| \geq 2$ and $|\gamma| \geq 1$ can concatenate to

the right, but cannot contribute to the recognition of a *passive item*, i.e., an item of the form $[A{\rightarrow}\bullet\beta\bullet, i, j]$. Hence the whole set

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

can be considered useless; these items can only be used to recognize further items in this set, but non of these items can be used to recognize a passive item. Hence we delete this set and discard all deduction steps that have one of these items as antecedent or as consequent. The deduction system $\mathbb{P}_{dVH3}$ for an arbitrary grammar is defined by

$$\begin{aligned}
\mathcal{I}^{(1)} &= \{[A{\rightarrow}\alpha\bullet X\bullet\gamma, i, j] \mid A{\rightarrow}\alpha X\gamma \in P \wedge 0 \leq i \leq j\},\\
\mathcal{I}^{(2)} &= \{[A{\rightarrow}\bullet X\beta\bullet\gamma, i, j] \mid A{\rightarrow}X\beta\gamma \in P \wedge 0 \leq i \leq j\},\\
\mathcal{I}^{(3)} &= \{[A{\rightarrow}\bullet\bullet, j, j] \mid A{\rightarrow}\varepsilon \in P \wedge j \geq 0\},\\
\mathcal{I}_{dVH3} &= \mathcal{I}^{(1)} \cup \mathcal{I}^{(2)} \cup \mathcal{I}^{(3)};\\
D^{Init} &= \{[a, j-1, j] \vdash [A{\rightarrow}\alpha\bullet a\bullet\gamma, j-1, j]\},\\
D^{\varepsilon} &= \{\vdash [B{\rightarrow}\bullet\bullet, j, j]\},\\
D^{Incl} &= \{[B{\rightarrow}\bullet\beta\bullet, i, j] \vdash [A{\rightarrow}\alpha\bullet B\bullet\gamma, i, j]\},\\
D^{Concat} &= \{[A{\rightarrow}\bullet\alpha\bullet X\gamma, i, j], [A{\rightarrow}\alpha\bullet X\bullet\gamma, j, k] \vdash [A{\rightarrow}\bullet\alpha X\bullet\gamma, i, k]\},\\
D_{dVH3} &= D^{Init} \cup D^{\varepsilon} \cup D^{Incl} \cup D^{Concat}.
\end{aligned}$$

□

It is left to the reader to verify **dVH1** $\stackrel{\mathbf{sf}}{\Longrightarrow}$ **dVH2** $\stackrel{\mathbf{sf}}{\Longrightarrow}$ **dVH3**.

Example 30 (dVH3 $\stackrel{\mathbf{sc}}{\Longrightarrow}$ buLC)
In order to show that **dVH3** can be filtered to **buLC**, we have to get rid of a tiny difference in notation. It should be clear that a single dotted item $[A{\rightarrow}\alpha\bullet\beta, i, j]$ and a double dotted item $[A{\rightarrow}\bullet\alpha\bullet\beta, i, j]$ are merely different notations for the same object; both refer to the same production for which $\alpha{\Rightarrow}^* a_{i+1} \ldots a_j$ holds. Then, clearly, $\mathcal{I}_{buLC} \subset \mathcal{I}_{dVH3}$.
It remains to be shown that $\vdash^*_{buLC} \subseteq \vdash^*_{dVH3}$. To this end it suffices to show that for every deduction step $y_1 \ldots, y_k \vdash x \in D_{buLC}$ it holds that $y_1 \ldots, y_k \vdash^*_{dVH3} x$.
For D^{Compl} in $\mathbb{P}_{buLC}$, an arbitrary deduction step

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

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

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

similarly for D^{Scan}. The other cases are trivial. Thus **dVH3** $\stackrel{\mathbf{sc}}{\Longrightarrow}$ **buLC**. □

Adding it all together, we have shown that **dVH1** $\stackrel{\mathbf{sc}}{\Longrightarrow}$ **LC**. The conclusion should *not* be, however, that de Vreught and Honig's algorithm is a sub-optimal version of (buttom-up) left-corner parsing. A subtle but decisive change took place in the seemingly harmless static filter **dVH1** $\stackrel{\mathbf{sf}}{\Longrightarrow}$ **dVH2**, where we laid down that the the concatenation of right-hand side elements is from left to right. A different, more general way to eliminate the redundancy in **dVH1** is to start expanding the right-hand side from the "most interesting" symbol, called the *head* of a production, rather than the leftmost symbol. This leads to so-called *Head Corner* (HC) parsers. A bottom-up head-corner parser that is very similar to the one defined by Satta and Stock[5] [SS89] can be obtained as a step contraction of the Vreught and Honig's algorithm along similar lines; top-down prediction can be added as in [SA93].

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

6 Related work and further applications

The notion of a chart parser has been introduced by Kay [Kay80]. Logical deduction as a basis for description of chart parsers is due to Pereira and Warren [PW83]. But our framework has a rather different emphasis. While the "Parsing as Deduction" approach is primarily interested in connecting the parsing logic with unification-based grammar formalisms, parsing schemata use deduction merely as a convenient notation for describing the relations between different context-free parsing algorithms. The difference in emphasis is illustrated by comparing Shieber's Ph.D. Thesis [Shi92] with the extension of parsing schemata to unification grammars in [Sik93]: the former describes an Earley-like parser for arbitrary unification grammars, whereas the latter describes arbitrary parsing algorithms for a simple PATR-like grammar.

An urgent question that needs to be addressed is: *how general is this framework*? We have simply supposed that parsing algorithms are based on items, but what about algorithms that do not use items? In [Sik93] it is argued at length that items should be understood as congruence classes of partial parse trees. Almost all parsing algorithms are *constructive* in the sense that a parse is created piece by piece. Hence, explicitly or implicitly, a trace is left of those pieces, the intermediate results of the parser. These intermediate results are not necessarily partial parse trees, but it must be objects that denote *relevant properties* of those partial parses. In the CYK algorithm, for example, a recognized item $[A, i, j]$ denotes the existence of *some* tree with root A and yield $a_{i+1} \ldots a_j$. Perhaps there are different trees with the specified root and yield. The internal structure of such a partial parse tree is not relevant for the further proceeding of the algorithm and therefore not represented. So one can see an item as a *partial specification* of a partial parse tree, or alternatively, as a *set of trees*, viz., all trees that conform to this partial specification. Different algorithms use different notations for items, simply because the salient properties of intermediate results that guide the parsing

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

process are different. All these items can be seen as simplified notations of a more fundamental kind of item that represents sets of trees sharing relevant properties of whatever kind.

An important class of parsers, notably Generalized LR parsers [Tom85], is based on the paradigm of a (nondeterministic) pushdown automaton. In a GLR parser, items are used at *compile time* to compute the parsing table, which makes it look rather different from an item-based parser. But it is no big deal to partly "uncompile" a GLR parser and make the recognized items visible at run time. An LR(0)-based GLR parser has an underlying parsing schema that is identical to the Earley schema, except for some techical restrictions on grammars imposed by the GLR machinery, i.e., **GLR(0)** $\stackrel{\text{ext}}{\Longrightarrow}$ **Earley**. Having uncovered this close relation, one can cross-fertilize variants of both algorithms; see [SL92] for a parallel parser that combines Tomita's graph-structured stack with the straightforward parallelization of bottom-up Earley.

A more theoretical application of parsing schemata is the simplification of correctness proofs. A parsing schema is rather more easy to prove correct, simply because there is much less to prove. The correctness of an algorithm is obtained by showing that it duly implements some correct schema. Notably for LR parsers this is a substantial simplification.

In this limited space we only discussed context-free parsing. It is not difficult to extend parsing schemata to unification grammars, provided that the grammar formalism has some notion of context-free backbone. The trend in unification grammars seems to be that the context-free backbone degenerates; more and more information is put into features. Nagata [Nag92] and Maxwell and Kaplan [MK93] have pointed out, however, that treating phrase structure grammar as functional constraints convenient for writing down grammars, but counterproductive for parsing efficiency. The parsing efficiency of unification grammars can be increased by retrieving a context-free backbone that covers more than just the *cat* feature. Context-free parsing, therefore, remains to be of interest for natural language processing.

7 Conclusion

We have given an introduction to the parsing schemata framework, that provides a well-defined level of abstraction between context-free grammars and context-free parsing algorithms. It is claimed that the framework covers all constructive parsing algorithms, both sequential and parallel.

Structural relations between parsing algorithms can investigated in a formal but nevertheless conceptually clear manner, by comparing the parsing schemata underlying these algorithms. Generalization and filtering relations have been formally defined and illustrated with a series of examples.

References

[dVH89] de Vreught JPM, Honig HJ (1989) A Tabular Bottom-Up Recognizer. Report 90-31, Dept. of Applied Mathematics and Informatics, Delft University of Technology, Delft, the Netherlands.

[dVH91] de Vreught JPM, Honig HJ (1991) Slow and fast parallel recognition. *2nd Int. Workshop on Parsing Technologies*, Cancun, Mexico, 127–135.

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

[GHR80] Graham SL, Harrison MA, Ruzzo WL (1980) An Improved Context-Free Recognizer. *ACM Transactions on Programming Languages and Systems* **2**: 415–462.

[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.
Reprinted in: Grosz B et al. (1982) *Readings in Natural Language Processing*, Morgan Kaufmann, Los Altos, Ca.

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

[MT+83] Matsumoto Y, Tanaka H, Hirakawa H, Miyoshi H, Yasukawa H (1983) BUP: a bottom-up parser embedded in Prolog. *New Generation Computing* **1**: 145–158.

[Nag92] Nagata M (1992) An Empirical Study on Rule Granularity and Unification Interleaving Toward an Efficient Unification-Based Parsing System. *14th Int. Conf. on Computational Linguistics* (*COLING'92*), Nantes, France, 177–183.

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

[PW83] Pereira FCN, Warren, DHD (1983) Parsing as Deduction. *21th Ann. Conf. of the ACL*, Cambridge, Mass., 137–144.

[RL70] Rosenkrantz DJ, Lewis PM (1970) Deterministic Left Corner Parsing. *11th Ann. Symp. on Switching and Automata Theory*, 139–152.

[SA93] Sikkel K, op den Akker R (1993) Predictive Head-Corner Chart Parsing. *3rd Int. Workshop on Parsing Technologies*, Tilburg/Durbuy, NL/Belgium, 267–276.

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

[Sik93] Sikkel K (1993) Parsing schemata. Ph.D. Thesis, University of Twente, Enschede, the Netherlands.

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

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

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

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

Reference to this paper:
K. Sikkel. How to compare the structure of parsing algorithms. In: G. Pighizzini, P. San Pietro (Eds.), Proc. ASMICS Workshop on Parsing Theory, Milano, Italy, October 1994. Technical Report 126–94, Dipartimento di Scienze dell'Informazione, Università di Milano, pp. 21–39.