A Deductive Approach to Dependency Parsing*

Carlos Gómez-Rodríguez
Departamento de Computación
Universidade da Coruña, Spain
cgomezr@udc.es

John Carroll and David Weir
Department of Informatics
University of Sussex, United Kingdom
{johnca,davidw}@sussex.ac.uk

Abstract

We define a new formalism, based on Sikkel's parsing schemata for constituency parsers, that can be used to describe, analyze and compare dependency parsing algorithms. This abstraction allows us to establish clear relations between several existing projective dependency parsers and prove their correctness.

1 Introduction

Dependency parsing consists of finding the structure of a sentence as expressed by a set of directed links (dependencies) between words. This is an alternative to constituency parsing, which tries to find a division of the sentence into segments (constituents) which are then broken up into smaller constituents. Dependency structures directly show head-modifier and head-complement relationships which form the basis of predicate argument structure, but are not represented explicitly in constituency trees, while providing a representation in which no non-lexical nodes have to be postulated by the parser. In addition to this, some dependency parsers are able to represent non-projective structures, which is an important feature when parsing free word order languages in which discontinuous constituents are common.

The formalism of parsing schemata (Sikkel, 1997) is a useful tool for the study of constituency parsers since it provides formal, high-level descriptions of parsing algorithms that can be used to prove their formal properties (such as correctness), establish relations between them, derive new parsers from existing ones and obtain efficient implementations automatically (Gómez-Rodríguez et al., 2007). The formalism was initially defined for context-free grammars and later applied to other constituency-based formalisms, such as tree-adjoining grammars (Alonso et al., 1999). However, since parsing schemata are defined as deduction systems over sets of constituency trees, they cannot be used to describe dependency parsers.

In this paper, we define an analogous formalism that can be used to define, analyze and compare dependency parsers. We use this framework to provide uniform, high-level descriptions for a wide range of well-known algorithms described in the literature, and we show how they formally relate to each other and how we can use these relations and the formalism itself to prove their correctness.

1.1 Parsing schemata

Parsing schemata (Sikkel, 1997) provide a formal, simple and uniform way to describe, analyze and compare different constituency-based parsers.

The notion of a parsing schema comes from considering parsing as a deduction process which generates intermediate results called *items*. An initial set of items is directly obtained from the input sentence, and the parsing process consists of the application of inference rules (*deduction steps*) which produce new items from existing ones. Each item contains a piece of information about the sentence's structure, and a successful parsing process will produce at least one *final item* containing a full parse tree for the sentence or guaranteeing its existence.

Items in parsing schemata are formally defined as sets of partial parse trees from a set denoted $Trees(G)$, which is the set of all the possible partial parse trees that do not violate the constraints imposed by a grammar G. More formally, an item set $\mathcal{I}$ is defined by Sikkel as a quotient set associated with an equivalence relation on $Trees(G)$.[1]

Valid parses for a string are represented by items containing complete *marked parse trees* for that string. Given a context-free grammar $G =$

*Partially supported by Ministerio de Educación y Ciencia and FEDER (TIN2004-07246-C03, HUM2007-66607-C04), Xunta de Galicia (PGIDIT07SIN005206PR, PGIDIT05PXIC-10501PN, PGIDIT05PXIC30501PN, Rede Galega de Proc. da Linguaxe e RI) and Programa de Becas FPU.

[1] While Shieber et al. (1995) also view parsers as deduction systems, Sikkel formally defines items and related concepts, providing the mathematical tools to reason about formal properties of parsers.

Proceedings of ACL-08: HLT, pages 968–976,
Columbus, Ohio, USA, June 2008.

(N, Σ, P, S), a *marked parse tree* for a string $w_1 \dots w_n$ is any tree $\tau \in Trees(G)/root(\tau) = S \wedge yield(\tau) = \underline{w}_1 \dots \underline{w}_n$[2]. An item containing such a tree for some arbitrary string is called a *final item*. An item containing such a tree for a particular string $w_1 \dots w_n$ is called a *correct final item* for that string.

For each input string, a parsing schema's deduction steps allow us to infer a set of items, called *valid items* for that string. A parsing schema is said to be *sound* if all valid final items it produces for any arbitrary string are correct for that string. A parsing schema is said to be *complete* if all correct final items are valid. A *correct* parsing schema is one which is both sound and complete. A correct parsing schema can be used to obtain a working implementation of a parser by using deductive engines such as the ones described by Shieber et al. (1995) and Gómez-Rodríguez et al. (2007) to obtain all valid final items.

2 Dependency parsing schemata

Although parsing schemata were initially defined for context-free parsers, they can be adapted to different constituency-based grammar formalisms, by finding a suitable definition of $Trees(G)$ for each particular formalism and a way to define deduction steps from its rules. However, parsing schemata are not directly applicable to dependency parsing, since their formal framework is based on constituency trees.

In spite of this problem, many of the dependency parsers described in the literature are constructive, in the sense that they proceed by combining smaller structures to form larger ones until they find a complete parse for the input sentence. Therefore, it is possible to define a variant of parsing schemata, where these structures can be defined as items and the strategies used for combining them can be expressed as inference rules. However, in order to define such a formalism we have to tackle some issues specific to dependency parsers:

• Traditional parsing schemata are used to define grammar-based parsers, in which the parsing process is guided by some set of rules which are used to license deduction steps: for example, an Earley *Predictor* step is tied to a particular grammar rule, and can only be executed if such a rule exists. Some dependency parsers are also grammar-based: for example, those described by Lombardo and Lesmo (1996), Barbero et al. (1998) and Kahane et al. (1998) are tied to the formalizations of dependency grammar using context-free like rules described by Hays (1964) and Gaifman (1965). However, many of the most widely used algorithms (Eisner, 1996; Yamada and Matsumoto, 2003) do not use a formal grammar at all. In these, decisions about which dependencies to create are taken individually, using probabilistic models (Eisner, 1996) or classifiers (Yamada and Matsumoto, 2003). To represent these algorithms as deduction systems, we use the notion of *D-rules* (Covington, 1990). D-rules take the form $a \rightarrow b$, which says that word b can have a as a dependent. Deduction steps in non-grammar-based parsers can be tied to the D-rules associated with the links they create. In this way, we obtain a representation of the semantics of these parsing strategies that is independent of the particular model used to take the decisions associated with each D-rule.

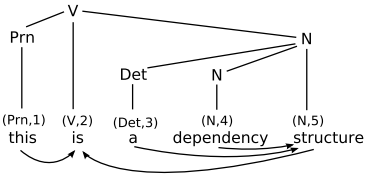

Figure 1: Representation of a dependency structure with a tree. The arrows below the words correspond to its associated dependency graph.

• The fundamental structures in dependency parsing are *dependency graphs*. Therefore, as items for constituency parsers are defined as sets of partial constituency trees, it is tempting to define items for dependency parsers as sets of partial dependency graphs. However, predictive grammar-based algorithms such as those of Lombardo and Lesmo (1996) and Kahane et al. (1998) have operations which postulate rules and cannot be defined in terms of dependency graphs, since they do not do any modifications to the graph. In order to make the formalism general enough to include these parsers, we define items in terms of sets of partial dependency *trees* as shown in Figure 1. Note that a dependency graph can always be extracted from such a tree.

• Some of the most popular dependency parsing algorithms, like that of Eisner (1996), work by connecting *spans* which can represent *disconnected* dependency graphs. Such spans cannot be represented by a single dependency tree. Therefore, our formalism allows items to be sets of *forests* of partial dependency trees, instead of sets of trees.

[2] $\underline{w}_i$ is shorthand for the *marked terminal* (w_i, i). These are used by Sikkel (1997) to link terminal symbols to string positions so that an input sentence can be represented as a set of trees which are used as initial items (hypotheses) for the deduction system. Thus, a sentence $w_1 \dots w_n$ produces a set of hypotheses $\{\{w_1(\underline{w}_1)\}, \dots, \{w_n(\underline{w}_n)\}\}$.

Taking these considerations into account, we define the concepts that we need to describe item sets for dependency parsers:

Let Σ be an alphabet of terminal symbols.

Partial dependency trees: We define the set of *partial dependency trees* (*D-trees*) as the set of finite trees where children of each node have a left-to-right ordering, each node is labelled with an element of $\Sigma \cup (\Sigma \times \mathbb{N})$, and the following conditions hold:

- All nodes labelled with marked terminals $\underline{w_i} \in (\Sigma \times \mathbb{N})$ are leaves,
- Nodes labelled with terminals $w \in \Sigma$ do not have more than one daughter labelled with a marked terminal, and if they have such a daughter node, it is labelled $\underline{w_i}$ for some $i \in \mathbb{N}$,
- Left siblings of nodes labelled with a marked terminal $\underline{w_k}$ do not have any daughter labelled $\underline{w_j}$ with $j \geq k$. Right siblings of nodes labelled with a marked terminal $\underline{w_k}$ do not have any daughter labelled $\underline{w_j}$ with $j \leq k$.

We denote the root node of a partial dependency tree t as $root(t)$. If $root(t)$ has a daughter node labelled with a marked terminal $\underline{w_h}$, we will say that $\underline{w_h}$ is the *head* of the tree t, denoted by $head(t)$. If all nodes labelled with terminals in t have a daughter labelled with a marked terminal, t is *grounded*.

Relationship between trees and graphs: Let $t \in$ *D-trees* be a partial dependency tree; $g(t)$, its associated dependency graph, is a graph (V, E)

- $V = \{\underline{w_i} \in (\Sigma \times \mathbb{N}) \mid \underline{w_i}$ is the label of a node in $t\}$,
- $E = \{(\underline{w_i}, \underline{w_j}) \in (\Sigma \times \mathbb{N})^2 \mid C, D$ are nodes in t such that D is a daughter of C, $\underline{w_j}$ the label of a daughter of C, $\underline{w_i}$ the label of a daughter of $D\}$.

Projectivity: A partial dependency tree $t \in$ *D-trees* is *projective* iff $yield(t)$ cannot be written as $\ldots \underline{w_i} \ldots \underline{w_j} \ldots$ where $i \geq j$.

It is easy to verify that the dependency graph $g(t)$ is projective with respect to the linear order of marked terminals $\underline{w_i}$, according to the usual definition of projectivity found in the literature (Nivre, 2006), if and only if the tree t is projective.

Parse tree: A partial dependency tree $t \in$ *D-trees* is a *parse tree* for a given string $w_1 \ldots w_n$ if its yield is a permutation of $\underline{w_1} \ldots \underline{w_n}$. If its yield is exactly $\underline{w_1} \ldots \underline{w_n}$, we will say it is a *projective parse tree* for the string.

Item set: Let $\delta \subseteq$ *D-trees* be the set of dependency trees which are acceptable according to a given grammar G (which may be a grammar of D-rules or of CFG-like rules, as explained above). We define an *item set* for dependency parsing as a set $\mathcal{I} \subseteq \Pi$, where Π is a partition of 2^{δ}.

Once we have this definition of an item set for dependency parsing, the remaining definitions are analogous to those in Sikkel's theory of constituency parsing (Sikkel, 1997), so we will not include them here in full detail. A *dependency parsing system* is a deduction system $(\mathcal{I}, H, D)$ where $\mathcal{I}$ is a dependency item set as defined above, H is a set containing *initial items* or *hypotheses*, and $D \subseteq (2^{(H \cup \mathcal{I})} \times \mathcal{I})$ is a set of *deduction steps* defining an inference relation $\vdash$.

Final items in this formalism will be those containing some forest F containing a parse tree for some arbitrary string. An item containing such a tree for a particular string $w_1 \ldots w_n$ will be called a *correct final item* for that string in the case of nonprojective parsers. When defining projective parsers, correct final items will be those containing *projective* parse trees for $w_1 \ldots w_n$. This distinction is relevant because the concepts of soundness and correctness of parsing schemata are based on correct final items (cf. section 1.1), and we expect correct projective parsers to produce only projective structures, while nonprojective parsers should find all possible structures including nonprojective ones.

3 Some practical examples

3.1 Col96 (Collins, 96)

One of the most straightforward projective dependency parsing strategies is the one described by Collins (1996), directly based on the CYK parsing algorithm. This parser works with dependency trees which are linked to each other by creating links between their heads. Its item set is defined as $\mathcal{I}_{Col96} = \{[i, j, h] \mid 1 \leq i \leq h \leq j \leq n\}$, where an item $[i, j, h]$ is defined as the set of forests containing a single projective dependency tree t such that t is grounded, $yield(t) = \underline{w_i} \ldots \underline{w_j}$ and $head(t) = \underline{w_h}$.

For an input string $w_1 \ldots w_n$, the set of hypotheses is $H = \{[i, i, i] \mid 0 \leq i \leq n+1\}$, i.e., the set of forests containing a single dependency tree of the form $w_i(\underline{w_i})$. This same set of hypotheses can be used for all the parsers, so we will not make it explicit for subsequent schemata.[3]

The set of final items is $\{[1, n, h] \mid 1 \leq h \leq n\}$: these items trivially represent parse trees for the input sentence, where w_h is the sentence's head. The deduction steps are shown in Figure 2.

[3]Note that the words w_0 and w_{n+1} used in the definition do not appear in the input: these are dummy terminals that we will call beginning of sentence (BOS) and end of sentence (EOS) marker, respectively; and will be needed by some parsers.

Col96 (Collins,96):

$$\textit{R-Link}\ \frac{\overset{[i,j,h_1]}{[j+1,k,h_2]}}{[i,k,h_2]}\ w_{h_1} \rightarrow w_{h_2}$$

$$\textit{L-Link}\ \frac{\overset{[i,j,h_1]}{[j+1,k,h_2]}}{[i,k,h_1]}\ w_{h_2} \rightarrow w_{h_1}$$

Eis96 (Eisner, 96):

$$\textit{Initter}\ \frac{[i,i,i] \qquad [i+1,i+1,i+1]}{[i,i+1,F,F]}$$

$$\textit{R-Link}\ \frac{[i,j,F,F]}{[i,j,T,F]}\ w_i \rightarrow w_j$$

$$\textit{L-Link}\ \frac{[i,j,F,F]}{[i,j,F,T]}\ w_j \rightarrow w_i$$

$$\textit{CombineSpans}\ \frac{\overset{[i,j,b,c]}{[j,k,not(c),d]}}{[i,k,b,d]}$$

ES99 (Eisner and Satta, 99):

$$\textit{R-Link}\ \frac{[i,j,i] \qquad [j+1,k,k]}{[i,k,k]}\ w_i \rightarrow w_k$$

$$\textit{L-Link}\ \frac{[i,j,i] \qquad [j+1,k,k]}{[i,k,i]}\ w_k \rightarrow w_i$$

$$\textit{R-Combiner}\ \frac{[i,j,i] \qquad [j,k,j]}{[i,k,i]}$$

$$\textit{L-Combiner}\ \frac{[i,j,j] \qquad [j,k,k]}{[i,k,k]}$$

YM03 (Yamada and Matsumoto, 2003):

$$\textit{Initter}\ \frac{[i,i,i] \qquad [i+1,i+1,i+1]}{[i,i+1]}$$

$$\textit{R-Link}\ \frac{\overset{[i,j]}{[j,k]}}{[i,k]}\ w_j \rightarrow w_k \qquad \textit{L-Link}\ \frac{\overset{[i,j]}{[j,k]}}{[i,k]}\ w_j \rightarrow w_i$$

LL96 (Lombardo and Lesmo, 96):

$$\textit{Initter}\ \frac{}{[(.S),1,0]}\ {}^{*(S)\in P} \qquad \textit{Predictor}\ \frac{[A(\alpha.B\beta),i,j]}{[B(.\gamma),j+1,j]}\ B(\gamma)\in P$$

$$\textit{Scanner}\ \frac{[A(\alpha.\star\beta),i,h-1] \qquad [h,h,h]}{[A(\alpha\star.\beta),i,h]}\ w_h \text{ IS } A$$

$$\textit{Completer}\ \frac{[A(\alpha.B\beta),i,j] \qquad [B(\gamma.),j+1,k]}{[A(\alpha B.\beta),i,k]}$$

Figure 2: Deduction steps of the parsing schemata for some well-known dependency parsers.

As we can see, we use D-rules as side conditions for deduction steps, since this parsing strategy is not grammar-based. Conceptually, the schema we have just defined describes a recogniser: given a set of D-rules and an input string $w_i \dots w_n$, the sentence can be parsed (projectively) under those D-rules if and only if this deduction system can infer a correct final item. However, when executing this schema with a deductive engine, we can recover the parse forest by following back pointers in the same way as is done with constituency parsers (Billot and Lang, 1989).

Of course, boolean D-rules are of limited interest in practice. However, this schema provides a formalization of a parsing strategy which is independent of the way linking decisions are taken in a particular implementation. In practice, statistical models can be used to decide whether a step linking words a and b (i.e., having $a \rightarrow b$ as a side condition) is executed or not, and probabilities can be attached to items in order to assign different weights to different analyses of the sentence. The same principle applies to the rest of D-rule-based parsers described in this paper.

3.2 Eis96 (Eisner, 96)

By counting the number of free variables used in each deduction step of Collins' parser, we can conclude that it has a time complexity of $O(n^5)$. This complexity arises from the fact that a parentless word (head) may appear in any position in the partial results generated by the parser; the complexity can be reduced to $O(n^3)$ by ensuring that parentless words can only appear at the first or last position of an item. This is the principle behind the parser defined by Eisner (1996), which is still in wide use today (Corston-Oliver et al., 2006; McDonald et al., 2005a).

The item set for Eisner's parsing schema is $\mathcal{I}_{Eis96} = \{[i,j,T,F] \mid 0 \leq i \leq j \leq n\} \cup \{[i,j,F,T] \mid 0 \leq i \leq j \leq n\} \cup \{[i,j,F,F] \mid 0 \leq i \leq j \leq n\}$, where each item $[i,j,T,F]$ is defined as the item $[i,j,j] \in \mathcal{I}_{Col96}$, each item $[i,j,F,T]$ is defined as the item $[i,j,i] \in \mathcal{I}_{Col96}$, and each item $[i,j,F,F]$ is defined as the set of forests of the form $\{t_1,t_2\}$ such that t_1 and t_2 are grounded, $head(t_1) = \underline{w}_i$, $head(t_2) = \underline{w}_j$, and $\exists k \in \mathbb{N}(i \leq k < j)/yield(t_1) = \underline{w}_i \dots \underline{w}_k \wedge yield(t_2) = \underline{w}_{k+1} \dots \underline{w}_j$.

Note that the flags b, c in an item $[i,j,b,c]$ indicate whether the words in positions i and j, respectively, have a parent in the item or not. Items with one of the flags set to T represent dependency trees where the word in position i or j is the head, while items with both flags set to F represent pairs of trees headed at positions i and j, and therefore correspond to disconnected dependency graphs.

Deduction steps[4] are shown in Figure 2. The set of final items is $\{[0,n,F,T]\}$. Note that these items represent dependency trees rooted at the BOS marker w_0, which acts as a "dummy head" for the sentence. In order for the algorithm to parse sentences correctly, we will need to define D-rules to allow w_0 to be linked to the real sentence head.

3.3 ES99 (Eisner and Satta, 99)

Eisner and Satta (1999) define an $O(n^3)$ parser for split head automaton grammars that can be used

[4] Alternatively, we could consider items of the form $[i, i+1, F, F]$ to be hypotheses for this parsing schema, so we would not need an *Initter* step. However, we have chosen to use a standard set of hypotheses valid for all parsers because this allows for more straightforward proofs of relations between schemata.

for dependency parsing. This algorithm is conceptually simpler than Eis96, since it only uses items representing single dependency trees, avoiding items of the form $[i, j, F, F]$. Its item set is $\mathcal{I}_{ES99} = \{[i, j, i] \mid 0 \leq i \leq j \leq n\} \cup \{[i, j, j] \mid 0 \leq i \leq j \leq n\}$, where items are defined as in Collins' parsing schema.

Deduction steps are shown in Figure 2, and the set of final items is $\{[0, n, 0]\}$. (Parse trees have w_0 as their head, as in the previous algorithm).

Note that, when described for head automaton grammars as in Eisner and Satta (1999), this algorithm seems more complex to understand and implement than the previous one, as it requires four different kinds of items in order to keep track of the state of the automata used by the grammars. However, this abstract representation of its underlying semantics as a dependency parsing schema shows that this parsing strategy is in fact conceptually simpler for dependency parsing.

3.4 YM03 (Yamada and Matsumoto, 2003)

Yamada and Matsumoto (2003) define a deterministic, shift-reduce dependency parser guided by support vector machines, which achieves over 90% dependency accuracy on section 23 of the Penn treebank. Parsing schemata are not suitable for directly describing deterministic parsers, since they work at a high abstraction level where a set of operations are defined without imposing order constraints on them. However, many deterministic parsers can be viewed as particular optimisations of more general, nondeterministic algorithms. In this case, if we represent the actions of the parser as deduction steps while abstracting from the deterministic implementation details, we obtain an interesting nondeterministic parser.

Actions in Yamada and Matsumoto's parser create links between two target nodes, which act as heads of neighbouring dependency trees. One of the actions creates a link where the left target node becomes a child of the right one, and the head of a tree located directly to the left of the target nodes becomes the new left target node. The other action is symmetric, performing the same operation with a right-to-left link. An $O(n^3)$ nondeterministic parser generalising this behaviour can be defined by using an item set $\mathcal{I}_{YM03} = \{[i, j] \mid 0 \leq i \leq j \leq n + 1\}$, where each item $[i, j]$ is defined as the item $[i, j, F, F]$ in $\mathcal{I}_{Eis96}$; and the deduction steps are shown in Figure 2.

The set of final items is $\{[0, n + 1]\}$. In order for this set to be well-defined, the grammar must have no D-rules of the form $w_i \rightarrow w_{n+1}$, i.e., it must not allow the EOS marker to govern any words. If this is the case, it is trivial to see that every forest in an item of the form $[0, n + 1]$ must contain a parse tree rooted at the BOS marker and with yield $\underline{w}_0 \ldots \underline{w}_n$.

As can be seen from the schema, this algorithm requires less bookkeeping than any other of the parsers described here.

3.5 LL96 (Lombardo and Lesmo, 96) and other Earley-based parsers

The algorithms in the above examples are based on taking individual decisions about dependency links, represented by D-rules. Other parsers, such as that of Lombardo and Lesmo (1996), use grammars with context-free like rules which encode the preferred order of dependents for each given governor, as defined by Gaifman (1965). For example, a rule of the form $N(Det * PP)$ is used to allow N to have Det as left dependent and PP as right dependent.

The algorithm by Lombardo and Lesmo (1996) is a version of Earley's context-free grammar parser (Earley, 1970) using Gaifman's dependency grammar, and can be written by using an item set $\mathcal{I}_{LomLes} = \{[A(\alpha.\beta), i, j] \mid A(\alpha\beta) \in P \wedge 1 \leq i \leq j \leq n\}$, where each item $[A(\alpha.\beta), i, j]$ represents the set of partial dependency trees rooted at A, where the direct children of A are $\alpha\beta$, and the subtrees rooted at α have yield $\underline{w}_i \ldots \underline{w}_j$. The deduction steps for the schema are shown in Figure 2, and the final item set is $\{[(S.), 1, n]\}$.

As we can see, the schema for Lombardo and Lesmo's parser resembles the Earley-style parser in Sikkel (1997), with some changes to adapt it to dependency grammar (for example, the *Scanner* always moves the dot over the head symbol $*$).

Analogously, other dependency parsing schemata based on CFG-like rules can be obtained by modifying context-free grammar parsing schemata of Sikkel (1997) in a similar way. The algorithm by Barbero et al. (1998) can be obtained from the left-corner parser, and the one by Courtin and Genthial (1998) is a variant of the head-corner parser.

3.6 Pseudo-projectivity

Pseudo-projective parsers can generate non-projective analyses in polynomial time by using a projective parsing strategy and postprocessing the results to establish nonprojective links. For example, the algorithm by Kahane et al. (1998) uses a projective parsing strategy like that of LL96, but using the following initializer step instead of the

Initter and *Predictor*:[5]

Initter $\frac{}{[A(\alpha), i, i-1]}\ A(\alpha) \in P \wedge 1 \leq i \leq n$

4 Relations between dependency parsers

The framework of parsing schemata can be used to establish relationships between different parsing algorithms and to obtain new algorithms from existing ones, or derive formal properties of a parser (such as soundness or correctness) from the properties of related algorithms.

Sikkel (1994) defines several kinds of relations between schemata, which fall into two categories: *generalisation* relations, which are used to obtain more fine-grained versions of parsers, and *filtering* relations, which can be seen as the reverse of generalisation and are used to reduce the number of items and/or steps needed for parsing. He gives a formal definition of each kind of relation. Informally, a parsing schema can be generalised from another via the following transformations:

- Item refinement: We say that $P_1 \xrightarrow{ir} P_2$ (P_2 is an item refinement of P_1) if there is a mapping between items in both parsers such that single items in P_1 are broken into multiple items in P_2 and individual deductions are preserved.
- Step refinement: We say that $P_1 \xrightarrow{sr} P_2$ if the item set of P_1 is a subset of that of P_2 and every single deduction step in P_1 can be emulated by a sequence of inferences in P_2.

On the other hand, a schema can be obtained from another by filtering in the following ways:

- Static/dynamic filtering: $P_1 \xrightarrow{sf/df} P_2$ if the item set of P_2 is a subset of that of P_1 and P_2 allows a subset of the direct inferences in P_1[6].
- Item contraction: The inverse of item refinement. $P_1 \xrightarrow{ic} P_2$ if $P_2 \xrightarrow{ir} P_1$.
- Step contraction: The inverse of step refinement. $P_1 \xrightarrow{sc} P_2$ if $P_2 \xrightarrow{sr} P_1$.

All the parsers described in section 3 can be related via generalisation and filtering, as shown in Figure 3. For space reasons we cannot show formal proofs of all the relations, but we sketch the proofs for some of the more interesting cases:

[5]The initialization step as reported in Kahane's paper is different from this one, as it directly consumes a nonterminal from the input. However, using this step results in an incomplete algorithm. The problem can be fixed either by using the step shown here instead (bottom-up Earley strategy) or by adding an additional step turning it into a bottom-up Left-Corner parser.

[6]Refer to Sikkel (1994) for the distinction between static and dynamic filtering, which we will not use here.

4.1 YM03 $\xrightarrow{sr}$ Eis96

It is easy to see from the schema definitions that $\mathcal{I}_{YM03} \subseteq \mathcal{I}_{Eis96}$. In order to prove the relation between these parsers, we need to verify that every deduction step in YM03 can be emulated by a sequence of inferences in Eis96. In the case of the *Initter* step this is trivial, since the *Initter*s of both parsers are equivalent. If we write the *R-Link* step in the notation we have used for Eisner items, we have

R-Link $\frac{[i,j,F,F] \qquad [j,k,F,F]}{[i,k,F,F]}\ w_j \rightarrow w_k$

This can be emulated in Eisner's parser by an *R-Link* step followed by a *CombineSpans* step:

$[j,k,F,F] \vdash [j,k,T,F]$ (by *R-Link*),

$[j,k,T,F], [i,j,F,F] \vdash [i,k,F,F]$ (by *CombineSpans*).

Symmetrically, the *L-Link* step in YM03 can be emulated by an *L-Link* followed by a *CombineSpans* in Eis96.

4.2 ES99 $\xrightarrow{sr}$ Eis96

If we write the *R-Link* step in Eisner and Satta's parser in the notation for Eisner items, we have

R-Link $\frac{[i,j,F,T] \qquad [j+1,k,T,F]}{[i,k,T,F]}\ w_i \rightarrow w_k$

This inference can be emulated in Eisner's parser as follows:

$\vdash [j,j+1,F,F]$ (by *Initter*),

$[i,j,F,T], [j,j+1,F,F] \vdash [i,j+1,F,F]$ (*CombineSpans*),

$[i,j+1,F,F], [j+1,k,T,F] \vdash [i,k,F,F]$ (*CombineSpans*),

$[i,k,F,F] \vdash [i,k,T,F]$ (by *R-Link*).

The proof corresponding to the *L-Link* step is symmetric. As for the *R-Combiner* and *L-Combiner* steps in ES99, it is easy to see that they are particular cases of the *CombineSpans* step in Eis96, and therefore can be emulated by a single application of *CombineSpans*.

Note that, in practice, the relations in sections 4.1 and 4.2 mean that the ES99 and YM03 parsers are superior to Eis96, since they generate fewer items and need fewer steps to perform the same deductions. These two parsers also have the interesting property that they use disjoint item sets (one uses items representing trees while the other uses items representing pairs of trees); and the union of these disjoint sets is the item set used by Eis96. Also note that the optimisation in YM03 comes from contracting deductions in Eis96 so that linking operations are immediately followed by combining operations; while ES99 does the opposite, forcing combining operations to be followed by linking operations.

4.3 Other relations

If we generalise the linking steps in ES99 so that the head of each item can be in any position, we obtain a

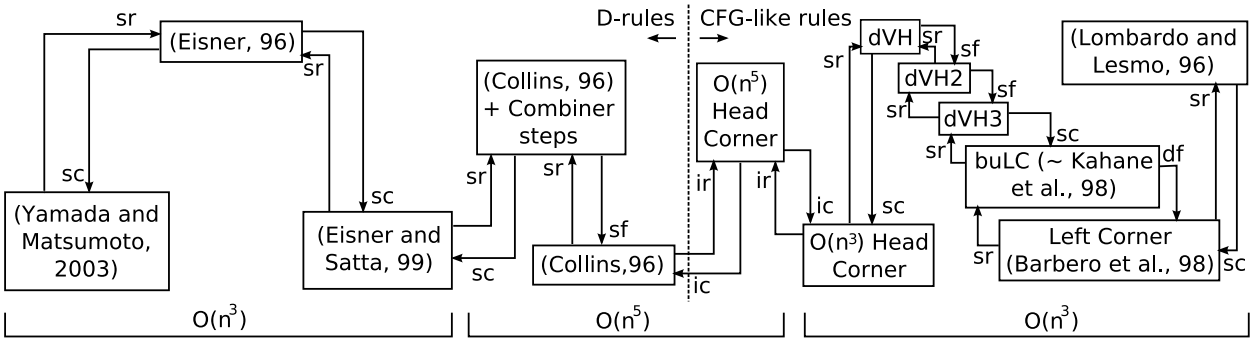

Figure 3: Formal relations between several well-known dependency parsers. Arrows going upwards correspond to generalisation relations, while those going downwards correspond to filtering. The specific subtype of relation is shown in each arrow's label, following the notation in Section 4.

correct $O(n^5)$ parser which can be filtered to Col96 just by eliminating the $Combiner$ steps.

From Col96, we can obtain an $O(n^5)$ head-corner parser based on CFG-like rules by an item refinement in which each Collins item $[i, j, h]$ is split into a set of items $[A(\alpha.\beta.\gamma), i, j, h]$. Of course, the formal refinement relation between these parsers only holds if the D-rules used for Collins' parser correspond to the CFG rules used for the head-corner parser: for every D-rule $B \rightarrow A$ there must be a corresponding CFG-like rule $A \rightarrow \ldots B \ldots$ in the grammar used by the head-corner parser.

Although this parser uses three indices i, j, h, using CFG-like rules to guide linking decisions makes the h indices unnecessary, so they can be removed. This simplification is an item contraction which results in an $O(n^3)$ head-corner parser. From here, we can follow the procedure in Sikkel (1994) to relate this head-corner algorithm to parsers analogous to other algorithms for context-free grammars. In this way, we can refine the head-corner parser to a variant of de Vreught and Honig's algorithm (Sikkel, 1997), and by successive filters we reach a left-corner parser which is equivalent to the one described by Barbero et al. (1998), and a step contraction of the Earley-based dependency parser LL96. The proofs for these relations are the same as those described in Sikkel (1994), except that the dependency variants of each algorithm are simpler (due to the absence of epsilon rules and the fact that the rules are lexicalised).

5 Proving correctness

Another useful feature of the parsing schemata framework is that it provides a formal way to define the correctness of a parser (see last paragraph of Section 1.1) which we can use to prove that our parsers are correct. Furthermore, relations between schemata can be used to derive the correctness of a schema from that of related ones. In this section, we will show how we can prove that the YM03 and ES99 algorithms are correct, and use that fact to prove the correctness of Eis96.

5.1 ES99 is correct

In order to prove the correctness of a parser, we must prove its soundness and completeness (see section 1.1). Soundness is generally trivial to verify, since we only need to check that every individual deduction step in the parser infers a correct consequent item when applied to correct antecedents (i.e., in this case, that steps always generate non-empty items that conform to the definition in 3.3). The difficulty is proving completeness, for which we need to prove that all correct final items are valid (i.e., can be inferred by the schema). To show this, we will prove the stronger result that all correct items are valid.

We will show this by strong induction on the *length* of items, where the length of an item $\iota = [i, k, h]$ is defined as $length(\iota) = k - i + 1$. Correct items of length 1 are the hypotheses of the schema (of the form $[i, i, i]$) which are trivially valid. We will prove that, if all correct items of length m are valid for all $1 \leq m < l$, then items of length l are also valid.

Let $[i, k, i]$ be an item of length l in $\mathcal{I}_{ES99}$ (thus, $l = k - i + 1$). If this item is correct, then it contains a grounded dependency tree t such that $yield(t) = \underline{w}_i \ldots \underline{w}_k$ and $head(t) = \underline{w}_i$.

By construction, the root of t is labelled w_i. Let w_j be the rightmost daughter of w_i in t. Since t is projective, we know that the yield of w_j must be of the form $w_l \ldots w_k$, where $i < l \leq j \leq k$. If $l < j$, then w_l is the leftmost transitive dependent of w_j in t, and if $k > j$, then we know that w_k is the rightmost transitive dependent of w_j in t.

Let t_j be the subtree of t rooted at w_j. Let t_1 be the tree obtained from removing t_j from t. Let t_2 be

the tree obtained by removing all the children to the right of $\underline{w_j}$ from t_j, and t_3 be the tree obtained by removing all the children to the left of $\underline{w_j}$ from t_j. By construction, t_1 belongs to a correct item $[i, l-1, i]$, t_2 belongs to a correct item $[l, j, j]$ and t_3 belongs to a correct item $[j, k, j]$. Since these three items have a length strictly less than l, by the inductive hypothesis, they are valid. This allows us to prove that the item $[i, k, i]$ is also valid, since it can be obtained from these valid items by the following inferences:
$[i, l-1, i], [l, j, j] \vdash [i, j, i]$ (by the *L-Link* step),
$[i, j, i], [j, k, j] \vdash [i, k, i]$ (by the *L-Combiner* step).
This proves that all correct items of length l which are of the form $[i, k, i]$ are correct under the inductive hypothesis. The same can be proved for items of the form $[i, k, k]$ by symmetric reasoning, thus proving that the ES99 parsing schema is correct.

5.2 YM03 is correct

In order to prove correctness of this parser, we follow the same procedure as above. Soundness is again trivial to verify. To prove completeness, we use strong induction on the length of items, where the length of an item $[i, j]$ is defined as $j - i + 1$.

The induction step is proven by considering any correct item $[i, k]$ of length $l > 2$ ($l = 2$ is the base case here since items of length 2 are generated by the *Initter* step) and proving that it can be inferred from valid antecedents of length less than l, so it is valid. To show this, we note that, if $l > 2$, either w_i has at least a right dependent or w_k has at least a left dependent in the item. Supposing that w_i has a right dependent, if t_1 and t_2 are the trees rooted at w_i and w_k in a forest in $[i, k]$, we call w_j the rightmost daughter of w_i and consider the following trees:

v = the subtree of t_1 rooted at w_j, u_1 = the tree obtained by removing v from t_1, u_2 = the tree obtained by removing all children to the right of w_j from v, u_3 = the tree obtained by removing all children to the left of w_j from v.

We observe that the forest $\{u_1, u_2\}$ belongs to the correct item $[i, j]$, while $\{u_3, t_2\}$ belongs to the correct item $[j, k]$. From these two items, we can obtain $[i, k]$ by using the *L-Link* step. Symmetric reasoning can be applied if w_i has no right dependents but w_k has at least a left dependent, and analogously to the case of the previous parser, we conclude that the YM03 parsing schema is correct.

5.3 Eis96 is correct

By using the previous proofs and the relationships between schemata that we explained earlier, it is easy to prove that Eis96 is correct: soundness is, as always, straightforward, and completeness can be proven by using the properties of other algorithms. Since the set of final items in Eis96 and ES99 are the same, and the former is a step refinement of the latter, the completeness of ES99 directly implies the completeness of Eis96.

Alternatively, we can use YM03 to prove the correctness of Eis96 if we redefine the set of final items in the latter to be of the form $[0, n+1, F, F]$, which are equally valid as final items since they always contain parse trees. This idea can be applied to transfer proofs of completeness across any refinement relation.

6 Conclusions

We have defined a variant of Sikkel's parsing schemata formalism which allows us to represent dependency parsing algorithms in a simple, declarative way[7]. We have clarified relations between parsers which were originally described very differently. For example, while Eisner presented his algorithm as a dynamic programming algorithm which combines spans into larger spans, Yamada and Matsumoto's works by sequentially executing parsing actions that move a focus point in the input one position to the left or right, (possibly) creating a dependency link. However, in the parsing schemata for these algorithms we can see (and formally prove) that they are related: one is a refinement of the other.

Parsing schemata are also a formal tool that can be used to prove the correctness of parsing algorithms. The relationships between dependency parsers can be exploited to derive properties of a parser from those of others, as we have seen in several examples.

Although the examples in this paper are centered in projective dependency parsing, the formalism does not require projectivity and can be used to represent nonprojective algorithms as well[8]. An interesting line for future work is to use relationships between schemata to find nonprojective parsers that can be derived from existing projective counterparts.

[7] An alternative framework that formally describes some dependency parsers is that of transition systems (McDonald and Nivre, 2007). This model is based on parser configurations and transitions, and has no clear relationship with the approach described here.

[8] Note that spanning tree parsing algorithms based on edge-factored models, such as the one by McDonald et al. (2005b) are not constructive in the sense outlined in Section 2, so the approach described here does not directly apply to them. However, other nonprojective parsers such as (Attardi, 2006) follow a constructive approach and can be analysed deductively.

References

Miguel A. Alonso, Eric de la Clergerie, David Cabrero, and Manuel Vilares. 1999. Tabular algorithms for TAG parsing. In *Proc. of the Ninth Conference on European chapter of the Association for Computational Linguistics*, pages 150–157, Bergen, Norway. ACL.

Giuseppe Attardi. 2006. Experiments with a Multilanguage Non-Projective Dependency Parser. In *Proc. of the Tenth Conference on Natural Language Learning (CoNLL-X)*, pages 166–170, New York, USA. ACL.

Cristina Barbero, Leonardo Lesmo, Vincenzo Lombarlo, and Paola Merlo. 1998. Integration of syntactic and lexical information in a hierarchical dependency grammar. In *Proc. of the Workshop on Dependency Grammars*, pages 58–67, ACL-COLING, Montreal, Canada.

Sylvie Billot and Bernard Lang. 1989. The structure of shared forest in ambiguous parsing. In *Proc. of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 143–151, Vancouver, British Columbia, Canada, June. ACL.

Michael John Collins. 1996. A new statistical parser based on bigram lexical dependencies. In *Proc. of the 34th annual meeting on Association for Computational Linguistics*, pages 184–191, Morristown, NJ, USA. ACL.

Simon Corston-Oliver, Anthony Aue, Kevin Duh, and Eric Ringger. 2006. Multilingual dependency parsing using Bayes Point Machines. In *Proc. of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 160–167, Morristown, NJ, USA. ACL.

Jacques Courtin and Damien Genthial. 1998. Parsing with dependency relations and robust parsing. In *Proc. of the Workshop on Dependency Grammars*, pages 88–94, ACL-COLING, Montreal, Canada.

Michael A. Covington. 1990. A dependency parser for variable-word-order languages. Technical Report AI-1990-01, Athens, GA.

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

Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proc. of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 457–464, Morristown, NJ, USA. ACL.

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proc. of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 340–345, Copenhagen, August.

Haim Gaifman. 1965. Dependency systems and phrase-structure systems. *Information and Control*, 8:304–337.

Carlos Gómez-Rodríguez, Jesús Vilares, and Miguel A. Alonso. 2007. Compiling declarative specifications of parsing algorithms. In *Database and Expert Systems Applications*, volume 4653 of *Lecture Notes in Computer Science*, pages 529–538, Springer-Verlag.

David Hays. 1964. Dependency theory: a formalism and some observations. *Language*, 40:511–525.

Sylvain Kahane, Alexis Nasr, and Owen Rambow. 1998. Pseudo-projectivity: A polynomially parsable non-projective dependency grammar. In *COLING-ACL*, pages 646–652.

Vincenzo Lombardo and Leonardo Lesmo. 1996. An Earley-type recognizer for dependency grammar. In *Proc. of the 16th conference on Computational linguistics*, pages 723–728, Morristown, NJ, USA. ACL.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005a. Online large-margin training of dependency parsers. In *ACL '05: Proc. of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 91–98, Morristown, NJ, USA. ACL.

Ryan McDonald, Fernando Pereira, Kiril Ribarov and Jan Hajič. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *HLT '05: Proc. of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. ACL.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the Errors of Data-Driven Dependency Parsing Models. In *Proc. of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 122–131.

Joakim Nivre. 2006. *Inductive Dependency Parsing (Text, Speech and Language Technology)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Stuart M. Shieber, Yves Schabes, and Fernando C.N. Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36.

Klaas Sikkel. 1994. How to compare the structure of parsing algorithms. In G. Pighizzini and P. San Pietro, editors, *Proc. of ASMICS Workshop on Parsing Theory. Milano, Italy, Oct 1994*, pages 21–39.

Klaas Sikkel. 1997. *Parsing Schemata — A Framework for Specification and Analysis of Parsing Algorithms*. Texts in Theoretical Computer Science — An EATCS Series. Springer-Verlag, Berlin/Heidelberg/New York.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proc. of 8th International Workshop on Parsing Technologies*, pages 195–206.