9

Partial Parsing

9.1 Is Syntax Necessary?

The description of language in terms of layers – sounds, words, and syntax – can suggest that a parse tree is a necessary step to obtain the semantic representation of a sentence. Yet, many industrial applications do not rely on syntax as we presented it before. The reason is that it is difficult to build a syntactic parser with large grammatical coverage, expensive in terms of resources, and sometimes it is not worth the cost.

Some applications need only to detect key words, as in some telephone speech servers. There, the speech recognition module spots meaningful words and sets the others aside. It enables the system to deal with the noisy environment or the fragmented nature of speech by telephone. Other applications rely on the detection of word groups such as noun phrases. Although sentences are not fully parsed, the result is sufficient to make use of it. Information retrieval and extraction are typical applications relying on group detection techniques.

In this chapter, we will examine a collection of techniques to extract incomplete syntactic representations. These techniques are generally referred to as partial or shallow parsing and operate on groups of words, often called **chunks**. Some of them just carry out the detection of key words or specific word patterns. Others use phrase-structure rules describing groups such as noun groups or verb groups. Finally, some techniques are an extension of part-of-speech tagging and resort to similar methods.

9.2 Word Spotting and Template Matching

9.2.1 ELIZA

A first shallow technique consists in matching predefined templates. It appeared with the popular ELIZA program that mimics a dialogue between a psychotherapist and his/her patient (Weizenbaum 1966). In fact, ELIZA understands merely nothing. She "spots" a handful of words or patterns such as *yes*, *no*, *why*, *I'm afraid of X*, *I like X*,

etc., where *X* is a name or any group of words. When a template matches the user's sentence, ELIZA has a set of ready-made answers or questions mapped onto it. When no template matches, ELIZA tries to guess whether the sentence is a declaration, a negation, or an interrogation, and has repartees like *in what way*, *can you think of a specific example*, *go on*, etc. It enables the machine to follow the conversation with a semblance of realism. Table 9.1 shows some user/psychotherapist templates.

Table 9.1. Some ELIZA templates.

User	Psychotherapist
... I like X...	*Why do you like X?*
... I am X...	*How long have you been X?*
... father...	*Tell me more about your father*

ELIZA's dialogue pays a specific attention to words like *mother* and *father*. Whenever one of these words occurs, ELIZA asks for more details. We remind the reader that this program was created when Freudian theories were still very influential. Although the approach is now considered simplistic, at best, the psychoanalytical settings secured ELIZA a mainstream popularity.

9.2.2 Word Spotting in Prolog

A word spotting program can easily be written using DCG rules. Utterances are modeled as phrase-structure rules consisting of a beginning, the word or pattern to search, and an end. The translation into a DCG rule is straightforward:

```
utterance(U) --> beginning(B), [the_word], end(E).
```

Each predicate has a variable that unifies with the part of the utterance it represents. Variables `B` and `E` unify respectively with the beginning and the end of the utterance. The variable `U` is used to build the system answer as in the templates in Table 9.1.

Prolog translates the DCG rules into clauses when they are consulted. It adds two arguments to each predicate, and the previous rule expands into:

```
utterance(U, L1, L) :-
  beginning(B, L1, L2),
  c(L2, the_word, L3),
  end(E, L3, L).
```

We saw in Chap. 8 that each predicate in the rule covers a word sequence, and that it corresponds to the difference of the two new arguments: `L1` minus `L` corresponds to `utterance`; `L1` minus `L2` corresponds to `beginning`; `L3` minus `L` corresponds to `end`. Figure 9.1 shows the composition of the utterance with respect to the new lists.

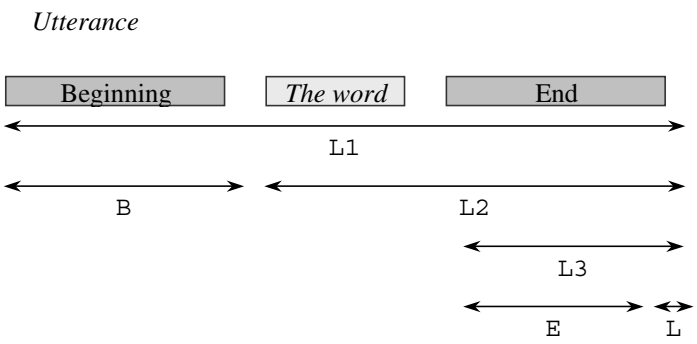

Fig. 9.1. The composition of utterance.

To match `B` and `E`, the trick is to define `beginning/3` and `end/3` as `append`-like predicates:

```
beginning(X, Y, Z) :- append(X, Z, Y).
end(X, Y, Z) :- append(X, Z, Y).
```

ELIZA is then a loop that reads the user input, tries to find a matching utterance, and answers with the corresponding template. It stops when the user writes the word *bye*. The next program is a simplified version of ELIZA. It matches the user/psychoanalyst pairs in Table 9.1.

```
%% A simplified version of ELIZA in Prolog
%%

% The main loop reads the input and calls process/1
% It stops when the input is the word bye.

eliza :-
  write('Hello, I am ELIZA. How can I help you?'),
  nl,
  repeat,
  write('> '),
  tokenize(In),
  process(In).

% process/1 accepts the user's utterance.
% It either terminates or tries to find a
% template matching the user's utterance

process([bye | _]) :-
  write('ELIZA: bye'), nl, !.
process(In) :-
  utterance(Out, In, []),
```

```
    !,
    write('ELIZA: '), write_answer(Out),
    fail.

% utterance is a set of DCG rules describing
% templates

utterance(['Why', aren, ''', t, you | Y]) -->
  ['I', am, not], end(Y).
utterance(['How', long, have, you, been | Y]) -->
  ['I', am], end(Y).
utterance(['Why', do, you, like | Y]) -->
  ['I', like], end(Y).
utterance(['Do', you, often, think, of | Y]) -->
  beginning(_), ['I', remember], end(Y).
utterance(['Please', tell, me, more, about, your,
    X, .]) -->
  beginning(_), [X], end(_), {important(X)}.
utterance(['Why', are, so, negative, '?']) -->
  [no].
utterance(['Tell', me, more, .]) -->
  [_].
utterance(['Please', go, on, .]) -->
  beginning(_).

% The Prolog clauses beginning/3 and end/3
% link the pieces of the utterance

beginning(X, Y, Z) :- append(X, Z, Y).
end(X, Y, Z) :- append(X, Z, Y).

% The Freudian facts
important(father).
important(mother).
important(brother).
important(sister).

% write_answer/1 prints the content of the list
% containing Eliza's answer.

write_answer([Head | Tail]) :-
  write(Head), write(' '),
  write_answer(Tail).
 write_answer([]) :- nl.
```

The next lines are a sample dialogue with ELIZA:

```
?- eliza.
Hello, I am ELIZA. How can I help you?
> yes
ELIZA: Tell me more .
> I like apples
ELIZA: Why do you like apples
>I went out with my father
ELIZA: Please tell me more about your father .
> no
ELIZA: Why are so negative ?
> I am cold
ELIZA: How long have you been cold
> bye
ELIZA: bye

Yes
?-
```

9.3 Multiword Detection

9.3.1 Multiwords

While ELIZA has no real application, the techniques we used in it can serve to detect specific patterns in texts or in speech. Finding multiple word expressions or multiwords is an example.

Multiwords – or multiword expressions (MWE) – are sequences of two or more words that act as a single lexical unit. They include proper nouns (names) of persons, companies, organizations, temporal expressions describing times and dates, and numerical expressions. Multiwords also include complex prepositions, adverbs, conjunctions, or phrasal verbs where each of the words taken separately cannot be clearly understood (Table 9.2). Multiwords corresponding to people or organization names are frequent in the press and the media, where new denominations surge and quickly disappear.

Although the identification of multiwords may seem intuitive, there are many tricky cases. In addition, people do not always agree on their exact definition.

9.3.2 A Standard Multiword Annotation

In the 1990s, The US Department of Defense organized series of competitions to measure the performance of commercial and academic systems on multiword detection. It called them the Message Understanding Conferences (MUC). To help benchmarking the various systems, MUC-6 and MUC-7 defined an annotation scheme that

Table 9.2. Multiwords in English and French.

Type	English	French
Prepositions	*to the left hand side*	*À gauche de*
Adverbs	*because of*	*à cause de*
Conjunctions		
Names	*British gas plc.*	*Compagnie générale d'électricité SA*
Titles	*Mr. Smith*	*M. Dupont*
	The President of the United States	*Le président de la République*
Verbs	*give up*	*faire part*
	go off	*rendre visite*

was shared by all the participants. This annotation has subsequently been adopted by commercial applications.

The MUC annotation restricts the annotation to information useful for its main funding source: the US military. It considers named entities (persons, organizations, locations), time expressions, and quantities. The annotation scheme defines a corresponding XML element for each of these three classes: `<ENAMEX>`, `<TIMEX>`, and `<NUMEX>` (Chinchor 1997), with which it brackets the relevant phrases in a text. The phrases can be real multiwords, consisting of two or more words, or they can be restricted to a single word.

The `<ENAMEX>` element identifies proper nouns and uses a `TYPE` attribute with three values to categorize them: `ORGANIZATION`, `PERSON`, and `LOCATION`, as in

```
the <ENAMEX TYPE="PERSON">Clinton</ENAMEX> government
<ENAMEX TYPE="ORGANIZATION">Bridgestone Sports Co.</ENAMEX>
<ENAMEX TYPE="ORGANIZATION">European Community</ENAMEX>
<ENAMEX TYPE="ORGANIZATION">University of California</ENAMEX>
in <ENAMEX TYPE="LOCATION">Los Angeles</ENAMEX>
```

The `<TIMEX>` element identifies time expressions and uses a `TYPE` attribute to distinguish between `DATE` and `TIME`, as in

```
<TIMEX TYPE="TIME">twelve o'clock noon</TIMEX>
<TIMEX TYPE="TIME">5 p.m. EST</TIMEX>
<TIMEX TYPE="DATE">January 1990</TIMEX>
```

The `<NUMEX>` element is used to bracket quantities. It has also a `TYPE` attribute to categorize `MONEY` and `PERCENT`, as in

```
<NUMEX TYPE="MONEY">20 million New Pesos</NUMEX>
<NUMEX TYPE="MONEY">$42.1 million</NUMEX>
<NUMEX TYPE="MONEY">million-dollar</NUMEX> conferences
<NUMEX TYPE="PERCENT">15 pct</NUMEX>
```

9.3.3 Detecting Multiwords with Rules

The detection of multiwords with rules is an extension of word spotting. Just as for word spotting, we represent multiwords using DCG rules. We use variables and Prolog code to extract them from the word stream and annotate them.

Compounded prepositions, conjunctions, and phrasal verbs are often listed in dictionaries and can be encoded as Prolog constants. Other multiwords raise more problems. Their identification generally requires specialized dictionaries of surnames, companies, countries, and trademarks. Some of these dictionaries, called **gazetteers**, are available on the Internet. They are built from the compilation of lexical sources such as economic and legal newspapers, directories, or Internet Web sites.

The extraction of multiwords also relies on hints that vary according to the type of entity to detect. Locations may include words such as *Ocean*, *Range*, *River*, etc. Legal denominations will be followed by acronyms such as *Ltd*, *Corp.*, *SA*, and *GMBH*. Persons might be preceded by titles such as *Mr.*, *Mme*, *Herr*, *Dr.*, by a surname, or have a capitalized initial. Currency phrases will include a sign such as €, $, £, etc., and a number. Such techniques can be applied to any measuring expression: length, time, etc.

Let us write rules to detect the phrasal verb *give up*, the French title *M. XXXX*, such as *M. Dupont*, and the European money worth *XXXX euros*, such as *200 euros*. As a result, the detector appends the multiword parts using an underscore character: `give_up`, or builds a list with surrounding XML tags `[<ENAMEX>, 'M.', 'Dupont', </ENAMEX>]`, and `[<NUMEX>, 200, euros, </NUMEX>]`. The corresponding rules are:

```
multiword(give_up) --> [give, up].
multiword(['<ENAMEX>', 'M.', Name, '</ENAMEX>']) -->
  ['M.'], [Name],
  {
    name(Name, [Initial | _]),
    Initial >= 65, % must be an upper-case letter
    Initial =< 90
  }.
multiword(['<NUMEX>', Value, euros, '</NUMEX>']) -->
  [Value], [euros],
  {
    number(Value)
  }.
```

9.3.4 The Longest Match

Among the set of multiwords we want to detect, some may have a common suffix, as for the phrases *in front* and *in front of*. This corresponds to the rules:

```
multiword(in_front) --> [in, front].
multiword(in_front_of) --> [in, front, of].
```

With the sentence

The car in front of the house

rules as they are ordered above yield two solutions. The first multiword to be matched is *in front*, and if Prolog backtracks, it will find *in front of*. A backtracking strategy is not acceptable in most cases. What we generally want is the longest possible match (Table 9.3).

Table 9.3. Longer matches are preferred.

	English	**French**
Competing multiwords	*in front of*	*en face de*
	in front	*en face*
Examples	*The car in front*	*La voiture en face*
	In front of me	*En face de moi*

Prolog interpreters consider rules sequentially and downwards (from the beginning to the end). We implement the longest match by ordering the DCG rules properly. When several multiwords compete, i.e., have the same beginning, the longest one must be searched first, as in the sequence:

```
multiword(in_front_of) --> [in, front, of].
multiword(in_front) --> [in, front].
```

9.3.5 Running the Program

Now we will write a rule to embed the multiword description. If the word stream contains a multiword, it should be modeled as a beginning, the multiword, and an end, as in ELIZA. Its transcription into a DCG rule is straightforward:

```
word_stream_multiword(Beginning, Multiword, End) -->
  beginning(Beginning),
  multiword(Multiword),
  end(End).
```

Extracting the list of multiwords means that the whole word stream must be matched against the rule set. The multiword detector scans the word stream from the beginning, and once a multiword has been found, it starts again with the remaining words.

`multiword_detector/2` is a Prolog predicate. It takes the word stream `In` as the input and the multiword list `Out` as the output. It searches a multiword within the word stream using the `word_stream_multiword` DCG rule.

Each word_stream_multiword rule is translated into a Prolog predicate when consulted and two new variables are added. Thus, word_stream_multiword is of arity 5 in the multiword_detector rule. The two last variables are unified respectively to the input list and to the empty list.

When word_stream_multiword reaches a multiword, Beginning is unified with the beginning of the word stream and End with the rest. The program is called recursively with End as the new input value.

```
multiword_detector(In, [Multiword | Out]) :-
  word_stream_multiword(Beginning, Multiword, End,
    In, []),
  multiword_detector(End, Out).
  multiword_detector(_, []).
```

Using the detector with the sentence *M. Dupont was given 500 euros in front of the casino* results into ['<ENAMEX>', 'M. ', 'Dupont', '</ENAMEX>'], ['<NUMEX>', 500, euros, '</NUMEX>'], and in_front_of:

```
?- multiword_detector(['M.', 'Dupont', was, given,
500, euros, in, front, of, the, casino], Out).
Out = [[<ENAMEX>, M., Dupont, </ENAMEX>],
[<NUMEX>, 500, euros, </NUMEX>], in_front_of]
```

The result is a list containing sublists. The flatten/2 predicate can replace recursively all the sublists by their elements and transform them into a flat list.

```
?- flatten([['<ENAMEX>', 'M. ', Dupont,
'</ENAMEX>'], ['<NUMEX>', 500, 'DM', '</NUMEX>'],
in_front_of], Out).
Out = [<ENAMEX>, M., Dupont, </ENAMEX>, <NUMEX>,
500, DM, </NUMEX>, in_front_of]
```

The multiword detector can be modified to output the whole stream. That is, the multiwords are tagged and other words remain unchanged. In this program, Beginning is appended to the multiword Multiword that has been detected to form the Head of the word stream. The Head and the result of the recursive call Rest form the Output. We must not forget the End in the termination fact.

```
multiword_detector(In, Out) :-
  word_stream_multiword(Beginning, Multiword, End,
    In, []),
  !,
  multiword_detector(End, Rest),
  append(Beginning, [Multiword], Head),
  append(Head, Rest, Out).
  multiword_detector(End, End).
```

Let us now execute a query with this new detector with flatten/2:

```
?- multiword_detector(['M.', 'Dupont', was, given,
500, euros, in, front, of, the, casino], Res),
flatten(Res, Out).
Out = [<ENAMEX>, M., Dupont, </ENAMEX>, was,
given, <NUMEX>, 500, euros, </NUMEX>, in_front_of,
the, casino]
```

9.4 Noun Groups and Verb Groups

The word detection techniques enabled us to search certain word segments, with no consideration of their category or part of speech. The detection can extend to syntactic patterns.

The two most interesting word groups are derived from the two major parts of speech: the noun and the verb. They are often called **noun groups** and **verb groups**, although **noun chunks** and **verb chunks** are also widely used. In a sentence, noun groups (Table 9.4) and verb groups (Table 9.5) correspond to verbs and nouns and their immediate depending words. This is often understood, although not always, as words extending from the beginning of the constituent to the head noun or the head verb. That is, the groups include the headword and its dependents to the left. They exclude the postmodifiers. For the noun groups, this means that modifying prepositional phrases or, in French, adjectives to the right of the nouns are not part of the groups.

The principles we exposed above are very general, and exact definitions of groups may vary in the literature. They reflect different linguistic viewpoints that may coexist or compete. However, when designing a parser, precise definitions are of primary importance. Like for part-of-speech tagging, hand-annotated corpora will solve the problem. Most corpora come with annotation guidelines. They are usually written before the hand-annotation process. As definitions are often difficult to formulate the first time, they are frequently modified or complemented during the annotation process. Guidelines normally contain definitions of groups and examples of them. They should be precise enough to enable the annotators to bracket consistently the groups. The guidelines will provide the grammar writer with accurate definitions, or when using machine learning techniques, the annotated texts will encapsulate the linguistic knowledge about groups and make it accessible to the automatic analysis.

Table 9.4. Noun groups.

English	French	German
The waiter *is bringing* ***the very big dish*** *on* ***the table***	***Le serveur*** *apporte* ***le très grand plat*** *sur* ***la table***	***Der Ober*** *bringt* ***die sehr große Speise*** *an* ***dem Tisch***
Charlotte *has eaten* ***the meal*** *of* ***the day***	***Charlotte*** *a mangé* ***le plat*** *du* ***jour***	***Charlotte*** *hat* ***die Tagesspeise*** *gegessen*

Table 9.5. Verb groups.

English	French	German
The waiter ***is bringing*** *the very big dish on the table*	*Le serveur* ***apporte*** *le très grand plat sur la table*	*Der Ober* ***bringt*** *die sehr große Speise an den Tisch*
Charlotte ***has eaten*** *the meal of the day*	*Charlotte* ***a mangé*** *le plat du jour*	*Charlotte* ***hat*** *die Tagesspeise* ***gegessen***

9.4.1 Groups Versus Recursive Phrases

The rationale behind word group detection is that a group structure is simpler and more tractable than that of a sentence. Group detection uses a local strategy that can accept errors without making subsequent analyses of the rest of the sentence fail. It also leaves less room for ambiguity because it sets aside the attachment of prepositional phrases. As a result, partial parsers are more precise. They can capture roughly 90% of the groups successfully (Abney 1996).

Like for complete sentences, phrase-structure rules can describe group patterns. They are easier to write, however, because verb groups and noun groups have a relatively rigid and well-defined structure. In addition, local rules usually do not describe complex recursive linguistic structures. That is, there is no subgroup inside a group and, for instance, the noun group is limited to a unique head noun. This makes the parser very fast. Moreover, in addition to phrase-structure rules, finite-state automata or regular expressions can also describe group structures.

9.4.2 DCG Rules to Detect Noun Groups

A noun group consists of an optional determiner *the*, *a*, or determiner phrase such as *all of the*, one or more optional adjectives, and one or more nouns. It can also consist of a pronoun or a proper noun – a name. This definition is valid in English. In German, sequences of nouns usually form a single word through compounding. In French, noun groups also include adjectives to the right of the head noun that we set aside.

The core of the noun group is a sequence of nouns also called a nominal expression. A first possibility would be to write as many rules as we expect nouns. However, this would not be very elegant. A recursive definition is more concise: a nominal is then either a noun or a noun and a nominal. Symbols `noun` and `nominal` have variables that unify with the corresponding word. This corresponds to the rules:

```
nominal([NOUN | NOM]) --> noun(NOUN), nominal(NOM).
nominal([N]) --> noun(N).
```

The simplest noun groups consist of a determiner and a nominal. The determiners are the articles, the possessive pronouns, etc. They are sometimes more complex phrases that we set aside here. Determiners are optional and the group definition must also represent its absence. A noun group can also be a proper noun or a pronoun:

```
% noun_group(-NounGroup)
% detects a list of words making a noun group and
% unifies NounGroup with it

noun_group([D | N]) --> det(D), nominal(N).
noun_group(N) --> nominal(N).
noun_group([PN]) --> proper_noun(PN).
noun_group([PRO]) --> pronoun(PRO).
```

The adjective group serves as an auxiliary in the description of noun group. It can feature one or more adjectives and be preceded by an adverb. If we set aside the commas, this corresponds to:

```
% adj_group(-AdjGroup)
% detects a list of words making an adjective
% group and unifies AdjGroup with it

adj_group_x([RB, A]) --> adv(RB), adj(A).
adj_group_x([A]) --> adj(A).

adj_group(AG) --> adj_group_x(AG).
adj_group(AG) -->
  adj_group_x(AGX),
  adj_group(AGR),
  {append(AGX, AGR, AG)}.
```

Past participles and gerunds can replace adjectives as in *A flying object* or *The endangered species*:

```
adj(A) --> past_participle(A).
adj(A) --> gerund(A).
```

We must be aware that these rules may conflict with a subsequent detection of verb groups. Compare the ambiguous phrase *detected words* in *the detected words* and *The partial parser detected words*.

Adjectives can precede the noun. Using the adjective group, we add two rules to the noun group:

```
noun_group(NG) -->
  adj_group(AG), nominal(NOM),
  {append(AG, NOM, NG)}.
noun_group(NG) -->
  det(D), adj_group(AG), nominal(NOM),
  {append([D | AG], NOM, NG)}.
```

9.4.3 DCG Rules to Detect Verb Groups

Verb groups can be written in a similar way. In English, the simplest group consists of a single tensed verb:

```
verb_group([V]) --> tensed_verb(V).
```

Verb groups also include adverbs that may come before the verb:

```
verb_group([RB, V]) --> adv(RB), tensed_verb(V).
```

Verb groups can combine auxiliary and past participles, or auxiliary and gerund, or modal and infinitive, or *to* and infinitive, or be simply an auxiliary:

```
verb_group([AUX, V]) --> aux(AUX),past_participle(V).
verb_group([AUX, G]) --> aux(AUX), gerund(G).
verb_group([MOD, I]) --> modal(MOD), infinitive(I).
verb_group([to, I]) --> [to], infinitive(I).
verb_group([AUX]) --> aux(AUX).
```

Verb groups can include adverbs and have more auxiliaries:

```
verb_group([AUX, RB, V]) -->
  aux(AUX), adv(RB), past_participle(V).
verb_group([AUX1, AUX2, V]) -->
  aux(AUX1), aux(AUX2), past_participle(V).
verb_group([MOD, AUX, V]) -->
  modal(MOD), aux(AUX), past_participle(V).
```

Now let us write a rule that describes a group inside a word stream: `word_stream_group`. As for with the multiwords, such a stream consists of a beginning, the group, and an end. Its transcription into a DCG rule is:

```
word_stream_group(Beginning, Group, End) -->
  beginning(Beginning),
  group(Group),
  end(End).
```

Finally, a group can either be a noun group or a verb group. As for multiwords, noun groups and verb groups are annotated using the XML tags `<NG>` and `<VG>`:

```
group(NG) -->
  noun_group(Group),
  {append(['<NG>' | Group], ['</NG>'], NG)}.
group(VG) -->
  verb_group(Group),
  {append(['<VG>' | Group], ['</VG>'], VG)}.
```

9.4.4 Running the Rules

Let us write a Prolog program using an approximation of the longest match algorithm to run the rules. The program is similar to the multiword detector:

```
group_detector(In, Out) :-
  word_stream_group(Beginning, Group, End, In, []),
  group_detector(End, Rest),
  append(Beginning, [Group], Head),
  append(Head, Rest, Out).
group_detector(End, End).
```

Since these rules match the longest segments first, they must be written from the longest to the shortest.

Although the grammar is certainly not comprehensive, it can fare reasonably well for a first step. We shall apply it to a text from the *Los Angeles Times* "Flying Blind With the Titans", December 17, 1996):

> *Critics question the ability of a relatively small group of big integrated prime contractors to maintain the intellectual diversity that formerly provided the Pentagon with innovative weapons. With fewer design staffs working on military problems, the solutions are likely to be less varied.*

The lexical rules for this text are:
The query results in:

```
?- group_detector([critics, question, the,
ability, of, a, relatively, small, group, of, big,
integrated, prime, contractors, to, maintain, the,
intellectual, diversity, that, formerly, provided,
the, pentagon, with, innovative, weapons, with,
fewer, design, staffs, working, on, military,
problems, the, solutions, are, likely, to, be,
less, varied], L), flatten(L, Out).


Out = [<NG>, critics, </NG>, <VG>, question,
</VG>, <NG>, the, ability, </NG>, of, <NG>, a,
relatively, small, group, </NG>, of, <NG>, big,
integrated, prime, contractors, </NG>, <VG>, to,
maintain, </VG>, <NG>, the, intellectual,
diversity, </NG>, that, <VG>, formerly, provided,
</VG>, <NG>, the, pentagon, </NG>, with, <NG>,
innovative, weapons, </NG>, with, <NG>, fewer,
design, staffs, </NG>, working, on, <NG>,
military, problems, </NG>, <NG>, the, solutions,
</NG>, <VG>, are, </VG>, likely, <VG>, to, be,
</VG>, less, varied]
```

```
det(the) --> [the].
det(a) --> [a].
det(null_det) --> [].
noun(critics) --> [critics].
noun(ability) --> [ability].
noun(group) --> [group].
noun(contractors) -->
       [contractors].
noun(diversity) -->
       [diversity].
noun(pentagon) -->
       [pentagon].
noun(weapons) --> [weapons].
noun(design) --> [design].
noun(staffs) --> [staffs].
noun(problems) -->
       [problems].
noun(solutions) -->
       [solutions].
adv(relatively) -->
       [relatively].
adv(formerly) --> [formerly].
adv(likely) --> [likely].
adv(less) --> [less].
adj(small) --> [small].
adj(big) --> [big].
adj(prime) --> [prime].
adj(intellectual) -->
       [intellectual].
adj(innovative) -->
       [innovative].
adj(military) --> [military].
adj(fewer) --> [fewer].
infinitive(be) --> [be].
infinitive(maintain) -->
       [maintain].
tensed_verb(question) -->
       [question].
tensed_verb(provided) -->
       [provided].
past_participle(integrated) -->
       [integrated].
past_participle(varied) -->
       [varied].
aux(are) --> [are].
```

Though our detector misses groups, we realize that a limited effort has rapidly produced results.

9.5 Group Detection as a Tagging Problem

Group detection results in bracketing a word sequence with opening and closing annotations. This can be recast as a tagging problem. However, the detector inserts brackets between words instead of assigning tags to words. The most intuitive annotation is then probably to tag intervals. We can use algorithms very similar to part-of-speech tagging. They give us an alternate method to DCG rules describing verb groups and noun groups.

For the sake of simplicity, we will only present the noun group detection. Verb group detection uses exactly the same method. We first describe which tags to use to annotate the intervals. We will then see that we can equivalently tag the words instead of the gaps.

9.5.1 Tagging Gaps

Below are examples of noun group bracketing from Ramshaw and Marcus (1995). They insert brackets between the words where appropriate.

[$_{NG}$ The government $_{NG}$] has [$_{NG}$ other agencies and instruments $_{NG}$] for pursuing [$_{NG}$ these other objectives $_{NG}$] .

Even [$_{NG}$ Mao Tse-tung $_{NG}$] [$_{NG}$'s China $_{NG}$] began in [$_{NG}$ 1949 $_{NG}$] with [$_{NG}$ a partnership $_{NG}$] between [$_{NG}$ the communists $_{NG}$] and [$_{NG}$ a number $_{NG}$] of [$_{NG}$ smaller, non-communists parties $_{NG}$] .

If we only consider noun groups, the tagset must include opening and ending brackets. There must also be a tag to indicate a separation between two contiguous noun groups. The rest of the gaps are to be labeled with a "no bracket" tag.

As noun group detection usually considers nonrecursive sequences, we avoid nested brackets, as in this sequence: `[ . . .  [` or in this one: `]  . . . ]`. To check nesting while processing the stream, we must make a distinction between a "no bracket" inside a group and "no bracket" outside a group. The tagger can then prevent an inside "no bracket" to be followed by a closing bracket. We complement the tagset with no bracket tags denoting either we are within a group or outside (Table 9.6)

Table 9.6. Tagset to annotate noun groups.

Beginning	End	Between	No bracket (outside)	No bracket (inside)
$[_{NG}$	$_{NG}]$	$_{NG}][_{NG}$	$Outside$	$Inside$

In addition to nested groups, other inconsistencies can occur, such as the sequences:

- `[ Outside`
- `] Inside` or
- `Outside ]`

The tagger must keep track of the preceding bracket to refuse illegal tags.

9.5.2 Tagging Words

Instead of tagging the gaps, we can equivalently tag the words. Ramshaw and Marcus (1995) defined a tagset of three elements {`I`, `O`, `B`}, where `I` means that the word is inside a noun group, `O` means that the word is outside, and `B` means that the word is at the beginning of a noun group that immediately follows another noun group. Using this tagging scheme, an equivalent annotation of the sentences in Sect. 9.5.1 is:

The/`I` *government*/`I` *has*/`O` *other*/`I` *agencies*/`I` *and*/`I` *instruments*/`I` *for*/`O` *pursuing*/`O` *these*/`I` *other*/`I` *objectives*/`I` *./*`O`

Even/`O` *Mao*/`I` *Tse-tung*/`I` *'s*/`B` *China*/`I` *began*/`O` *in*/`O` *1949*/`I` *with*/`O` *a*/`I` *partnership*/`I` *between*/`O` *the*/`I` *communists*/`I` *and*/`O` *a*/`I` *number*/`I` *of*/`O` *smaller*/`I` *,*/`I` *non-communists*/`I` *parties*/`I` *./*`O`

As in the case for gap tagging, some inconsistencies can occur, such as the sequence: `O B`. The tagger can refuse such sequences, mapping them to a plausible annotation. That is, in the example above, to change the `B` tag into an `I` tag.

As with part-of-speech tagging, group detection uses statistical and symbolic rules methods. Both statistics and rules are learned from hand-annotated corpora. Church (1988) first addressed group detection as a tagging problem and used statistical methods. Church tagged the gaps with brackets. Ramshaw and Marcus (1995) used a symbolic strategy. They adapted Brill's (1995) algorithm to learn rules to detect groups from annotated corpora. They used the {`I`, `O`, `B`} tagset.

9.5.3 Using Symbolic Rules

The symbolic rules algorithm is very similar to that of Brill's part-of-speech tagging method. The initial tagging considers the part of speech of the word and assigns the group annotation tag that is most frequently associated with it, that is, `I`, `O`, or `B`. Then, rules applied sequentially modify annotation tags.

Rules consider the immediate context of the tag to be modified, spanning a few words to the left and a few words to the right of the current word. More precisely, they take into account group annotation tags, parts-of-speech tags, and words around the current word. When the context of the current word matches that of the rule being applied, the current tag is altered.

Ramshaw and Marcus (1995) applied a set of 100 templates using a combination of 10 word contexts and 10 part-of-speech contexts, 20 templates in total, and 5 group annotation tag contexts spanning up to three words to the left and to the right:

- $\texttt{W}_0$, $\texttt{W}_{-1}$, $\texttt{W}_1$ being respectively the current word, the first word to the left, and the first word to the right.
- $\texttt{P}_0$, $\texttt{P}_{-1}$, $\texttt{P}_1$ being respectively the part of speech of the current word, of the first word to the left, and of the first word to the right.
- $\texttt{T}_0$, $\texttt{T}_{-1}$, $\texttt{T}_1$ being respectively the group annotation tag of the current word, of the first word to the left, and of the first word to the right.

Table 9.7 shows the complete set of templates. Word and part-of-speech templates are the same.

After training the rules on the Penn Treebank using its part-of-speech tagset, they could retrieve more than 90% of the noun groups. The five most productive rules are given in Table 9.8. The first rule means that an `I` tag is changed into an `O` tag when the current part of speech is an adjective (`JJ`) and the following word is tagged `O`. The second rule sets the tag to `B` if the two previous tags are `I` and the current word's part of speech is a determiner (`DT`).

9.5.4 Using Statistical Tagging

The maximum likelihood estimator determines the optimal sequence of gap tags $G = g_2, g_3, ..., g_n$, given a sequence of part-of-speech tags $T = t_1, t_2, t_3, ..., t_n$ and of

Table 9.7. Patterns used in the templates.

Word patterns		Noun group patterns	
Pattern	**Meaning**	**Pattern**	**Meaning**
W_0	Current word	T_0	Current noun group tag
W_{-1}	First word to left	T_{-1}, T_0	Tag bigram to left of current word
W_1	First word to right	T_0, T_1	Tag bigram to right of cur. word
W_{-1}, W_0	Bigram to left of current word	T_{-2}, T_{-1}	Tag bigram to left of current word
W_0, W_1	Bigram to right of current word	T_1, T_2	Tag bigram to right
W_{-1}, W_1	Surrounding words		
W_{-2}, W_{-1}	Bigram to left		
W_1, W_2	Bigram to right		
$W_{-1,-2,-3}$	Words 1 or 2 or 3 to left		
$W_{1,2,3}$	Words 1 or 2 or 3 to right		

Table 9.8. The five first rules from Ramshaw and Marcus (1995).

Pass	Old tag	Context	New tag
1	I	T_1 = O, P_0 = JJ	O
2	-	T_{-2} = I, T_{-1} = I, P_0 = DT	B
3	-	T_{-2} = O, T_{-1} = I, P_{-1} = DT	I
4	I	T_{-1} = I, P_0 = WDT	B
5	I	T_{-1} = I, P_0 = PRP	B

words $W = w_1, w_2, w_3, ..., w_n$. It maximizes Eq. (9.1), where w_{i-1} and w_i are the words before and after each gap together with the surrounding parts of speech: t_{i-1} and t_i. Church (1988) used a simpler equation in considering parts of speech only, see Eq. (9.2).

$$P(G) = \prod_{i=2}^{n} P(g_i|w_{i-1}, t_{i-1}, w_i, t_i). \tag{9.1}$$

$$P(G) = \prod_{i=2}^{n} P(g_i|t_{i-1}, t_i). \tag{9.2}$$

Finally, the equation can take the preceding tag into account to prevent illegal transitions. That is, to assign:

$$P(g_i|t_{i-1}, t_i, g_{i-1})$$

to 0 when $g_i = [$ and $g_{i-1} = Outside$, for instance.

9.6 Cascading Partial Parsers

We saw that partial phrase-structure rules could detect multiwords and groups. We can combine both detectors into a multilevel parser and add more layers. A tokenizer

is necessary to read the text before it can be passed to the parsers. The applications generally use a part-of-speech tagger before the group detector (or **chunker**) and sometimes a morphological parser. The parser's structure is then a pipeline of analyzers, where each parsing level has a definite task to achieve. This technique is referred to as cascaded parsing.

With this approach, the exact number and nature of levels of cascaded parsers depends on the application and the expected result. In addition, some layers are relatively generic, while others are more specific and depend on the application goal. However, the principle is that one level uses the output of the lower level and passes on the result to the next layer (Fig. 9.2).

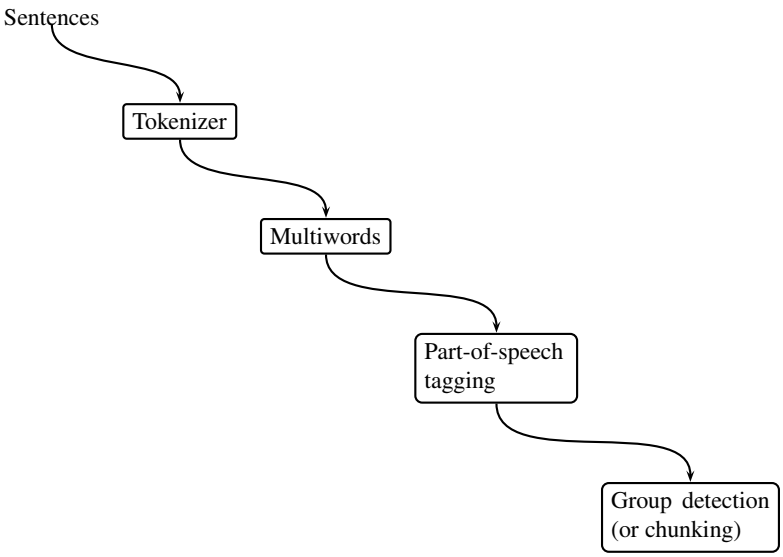

Fig. 9.2. A cascade of partial parsers.

9.7 Elementary Analysis of Grammatical Functions

9.7.1 Main Functions

In a previous section, we named groups according to the part of speech of their main word, that is, noun groups and verb groups. We can also consider their grammatical function in the sentence. We already saw that main functions (or relations) are subject, direct object, and indirect object. An accurate detection of function is difficult, but we can write a simplified one using cascaded parsing and phrase-structure rules.

We can recognize grammatical functions using a layer above those we have already described and thus complement the cascade structure. In English, the subject

is generally the first noun group of a sentence in the active voice. It is marked with the nominative case in German, while case inflection is limited to pronouns in English and French. The direct object is the noun group just after the verb if there is no preposition in-between. It is marked with the accusative case in German.

We will now write a small set of DCG rules to encode this simplified description. The structure of a simple sentence consists of a subject noun group, a verb group in the active voice, and an object noun group. It corresponds to the rules:

```
sentence(S, V, O) -->
  subject(S),
  verb(V, active),
  object(O),
  ['.'].

subject(S) --> noun_group(S).

object(O) --> noun_group(O).

verb(V, active) --> verb_group(V, active).
```

We must modify the description of verbs in the terminal symbols to add an active/passive feature.

9.7.2 Extracting Other Groups

The Subject–Verb–Object relation is the core of most sentences. However, before extracting them, it is useful to skip some groups between them. Among the groups, there are prepositional phrases and embedded clauses, as in the two sequences: subject, prepositional groups, verb and subject, relative clause, verb.

A prepositional group can be defined as a preposition followed by a noun group. Using a DCG rule, this translates into:

```
prep_group([P | [NG]]) --> prep(P), ng(NG).
```

The detection of prepositional groups is a new layer in the cascade structure. A new rule describing `ng` as a terminal symbol is then necessary to be consistent with the noun groups detected before:

```
ng(['<NG>'| NG]) --> [['<NG>'| NG]].
```

Embedded clauses can be relative, infinitive, or subordinate. Here we will only consider relative and infinitive clauses that may modify a noun.

A relative clause is an embedded sentence whose subject or object has been replaced with a relative pronoun. The relative pronoun comes in front of the clause. For simple clauses, this translates into two rules:

```
%Relative clause: The relative pronoun is the subject
relative_clause(RC) -->
  relative_pronoun(R),
  vg(VG),
  ng(NG),
  {append([R | [VG]], [NG], RC)}.

% Relative clause: The relative pronoun is the object
relative_clause(RC) -->
  relative_pronoun(R),
  ng(NG),
  vg(VG),
  {append([R | [NG]], [VG], RC)}.
```

An infinitive clause is simply a verb phrase set in the infinitive. For simple examples, it translates into a verb group possibly followed by a noun group, where the verb group begins with *to*:

```
infinitive_clause([['<VG>', to | VG], NG]) -->
  vg(['<VG>', to | VG]),
  ng(NG).
infinitive_clause([['<VG>', to | VG]]) -->
  vg(['<VG>', to | VG]).
```

Like for noun groups, we must describe verb groups as a terminal symbol:

```
vg(['<VG>' | VG]) --> [['<VG>' | VG]].
```

Now let us write the rules to describe the modifiers and annotate them:

```
modifier(MOD) -->
  prep_group(PG),
  {append(['<PG>' | PG], ['</PG>'], MOD)}.
modifier(MOD) -->
  relative_clause(RC),
  {append(['<RC>' | RC], ['</RC>'], MOD)}.
modifier(MOD) -->
  infinitive_clause(IC),
  {append(['<IC>' | IC], ['</IC>'], MOD)}.
```

Finally, we write the detector to run the program:

```
modifier_detector(In, Out) :-
  word_stream_modifier(Beginning, Group, End, In,[]),
  modifier_detector(End, Rest),
  append(Beginning, [Group], Head),
  append(Head, Rest, Out).
modifier_detector(End, End).
```

```
word_stream_modifier(Beginning, Group, End) -->
  beginning(Beginning),
  modifier(Group),
  end(End).
```

Let us apply these rules on the first sentence of the *Los Angeles Times* excerpt. We must add prepositions and a relative pronoun to the vocabulary:

```
prep(of) --> [of].
prep(with) --> [with].

relative_pronoun(that) --> [that].
```

And the query yields:

```
?- modifier_detector([[<NG>, critics, </NG>],
[<VG>, question, </VG>], [<NG>, the, ability,
</NG>], of, [<NG>, a, relatively, small, group,
</NG>], of, [<NG>, big, integrated, prime,
contractors, </NG>], [<VG>, to, maintain, </VG>],
[<NG>, the, intellectual, diversity, </NG>], that,
[<VG>, formerly, provided, </VG>], [<NG>, the,
pentagon, </NG>], with, [<NG>, innovative,
weapons, </NG>], with, [<NG>, fewer, design,
staffs, </NG>], working, on, [<NG>, military,
problems, </NG>], [<NG>, the, solutions, </NG>],
[<VG>, are, </VG>], likely, [<VG>, to, be, </VG>],
less, varied], O).

O = [[<NG>, critics, </NG>], [<VG>, question,
</VG>], [<NG>, the, ability, </NG>], [<PG>, of,
[<NG>, a, relatively, small, group, </NG>],
</PG>], [<PG>, of, [<NG>, big, integrated, prime,
contractors, </NG>], </PG>], [<IC>, [<VG>, to,
maintain, </VG>], [<NG>, the, intellectual,
diversity, </NG>], </IC>], [<RC>, that, [<VG>,
formerly, provided, </VG>], [<NG>, the, pentagon,
</NG>], </RC>], [<PG>, with, [<NG>, innovative,
weapons, </NG>], </PG>], [<PG>, with, [<NG>,
fewer, design, staffs, </NG>], </PG>], working,
on, [<NG>, military, problems, </NG>], [<NG>, the,
solutions, </NG>], [<VG>, are, </VG>], likely,
[<IC>, [<VG>, to, be, </VG>], </IC>], less, varied]
```

Prepositional phrases and relative clauses are labeled with `<PG>`, `<IC>`, and `<RC>` tags. Remaining groups are `[<NG>, critics, </NG>]`, `[<VG>,ques-`

tion, </VG>], and [<NG>, the, ability, </NG>], which correspond to heads of the subject, main verb, and the object of the sentence.

9.8 An Annotation Scheme for Groups in French

The PEAS initiative (Protocole d'évaluation des analyseurs syntaxiques, Gendner et al. 2003) defines an XML annotation scheme for syntactic groups (chunks) and functional relations for French. It was created to reconcile different annotation practices and enable the evaluation of parsers. We present here the chunk annotation that applies to continuous, nonrecursive constituents.

The PEAS annotation identifies six types of chunks:

1. verb groups (noyau verbal): <NV></NV>
2. noun groups (groupe nominal): <GN></GN>
3. prepositional groups: <GP></GP>
4. adjective groups: <GA></GA>
5. adverb groups: <GR></GR>
6. verb groups starting with a preposition: <PV></PV>

The sentence *En quelle année a-t-on vraiment construit la première automobile?* 'Which year the first automobile was really built?' is bracketed as

<GP> En quelle année </GP> <NV> a –t-on </NV> <GR> vraiment </GR> <NV> construit </NV> <GN> la première automobile</GN> ?

The annotation first identifies the sentence in the corpus:

```
<E id="2"> En quelle année a -t-on vraiment
construit la première automobile ? </E>
```

The second step tokenizes the words:

```
<DOCUMENT fichier="Guide.1">
  <E id="E2">
    <F id="E2F1">En</F>
    <F id="E2F2">quelle</F>
    <F id="E2F3">année</F>
    <F id="E2F4">a</F>
    <F id="E2F5">-t-on</F>
    <F id="E2F6">vraiment</F>
    <F id="E2F7">construit</F>
    <F id="E2F8">la</F>
    <F id="E2F9">première</F>
    <F id="E2F10">automobile</F>
    <F id="E2F11">?</F>
  </E>
</DOCUMENT>
```

using the DTD

```
<!ELEMENT DOCUMENT ( E+ ) >
<!ATTLIST DOCUMENT fichier NMTOKEN #REQUIRED >
<!ELEMENT E ( F)+>
<!ATTLIST E id NMTOKEN #REQUIRED >
<!ELEMENT F ( #PCDATA ) >
<!ATTLIST F id ID #REQUIRED >
```

The third step brackets the groups:

```
<DOCUMENT fichier="Guide.1.ph1.IR.xml">
  <E id="E2">
    <Groupe type="GP" id="E2G1">
     <F id="E2F1">En</F>
     <F id="E2F2">quelle</F>
     <F id="E2F3">année</F>
    </Groupe>
    <Groupe type="NV" id="E2G2">
      <F id="E2F4">a</F>
      <F id="E2F5">-t-on</F>
    </Groupe>
    <Groupe type="GR" id="E2G3">
     <F id="E2F6">vraiment</F>
    </Groupe>
    <Groupe type="NV" id="E2G4">
      <F id="E2F7">construit</F>
    </Groupe>
    <Groupe type="GN" id="E2G5">
      <F id="E2F8">la</F>
      <F id="E2F9">première</F>
      <F id="E2F10">automobile</F>
    </Groupe>
      <F id="E2F11">?</F>
  </E>
</DOCUMENT>
```

using the DTD

```
<!ELEMENT DOCUMENT ( E+ ) >
<!ATTLIST DOCUMENT fichier NMTOKEN #REQUIRED >
<!ELEMENT E ( F | Groupe )+>
<!ATTLIST E id NMTOKEN #REQUIRED >
<!ELEMENT Groupe ( F+ ) >
<!ATTLIST Groupe id ID #REQUIRED >
<!ATTLIST Groupe type ( GA | GN | GP | GR | NV |
```

```
PV ) #REQUIRED >
<!ELEMENT F ( #PCDATA ) >
<!ATTLIST F id ID #REQUIRED >
```

9.9 Application: The FASTUS System

9.9.1 The Message Understanding Conferences

The FASTUS system was designed at the Stanford Research Institute to extract information from free-running text (Hobbs et al. 1997, Appelt et al. 1993). It was implemented within the course of the Message Understanding Conferences (MUCs) that we introduced in Sect. 9.3.2. MUCs were organized to measure the performance of information extraction systems. They were held regularly until MUC-7 in 1997, under the auspices of DARPA, an agency of the US Department of Defense. The performances improved dramatically in the beginning and then stabilized. DARPA discontinued the competitions when it realized that the systems were no longer improving.

MUCs are divided into a set of tasks that have changed over time. The most basic task is to extract people and company names. The most challenging one is referred to as information extraction. It consists of the analysis of pieces of text ranging from one to two pages, the identification of entities or events of a specified type, and filling a predefined template with relevant information from the text. Information extraction then transforms free texts into tabulated information. Here is an example news wire cited by Hobbs et al. (1997) and its corresponding filled template drawn from MUC-3 (Table 9.9):

> *San Salvador, 19 Apr 89 (ACAN-EFE) – [TEXT] Salvadoran President-elect Alfredo Cristiani condemned the terrorist killing of Attorney General Roberto Garcia Alvarado and accused the Farabundo Marti National Liberation Front (FMLN) of the crime.*
>
> *...*
>
> *Garcia Alvarado, 56, was killed when a bomb placed by urban guerrillas on his vehicle exploded as it came to a halt at an intersection in downtown San Salvador.*
>
> *...*
>
> *Vice President-elect Francisco Merino said that when the attorney general's car stopped at a light on a street in downtown San Salvador, an individual placed a bomb on the roof of the armored vehicle.*
>
> *...*
>
> *According to the police and Garcia Alvarado's driver, who escaped unscathed, the attorney general was traveling with two bodyguards. One of them was injured.*

Table 9.9. A template derived from the previous text. After Hobbs et al. (1997).

Template slots	Information extracted from the text
Incident: Date	19 Apr 89
Incident: Location	El Salvador: San Salvador (city)
Incident: Type	Bombing
Perpetrator: Individual ID	*urban guerrillas*
Perpetrator: Organization ID	*FMLN*
Perpetrator: Organization confidence	Suspected or accused by authorities: *FMLN*
Physical target: Description	*vehicle*
Physical target: Effect	Some damage: *vehicle*
Human target: Name	*Roberto Garcia Alvarado*
Human target: Description	*Attorney general*: *Roberto Garcia Alvarado* *driver* *bodyguards*
Human target: Effect	Death: *Roberto Garcia Alvarado* No injury: *driver* Injury: *bodyguards*

9.9.2 The Syntactic Layers of the FASTUS System

FASTUS uses partial parsers that are organized as a cascade of finite-state automata. It includes a tokenizer, a multiword detector, and a group detector as first layers. The detector uses a kind of longest match algorithm. Verb groups are tagged with active, passive, gerund, and infinitive features. Then FASTUS combines some groups into more complex phrases. Complex groups include notably the combination of adjacent nouns groups (appositives):

The joint venture, Bridgestone Sports Taiwan Co.
First noun group Second noun group

of noun groups separated by prepositions *of* or *for* (noun postmodifiers):

The board of directors

and of noun group conjunctions:

a local concern and a Japanese trading house

Complex groups also include verb expressions such as:

plan to set up
announced a plan to form

Such complex groups can be found in French and German, where they have often a one-word counterpart in another language:

mettre une lettre à la poste 'mail a letter'
jemanden kennen lernen 'know somebody'

They merely reduce to a single semantic entity that is formed differently from one language to another.

FASTUS' upper layers then deal with grammatical functions and semantics. FASTUS attempts to reduce sentences to a basic pattern consisting of a subject, a verb, and an object. Finally, FASTUS assigns a sense to some groups by annotating them with a semantic category such as company, product, joint venture, location, and so on.

SRI first used a full parser called TACITUS, and FASTUS as a front-end to offload it of some tasks. Seeing the excellent results and speed of FASTUS, SRI completely replaced TACITUS with FASTUS. It had a considerable influence on the present evolution of parsing techniques. FASTUS proved that the local and cascade approach was more efficient and much faster than other global analyses for information extraction. It had a considerable number of followers.

9.9.3 Evaluation of Information Extraction Systems

The MUCs introduced a metric to evaluate the performance of information extraction systems using three figures: recall, precision, and the F-measure. This latter metric, originally borrowed from library science, proved very generic to summarize the overall effectiveness of a system. It has been used in many other fields of language processing since then.

To explain these figures, let us stay in our library and imagine we want to retrieve all the documents on a specific topic, say *morphological parsing*. An automatic system to query the library catalog will, we hope, return some of them, but possibly not all. On the other hand, everyone who has searched a catalog knows that we will get irrelevant documents: *morphological pathology*, *cell morphology*, and so on. Table 9.10 summarizes the possible cases into which documents fall.

Table 9.10. Documents in a library returned from a catalog query and split into relevant and irrelevant books.

	Relevant documents	**Irrelevant documents**
Retrieved	A	B
Not retrieved	C	D

Recall measures how much relevant information the system has retrieved. It is defined as the number of relevant documents retrieved by the system divided by number of relevant documents in the library:

$$\text{Recall} = \frac{A}{A \cup C}.$$

Precision is the accuracy of what has been returned. It measures how much of the information is actually correct. It is defined as the number of correct documents returned divided by the total number of documents returned.

$$\text{Precision} = \frac{A}{A \cup B}.$$

Recall and precision are combined into the F**-measure**, which is defined as the harmonic mean of both numbers:

$$F = \frac{2PR}{P + R}.$$

The F-measure is a composite metric that reflects the general performance of a system. It does not privilege precision at the expense of recall, or vice versa. An arithmetic mean would have made it very easy to reach 50% using, for example, very selective rules with a recall of 100 and a precision of 0.

Using a β-coefficient, it is possible to give an extra weight to either precision, $\beta > 1$, or recall, $\beta < 1$, however:

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}.$$

Finally, a **fallout** figure is also sometimes used that measures the proportion of irrelevant documents that have been selected.

$$\text{Fallout} = \frac{B}{B \cup D}.$$

9.10 Further Reading

Partial or shallow parsing has attracted a considerable interest in the 1990s and has renewed the field. This is largely due to the simplicity of the methods it involves. It is also due to its recent successes in information extraction competitions such as the MUCs (see, for instance, MUC-5 (1993)). The definition of the Named Entity annotation can be read from the MUC-7 web page: www.itl.nist.gov/iaui/894.02/-related_projects/muc/proceedings/muc_7_toc.html.

One of the first partial parsing systems is due to Ejerhed (1988). Appelt et al. (1993) describe with eloquence the history and structure of the FASTUS system. Abney (1994) has surveyed partial parsing with much detail and provides a comprehensive bibliography of 200 papers! Roche and Schabes (1997) and Kornai (1999) are other sources for partial parsing techniques.

Partial parsing was the topic of a series of conferences on Computational Natural Language Learning (CoNLL). Each year, the CoNLL conference organizes a "shared task" where it provides an annotated training set. Participants can train their system on this set, evaluate it on a common test set, and report a description of their algorithms and results in the proceedings. In 1999, the shared task was dedicated to noun group chunking (http://www.cnts.ua.ac.be/conll99/npb/); in 2000, it was extended to other chunks (http://www.cnts.ua.ac.be/conll2000/chunking/), and in 2001, the topic was the identification of clauses (http://www.cnts.ua.ac.be/conll2001/clauses/). The

CoNLL sites and proceedings are extremely valuable as they provide data sets, annotation schemes, a good background literature, and an excellent idea of the state of the art.

The development of partial parsing has been driven by applications without concern for a specific linguistic framework. This is a notable difference from many other areas of language processing, where theories abound. Functional and dependency grammars (Tesnière 1966, Mel'cuk 1988) may offer background and provide readers with a sound theory perspective.

Exercises

9.1. Complement the ELIZA program and add possible templates and answers.

9.2. Implement a multiword detector to detect dates in formats such as in English: 04/04/1997 or April 4, 1997, and in French: 20/04/1997 or 20 avril 1997.

9.3. Complement the noun group grammar and write down the vocabulary to recognize the noun groups of the text:
The big tobacco firms are fighting back in the way that served them well for 40 victorious years, pouring their wealth into potent, relentless legal teams. But they are also starting to talk of striking deals – anathema for those 40 years, and a sure sign that, this time, victory is less certain.
(*The Economist*, no. 8004, 1997).

9.4. See Exercise 9.3; do the same for verb groups.

9.5. Write a noun group grammar to parse the French text:
Les limites de la régulation de l'audiovisuel sont clairement définies aujourd'hui par la loi. C'est le principal handicap du CSA : son champ d'action est extrêmement limité. Alors que la télévision numérique prend son essor, le CSA, dont les compétences s'arrêtent au câble et à l'hertzien, n'a aucun pouvoir pour contrôler ou sanctionner la télévision de demain formée par les chaînes satellitaires.
(*Le Monde*, mercredi 3 septembre 1997).

9.6. See Exercise 9.5; do the same for verb groups.

9.7. Write a noun group grammar to parse the German text:
Die Freude über das neue große Europa wird also nur von kurzer Dauer sein. Die Probleme, die sich aus einer Union der 25 ergeben, dürften dagegen Regierungen und Völker über Jahre hinweg in Atem halten. Zunächst einmal wird es alles andere als leicht sein, die 10 neuen Mitgliedsstaaten zu integrieren. Die Migrationswellen, die von ihnen ausgehen, werden der „alten" EU reichlich Kopfschmerzen bereiten. Vor allem stellt sich der Entscheidungsprozess innerhalb der Union künftig noch weitaus schwieriger dar.
(*Die Zeit*, 30 April 2004).

9.8. See Exercise 9.7; do the same for verb groups.

9.9. Adapt the Prolog code of Brill's tagger from Chap. 6 so that it can detect noun groups.

9.10. Write rules that detects some complex noun groups:
- Adjacent nouns groups linked by the prepositions *of* or *for*
- Noun group conjunctions

9.11. Find press wires on football matches on the Web and implement a program to retrieve teams' names and final scores. Use a base of football team names, and adopt a cascaded architecture.