Chapter 9

Parsing Strategies

1. Introduction

Sentence analysis is fundamental to most NLP applications -- it is an essential component of, for instance, natural language interfaces, translation systems, text summarisation systems, and grammar checkers. This chapter and the next describe sentence analysis, focusing on Systemic sentence analysis. This chapter outlines the basic analysis strategies (for Systemic and non-Systemic analysis), and chapter 10 describes the approach taken in the WAG analyser.

The term 'analysis' will be used to describe the construction of a description on any stratum derived from a description on a lower stratum. We thus have 'graphological analysis', where we break up an input text into graphological units (spellings, sentences etc.); lexico-grammatical analysis, whereby we produce a lexico-grammatical representation based upon a graphological analysis; and micro-semantic analysis, whereby we produce a micro-semantic representation derived from the lexico-grammatical and graphological descriptions. We will use the term *parsing* to refer specifically to lexico-grammatical analysis.

2. Parsing Control Strategies

Over the years, many different approaches have been tried to apply a set of grammar rules to the analysis of a sentence. These various approaches can be split up on a number of dimensions, each dimension concerning different priorities in the construction of a parse tree. This section will look at three such dimensions:

- Top-Down *vs.* Bottom-Up;
- Depth-First *vs.* Breadth-First;
- Node Selection Strategies.

2.1 Top-Down *vs.* Bottom-Up

A **top-down** approach starts at the root of the tree (traditionally the sentence), and builds down to the leaves of the tree (words, or in some cases, morphemes). A bottom-up approach starts with the leaves and builds upwards towards the sentence.

Palme (1981) comments on these approaches :

> "The top-down algorithm is theoretically based on the idea of using a generative grammar to produce all possible sentences in a language until one is found which fits the input sentence. This would in most cases take too much time, but there are ways to restrict the method to the most fruitful possibilities." (p208)

"The bottom-up algorithm... tries to combine elements in the input sentence in different ways until a tree covering the whole sentence is found." (p204) "[It] starts with the single words in the sentence. These are grouped together (correlated) into larger and larger units until the whole sentence is grouped together" (p208)

Left-Corner Parsing: There are some parsing algorithms which use a mixture of top-down and bottom-up strategies. The Left-Corner parsing algorithm for instance invokes a rule bottom-up and finishes it top-down. Such a parser "...builds sentence structure in a left-to-right, bottom-to-top fashion, piecing together the left corner of a structural description first." (Petrick 1989, p690).

2.2 Depth-First *vs.* Breadth-First

A *depth-first* approach moves as quickly as possible between root and leaves (in a top-down approach) or leaves and root (in a bottom-up approach). A *breadth-first* approach explores all branches at each level before going up/down a level.

Ravelli (1987) say that this is a difference between:

"whether to explore one path to its limits before trying others (for example, to determine everything necessary about one word of the input, before proceeding to the remaining words), or whether to explore all paths simultaneously (for example, to carry out the first stage of the process on all the words of the input, before proceeding to the next stage" (p21)

De Roeck (1983, p15) comments that a depth-first approach stresses the vertical dimension of a tree, breadth-first its horizontal dimension, taking into consideration all nodes at the same level in the tree.

2.3 Combinations of Strategies

The above discussion has defined two strategic alternatives in the design of a parsing algorithm. There are thus four combinations of these strategies:

- Top-Down/Depth-First
- Top-Down/Breadth-First
- Bottom-Up/Depth-First
- Bottom-Up/Breadth-First

Figure 9.1 exemplifies these strategy combinations, by showing how a parse-tree (for “the cat sat”) is built up over a series of steps. The discussion below shows how the top-down/depth-first tree is constructed. During processing there will usually be various incorrect hypotheses made -- such occurrences are ignored in this example:

1. Hypothesise a clause unit.
2. Using the resources, we hypothesise that the left-most element of the clause is the Subject role, which is filled by a nominal-group.
3. Since we are using a depth-first strategy, we choose to expand the Subject before hypothesising other elements of the clause. Again using the resources, we hypothesise that the left-most element of the nominal-group is the Deictic role, which is filled by a determiner. Since this hypothesis involves a word-rank element, we can check the hypothesis against the input string. Since the first element -- “the” -- is indeed a determiner, we mark this element as recognised, and proceed.
4. Given that the Deictic element is complete, we now proceed to hypothesise the second element of the nominal-group. The resources predict a Head role, filled by a noun. This element matches the second element of the input string.

5. The resources allow a Deictic ^ Head structure to be a complete nominal-group, so the Subject is hypothesised complete. The parser thus predicts the second element of the clause, a Pred role, filled by a verb. This item matches the third and final item on the input string.
6. Since the resources allow Subj ^ Pred to be a complete clause structure, and all items in the input string are incorporated in the structure, the parser returns the completed structure as a valid parse of the input string.

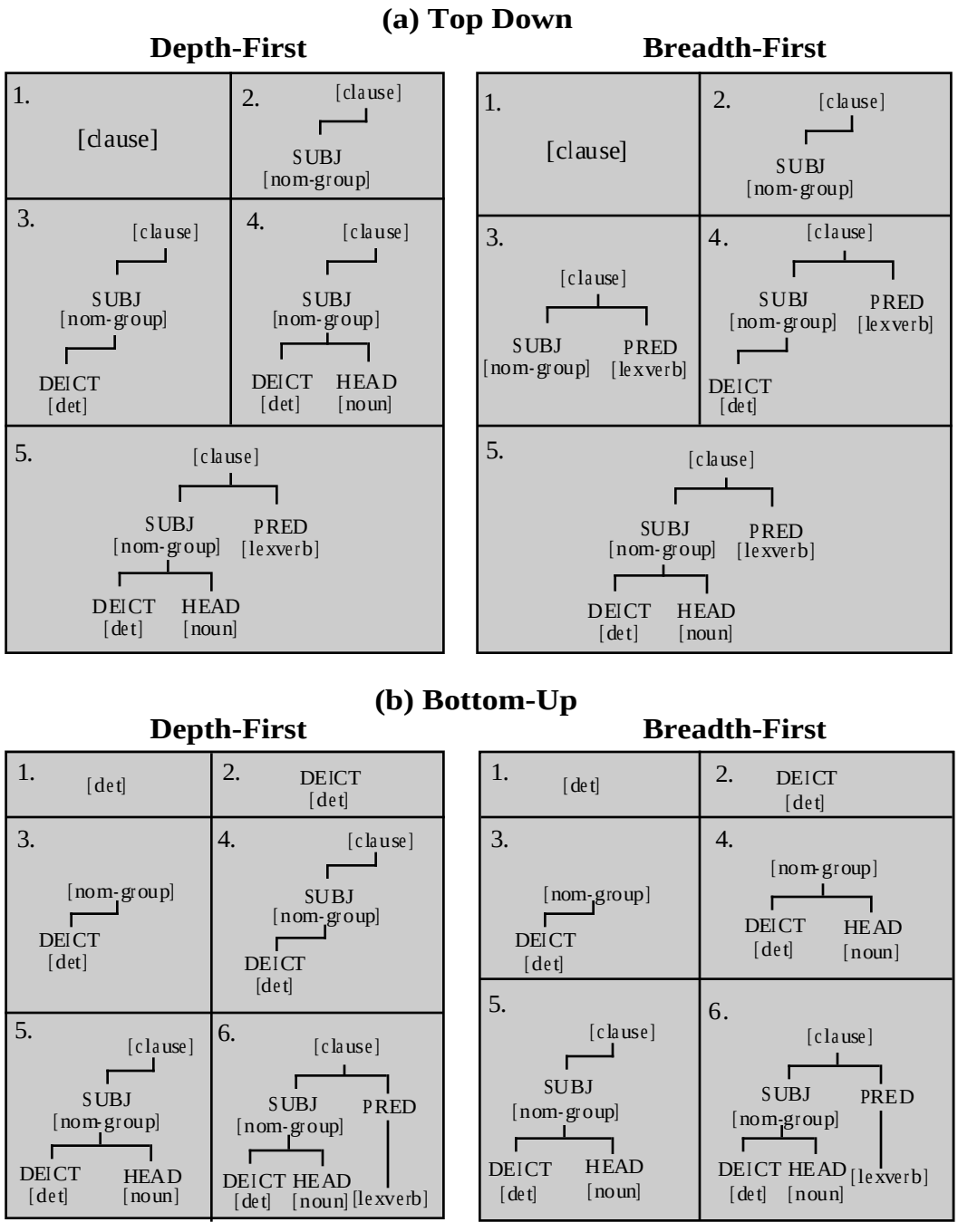

Figure 9.1: Various Parsing Strategies

2.4 Node Selection Strategies

The strategy combinations we have defined above do not limit all the possibilities in the construction of a parse tree. For instance, when parsing bottom-up and depth-first, these strategies do not say which word in the input string we should start with. We could start with the first, but this is only one possibility. When parsing top-down, any of the possible constituents of the present unit could be expanded first. A further restriction on the parsing process is needed to nominate which node of the parse tree should be expanded first. Several possible strategies exist:

- **Left-to-right Expansion**: the left-most node is expanded first. In a bottom-up parser, this means incorporating words of the input string starting from the left-most, and proceeding until the right-most is incorporated. For a top-down parser, this means expanding the left-most elements of the present unit's constituents before those to the right. Michael Covington (Internet discussion), notes that parsers can also operate in a right-to-left manner (an earlier parser of mine operated in this manner as well).
- **Head-driven Expansion**: In a head-driven (sometimes *keyword*) parser, some key item at each level of structure (the "head") is expanded first, since this unit is seen as the controller of all other units at this level. For instance, a bottom-up key-search parser would start by parsing the verb, since this item controls the presence and nature of other structural elements (e.g., a transitive verb expects a Complement, while an intransitive one does not). Nouns may be selected as a second-level key, since they control much of the structure of the nominal group. Ravelli (1987) summarises the head-driven approach:

  "first scan the input for a particular item, then begin the analysis at that item... This option needs to be motivated by a theoretical belief that the chosen item is in some way a 'key' to analysing the input. For instance, it may be felt that the particular nature of the verb will be the most important information to the parser in, say, segmenting the constituents of the clause. Thus the parsing device will begin its analysis at the verb." (p22).
- **Resource-driven Expansion**: A resource-driven approach uses the grammatical resources to direct the analysis, not the representations. The resources thus decide which node is to be expanded first. For instance, a shift-reduce parser[1] tries to successively apply all the rules of the grammar to the partially developed parse tree. The parse tree is expanded at the point at which the rule fits the tree (which could be at any point in the parse tree). The input string is thus not necessarily processed left to right (although some shift reduce parsers do work left to right, by applying all the rules to the left-most expansion point in the parse tree, before advancing to the next point).

[1] A popular bottom-up parsing strategy - see, for instance, Aho & Ullman (1972).

2.5 Evaluation of Strategies

This section evaluates the various strategies discussed above.

2.5.1 Top-Down *vs.* Bottom-up Strategies

Gough (1987, p161) notes that "the most powerful of the bottom-up parsing methods, the so-called LR(k) parsers, accept a wider class of languages than do the most powerful top-down parsers.".

Also, top-down parsing tends to be less efficient than bottom-up parsing, because a large degree of hypothesis making may be required before the input string is reached. The more complex the grammar, the higher the degree of hypothesis-making. Bottom-up algorithms start with the input sentence (data-driven), and thus tend to be more efficient - they do not need to hypothesise about what could be in the sentence.

On the other hand, Gough (1987, p161) notes that top-down methods are simpler to implement, and easier to understand. Systemic grammar, with its 'realisation' metaphor, is well-suited to top-down parsing. I implemented various top-down parsers before switching to bottom-up approaches for efficiency reasons.

There thus tends to be a trade-of between these strategies: the implementational simplicity of top-down parsers *vs*. the processing efficiency of bottom-up parsers. For large-size grammars, the lower efficiency of top-down approaches may become prohibitive.

2.5.2 Depth-First *vs.* Breadth-First Strategies

Ravelli (1987) notes that for top-down parsing, depth-first building is the preferred option, since "reaching the input as fast as possible will give the earliest indication of error in hypothesis." (p21).

For bottom-up parsing, breadth-first is usually preferred, since these approaches wait to see which other words are present, and how they are structurally related, before climbing up a level. Depth-first approaches are more rash, building up towards the sentence level based on just the first word of the sentence. A single word generally provides insufficient evidence to make decisions at higher levels. For either the top-down or bottom-up case, the best option is to see the input text as quickly as possible.

2.5.3 Node-Selection Strategies

The head-driven (keyword search) strategy can be more efficient, since it expands those nodes of the parse-tree which are most likely to constrain the structure of neighbouring units. However, this strategy requires the resources to contain a statement of what the key, or head, elements are. Some grammars already contain such information (e.g., dependency grammars). For grammars which do not automatically provide this information, a separate resource would need to be provided.

3. Dealing with Non-Determinism

During the construction of the structure tree, there will be times where the parser has a choice as to how to build the structure. This may be due to lexical ambiguity (multiple meanings of words, e.g., after seeing only the first two words of "the orange car" we don't know whether we have det^adj or det^noun); or structural ambiguity (multiple ways in which items can attach to each other, e.g., in "the guard hit the prisoner from far away", the prepositional-phrase "from far away" could modify the noun or the verb). There are several ways to handle ambiguity, and these are discussed below.

3.1 Deterministic Methods

Deterministic methods attempt to resolve ambiguity by somehow determining which of the choices will be correct before parsing further. Commonly 'look-ahead' is used, which involves using the rest of the input string to make a decision. This approach was first used successfully by Marcus (1980). See Sampson (1983) for a good summary.

Note that look-ahead can only be used to resolve local ambiguity. Where the sentence as a whole is ambiguous, no amount of look-ahead will help. Often, a deterministic parser might turn to heuristic criteria to resolve the ambiguity before preceding (see discussion in 3.4 below).

3.2 Non-deterministic Methods

A non-deterministic approach doesn't attempt to resolve the ambiguity, but rather tries all alternatives. The alternatives can be tested serially (*backtracking*), or simultaneously (*parallel parsing*):

1. **Backtracking**: At each choice point, one path is chosen and parsing continues. If, at a later stage the parse fails, then the parser *backtracks* to the last decision point and tries an alternative choice.
2. **Parallel parsing**: At each choice point all choices are proceeded with - the parser keeps track of each possible construction, and develops them all in parallel. One particularly powerful parallel parsing technique is called *chart parsing*. It will be discussed below.

3.3 Avoiding Repeated Analyses

Most non-deterministic strategies suffer from a problem: when structural ambiguity occurs within the parsing process, the system will analyse some components of the sentence more than once. This is true of both backtracking and parallel parsing techniques. For instance, given "The horse raced past the barn fell", a backtracking parser would process "The horse raced past the barn" as a clause, then fail to handle "fell". Processing would then return to the last choice point, where an alternative choice is made, and then proceed from there. The second analysis would correctly handle "The horse raced past the barn" as a single nominal group, and properly handle "fell" as the verb. However, in the second attempt, a backtracking parser would re-analyse parts of the sentence which are the same for both analyses. The various possible analyses of "past the barn" would remain the same over both passes.

A parser can use a *well-formed substring table* (WFST) to avoid much of the repetition: -- "a mechanism enabling a parser to keep a record of structures it has already found, so that it can avoid looking for them again" (Gazdar & Mellish 1989, p179). Successful parses of each sequence of words are stored away in the table. If the parser later needs to re-analyse the same sequence of words, the successful analyses can be recovered from this table, rather than re-calculated.

A *chart-table* offers one extension to this idea:

> "The further extension represented in charts enables a parser, in addition, to record information about goals it has adopted. Recorded goals may have been unsuccessful or may still be under exploration, but in either case it would be inefficient for the parser to start pursuing them again from scratch." (Gazdar & Mellish 1989, p179).

While a WFST records successful parses only, a chart table records in addition analyses still under construction. For instance, if we have so far recognised a Deictic and Numerator roles of a nominal group, the chart would record this partial structure, as well as hypotheses about what could continue the structure.

The WAG parser uses a chart parser, although the notion has been modified for Systemic grammars. For further reading on chart parsing, see Kay (1980, 1985), Varille (1983), or Gazdar & Mellish (1989). Jay Early developed the first chart-like algorithm (Early 1970). Masara Tomita developed an alternative data-structure for storing alternative analyses, called *graph-structured stacks* (see Tomita 1985).

3.4 Exhaustive *vs.* Non-Exhaustive Search

Another choice between parsing strategies involves the exhaustiveness of the parse:

- **First-Analysis**: the parse stops on the first successful analysis of the sentence.
- **Exhaustive-Search**: The parse finds all analyses of a sentence.

The choice between the algorithms is a choice between speed (First-Analysis is quicker) and accuracy (the first sentence is not always the appropriate one). For the First-Analysis parsers, several heuristics have been used to increase the likelihood of reaching the correct analysis first. For instance:

Probability: each option in parsing (branching, e.g., optional elements, rule application, lexical ambiguity) has a probability associated with it. The parser chooses the highest probability option at each choice-point (see, for instance, O'Donoghue 1991a).

Precedence Rules: rather than probabilities, precedence rules state which interpretation is preferred in a particular setting, e.g., to resolve the pp-attachment problem, some verbs and nouns can be given priority for the attachment of particular prepositional-phrases. Thus, motion-verbs have precedence to attach Origin prep-phrases, e.g., in "She drove the car from Italy.", the prepositional-phrase would be attached to the verb, rather than to the noun (cf. Ford, Bresnan & Kaplan 1982; also Huang Xuiming - personal communication).

For descriptions of systems using such heuristics, see for instance, Kimball 1973; Ford, Bresnan & Kaplan 1982; Huyck & Lytinen 1993.

4. Summary

This chapter has outlined the basic alternatives in parsing systems, including:

a) **Differences in control strategies**:

- Top-down *vs.* bottom-up;
- Depth-first *vs.* breadth-first;
- Node selection strategies.

b) **Handling determinism in parsing**:

- Deterministic parsing methods;
- Non-Deterministic parsing methods (backtracking, parallel parsing, chart parsing).

c) **Search exhaustiveness.**

This chapter has provided background for more specific discussion of Systemic parsing in the following chapter.