GUIDES AND ORACLES FOR LINEAR-TIME PARSING

Martin Kay

Stanford University and Xerox Palo Alto Research Center

kay@parc.xerox.com

Abstract

If chart parsing is taken to include the process of reading out solutions one by one, then it has cxponential complexity. The stratagem of separating read-out from chart construction can also be applied to other kinds of parser, in particular, to left-corner parsers that use early composition. When a limit is placed on the size of the stack in such a parser, it becomes context-free equivalent. However, it is not practical to profit directly from this observation because of the large state sets that are involved in otherwise ordinary situations. It may be possible to overcome these problems by means of a *guide* constructed from a weakened version of the initial grammar.

A recognition procedure for a language is a method of determining whether a given string belongs to the language. In the context-free case, it clearly reduces to showing that the string is a phrase of a particular category, the goal category of the grammar. A string α belongs to category C if either, α consists of the single symbol C, or there is a rule $C \longrightarrow c_1 \ldots c_n$ and α is the concatenation of strings that are phrases of categories $c_1 \ldots c_n$, in that order. The proof that a string is a phrase of a given category can be summarized in an ordered tree with nodes named for grammar symbols and this is what we refer to as the structure of the string according to the grammar. The root is named for the grammar's distinguished symbol and the daughters of a node labeled C are labeled, from left to right, $c_1 \ldots c_n$, given that $C \longrightarrow c_1 \ldots c_n$ is a grammar rule.

Concretely, a rule $s \longrightarrow np\ aux\ vp$ can be transcribed directly into Prolog as a *definite-clause grammar* (DCG) somewhat as follows:

```
wof(s, A, D) :-
  wof(np, A, B),
  wof(aux, B, C),
  wof(vp, C, D).
```

There is a word or phrase (wof) of category s stretching from point A to point D in the string if, for some points B and C between A and D, there is a phrase of category np from A to B, of category aux from B to C, and of category vp from C to D. A terminal symbol, say *dog,* is recognized as belonging to category n by virtue of the clause:

```
wof(dog, [dog | X], X).
```

This is based on the convention of using suffixes of the string as names of points in it. In particular, this clause says that a string consists of a noun followed by a string X if it consists of the word *dog* followed by X.

With these, and few more obvious definitions, the Prolog interpreter will be able to prove the proposition

```
wof(s, [the, dog, will, chase, the cat], [])
```

The proof will be carried out in accordance with the so-called *recursive-descent*, or *top-down backtracking* strategy suggested by our initial definitions. In order to show that a string is a word or phrase of category s, the procedure is to first show that it begins with a phrase of category np, and then to show that the remainder of the string consists of the *aux* phrase followed by a vp. Each of these steps consists of a recursive application of the same procedure.

To get the structure of a string, one must arrange to capture the control structure of the recognition process, and this can be done in a variety of ways. To capture all the possible structures of a string, it is necessary to behave on success just as one does on failure, by backing up to the most recent choice point with hitherto unexplored branches.

As an effective recognition or parsing algorithm, the flaws of DCG are well known. The two principal ones are (1) that the assymptotic time complexity is exponential in the length of the string, and (2) that the procedure does not terminate when a phrase of a given category can have an initial substring that must be analyzed as belonging to that same category.

The information that the recognition procedure amasses about the a string can be summarized in the manner exemplified below:

```
oracle(s, [the, dog, will, chase, the, cat], []).
oracle(np, [the, dog], [will, chase, the, cat]).
oracle(det, [the], [dog, will, chase, the, cat]).
oracle(vp, [will, chase, the, cat], []).
...
```

Suppose that the clauses embodying the grammar are augmented as follows.

```
wof(s, A, D) :-
  oracle(np, A, B),
  wof(np, A, B),
  oracle(aux, B, C),
  wof(aux, B, C),
  oracle(vp, C, D)
  wof(vp, C, D).
```

With this augmented grammar and the oracle, the process of recognition is completely trivialized—in fact the first `oracle` clause is all we need for recognition. Since the oracle does not provide structures, however, the control structure of the recursive-decent analysis process must still be recorded if it is required to parse the string. Notice, however, that the existence of such an oracle would eliminate one of the problems with recursive-decent analysis, namely failure to terminate in cases of left recursion, and it alleviates the other by removing from the search space all moves that do not belong to successful paths. It is this last property that motivates the use of the term "oracle".

The interest in recasting top-down syntactic analysis in this way comes from the analogy that can be drawn to chart parsing. The oracle is essentially a chart and the `wof` grammar predicate supplies the read-out procedure. It is usual to include more information in the chart so that the read-out procedure does not have to have information from the grammar rules. In the present formulation, edges contain no information about the members of a phrase, so that polynomial complexity is achieved automatically without having to conflate edges with the same category symbol and string coverage, but different internal structures.

Computational linguists are generally comfortable with the claim that chart parsing with a context-free grammar has polynomial assymptotic time complexity. Since a context-free grammar can assign a number of structures to a string that increases as an exponential function of its length, we assume that there is a tacit agreement not to count the read-out procedure, but only the process of building the chart that will serve as an oracle for the read-out procedure. How this is justified in detail is not clear. Intuitively, however, dividing the parsing process into a first stage in which a data base of grammatical information about the string is constructed and a second stage in which individual analyses are read out has the advantage of allowing different parsing algorithms to be seen as idffering in complexity on the basis of the first, and intuitively more interesting part of the process.

In the balance of this paper, we outline a parsing algorithm that is very different from chart parsing in its details, but similar in that it proceeds in two stages, one in which a data base is constructed at quite attractive cost in complexity, and one in which individual analyses are read out by a simple, oracle-driven, backtracking parser. It will turn out, however, that the approach can form the basis of a practical parser only if the influence of grammar size on the overall process can be brought under control. For this purpose, we introduce the notion of a *guide*, which is a weak form of an oracle. If an oracle is available at a particular branch in a process, it can be counted on to eliminate all choices that do not lead to a successful outcome. A guide will, in general, not eliminate all unproductive choices, but it can be counted on not to eliminate any choices that do could lead to a successful outcome. As an example of a guide, consider the weakened form of chart represented in the following clauses:

```
guide([the, dog, will, chase, the, cat], []).
guide([the, dog], [will, chase, the, cat]).
guide([the], [dog, will, chase, the, cat]).
guide([will, chase, the, cat], []).
  ...
```

and a read-out procedure based on clauses like the following:

```
wof(s, A, D) :-
  guide(A, B),
  wof(np, A, B),
  guide(B, C),
  wof(aux, B, C),
  guide(C, D)
   wof(vp, C, D).
```
This chart shows where there are phrases in the string, but does not give their grammatical category. It is sufficient, however, to eliminate problems arising from left-recursive grammars.

The scheme we will outline provides analyses of strings in linear time with a context-free grammar. There is good reason to believe that this is not possible if all the structures allowed by the grammar are to be recovered, and our scheme will indeed ignore certain structures. However, there is also good reason to believe that the structures that we shall ignore are also not accessible to humans and, if this is the case, then nothing but good can come from leaving them out of account.

Abney and Johnson (1989) have shown that a left-corner parser with early composition uses stack space in proportion to the amount of center embedding in the structure. Such a parser is clearly also equivalent to a finite-state automaton which can recognize a string in linear time. One problem with this is that a finite-state automaton can serve only as a recognizer, and not as a parser. However, a recognizer and can serve as an oracle for parser[1]. The idea is simply to scan the string to be parsed from right to left, using a finite-state automaton that recognizes the reverse of the depth-limited version of the context-free language and to associate with the space between each pair of words the state of the machines. When read from left to right, this sequence of states serves as an oracle for the left-corner parser with the early composition and a finite stack.

Unfortunately, even for modest sized grammars, and an early limit on stack size, the numbers states in the automaton is unmanageably large so that it cannot be represented explicitly and therefore cannot be made deterministic. But, while undoubtedly a setback, this does not entirely upset the plan. Recognition with a nondeterministic automaton is possible in linear time if an appropriate strategy is employed for caching the states reached following a given substring. This follows from the fact that the number of alternative states that the automaton can be in after a given sub-string is limited by properties of the automaton and not by the length of string. However, even this is not enough to bring the cost of the computation within reasonable bounds because the number of stacks configurations that are possible following even a fairly short string can also be unmanageably large.

The idea of providing an oracle to control the construction of the sequence of state sets that will, in its turn, serve as an oracle in reading out the structures of the string suggests itself, but it is difficult to see how such an oracle would differ from the structure it is intended to help assemble. The intuition is that a useful oracle must contain only a part of the information in the structure whose assembly it controls. However, the possibility of a guide may be more promising. The idea will be to construct a weaker version of the context free grammar, which assigns to any given string a superset to of the structures that are signed by the original grammar, but which gives rises to an automaton with a smaller set of states. These states map in a systematic way onto those of the original automaton and, when this is applied to the string, the only states that will be considered at a given point will be those corresponding to states through which the smaller automaton has passed.

A simple way to construct a weakend version of a grammar is to construct a partition that symbols and to map each class in the partition onto a single symbol. The rules of the weekend grammar are simply the images are under this mapping of the rules in the original grammar. The new grammar will be weeker to the extent that it derives from a smaller number of a larger classes . Since the total number of symbols in the weakened grammar is smaller than that in the original grammar, so is the number of possible stack configurations.

The picture we now have is of a parser that proceeds in three phases. First, it scans the string from left to right using a left corner recognizer based on the weakened grammar, annotating the spaces between the words with the sets of states that the automaton is in at that point. Next, it scans the string from right to left using the left-corner recognizer based on the full grammar and allowing states to be entered only if they map onto members of the list of states associated with that point in the string in the preceding phase. The reason for the reversal of direction is simply to ensure that the states on each list that is encountered are reachable from the other end of the string, thus providing a guide for the present scan that is, to the extent possible, predictive. Following the practice in chart parsing, we declare these first two phases to constitute a parser and

1. This is reminiscent of the way the first member of a bimachine (Schützenberger, 1961) is used to control the operation of the second member.

declare its assymptotic in complexity to be linear . The third phase reads out structures using the original context free grammar and the left-corner parser with the early composition whose stack states must be chosen from those associated with the string in the second phase.

One apparently minor matter remains, namely how to construct a weakened version of a particular grammar that will serve as an effective guide, in this process. Surprisingly, this proves to be the sticking point. One possibility would be simply to construct the partition of the grammar symbols in a random fashion. Another would be to eliminate the distinctions made by X-bar theory, collapsing, for example, N, N-bar, and NP onto the same symbol. Yet another would be to eliminate "minor" grammatical matters , such as agreement from the first scan. The disturbing fact is that none of these things can be counted on to give a weakened grammar with desirable properties. Either the grammar remains essentially unchanged, or it reduces to one that accepts almost everything. Minor changes can easily cause it to move, almost chaotically, from one of these conditions to the other. I offer this as a challenge to the parsing community.

References

Abney, S. P. and M. Johnson (1989). “Memory requirements and local ambiguities of parsing strategies.” *Journal of Psycholinguistic Research* 18(1): 129-144.

Schützenberger, Marcel Paul. 1961. A remark on finite transducers. *Information and Control*, 4. pp. 185-187.