

Chapter 1

Introduction

This book provides several theoretical and practical tools that extend the applicability of Sikkel's theory of parsing schemata (Sikkel, 1997) in several different directions.

First, a compilation technique is defined that can be used to obtain efficient implementations of parsers automatically from their corresponding schemata. This makes it possible to use parsing schemata to prototype and test parsing algorithms, without the need of manually converting the formal representation to an efficient implementation in a programming language.

Second, the range of parsing algorithms that can be defined by means of schemata is extended with the definition of new variants of the formalism that can deal with error-repair parsers and dependency-based parsers.

Apart from these tools themselves, the book also introduces several research results that have been obtained by using them. The compilation technique is used to obtain implementations of different parsers for context-free grammars and tree-adjoining grammars, and perform an empirical analysis of their behaviour with real-sized grammars. The extension of parsing schemata for error-repair parsing is used to define a transformation that can be employed to automatically add error-repair capabilities to parsers that do not have them. Finally, the extension of parsing schemata for dependency parsing is used to find formal relationships between several well-known dependency parsers, as well as to define novel algorithms for mildly non-projective dependency structures.

1.1 Motivation

Parsing schemata, introduced by Sikkel (1997), are high-level, declarative descriptions of parsing algorithms. A parsing schema describes a set of

intermediate results that can be obtained by a parser and a set of operations that it can use to generate new results from existing ones, but it imposes no constraints on the order in which to execute the operations or the data structures in which to store the results. We could say that schemata are descriptions of *what* to do, but they abstract away from *how* to do it.

This high abstraction level makes parsing schemata a useful tool to describe, analyse and compare different parsing algorithms: schemata provide simple and uniform descriptions of the parsers, and allow us to focus on their logic without worrying about implementation details. However, if we wished to apply the schemata formalism to practical parsing of natural language text, we would find several problems, and is it these problems that are addressed in this monograph:

- First of all, the high abstraction level of parsing schemata, which makes them useful from a theoretical standpoint, also rules out the possibility of executing them on a computer. If we wish to test an algorithm that we have defined by means of a schema, we must first implement it in a programming language, filling in the control structures and implementation details that were omitted in the schema. Therefore, schemata are useful as a formal framework for defining parsers and analysing their formal properties but not for prototyping them and studying their empirical behaviour.
- Parsing schemata can be used to represent algorithms that analyse input sentences with respect to a given grammar, so that they only will produce a complete analysis of an input sentence if it is grammatical. However, it is common in practical applications to find sentences that do not conform to the constraints of our grammar, since it is not possible to build a grammar that will recognise every possible structure that can appear in a natural language. Therefore, it is interesting to be able to define *robust* parsers that are able to obtain a complete, albeit approximate, analysis for ungrammatical sentences. The formalism of Sikkel (1997) does not allow the relaxation of grammar constraints in order to define this kind of parsers, since it is based on the assumption that all the possible intermediate results are parts of grammatically valid trees.
- The formalism of Sikkel (1997) is formally linked to *constituency-based* parsing, in which the structure of sentences is expressed by dividing them in blocks (*constituents*) which are, in turn, divided into smaller constituents. However, in recent years, there has been

a surge of interest in *dependency-based parsers*, in which the structure of a sentence is represented as a set of binary links (*dependencies*) between its words. Although many current state-of-the-art parsers are based on this assumption, dependency parsers cannot be represented by the formalism of Sikkel (1997).

The purpose and contribution of this monograph is to solve these problems, by defining a set of extensions and tools to make parsing schemata more useful in practice. In addition to this, research results obtained by using these extensions and tools are also presented.

1.2 Background

In this section, the contributions of this book are put into context by briefly introducing the problem of parsing natural language sentences and the formalism of parsing schemata.

1.2.1 Parsing natural language

In the context of computational linguistics, the process of *parsing*, or syntactic analysis, consists of finding the grammatical structure of natural language sentences. Given a sentence represented as a sequence of symbols, each corresponding to a word, a *parsing algorithm* (or simply *parser*) will try to find and output a representation of the syntactic structure of the sentence. The nature of this representation depends on the linguistic theory that the parser uses to describe syntax. In *phrase structure parsers*, or *constituency parsers*, sentences are analysed by dividing them into meaningful segments called *constituents*, which are, in turn, broken up into smaller constituents. The result of a constituency analysis can be represented with a *phrase structure tree* (or *constituency tree*), as can be seen in Figure 1.1a. On the other hand, in *dependency-based parsers*, the structure of a sentence is represented by a set of directed links (*dependencies*) between their words, which form a graph as can be seen in Figure 1.1b.

Parsing is a fundamental process in any natural language processing pipeline, since obtaining the syntactic structure of sentences provides us with information that can be used to extract meaning from them: constituents correspond to units of meaning, and dependency relations describe the ways in which they interact, such as who performed the action described in a sentence or which object is receiving the action. Thus, we can find

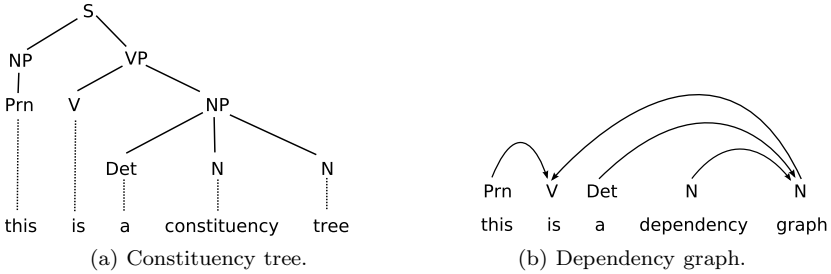


Fig. 1.1 Constituency and dependency structures.

parsers applied to many practical problems in natural language processing where some degree of semantic analysis is necessary or convenient, such as information retrieval (Vilares *et al.*, 2008), information extraction (Surdanu *et al.*, 2003), machine translation (Chiang, 2005; Shen *et al.*, 2008), textual entailment (Herrera *et al.*, 2005), or question answering (Bouma *et al.*, 2005; Amaral *et al.*, 2008).

The practical importance of parsing natural language text has motivated the development of a diverse range of algorithms for this purpose. Different parsers are better suited to different situations, so the question of which parser is better for a practical application depends on many factors, such as the language, domain, application goal or availability of linguistic resources.

According to the linguistic resources that they use, we can broadly classify parsing techniques into two categories: grammar-driven and data-driven approaches (Nivre, 2006b). In *grammar-driven* parsing, the set of valid sentences in a given language is represented by a *grammar*, which is a set of formal rules that describe the kinds of structures that can appear in the language. In a *data-driven* approach, no formal grammar is needed: instead, some learning technique is used to automatically infer linguistic knowledge from (possibly annotated) texts, that can later be used to parse other texts. However, most of the systems used in practice are not purely grammar-driven or data-driven: instead, they combine features of both approaches. For example, a formal grammar can be annotated with statistical information obtained from a corpus of annotated text, or a system may induce a grammar automatically from text data.

In parsers that use grammars, the kinds of rules that can be used to describe syntactic structures and the ways in which they are used to form sentences depend on the particular *grammatical formalism* that we use.

In the field of constituency parsing, *context-free grammars* (CFGs) are a widely-used formalism, since they are simple and can be parsed efficiently, in cubic time with respect to the length of the input (Younger, 1967; Earley, 1970). However, it has been shown that CFGs are not powerful enough to represent natural languages, since some linguistic phenomena (such as Dutch cross-serial dependencies) are not context-free (Joshi, 1985). This led to the development of *mildly context-sensitive* grammar formalisms (Joshi, 1985; Joshi *et al.*, 1991), which are formalisms that are general enough to account for linguistic phenomena that cannot be handled by CFGs, yet restricted enough to still be parsable in polynomial time. An example of this type of formalism is *tree-adjoining grammars* (TAGs), which are parsable in $O(n^6)$, where n is the length of the input (Vijay-Shanker and Joshi, 1985).

In the case of dependency-based parsing, it is also possible to use a grammar. For example, dependency grammars can be defined with the formalism by Hays (1964) and Gaifman (1965) or as bilexical grammars (Eisner, 2000). However, many dependency parsers (Yamada and Matsumoto, 2003; Nivre *et al.*, 2007b) are purely data-driven, and do not use a grammar at all.

1.2.2 Robustness in grammar-driven parsers

One of the biggest difficulties that arise when using grammar-driven approaches to process natural language sentences in practical domains is the problem of achieving *robustness*. Robustness can be defined as the capacity of a parser to analyse any input sentence (Nivre, 2006b), and it is a highly desirable property for any system dealing with unrestricted natural language text. The problem is that, when a parser that employs a formal grammar is used to analyse unrestricted text, it is likely to find sentences that do not belong to the formal language defined by the grammar. This can happen because of several reasons, including insufficient coverage (the sentence is syntactically correct, but our grammar is not detailed enough to recognise it) or errors in the sentence itself (not all the sentences in real texts are syntactically correct).

The methods that have been proposed to solve this problem fall mainly into two broad categories: those which try to parse well-formed fragments of the input when an analysis for the complete sentence cannot be found – partial parsers, like the one by Kasper *et al.* (1999) – and those that try to provide a complete parse for the input sentence by relaxing the constraints

imposed by the grammar. One of the approaches to achieve the latter is that of *error-repair parsers* (McKenzie *et al.*, 1995), which are algorithms that are able to find complete parse trees for sentences not covered by their grammar, by supposing that ungrammatical strings are versions of valid strings in which there are a number of errors. Under this assumption, the generation of an analysis for an ungrammatical sentence is based on the analysis of the grammatical sentence that produced it under this error model.

1.2.3 Parsing schemata

The formalism of *parsing schemata* (Sikkel, 1997) is a theoretical framework that can be used to describe different parsers in a simple and uniform way. Parsing schemata are based on the idea of considering the parsing process as a deduction process, which proceeds by generating intermediate results called *items*. This is similar to the view of parsing as deduction by Shieber *et al.* (1995), with the difference that Sikkel's framework formally defines items and related concepts, providing a mathematical basis that can be used to reason about formal properties of parsers. In particular, items in parsing schemata are sets of partial constituency trees that are licensed by the rules of a given grammar. Note that this means that parsing schemata are only applicable to grammar-based constituency parsers, and not to data-driven or dependency-based parsers. In fact, Sikkel's examples were focused in CFG parsers only, but parsing schemata have subsequently been used to describe TAG parsers as well (Alonso *et al.*, 1999, 2004).

In order to describe a parser by means of a parsing schema, we need to provide the following:

- A set of *initial items*, or *hypotheses*, which are obtained directly from each input sentence.
- A set of inference rules called *deduction steps*, that can be used to derive new items from existing ones. A parsing schema specifies these inference rules as a set, but makes no claim about the order in which they are to be executed. The set of deduction steps in a schema is given as a function of the rules in our grammar, so that the same parsing strategy will produce different results with different grammars, as expected.
- A set of *final items*, which are items that contain a full parse tree for the input or allow us to obtain it.

A given sentence belongs to the language defined by a grammar if a final item can be obtained from the initial items by some sequence of inferences using deduction steps. A generic deductive procedure as the one described by Shieber *et al.* (1995) can be used to try all the possible inferences in order to find final items, and the sequences of inferences that produce final items can be used to recover parse forests (Billot and Lang, 1989).

Items in parsing schemata are formally defined as sets of partial constituency trees, members of the quotient set of some equivalence relation between trees. Additionally, items can be augmented with additional information. For example, probabilities can be included in items to describe statistical parsers, or feature structures to describe unification-based parsers.

Parsing schemata are a useful theoretical tool to describe and study parsers for several reasons:

- They provide a simple and declarative way to represent parsing algorithms. A representation of a parser by means of a parsing schema is typically much more compact than, for example, a pseudocode representation. Properties of parsers such as their time and space computational complexities are easy to infer from their schemata.
- They represent parsers in a uniform way, and allow us to find and prove relationships between different algorithms. The mathematical framework provided by parsing schemata can also be used to prove the correctness of parsers, and proofs of correctness can be transferred between some parsers by using the formal relationships between them.
- They are at a high abstraction level, so that they focus on the semantics of the parser (what the parser does) and abstract away from implementation details (the particular data and control structures used by the algorithm). Note that this means that there is not a direct correspondence between parsing schemata and algorithms: a schema describes a parsing strategy that can sometimes be realised by different algorithms.

1.3 Outline of the book

The fundamental goal of this book is to provide theoretical and practical tools to extend the applicability of parsing schemata in practical natural language text parsing. In what follows, we list the main research

contributions presented in this work, and then outline the contents of each chapter of the book.

1.3.1 Contributions

- (1) A compiler is presented that automatically converts parsing schemata into efficient executable implementations of their corresponding parsing algorithms. The system is tested by compiling schemata for several well-known CFG parsers and running them on grammars from real-life corpora.
- (2) The parsing schemata compiler is used to conduct an empirical study of the performance of several TAG parsing strategies on a real-life, feature-based LTAG (Lexicalised Tree Adjoining Grammar): the XTAG English Grammar (XTAG Research Group, 2001). Note that previous comparative studies of TAG parsers in the literature were done with “toy” grammars, but there were no such studies with real-life, wide-coverage grammars. Additionally, the performance of TAG parsers is also compared to that of CFG parsers, and the empirical overhead introduced when using TAGs to parse context-free languages is quantified.
- (3) A theoretical extension of parsing schemata is defined to allow robust parsers under the error-repair paradigm to be described with schemata.
- (4) A transformation is defined that can be used to automatically add error-repair capabilities to the schemata of parsers that do not have them. By using this transformation, robust parsers and their implementations can be obtained automatically from non-robust parsing schemata.
- (5) A variant of the parsing schemata formalism is defined for dependency-based parsing, and used to describe and formally relate several well-known projective and non-projective dependency parsers in the literature. Parsing schemata are also given for the formalism of link grammar.
- (6) Novel parsing algorithms are defined for different sets of mildly non-projective dependency structures, including a new set of *mildly ill-nested* structures that includes all the sentences present in a number of dependency treebanks.

1.3.2 Structure of the book

This book is structured in five parts. The first part is introductory, containing this chapter which summarises the main goals and contributions of

the monograph, and a chapter defining the formalism of parsing schemata, that will be used throughout the book. The second part presents a compiler for parsing schemata and several empirical studies of constituency-based parsers conducted with it. The third part introduces an extension of parsing schemata that can be used to describe error-repair parsers. The fourth part is devoted to a variant of schemata for dependency-based parsers. Finally, the fifth part summarises conclusions and discusses future work.

A chapter-by-chapter breakdown of the parts follows:

1.3.2.1 *Part 1*

CHAPTER 2 formally introduces the framework of parsing schemata, which will be used throughout the book. The formalism is described here as defined by Sikkell (1997), serving as a starting point for the novel extensions presented in subsequent chapters. The definitions are complemented by concrete examples based on the Cocke-Younger-Kasami (CYK; Kasami, 1965; Younger, 1967) and Earley (1970) parsing algorithms, which will also be used recurrently through the rest of the chapters.

1.3.2.2 *Part 2*

CHAPTER 3 presents a compiler able to automatically transform parsing schemata into efficient Java implementations of their corresponding algorithms. The input to this system is a simple representation of a schema, practically coincident with the formal notation commonly used to denote them, as described in Chapter 2. The system performs an analysis of the deduction steps in the input schema in order to determine the best data structures and indexes to use, ensuring that the generated implementations are efficient. The system described is general enough to handle all kinds of schemata for different grammar formalisms, and it provides an extensibility mechanism allowing the user to define custom notational elements.

In CHAPTER 4, the compiler presented in Chapter 3 is used to generate implementations of three well-known CFG parsing algorithms and compare their empirical performance on several grammars taken from real-life corpora. These results show how different parsing algorithms are better suited to different grammars. Implementations are also generated for four TAG parsing algorithms, and used to analyse sentences with the XTAG grammar, a real-life, wide-coverage feature-based TAG. In order to be able to generate XTAG parsers, some transformations are made to the grammar, and TAG parsing schemata are extended with feature structure unification

support and a simple tree filtering mechanism. The data obtained is used to compare the empirical performance of these algorithms, being the first comparison of TAG parsers on a large-scale, wide-coverage grammar: previously to this work, existing comparisons were limited to “toy” grammars with a small number of rules. The results obtained from CFG and TAG parsers on real-life grammars are then complemented by studies performed on artificially-generated grammars, which allow us to evaluate the influence of string and grammar size in empirical computational complexity, as well as to the overhead caused by using TAG to parse context-free languages.

1.3.2.3 *Part 3*

CHAPTER 5 introduces error-repair parsing schemata: an extension of parsing schemata which can be used to define parsers that can robustly handle sentences with errors or inconsistencies. As the original theory of parsing schemata is based on the assumption that items are sets of partial constituency trees that follow the rules of a given grammar, the underlying concepts behind schemata have to be redefined in order to support these robust parsers. This extension of the parsing schema framework allows us to describe and compare error-repair parsers in a simple and uniform way, and provides a formal basis for proving their correctness and other properties. In addition, this framework is used to develop a generic technique for obtaining efficient error-correcting parsers based on regional error repair, and empirical performance results are provided.

In CHAPTER 6, the framework of error-repair parsing schemata is used to define a general transformation technique to automatically obtain robust, error-repair parsers from standard non-robust parsers. If our method is applied to a correct parsing schema verifying certain conditions, the resulting error-repair parsing schema is guaranteed to be correct. The required conditions are weak enough to be fulfilled by a wide variety of popular parsers used in natural language processing, such as CYK, Earley and Left-Corner. The schemata obtained from the transformation can be implemented with global, regional or local error-repair techniques, so that we may choose to obtain more efficient robust parsers by sacrificing the guarantee of obtaining all the optimal solutions.

1.3.2.4 *Part 4*

In CHAPTER 7, a variant of parsing schemata is defined that can be used to describe, analyse and compare dependency parsing algorithms. This

extension is used to establish clear relations between several existing projective dependency parsers and prove their correctness. Parsing schemata for non-projective dependency parsers are also presented. A variant of the formalism to represent parsers based on link grammar (Sleator and Temperley, 1991, 1993) is also shown, including examples of how some existing dependency parsers can be adapted to produce parsers for link grammar.

In CHAPTER 8, parsing schemata are used to solve the problem of efficiently parsing mildly non-projective dependency structures. Polynomial time parsing algorithms are presented for various mildly non-projective dependency formalisms, including well-nested structures with gap degree bounded by a constant k , and a new class of *mildly ill-nested* structures for gap degree k . The latter class includes all the gap degree k structures in a number of dependency treebanks.

1.3.2.5 Part 5

Finally, CHAPTER 9 contains a summary of the main conclusions derived from the research described in this book, as well as a discussion of possible lines for future research.

Note that the material in Part 1 (especially Chapter 2) is required to understand the rest of the parts, since it introduces notation that will be used throughout the book. Parts 2, 3 and 4 can be read or skipped independently of each other, but the material in the first chapter of each of these parts is required to understand the rest of the part.