Automatic Generation of Natural Language Parsers from Declarative Specifications[1]

Carlos Gómez-Rodríguez [a], Jesús Vilares [b] and Miguel A. Alonso [b]

[a] *Escuela Superior de Ingeniería Informática, Universidade de Vigo (Spain)*
e-mail: cgomezr@uvigo.es
[b] *Departamento de Computación, Universidade da Coruña (Spain)*
e-mail: {jvilares, alonso}@udc.es

1. Introduction

Parsing schemata, described in [2], provide a formal, simple and uniform way to describe, analyze and compare different parsing algorithms. 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 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. Almost all known parsing algorithms may be described by a parsing schema.

Parsing schemata are located at a higher abstraction level than algorithms. A schema specifies the steps that must be executed and the intermediate results that must be obtained in order to parse a given string, but it makes no claim about the order in which to execute the steps or the data structures to use for storing the results.

Their abstraction of low-level details makes parsing schemata very useful, allowing us to define and study parsers in a simple and straightforward way. However, when we want to actually test a parser by running it on a computer, we need to implement it in a programming language, so we have to abandon the high level of abstraction and worry about implementation details that were irrelevant at the schema level.

The technique presented in this paper automates this task, by compiling parsing schemata to Java language implementations of their corresponding parsers. The input to the compiler is a simple and declarative representation of a parsing schema, and the output is an efficient executable implementation of its associated parsing algorithm.

[1] Partially supported by Ministerio de Educación y Ciencia and FEDER (Grants TIN2004-07246-C03-01, TIN2004-07246-C03-02), Xunta de Galicia (Grants PGIDIT05PXIC30501PN, PGIDIT05PXIC10501PN and PGIDIT05SIN044E), and Programa de becas FPU (Ministerio de Educación y Ciencia).

2. From declarative descriptions to program code

These are the fundamental ideas behind our compilation process for parsing schemata:

- Each deductive step is compiled to a class containing code to match and search for antecedent items and produce the corresponding conclusions from the consequent.
- The step classes are coordinated by a deductive parsing engine, as the one described in [1]. This algorithm ensures a sound and complete deduction process, guaranteeing that all items that can be generated from the initial items will be obtained.
- In order to attain efficiency, an automatic analysis of the schema is performed in order to create indexes allowing fast access to items. As each different parsing schema needs to perform different searches for antecedent items, the index structures we generate are schema-specific. In this way, we guarantee constant-time access to items so that the computational complexity of our generated implementations is never above the theoretical complexity of the parsers.
- Since parsing schemata have an open notation, for any mathematical object can potentially appear inside items, the system includes an extensibility mechanism which can be used to define new kinds of objects to use in schemata.

3. Experimental results

We have used our technique to generate implementations of three popular parsing algorithms for context-free grammars: CYK, Earley and Left-Corner[1], and tested all of them with sentences from three different natural language grammars from real corpora: Susanne, Alvey and Deltra. Since we are interested in measuring and comparing the performance of the parsers, not the coverage of the grammars; we have generated random input sentences of different lengths for each of these grammars.

The obtained performance measurements show that the empirical computational complexity of the three algorithms is always below their theoretical worst-case complexity of $O(n^3)$, where n denotes the length of the input string. This empirical complexity is achieved thanks to the automatic indexing techniques used by the code generator, which guarantee constant-time access to items. Our results also show that not all algorithms are equally suitable for all grammars. CYK is the fastest algorithm for larger grammars thanks to its lower computational complexity with respect to grammar size when compared to Earley and Left-Corner; and the efficiency difference between the latter two heavily depends on the way the grammar has been designed. The compilation technique described in this paper is useful to prototype different natural language parsers and easily see which one is better suited for a given application.

References

[1] Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36, July-August 1995.

[2] Klaas Sikkel. *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, 1997.

[1] However, we must remark that we are not limited to working with context-free grammars, since parsing schemata can be used to represent parsers for other grammar formalisms as well.