THE ROLE OF SYNTAX IN INFORMATION EXTRACTION

Ralph Grishman
Computer Science Department
New York University
715 Broadway, 7th Floor
New York, NY 10003
grishman@cs.nyu.edu

INTRODUCTION

Our group at New York University has developed a number of information extraction systems over the past decade. In particular, we have been participants in the Message Understanding Conferences (MUCs) since MUC-1. During this time, while experimenting with many aspects of system design, we have retained a basic approach in which information extraction involves a phase of full syntactic analysis, followed by a semantic analysis of the syntactic structure [2]. Because we have a good, broad-coverage English grammar and a moderately effective method for recovering from parse failures, this approach held us in fairly good stead.

However, we have recently found ourselves at a disadvantage with respect to groups which performed more local pattern matching, in three regards:

1. **our systems were quite slow**

   In processing the language as a whole, our system is operating with only relatively weak semantic preferences. As a result, the process of building a global syntactic analysis involves a large and relatively unconstrained search space and is consequently quite expensive. In contrast, pattern matching systems assemble structure "bottom-up" and only in the face of compelling syntactic or semantic evidence, in a (nearly) deterministic manner.

   Speed was particularly an issue for MUC-6 because of the relatively short time frame (1 month for training). With a slow system, which can analyze only a few sentences per minute, it is possible to perform only one or at best two runs per day over the full training corpus, severely limiting debugging.

2. **global parsing considerations sometimes led to local errors**

   Our system was designed to generate a full sentence parse if at all possible. If not, it attempted a parse covering the largest substring of the sentence which it could. This global goal sometimes led to incorrect local choices of analyses; an analyzer which trusted local decisions could in many cases have done better.

3. **adding syntactic constructs needed for a new scenario was hard**

   Having a broad-coverage, linguistically-principled grammar meant that relatively few additions were needed when moving to a new scenario. However, when specialized constructs did have to be added, the task was relatively difficult, since these constructs had to be integrated into a large and quite complex grammar.

We considered carefully whether these difficulties might be readily overcome using an approach which was still based on a comprehensive syntactic grammar. It appeared plausible, although not certain, that problems (1) and (2) could be overcome within such an approach, by adopting a strategy of *conservative* parsing. A conservative parser would perform a reduction only if there was strong (usually, local) syntactic evidence or strong semantic support. In particular, chunking parsers, which built up small chunks using syntactic criteria and then assembled larger structures only if they were semantically licensed, might provide a suitable candidate.

In any case, problem (3) still loomed. Our Holy Grail, like that of many groups, is to eventually get the computational linguist out of the loop in adapting an information extraction system for a new scenario. This will be difficult, however, if the scenario requires the addition of some grammatical construct, albeit minor. It would require us to organize the grammar in such a way that limited additions could be made

by non-specialists without having to understand the entire grammar — again, not a simple task.

In order to better understand the proper role of syntax analysis, we decided to participate in the most recent MUC, MUC-6 (held in the fall of 1995), using a quite different approach, often referred to as "pattern matching", which has become increasingly popular among information extraction groups. In particular, we carefully studied the FASTUS system of Hobbs et al. [1], who have clearly and eloquently set forth the advantages of this approach. This approach can be viewed as a form of conservative parsing, although the high-level structures which are created are not explicitly syntactic.

THE SYSTEM

The goal of information extraction is to analyze a text (an article / a message) and to fill a template with information about a specified type of event. In the case of MUC-6, the task (the "scenario") was to identify instances of executives being hired or fired from corporations.[1]

Most of the stages of processing are performed one sentence at a time. First, each word in a sentence is looked up in a large English dictionary, Comlex Syntax, which provides syntactic information about each word. The system then performs several stages of pattern matching. The first stages deal primarily with name recognition — people's names, organization names, geographic names, and names of executive positions ("Executive Vice President for Recall and Precision"). The next stages deal with noun and verb groups. Basically, a *noun group* consists of a noun and its left modifiers: "the first five paragraphs", "the yellow brick road"; such groupings can generally be identified from syntactic information alone. A *verb group* consists of a verb and its related auxiliaries: "sleeps", "is sleeping", "has been sleeping", etc. All of these stages are basically scenario-independent (except for the recognition of executive positions, which was added for this scenario).

Next come the scenario-specific patterns. These include, in particular, patterns to recognize the scenario events, such as "Smith became president of General Motors", "Smith retired as president of General Motors", and "Smith succeeded Jones as president of General Motors". When such a pattern is matched, a corresponding event structure is generated, recording the type of event (for this scenario, hiring or firing) and the people and companies involved.

The next stage of processing is reference resolution. At this stage, pronouns and definite noun phrases which refer back to previously mentioned people or organizations are linked to these antecedents.

When all the sentences of an article have been analyzed, a final stage of processing assembles the information and generates a template in the format required for MUC.

The resulting system did quite well. With a limited development time (four weeks for this MUC) we were able to develop a system which obtained a recall of 47% and a precision of 70% (with a combined F measure of 56.4) on the test corpus. This was the best F score on the scenario template task, although several other systems (mostly with similar architectures) got scores that were not significantly lower.

THE ROLE OF SYNTAX

Although our system, and systems like it, are characterized as "pattern matching" systems, they really are doing a form of parsing: they analyze the sentence into a nested constituent structure. They differ from more conventional parsing systems (such as our earlier system) in

- not seeking a full-sentence analysis: they only build as much structure as is needed for the information extraction task, including selected clauses relevant to the scenario
- parsing conservatively and deterministically: they only build structures which have a high chance of being correct, either because of syntactic clues (for noun groups) or semantic clues (for clause structures); as a result, they are much faster than traditional parsers
- using semantic patterns for the final stage(s) of analysis

Overall, we profited from the use of the pattern-matching approach; our analyzer was considerably faster, and we avoided some of the parsing errors which result from trying to obtain complete sentence analyses with a syntactic grammar. On the other hand, we also experienced first-hand some of the shortcomings of the semantic pattern approach. Syntax analysis provides two main benefits: it provides generalizations of linguistic structure across different semantic relations (for example, that the structure of a main clause is basically the same whether the verb is

[1] For a description of MUC-6, see the papers "Design of the MUC-6 Evaluation" and "Overview of the Results of the MUC-6 Evaluation" in this volume; for a more detailed description of the NYU system, see our paper "The NYU System for MUC-6 or Where's the Syntax?" in *Proc. of the Sixth Message Understanding Conference*, Morgan Kaufmann, 1996.

"to succeed" or "to fire"), and it captures paraphrastic relations between different syntactic structures (for example, between "X succeeds Y", "Y was succeeded by X", and "Y, who succeeded X"). These benefits are lost when we encode individual semantic structures. In particular, in our system, we had to separately encode the active, passive, relative, reduced relative, etc. patterns for each semantic structure. These issues are hardly new; they have been well known at least since the syntactic grammar vs. semantic grammar controversies of the 1970's.

How, then, to gain the benefits of clause-level syntax within the context of a partial parsing system? The approach we have adopted has been to introduce *clause level patterns* which are expanded by *metarules.*[2]

As a simple example of a clause-level pattern, consider

```
(defclausepattern runs
    "np-sem(C-person) vg(C-run)
             np-sem(C-company):
     person-at=1.attributes,
     verb-at=2.attributes,
     company-at=3.attributes"
    when-run)
```

This specifies a clause with a subject of class *C-person*, a verb of class *C-run* (which includes "run" and "head"), and an object of class *C-company.*[3] This is expanded into patterns for the active clause ("Fred runs IBM"), the passive clause ("IBM is run by Fred."), relative clauses ("Fred, who runs IBM, ..." and "IBM, which is headed by Fred, ..."), reduced relative clauses ("IBM, headed by Fred, ...") and conjoined verb phrases ("... and runs IBM", "and is run by Fred"). The expanded patterns also include pattern elements for sentence modifiers, so that they can analyze sentences such as "Fred, who last year ran IBM, ...".

Using `defclausepattern` reduced the number of patterns required and, at the same time, slightly improved coverage because — when we had been expanding patterns by hand — we had not included all expansions in all cases.

The `defclausepattern` procedure performs a rudimentary syntactic analysis of the input. In our example, it determines that np-sem(C-person) is the subject, vg(C-run) is the verb, and np-sem(C-company) is the object. This is a prerequisite for generating the various restructurings, such as the passive. We intend in the near future to expand `defclausepattern` to handle (parse) a richer set of patterns, including both sentence modifiers and a wider range of complements. In this way the power of clause-level syntax is provided to the pattern writer, without requiring the pattern writer to keep these details explicitly in mind.

The use of clause-level syntax to generate syntactic variants of a semantic pattern is even more important if we look ahead to the time when such patterns will be entered by users rather than computational linguists. We can expect a computational linguist to consider all syntactic variants, although it may be a small burden; we cannot expect the same of a typical user.

We expect that users would enter patterns by example, and would answer queries to create variants of the initial pattern. We have just begun to create such an interface, which allows a user to begin the process of pattern creation by entering an example and the correspoding event structure to be generated. The example is analyzed using the low-level patterns (such as the name and noun group patterns) and then translated into a clause-level pattern. The user can then manipulate the pattern, generalizing pattern elements and dropping some pattern elements. Using `defclausepattern`, the resulting pattern is then analyzed and its clause-level syntactic variants are generated. Though our initial tests are promising, a great deal of work will still be required on this interface to provide the full flexibility needed for creating a wide range of patterns.

Our work has indicated the ways in which we can continue to obtain the benefits of syntax analysis along with the performance benefits of the pattern matching approach. While we no longer have a monolithic grammar, we are still able to take advantage of the syntactic regularities of both noun phrases and clauses. Noun group syntax remains explicit, as one phase of pattern matching. Clause syntax is now utilized in the metarules for defining patterns and in the rules which analyze example sentences to produce patterns.

Acknowledgements

The development of our language analysis software and our participation in the MUCs has been supported by the Advanced Research Projects Agency under a series of contracts. This work is currently supported under Contract 94-FI57900-000 from the

[2] These have some kinship to the metarules of GPSG, which expand a small set of productions into a larger set involving the different clause-level structures.

[3] It also specifies that the attributes of these three constituents are to be bound to the variables `person-at`, `verb-at`, and `company-at`, and that the procedure `when-run` is to be invoked when this pattern is matched.

Office of Research and Development and under Contract DABT63-93-C-0058 from the Department of the Army.

References

[1] D. Appelt, J. Hobbs, J. Bear, D. Israel, M. Kameyama, and M. Tyson. SRI: Description of the JV-FASTUS System used for MUC-5. In *Proc. Fifth Message Understanding Conf. (MUC-5)*, Baltimore, MD, August 1993. Morgan Kaufmann.

[2] Ralph Grishman and John Sterling. New York University: Description of the PROTEUS System as used for MUC-5. In *Proc. Fifth Message Understanding Conf. (MUC-5)*, Baltimore, MD, August 1993. Morgan Kaufmann.