Grammar and Parsing

1 Parsing

For a given input string, a parser will attempt to find one or more parse trees for that string. (In fact, we'll see in the next lecture that, in systems where semantic translation (into logic) is interleaved with parsing, producing the parse trees may not be required. However, the basic operation of the parser remains the same.)

There are at least two different ways that parsers can operate:

Top-down parsing: The parser tries to build phrase-structure trees starting from their root nodes, adding branches and nodes until it reaches the leaves. This is a bit like backwards-chaining: you're working from a hypothesis towards the facts (words).

Bottom-up parsing: The parser builds phrase-structure trees from the leaves, adding branches and nodes until they reach the root. This is a bit like forwards-chaining: you're working from what you know (the words).

An advantage of bottom-up parsing arises in connection with parsing *fragmentary input*. We do not always communicate in full sentences. In most top-down parsers, the parser begins with an initial hypothesis about what the category at the root of the parse will be, usually S. Obviously, with this hypothesis, a fragmentary input cannot be parsed. Bottom-up parsing, however, is more amenable to parsing such fragments: it can work away at the words, and see what it comes up with.

It might also be more psychologically plausible to parse bottom-up. In fact, it is likely that human parsers work in both directions: incoming words are parsed bottom-up, but this sets up expectations (top-down hypotheses) about how subsequent words will be parsed.

There are other ways that parsers can differ. They might work from left-to-right or right-to-left. Working left-to-right would seem to have the greater plausibility. But a case can be made for bidirectionality here too: working outwards from important words in the sentence.

Finally, bear in mind that parsing is a search process. There will be choices. Depending on how we process the agenda, we may parse breadth-first, depth-first or something else. There's some bad news here too. Parsing using a CF-PSG is intractable: in the worst case, the number of parse trees is exponentially related to the length of the string. (In practice, average case behaviour on typical natural language grammars and typical input string lengths is more important, and is not prohibitively inefficient.)

We're going to look at a bottom-up parser which works from left-to-right and uses an agenda to manage its search.

2 Shift-Reduce Parsing

The parser we look at is by no means the best available for NLP. Its main advantage is that it is quite simple. It has also formed the basis of a number of variants that people claim are psychologically quite plausible ways of parsing. It is not, however, one of the more efficient parsers.

A shift-reduce parser employs two data structures: a *buffer* that contains the unprocessed part of the input string, and a *stack* on which a parse is gradually assembled. The state of these two data structures at any point in the algorithm is referred to as the parser's *configuration*.

At any step, the parser can carry out one of two operations. The two operations are:

Shift: In shift operations the next word of the input string is moved from the buffer to the top of the stack: if the next word of the input string is some w and the stack is presently $[\ldots\ X]$, where X is the top of the stack, then the stack becomes $[\ldots\ X\ w]$.

Reduce: In reduce operations, the parser finds an entry in the lexicon and replaces the word on the top of the stack by its category. Or it finds a rule in the grammar whose right-hand side matches the constituents on the stack, one by one, starting from the top and replaces these constituents by the category on the left-hand side of the rule: if there is a rule $A \rightarrow \alpha$ and the stack contains $[\ldots\ \alpha]$, where α is the sequence of symbols on the top of the stack, the categories α on the stack are popped and the left-hand side of the rule is pushed, and so the stack becomes $[\ldots\ A]$.

On reaching a configuration in which the input buffer is empty and the stack contains a single category, we have found a parse of the string as a phrase of that category.

In the lecture, we'll parse *"Ann saw Ben"* using the shift-reduce parser and using the grammar and lexicon given in the previous lecture. From the example, you will be able to see that shift-reduce parsers work bottom-up and left-to-right.

The example will also illustrate that at some steps of its processing the parser has a *choice* of operations and thus needs to search. These decision points are referred to as *conflicts* and can be subcategorised as follows:

Shift-reduce conflicts: In shift-reduce conflicts, the parser configuration is such that a shift is applicable and at least one reduce is also applicable. For example, in the configuration

[Name] ⟨*saw Ben*⟩

either the word *"saw"* could be shifted onto the stack or, given that the top of the stack matches the right-hand side of the rule NP → Name, the Name on the top of the stack can be reduced to NP.

Reduce-reduce conflicts: In reduce-reduce conflicts, the parser configuration is such that more than one reduction is applicable. For example, in the configuration

[NP *saw*] ⟨*Ben*⟩

there are two applicable reductions, one using *saw* : V and the other using *saw* : N. (Of course, a shift of *"Ben"* is also possible so there is a shift-reduce conflict in this configuration too.)

Faced with a conflict, a simple shift-reduce parser will have first to try one of the applicable operations and eventually return and try the others: some may lead to parses, others may prove fruitless.

3 Definite Clause Grammars

We now present another grammar formalism, the *Definite Clause Grammar (DCG)* formalism, that is much used in computational work and that is one representative of a class of grammar formalisms, the unification-based grammar formalisms, that form the basis of more linguistically adequate grammars than the CF-PSGs we have used up to now.

The grammar we gave in the previous lecture incorrectly says that the following ungrammatical strings are grammatical.

(1) *"This men saw Ann."*

(2) *"Us died."*

(3) *"I sees the mountain."*

(4) *"Ann died Ben."*

It fails to enforce agreement between Dets and Ns; it fails to enforce agreement between subject NPs and VPs; it fails to allow only subject NPs in subject position and object NPs in other positions; and it fails to distinguish different subcategories of verbs.

Of course, we could repair this by using more syntactic categories. For example, we could use SubjPro for words such as *"I"* and ObjPro for words such as *"me"*. Unfortunately, we would then need more rules, e.g. SubjNP $\rightarrow$ SubjPro, S $\rightarrow$ SubjNP VP, ObjNP $\rightarrow$ ObjPro and VP $\rightarrow$ V ObjNP.

But there is a much more elegant solution. Instead of using indivisible symbols to label syntactic categories (such as NP, VP, etc.), we use labels that have some internal structure to them.

Grammar

$$
\begin{aligned}
\text{S} &\rightarrow \text{NP(SUBJ}, x, y)\ \text{VP}(x, y) \\
\text{NP}(\_, 3, x) &\rightarrow \text{Name}(x) \\
\text{NP}(x, y, z) &\rightarrow \text{Pro}(x, y, z) \\
\text{NP}(\_, 3, x) &\rightarrow \text{Det}(x)\ \text{Nbar}(x) \\
\text{Nbar}(x) &\rightarrow \text{N}(x) \\
\text{Nbar}(x) &\rightarrow \text{Nbar}(x)\ \text{PP} \\
\text{VP}(x, y) &\rightarrow \text{V(INTRANS}, x, y) \\
\text{VP}(x, y) &\rightarrow \text{V(TRANS}, x, y)\ \text{NP(OBJ}, \_, \_) \\
\text{VP}(x, y) &\rightarrow \text{VP}(x, y)\ \text{PP} \\
\text{PP} &\rightarrow \text{P NP(OBJ}, \_, \_)
\end{aligned}
$$

Lexicon

Ann	Name(SING)	*I*	Pro(SUBJ, 1, SING)
Ben	Name(SING)	*me*	Pro(OBJ, 1, SING)
man	N(SING)	*we*	Pro(SUBJ, 1, PLU)
men	N(PLU)	*us*	Pro(OBJ, 1, PLU)
saw	N(SING)	*you*	Pro(_, 2, _)
telescope	N(SING)	*he*	Pro(SUBJ, 3, SING)
mountain	N(SING)	*him*	Pro(OBJ, 3, SING)
the	Det(_)	*she*	Pro(SUBJ, 3, SING)
this	Det(SING)	*her*	Pro(OBJ, 3, SING)
these	Det(PLU)	*it*	Pro(_, 3, SING)
with	P	*them*	Pro(OBJ, 3, PLU)
on	P	*they*	Pro(SUBJ, 3, PLU)

see	V(TRANS, 1, _), V(TRANS, 2, _), V(TRANS, 3, PLU)
sees	V(TRANS, 3, SING)
saw	V(TRANS, _, _)
die	V(INTRANS, 1, _), V(INTRANS, 2, _), V(INTRANS, 3, PLU)
dies	V(INTRANS, 3, SING)
died	V(INTRANS, _, _)

We'll explain the grammar in the lecture. One important point is that multiple occurrences of a variable *within a rule* must match the same expression. However, occurrences of the same variable name *in different rules* are distinct and need not match the same value. Computer scientists would say that variables are *local* to rules.

Another important point is that underscore (_) is a variable but two underscores, even in the same rule, do not have to be matched to the same value. We refer to underscore as the *anonymous variable*.

Where do such rules comes from? At present, they're written by knowledge engineers. Algorithms for learning rules in which the predicates are allowed to have arguments (e.g. FOIL and GOLEM) might be applicable, but research is only just beginning.

In parsing, when carrying out matching, we now, of course, use *unification*. For example, if we are parsing *"The men died"* and we have shifted *"the"* and reduced it to a Det and shifted *"men"* and reduce it to an N and then to an Nbar, we will be in the following configuration:

$$[\text{Det}(\_)\ \text{Nbar(PLU)}] \qquad \langle \textit{died} \rangle$$

We reduce using the rule NP$(_, 3, x) \rightarrow$ Det(x) Nbar(x). The multiple occurences of x in this rule force the argument of the Det and Nbar to be the same, and to be the same as the third argument of the NP. Thus, our next configuration is:

$$[\text{NP}(\_, 3, \text{PLU})] \qquad \langle \textit{died} \rangle$$

In the lecture, we'll draw a parse tree for *"I saw this man"*.

Exercises

1. (Past exam question) Here is a Context-Free Phrase-Structure Grammar and lexicon for a fragment of the English language:

S $\rightarrow$ NP VP	*this* : Det, Pro	*airport* : N
NP $\rightarrow$ Det N	*these* : Det, Pro	*airports* : N
NP $\rightarrow$ Pro	*the* : Det	*bus* : N
N $\rightarrow$ N N	*is* : Vbe	*buses* : N
VP $\rightarrow$ Vbe Adj	*are* : Vbe	*stop* : N
VP $\rightarrow$ Vbe NP	*busy* : Adj	*stops* : N

   (a) Draw the parse tree(s), if any, that this grammar and lexicon would assign to each of the following strings of words:

      i. *This bus stop is busy*
      ii. *This is the airport bus stop*
      iii. *This stops the airport bus*
      iv. *These is the stop bus*

   (b) The grammar and lexicon above are being used to parse the string *"The bus is busy"* using a *shift-reduce parser*. The parser is currently in the following configuration:

$$[\text{NP}] \qquad \langle \textit{is busy} \rangle$$

   The stack is on the left and the input buffer on the right.

   Draw a search tree with the above configuration as its root to show how the rest of the parse proceeds. Each node in your tree will be a new configuration, produced by either shifting or reducing. Assume breadth-first search.

   (c) The grammar and lexicon above incorrectly parse many strings that are not grammatical sentences of English. Rewrite the grammar as a Definite Clause Grammar (DCG) and lexicon so that the grammatical strings still parse but the ungrammatical ones do not.

   (d) Using your Definite Clause Grammar and lexicon, draw a parse tree for the following:

      The buses are busy

   (e) Using your Definite Clause Grammar and lexicon, draw an incomplete parse tree and explain why your grammar does not parse the following:

      The buses is busy

2. (Past exam question) Here is a Definite Clause Grammar and lexicon for a fragment of the English language. (x, y and z are variables; _ is the anonymous variable.)

S(statement) $\rightarrow$ NP(x, y) VP(finite, x, y)	did	:	V(aux, finite, _, _)
S(question) $\rightarrow$ V(aux, finite, x, y) NP(x, y) VP(base, _, _)	the	:	Det(_)
NP(3, x) $\rightarrow$ Det(x) N(x)	Italian	:	Adj
N(x) $\rightarrow$ Adj N(x)	wine	:	N(singular)
N(x) $\rightarrow$ N(_) N(x)	merchants	:	N(plural)
VP(x, y, z) $\rightarrow$ V(intrans, x, y, z)	arrive	:	V(intrans, base, _, _)

(a) This grammar and lexicon assign two different phrase-structures to the following sentence:

"did the Italian wine merchants arrive"

Draw *parse trees* for **both** phrase-structures.

(b) According to this grammar and lexicon the following sentences are not syntactically well-formed:

"the merchants brought the wine"
"did the merchants bring the wine"

Give the grammar rules and lexical entries that you would add so that these sentences can be parsed, without allowing any ungrammatical strings to be parsed.

(There is an answer that requires only one new grammar rule and two new lexical entries. However, marks are available for other answers.)

(c) The grammar tries to determine the *'force'* of speech acts based on the syntactic structure of sentences. For example, the root of the parse tree for the sentence *"the merchants arrive"* is labelled S(statement).

Explain, using one or more example sentences (not necessarily sentences that use this grammar and lexicon), why this approach to determining speech act 'force' is inadequate in general.

(d) Write a brief (3 or 4 paragraph) discussion of the relationship between *natural language understanding* and *knowledge engineering*.