Natural Language Processing & Applications
Syntax

1 Introduction to the syntax of English

Recall that in my definition, **grammar** includes the rules governing word formation (**morphology**) as well as those governing sentence formation (**syntax**). English has a fairly simple morphology, but a very complex syntax. How complex has been revealed by recent linguistic research, particularly since the pioneering work of Noam Chomsky. In the rest of this module, only VERY simple sentences will be considered in any detail. More specifically I will concentrate on:

- Sentences which contain a single main verb (with or without auxiliaries). Thus I will not consider sentences containing 'subordinate clauses', such as *The man who came yesterday stayed for dinner*, which can be analysed as a combination of *The man came yesterday* and *The man stayed for dinner*.
- Sentences whose main verb expresses an action, rather than for example a perception or an identity. Such sentences can usually be questioned by asking *What did X do (to Y)?* Thus *Lions kill deer* is an action sentence, since I can ask *What do lions do to deer?* On the other hand, *lions like eating deer* is not an action sentence since it does not describe any action that lions perform, and can't be queried by asking *What do lions do?*

The first step in constructing a grammar for the syntax of the English language is to divide words into **categories** ('parts of speech' in old-fashioned terminology). There are two kinds of evidence which suggest that words fall into distinct categories. The first is **morphological**, in particular inflectional morphology. Inflections alter the **properties** of words (e.g. changing number from singular to plural), without changing the lexeme. Which inflection a word has reflects its category. For example, only some words can have *-ed* added to them (changing tense from present to past). These words can be put into one category (verb).

The second kind of evidence is **distributional**. For example, all the words which can fill the blank in the following sentence can be put into another category (noun).

He has no __

Using a combination of morphological and distributional evidence, it is possible to assign an English word to one (or more) of a number of categories. However, there is rarely one definitive test.

The main categories which I shall use are listed below.

- **Noun** (N). Morphologically, many (but not all) English nouns can form plurals by adding *-s*. Distributionally, many (but not all) nouns can fill in the blank in the sentence:

  He has no __.

  Car, *friend* and *idea* fall in the noun category by both these tests: all three can have *-s* added and all three can complete the above sentence. 'Proper nouns' such as *John* or *Chomsky* don't meet either of these tests (but do meet others).

  In English, nouns are either singular (e.g. *car*, *child*) or plural (e.g. *cars*, *children*). In other Indo-European languages, nouns can have gender and case. Gender is often shown by the choice of the word for *the*. Thus in French, *the woman* is *la femme*, because *la* goes only with feminine words, whereas *the boy* is *le garçon*, because *le* goes only with masculine words. (For a discussion of case, see pronouns below.)

- **Verb** (V). Morphologically, almost all English verbs can have *-ing* added. Distributionally, many verbs can occur in one or other of the positions marked __ below to form a complete sentence.

  They can __.
  They can __ them.

  Stay, *cry* and *see* fall into the verb category by both these tests. In English (and other Indo-European languages), verbs have person and number. Thus *am* is first person singular because it agrees only with *I* which is the first person singular. English verbs other than *be*

change only in the third person singular of the present tense: *I/you/we/they eat*, but *he/she/it eat**s***. Verbs may also show morphological changes with tense. Thus *kill* is present tense, *killed* is past tense.

- **Adjective** (A). Morphologically, many adjectives can have *-er* or *-est* added (although this is not exactly inflectional morphology). Distributionally, many adjectives can complete the sentence:

  They are very __.

  Tall, *pretty* and *kind* are adjectives by both tests; *careful* only by the second. In many Indo-European languages, but not English, adjectives can have number, gender and case, like nouns.

- **Preposition** (P). Morphologically, prepositions are invariant. It's not easy to give a simple distributional test for prepositions. A word which is not a verb but can immediately precede the word *them* is usually a preposition. For example:

  I am against them.
  She walked into them.
  He painted a picture of them.

  By this test, *against*, *into* and *of* are prepositions.

- **Determiner** (DET). English determiners have no regular morphological variations. Distributionally, they can fill the position marked __ below.

  First speaker: *What are you looking for?*
  Second speaker: __ *hat.*

  Thus *a*, *the*, *this* or *my* are determiners. Semantically, determiners 'determine' which entity is involved. In the example above, *a hat* means 'any old hat, no particular hat'; *the hat* means 'the hat that we both know about'. In many Indo-European languages, determiners change with number, gender and case. In English, a few show changes with number, e.g. *this car*, *these cars.*

- **Pronoun** (PRN). Morphologically, most English pronouns change 'case'. In the sentence *John likes Mary but Mary doesn't like John*, the (proper) nouns *John* and *Mary* don't change depending on who likes whom. However, if we substitute the pronouns *he* and *she* for *John* and *Mary*, the sentence is incorrect (in SEE): **He likes she but she doesn't like he*. Neither can we consistently substitute *him* and *her*: **Him likes her but her doesn't like him*. *He* and *she* must be used for the liker, *him* and *her* for the liked: *He likes her but she doesn't like him.*

  Distributionally, a pronoun can substitute for a noun which is not preceded by a determiner or adjective(s). English pronouns have only two cases: one used for the subject of the sentence (i.e. when immediately followed by a verb in a 'normal' sentence) and one used elsewhere. Traditionally the cases are called 'nominative' and 'accusative'. Some other Indo-European languages have more cases, and nouns and adjectives may also change with case.

- **Auxiliary** (AUX). Historically, English auxiliaries were derived from verbs and some still show verb-like morphology (adding *-s* for example), while others are invariant. Distributionally, auxiliaries are immediately followed by a verb in a positive statement, can be inverted to form a question and can be immediately followed by *not* (or some contracted form like *-n't*) to form a negative. Thus:

  I can speak English. Can I speak English? I can't speak English.
  He has spoken English. Has he spoken English? He has not spoken English.
  *I speak English. *Speak I English? *I speak not English.*

  Here *can* and *has* are auxiliaries whereas *speak* and its variant *spoken* are verbs.

The table below summarizes these categories and the morphological changes each can show.

E means that English almost always shows morphological changes; (E) that it is sometimes does; * that most other Indo-European languages show morphological changes but English doesn't.

	Case	Gender	Number	Person	Tense
Determiner	*	*	(E)		
Adjective	*	*	*		
Noun	*	*	E		
Pronoun	E	E	E	E	
Verb			E	E	E
Auxiliary			(E)	(E)	E
Preposition					

Note that there are other categories (such as conjunctions, adverbs, complementizers); also categories can be subdivided (e.g. proper nouns versus common nouns, modal versus non-modal auxiliaries). Words can belong to more than one category. For example, *hides* is a noun in *The ducks are easier to see from the hides* and a verb in *The duck hides her eggs*. *Is* is an auxiliary in *John is speaking English*, but a verb in *John is English*.

Within a sentence, words appear to form groups or **phrases**. Consider these sentences:

I saw men.
I saw the men.
I saw the angry men.
I saw the angry men with banners.
I saw the angry men with their banners.
I saw the angry men with their black banners.

In the first sentence, *men* is a noun. In the other sentences, all the words which replace *men* form a single phrase centred around (or 'headed by') this noun. If we allow a single noun to constitute a noun phrase (NP), then we can say that all the sentences have the structure I_{PRN} saw_{V} $—_{NP}$. If a single pronoun is also allowed to form a noun phrase, then *I saw them* also fits this pattern.

The last three sentences above contain a prepositional phrase (PP), i.e. a phrase headed by a preposition – in this case the preposition *with*. We can analyse *I saw the angry men with their black banners* as I_{PRN} saw_{V} (*the angry men* (*with their black banners*)$_{PP}$)$_{NP}$. The PP in turn contains the NP *their black banners*, giving I_{PRN} saw_{V} (*the angry men* (*with* (*their black banners*)$_{NP}$)$_{PP}$)$_{NP}$.

The sentence *They banged the van with their black banners* requires a different analysis, namely *They*$_{PRN}$ *banged*$_{V}$ (*the van*)$_{NP}$ (*with their black banners*)$_{PP}$. One way of deciding on the bracketing is to note that a pronoun is really a pro-(noun phrase). If I say *I saw the angry men with their black banners* and you reply *I saw them too*, the pronoun *them* refers to the WHOLE NP *the angry men with their black banners*, whereas if I say *They banged the van with their black banners* and you reply *Yes, they banged it really hard*, the pronoun *it* refers only to *the van*.

The final kind of phrase I want to consider is less obvious (to me anyway). Consider the sentence *Careful owners should keep their cars in their garages*. Given the discussion so far, this can be analysed as (*Careful owners*)$_{NP}$ *should keep* (*their cars*)$_{NP}$ (*in their garages*)$_{PP}$. It's tempting to put *should* and *keep* together into some kind of phrase. However, linguists usually prefer to form a larger verb phrase, consisting of any auxiliaries which may be present, the verb itself, and any complements – i.e. any NPs or PPs associated with the meaning of the verb. Thus *Careful owners should keep their cars in their garages* will be analysed as (*Careful owners*)$_{NP}$ (*should keep their cars in their garages*)$_{VP}$. One justification for this approach is the observation that auxiliaries in English can stand for the whole of a VP when defined in this way. For example, in the sentence *Careful owners should keep their cars in their garages, but they don't*, the auxiliary *don't* stands for the whole VP *don't keep their cars in their garages*.

Verb phrases covered in this module have the structure: [AUX] V VerbComps, i.e. an optional auxiliary, a verb, and then some verb complements. There is a relatively small set of valid verb complements, including the following:

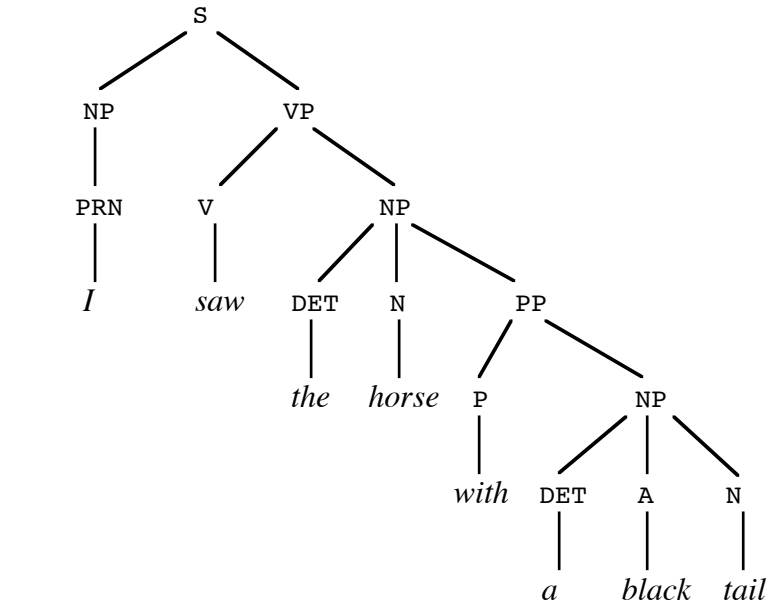
- nothing, e.g. ((*The man*)$_{NP}$ (*is*$_{AUX}$ *sleeping*$_{V}$)$_{VP}$)$_{S}$
- NP, e.g. ((*The woman*)$_{NP}$ (*ate*$_{V}$ (*the red tomatoes*)$_{NP}$)$_{VP}$)$_{S}$
- PP, e.g. ((*The girl*)$_{NP}$ (*sat*$_{V}$ (*on the sofa*)$_{PP}$)$_{VP}$)$_{S}$

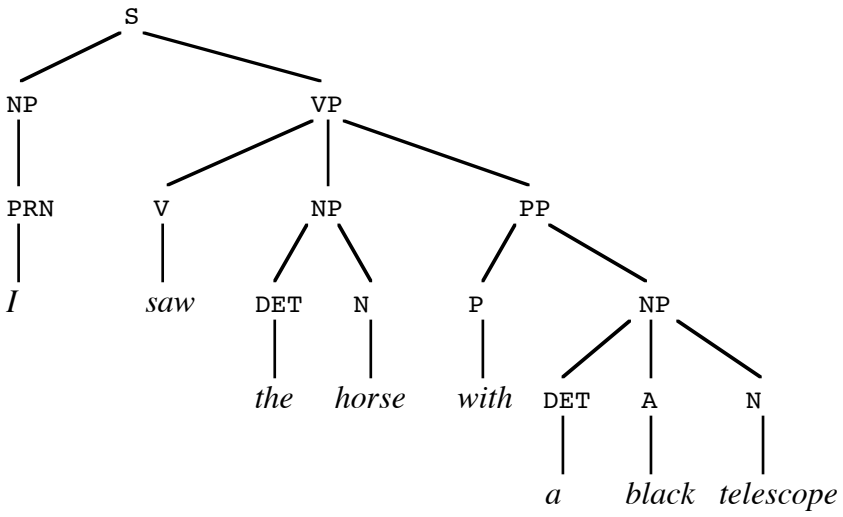
- NP PP, e.g. $((She)_{NP}\ (wrote_{V}\ (a\ letter)_{NP}\ (to\ her\ sister)_{PP})_{VP})_{S}$
- NP NP, e.g. $((Nobody)_{NP}\ (was_{AUX}\ telling_{V}\ (the\ police)_{NP}\ (the\ truth)_{NP})_{VP})_{S}$

To summarize:

- Words fall into a limited number of categories, which can be defined by morphological and distributional criteria.
- Sentences are composed of groups of words making up phrases. Phrases may contain other phrases. Phrases fall into a small set of types, the most important of which are NP (noun phrase), PP (prepositional phrase) and VP (verb phrase). Every phrase has a 'head' word which defines its type. An simple English sentence S is composed of a noun phrase followed by a verb phrase.

Note carefully the difference between sentences like *I saw the horse with a black tail* and *I saw the horse with a black telescope*. In the first sentence, the verb has one complement, the NP *the horse with a black tail*. In the second sentence, the verb has two complements, the first the NP *the horse*, the second the PP *with a black telescope*.[1] In tree form:





Given that English, at least, can be analysed in this way, the next question is whether this analysis can be made sufficiently formal to be handled by a computer program. For simple sentences, the answer is that it can.

---

[1] There is no unique way of describing phrases (and hence of constructing grammars to generate them). Many linguists insist on strictly binary branching, so would not use the analysis presented here.

2 A Formal Grammar

The syntax of a language can be described by a 'formal grammar' which consists of:

- A set of non-terminal symbols. In the notation used here, non-terminal symbols appear in `Courier` font. Those starting with capital letters (e.g. `S`, `NP`) can be expanded further by the grammar. Those starting with lower-case letters (e.g. `verb`, `det`) cannot be expanded further by the grammar, but must be replaced by actual words. They are thus 'pre-terminal' symbols.
- A start symbol – one of the non-terminal symbols.
- A set of terminal symbols. In describing the syntax of a sentence, these are words (which in the notation used here are italicized). The special symbol ø stands for 'nothing'.
- A set of productions (also called re-write rules).

In productions, → means 'can be re-written as', | means 'or'.

A grammar which describes a small subset of English sentences is given below. The start symbol is `S`.[2]

```
S → NP VP

NP → det SNP | SNP
SNP → noun | adj SNP

VP → verb VerbComps
VerbComps → ø | NP | PP | NP PP

PP → prep NP

det → the | this | that | my
adj → black | young | happy
noun → cat | man | table
prep → on
verb → killed | put | slept
```

Nonterminal symbols:

`S` = sentence
`NP` = noun phrase
`SNP` = simple noun phrase
`VP` = verb phrase
`VerbComps` = verb complements
`PP` = prepositional phrase

`det` = determiner
`adj` = adjective
`noun`
`prep` = preposition
`verb`

Generating sentences from this grammar means beginning with the start symbol `S` and successively re-writing non-terminal symbols until only terminal symbols remain. Where there are alternatives, any one can be chosen. For example:

```
S → NP VP → det SNP VP → det adj SNP VP → det adj adj SNP VP →
det adj adj noun VP → det adj adj noun verb VerbComps →
det adj adj noun verb NP PP → det adj adj noun verb det SNP PP →
det adj adj noun verb det noun PP →
det adj adj noun verb det noun prep NP →
det adj adj noun verb det noun prep det SNP →
det adj adj noun verb det noun prep det adj SNP →
det adj adj noun verb det noun prep det adj noun →→
the happy young man put the cat on the black table
```

(Note the use of a 'double arrow' to indicate steps have been left out.) Some other examples of sentences which this grammar generates are:

The young man killed the black cat.
The young cat slept.
**The young young young young cat slept.*
**The young table slept.*
**Young table slept.*
**The man slept the cat.*
**The man put the cat.*

Note that all these sentences are valid according to the grammar above. The asterisks indicate whether they are invalid in 'standard' English. I regard only the last three of these sentences as SYNTACTICALLY invalid, the previous two starred sentences being semantically invalid.

---

[2] Note that this is only A grammar. Many alternative grammars can describe the same subset of English; in this module, I will not attempt to evaluate or choose between grammars.

Recognizing sentences as valid according to a grammar involves using productions backwards until the start symbol is obtained. For example:

the man killed the black cat ←← det noun verb det adj noun ←
det noun verb det adj SNP ← det noun verb det SNP ←
det SNP verb det SNP ← det SNP verb NP ← det SNP verb VerbComps ←
det SNP VP ← NP VP ← S

Recognition is more difficult than generation, since the correct alternative must be chosen. For example, suppose we reached the stage det SNP VP and decided that the SNP was generated from NP, giving the derivation:

det SNP VP ← det NP VP ← det S

The problem now is that the sequence det S cannot have been derived from the start symbol. Hence we need to BACKTRACK and use an alternative derivation (i.e. that det SNP was generated from NP). Only if all backtracks fail can we decide that the sentence is not valid according to the grammar. Thus in general recognition requires a search process, backtracking when a wrong path is chosen.

The reason I have chosen to write 'pre-terminal' symbols in lower-case to distinguish them from other non-terminal symbols is that in practice we often don't want to include actual words (i.e. terminal symbols) in the grammar. Instead we handle pre-terminals by some kind of dictionary look-up. We can express this semi-formally by re-writing a production like:

det → *the* | *this* | *that* | *my*

as

det → {any word stored in the lexicon as a det}

or just

det → {det}

where braces {} are used to enclose informal text. The last production should be read as 'det can be re-written as any word stored in the lexicon as a det'. To show how words are stored in the lexicon I will write expressions of the form:

the : det
killed : verb

where a list of any number of pieces of information can appear after the colon.

3 Types of Grammar

The grammar given above is an example of a context-free phrase-structure grammar (CFG), since the left-hand side of every production contains a SINGLE NON-TERMINAL SYMBOL. There is a well-developed mathematical theory of grammars (which owes a great deal to the linguist Noam Chomsky). Grammars can be arranged into a hierarchy of types. Only two need concern us here: CFGs and context-sensitive phrase-structure grammars (CSGs).

CSGs are needed in phonology (as we saw earlier). Consider the following grammar which generates spoken plurals for a subset of English nouns.

PluralWord → Phone pluralMorpheme | Phone PluralWord
Phone → voicedPhone | voicelessPhone
voicedPhone pluralMorpheme → voicedPhone [z]
voicelessPhone pluralMorpheme → voicelessPhone [s]
voicedPhone → [b] | [d] | [g] | [æ] | [ɒ] | ...
voicelessPhone → [p] | [t] | [k] | ...

Using this grammar, we can generate the correct pronunciation for the plural word *cats*:

PluralWord → Phone PluralWord → Phone Phone PluralWord →
Phone Phone Phone pluralMorpheme →→
voicelessPhone voicedPhone voicelessPhone pluralMorpheme →→
[k] [æ] voicelessPhone pluralMorpheme →
[k] [æ] voicelessPhone [s] → [k] [æ] [t] [s] = [kæts]

Whereas [dɒgs] cannot be recognized using this grammar:

[d] [ɒ] [g] [s] ←← `Phone Phone voicedPhone` [s]

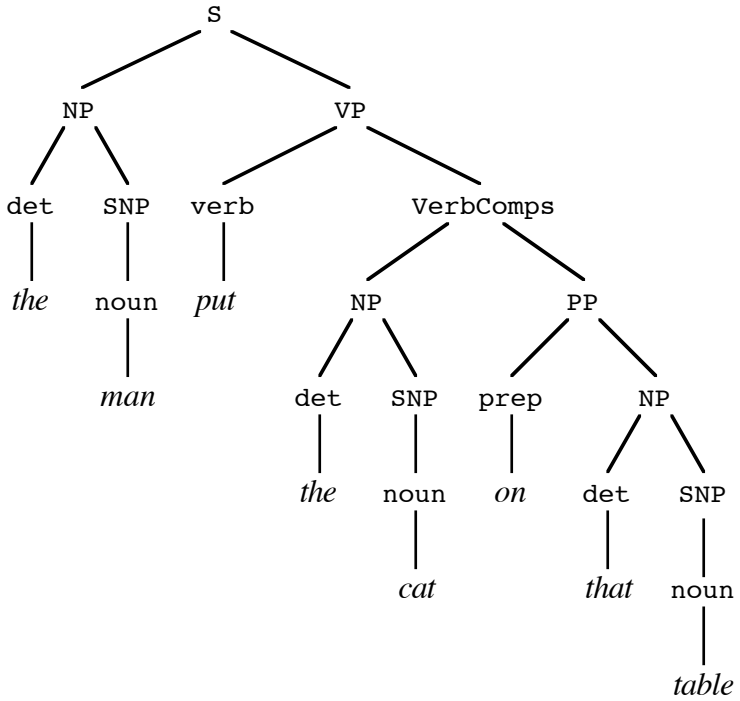
No further progress is possible.

This is a context-sensitive grammar, since the left-hand side of at least one production contains MORE THAN ONE SYMBOL; it is a phase-structure grammar since ONLY ONE OF THESE SYMBOLS IS REPLACED by the production. The general consensus among linguists seems to be that processing the syntax of NLs requires only varieties of CFGs, whereas phonological processing requires CSGs.

The output of a PSG (CF or CS) can be described by a TREE, since the right-hand side of each production has a single 'parent': the nonterminal which is replaced by the production. The diagram shows how our sample PSG generates the sentence *The man put the cat on that table*.

An alternative to the diagram is to 'bracket' the sentence as we did earlier, e.g.:

((*the man*)$_{\texttt{NP}}$ (*put* ((*the cat*)$_{\texttt{NP}}$ (*on* (*that table*)$_{\texttt{NP}}$)$_{\texttt{PP}}$)$_{\texttt{VerbComps}}$)$_{\texttt{VP}}$)$_{\texttt{S}}$



Analysing a sentence in this way goes beyond mere recognition and is termed **parsing**. Note that given a PSG, we can describe appropriate sequences of terminal symbols as 'being' the non-terminal symbol. For example, *the black cat* is an `NP` (= noun phrase) in the sense that it is generated from an `NP`.

4 Handling Agreement

There are a number of problems with the grammar I have defined so far. One is that if we add plural nouns and present tense verbs to the lexicon, the grammar does not handle agreement.

cats `: noun`
sleep `: verb`
sleeps `: verb`
the `: det`
that `: det`
`...`

`S` →→ *the cat sleep*
`S` →→ *that cats sleeps*

The solution to this problem is to introduce VARIABLES into the grammar to represent the properties of the words (as shown by the inflections). Variables are written in parentheses

after a nonterminal, i.e. as arguments to the nonterminal. I will always start variables with a capital letter (the Prolog convention). A special variable symbol is needed to mean 'don't care': this will be the underscore _. Provided we don't introduce pronouns (such as *I* or *you*), English determiners, nouns and verbs must agree only in number, being either singular or plural. The grammar above can be re-written, introducing a variable at appropriate points:

```
S → NP(N) VP(N)

NP(N) → det(N) SNP(N) | SNP(N)
SNP(N) → noun(N) | adj SNP(N)

VP(N) → verb(N) VerbComps
VerbComps → Ø | NP(_) | PP | NP(_) PP

PP → prep NP(_)
```

```
det(N) →  {det,N} [i.e. any word stored in the lexicon as a det of number N]
adj →     {adj}
noun(N) → {noun,N)}
prep →    {prep}
verb(N) → {verb,N}
```

Notice the use of the 'don't care' variable when agreement is not needed but the nonterminal needs an argument. The lexicon must be extended to store number for determiners, nouns and verbs.

the : det,_
this : det,s
these : det,p
that : det,s
those : det,p
my : det,_
black : adj
young : adj
happy : adj
cat : noun,s
cats : noun,p
man : noun,s
men : noun,p
table : noun,s
tables : noun,p
on : prep
killed : verb,_
kill : verb,p
kills : verb,s
put : verb,_
puts : verb,s
slept : verb,_
sleep : verb,p
sleeps : verb,s

With this grammar, agreement in number can be enforced in both generation and recognition.

In generation, when we first encounter a variable or _, it can be replaced by any of its valid values, a variable must be given the same value throughout that production. Suppose we choose N = s in the NP in first production. Then:

```
S → NP(s) VP(s)
```

Let's choose NP(s) to expand next. One of the possible expansions of NP(N) (from the second production above) is to det(N) SNP(N). However, we already have N = s in the lhs of the production, so the only possible expansion is to det(s) SNP(s):

```
S → NP(s) VP(s) → det(s) SNP(s) VP(s)
```

Carrying on in this way, making arbitrary choices where the productions have alternatives or we reach a lexicon entry, we can reach *this cat sleeps*:

```
det(s) SNP(s) VP(s) → det(s) SNP(s) verb(s) VerbComps →
det(s) SNP(s) verb(s) Ø → this SNP(s) verb(s) Ø →
this noun(s) verb(s) Ø →→ this cat sleeps [3]
```

In recognition, the values of variables will often be set by the lexicon entries:

```
these cats sleep ←← det(p) noun(p) verb(p) ←← det(p) SNP(p) verb(p) Ø
←← NP(p) verb(p) VerbComps ←← NP(p) VP(p) ← S
```

The process fails if we start from *this cats sleep*:

```
this cats sleep ←← det(s) noun(p) verb(p) ←← det(s) SNP(p) verb(p)
```

[3] An alternative algorithm for generation delays the assignment of a value to a variable until a preterminal is replaced by a lexicon entry. It is then important to use different names for variables which are distinct, in order to avoid forcing agreements where the grammar does not.

We can't work back from `det(s) SNP(p)` to `NP(N)` since the values of `N` would be inconsistent.

When the lexicon contains a don't care value (_), this will match any value when working backwards. Thus if we start from *the cats sleep*:

the cats sleep ←← `det(_) noun(p) verb(p)` ← `det(_) SNP(p) verb(p)` ← `NP(p) verb(p) ...`

Some other examples:

`S` →→ *this man kills cats*

these cats sleeps ←← `NP(p) VP(s)` – Now can't get back to `S`.

these cat sleeps ←← `det(p) noun(s) verb(s)` – Now can't get back to `S`.

The key principle is that within a production a variable can have only one value, no matter how it got that value.

Other agreements can be handled in exactly the same way. In the most morphologically complex Indo-European languages, all the components of an `NP` must agree in number, gender and case and the `NP` and `VP` must agree in person and number. To generate the full range of 'persons', pronouns must be included in the grammar.

Arguments are always in the order:
`P`(erson) - values `1 | 2 | 3`
`N`(umber) - values `s`(ingular) | `p`(lural)
`G`(ender) - values `m`(asculine) | `f`(eminine) | `n`(euter)
`C`(ase) - values `nom`(inative) | `acc`(usative) | `dat`(ive) | etc.

```
S → NP(P,N,G,nom) VP(P,N)

NP(P,N,G,C) → prn(P,N,G,C)
NP(3,N,G,C) → det(N,G,C) SNP(N,G,C)| SNP(N,G,C)
SNP(N,G,C) → noun(N,G,C) | adj(N,G,C) SNP(N,G,C)

VP(P,N) → verb(P,N) VerbComps
VerbComps → Ø | NP(_,_,_,acc) | PP | NP(_,_,_,acc) PP [+ others]

PP → prep NP(_,_,_,acc) | prep NP(_,_,_,dat) [+ other cases]

det(N,G,C)  →  {det,N,G,C}    [i.e. a word stored in the lexicon as a det of number
                               N, gender G and case C]
adj(N,G,C) →   {adj,N,G,C}
noun(N,G,C)  → {noun,N,G,C}
prn(P,N,G,C) → {prn,P,N,G,C}
prep  →        {prep}
verb(P,N)  →   {verb,P,N}
```

With appropriate adjustments (and lexicon), this grammar can be used to handle a wide range of IE languages. For Modern Greek or German, the grammar is essentially correct as it stands; Spanish, French and English show case only in pronouns, so that this argument can be removed from other predicates; English shows gender only in pronouns[4] and verb agreement is very limited (except in the verb *be*). Small changes in word order may also be needed. For example, French adjectives generally go after the noun, and in many IE languages (including French and Greek), when verb complements become `prn`s, they go before the verb not after.

The considerable expansion of the lexicon is a problem. For example, in a language with 3 genders, 2 numbers and 4 cases (like German or Modern Greek), an adjective can in principle occur in 3 × 2 × 4 = 24 different forms! In practice some of these are the same, but the

[4] And even here it does not show GRAMMATICAL gender, but 'referent' gender. The difference can be only be explained in another language. In Modern Greek the word for girl (*koritsi*) is neuter. When a pronoun refers back to 'the girl', there appears to be a choice of either the neuter form or the feminine form to give either agreement in grammatical gender or agreement in referent gender. The normal choice in Greek is the former.

problem is only reduced not eliminated. The solution is to use morphological processing in conjunction with the grammar. For example, we might replace a production like:

```
adj(N,G,C) → {any word stored in lexicon as adj,N,G,C}
```

by:

```
adj(N,G,C) → {look up a 'base' word in the lexicon and use
morphological rules to convert to number N, gender G and case C}
```

The stored 'base' form is usually the masculine singular nominative.

5 Handling other restrictions

The grammar will still accept sentences which are syntactically incorrect. For example:

S →→ *the man sleeps the cat*
S →→ *the man puts*

The problem here is that verbs like *sleep* and *put* cannot occur with any of the possible verb complements (VerbComps). *Sleep* cannot have a following NP, and *put* must be followed by an NP and an appropriate PP. The solution is much the same as that adopted to handle agreement by number: extend the grammar by adding variables and pick up the values for these variables from the lexicon. Consider the grammar for English presented above. The VP production was:

```
VP(N) → verb(N) VerbComps
VerbComps → Ø | NP(_) | PP | NP(_) PP
```

This can be re-written as:

```
VP(N) → verb(N,Comps) VerbComps(Comps)
VerbComps(none) → Ø
VerbComps(np) → NP(_)
VerbComps(pp) → PP
VerbComps(np_pp) → NP(_) PP
```

Note that the arguments to VerbComps when it is on the lhs of a production are just constants, shown in my notation by the initial lower-case letter. I didn't have to use np for example – any constant would do. These constants are then picked up from the lexicon:

put : verb,np_pp
sleep : verb,none

This will ensure that *put* can only be followed by NP plus PP and *sleep* only by nothing.

The problem with this simple approach is that we will have to duplicate lexicon entries if a verb can have more than one set of complements (e.g. *eat* should allow both none and np). A more realistic solution should allow for both lists of possible complements and morphological processing. Current approaches to syntax analysis tend to rely on relatively simple grammar rules combined with complex lexicons.

6 Parsing

So far we have only developed recognizers and generators. A parser must not only recognize a sentence as belonging to a grammar, but also return an analysis of its structure. It is in fact comparatively easy to write a grammar which does this, although hand-tracing such grammars is not always easy. Consider the grammar we have developed for an English noun phrase (NP):

```
NP(N) → det(N) SNP(N)
NP(N) → SNP(N)
SNP(N) → noun(N)
SNP(N) → adj SNP(N)
```

(I have divided the alternatives into separate productions.) In order to return an analysis of the structure of the input, each nonterminal must be supplemented by an additional variable

which returns a 'tree' (ultimately built from the right-hand sides of each production). I'll put the tree variable as the first argument. The tree can be represented using nested expressions. Thus the first production above could be re-written as:

```
NP(np(T1,T2),N) → det(T1,N) SNP(T2,N)
```

Suppose that when `det(T1,N)` is expanded `T1` acquires the value `det(`*the*`)` and that when `SNP(T2,N)` is expanded `T2` acquires the value `noun(`*cat*`)`. Then the production above builds the tree `np(det(`*the*`),noun(`*cat*`))`.

The full grammar for a `NP` can be written as:

```
NP(np(T1,T2),N) → det(T1,N) SNP(T2,N)
NP(np(T),N) → SNP(T,N)
SNP(snp(T),N) → noun(T,N)
SNP(snp(T1,T2),N) → adj(T1) SNP(T2,N)
```

The terminal predicates of this grammar can return the actual word as part of the tree:

```
det(det(Word),N)  →  {Word : det,N}
adj(adj(Word))  →    {Word : adj}
noun(noun(Word),N) → {Word : noun,N}
```

The 'informal' notation `{Word : det,N}` means any word `Word` stored in the lexicon as a `det` of number `N`.

these happy young cats ←←
`det(det(`*these*`),p) adj(adj(`*happy*`)) adj(adj(`*young*`)) noun(noun(`*cats*`),p)` ←
`det(det(`*these*`),p) adj(adj(`*happy*`)) adj(adj(`*young*`)) SNP(snp(noun(`*cats*`)),p)` ←
`det(det(`*these*`),p) adj(adj(`*happy*`)) SNP(snp(adj(`*young*`),snp(noun(`*cats*`))),p)` ←
`det(det(`*these*`),p) SNP(snp(adj(`*happy*`),snp(adj(`*young*`),snp(noun(`*cats*`)))),p)` ←
`NP(np(det(`*these*`),snp(adj(`*happy*`),snp(adj(`*young*`),snp(noun(`*cats*`)))),p)`

Two things are clear: we do not want to have to do this by hand, and trees in this linear form are hard to make sense of! Re-writing the tree in 'indented' format is perhaps clearer:

```
np
  det - these
  snp
    adj - happy
    snp
      adj - young
      snp
        noun - cats
```

We could also use the grammar to generate a noun phrase, having supplied the tree, e.g.

`NP(np(det(`*these*`),snp(adj(`*happy*`),snp(adj(`*young*`),snp(noun(`*cats*`)))),p)` →→
these happy young cats

Again this process clearly needs automating. A more useful alternative to returning the actual word in the tree is to return the 'base' word plus a list of 'key' properties the word possesses.[5] For example the phrase *these happy young cats* might be represented as something like:

`np(det(`*this*`+[p]),snp(adj(`*happy*`),snp(adj(`*young*`),snp(noun(`*cat*`+[p])))))`

7 Machine Translation

There are a number of distinct approaches to automating translation between NLs. The most obvious approach is to use a SINGLE intermediate representation, e.g. a syntax tree with extra information added.

For example, translation from English (E) to German (G) could start with an English grammar, extended to generate syntax trees. Inputting an English sentence using an English lexicon produces a syntax tree representing the English sentence. The syntax tree, as noted above, should store the 'base' word (i.e. lexeme) plus key properties, e.g. *cat*`+[p]` rather than

[5] What I mean by the word 'key' is explained later.

cats. The base words in the tree can then be mapped to their German equivalents using an English/German translation lexicon, with key properties copied across. The resulting syntax tree for German is then put back through a German grammar and a German lexicon to yield a German sentence.

'Lexicon' here includes appropriate morphological processing, in both languages.

English input sentence	→ E Grammar E Lexicon	Syntax Tree with English base words	→ E/G Lexicon	Syntax Tree with German base words	→ G Grammar G Lexicon	German output sentence

Translating from German to English can then be achieved by reversing this process.

For some pairs of sentences this approach clearly works. *The man sees the cats* might produce an English syntax tree of the form:

```
s
  np
    det = the+[s]
    snp
      noun = man+[s]
  vp
    verb = see+[3,s]
    np
    det = the+[p]
      snp
        noun = cat+[p]
```

Base words have after them a list of properties, here just `[Number]` for determiners and nouns and `[Person,Number]` for verbs. (For verbs, other properties, such as tense, would also be needed in a proper translation system.) (Note that by itself *the* is either singular or plural, but agreement within `NP` determines which is intended in the input sentence.) Translating the English base words to German base words and transferring number and person gives:

```
s
  np
    det = der+[s]
    snp
      noun = Herr+[s]
  vp
    verb = sehen+[3,s]
    np
      det = der+[p]
      snp
        noun = Katze+[p]
```

From this tree, the German grammar plus an appropriate German lexicon will generate *Der Herr sieht die Katzen*. If appropriate the grammar will enforce agreement in gender and case; neither need storing in the tree because they will automatically be picked up from the lexicon and the grammar respectively.

(Note that if gender and case were stored in the tree, they could not be copied across, since English doesn't have grammatical gender and only shows case in pronouns. Even if we were translating between two languages which do have gender and case in determiners and nouns, they aren't necessarily the same in each language.)

On the other hand, we do need to store number in the tree, since this is a purely semantic property, and cannot be predicted from the lexicon or the grammar of either language. Thus, number is a 'key' property in the language used earlier.

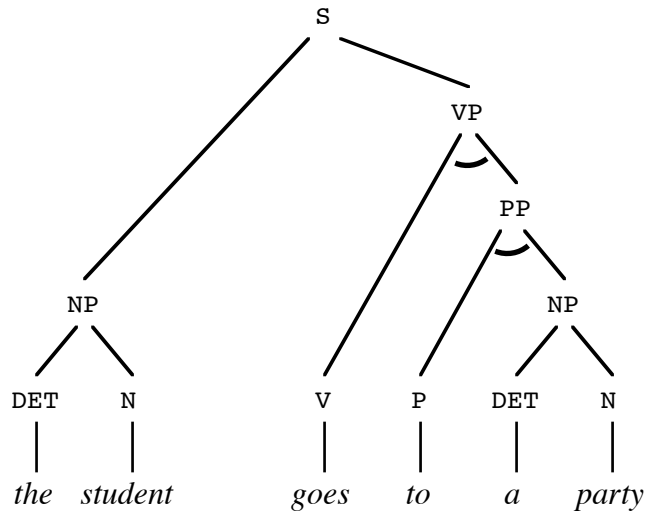
Using a single syntax tree may be successful for languages which are reasonably similar, such as English and German.[6] Where language pairs differ significantly in syntax, such as English and Japanese, the grammar of each language will more naturally generate different syntax trees, and transformation rules are needed to map one into the other. Thus English (E) to Japanese (J) translation might involve the process:

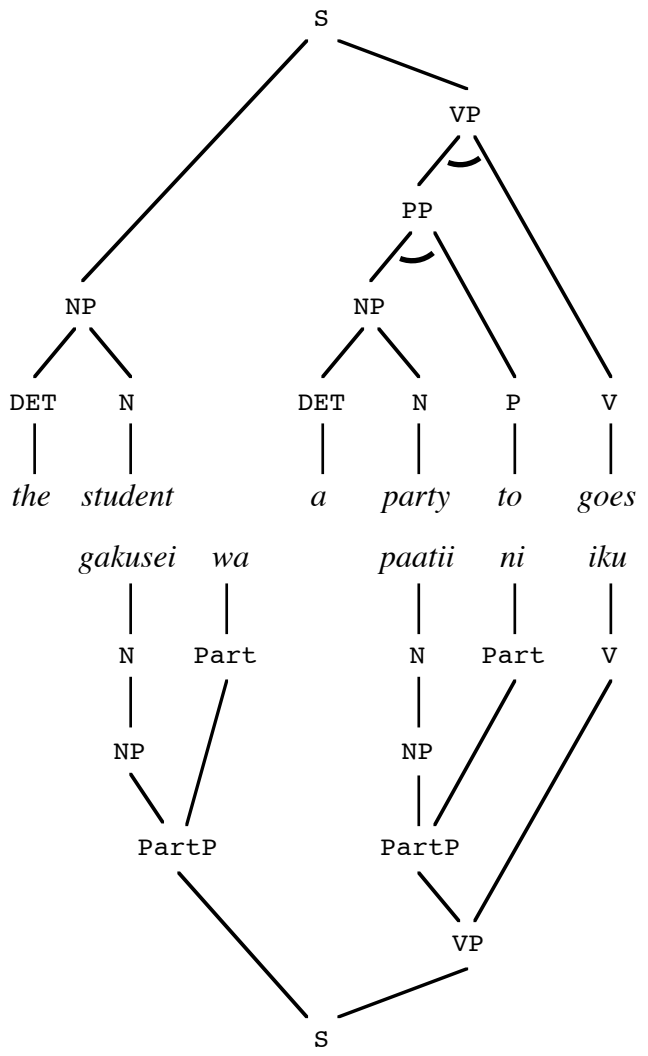
---

6 Even these very similar languages have significantly different word ordering in subordinate clauses.

English input sentence	→ E Grammar E Lexicon	E Syntax Tree with English base words	→ E/J Rules E/J Lexicon	J Syntax Tree with Japanese base words	→ J Grammar J Lexicon	Japanese output sentence

Consider the English sentence *The student goes to a party*. Putting the sentence through an English grammar might yield a tree such as the following (for simplicity I haven't here represented words as base + key properties, but this would be done in a real system).





Construction of the Japanese syntax tree could begin by swapping the subtrees marked with an arc, since Japanese has the verb last in verb phrases, and the Japanese equivalent of

prepositions come after the noun phrase to which they refer. The top of the second diagram shows the transformed tree.

Generating the Japanese syntax tree shown upside down in the bottom of the second diagram is now relatively straightforward. The English PP is converted to a Japanese 'particle phrase'. The English subject NP is also converted to a PartP, by inserting a 'subject marker'. Japanese doesn't have obligatory determiners, so these disappear and the English NPs in this example become bare nouns in the corresponding Japanese NPs. 1:1 translation of the relevant words followed by the use of Japanese grammar in reverse produces the translation *Gakusei wa paatii ni iku* (literally "student subject-marker party to goes").

It should be clear that this process is sufficiently rule-based so that it can be programmed.

Using this method to translate among a set of languages, we need a grammar for each language plus for each pair a 'translation lexicon' and a set of transformation rules. So for 10 languages we would need 10 grammars, 10 monolingual lexicons, 45 translation lexicons and 45 sets of transformation rules (assuming these last are bi-directional – otherwise we need 90 of each).

One way of reducing complexity would seem to be to use a common 'interlingua'. Translating English to German or English to Japanese would then require English to Interlingua followed by Interlingua to German or Interlingua to Japanese. Now for 10 languages only 10 translation lexicons and 10 sets of transformation rules are needed. However, since the 'interlingua' would need to be able to represent ALL information present in a sentence in ANY of the languages, finding or constructing an interlingua is a difficult or even impossible task.

Attractive though it seems, this whole approach suffers from a number of very serious difficulties, if not fundamental flaws.

In the English to Japanese example discussed above, the word *student* can be translated by a number of different Japanese words, depending partly on the type of institution the student attends. Japanese verbs have formal and informal morphological variants, as noted in the introduction to this module, which make a distinction which does not exist in English. Hence the translation of *goes* as *iku* (informal) rather than *ikimasu* (formal) might or might not be appropriate.

As a minimum, then, contextual information based on the sentence and passage being translated (including perhaps the social setting) will be needed to enable the correct lexeme to be selected from the set of possible translations.

However, even with full contextual information, 1:1 word translation may not be possible when the sentence in one language revolves around a distinction which is not made in the other. For example, English clearly distinguishes the words *rat* and *mouse*. In a number of European languages, including everyday Modern Greek, the same word (*pondiki*) is normally used for both kinds of animal. So how should the English sentence *That's not a rat, it's a mouse* be translated into everyday Greek? (There are of course scientific terms which a Greek biologist would use, but these are not appropriate to everyday language.)

- Even where 1:1 translation of the underlying lexemes is possible, there may be problems with properties of the words which affect meaning, such as number or person. So far it has been assumed that these can just be copied from one language to another. However, this will sometimes fail. For example, the word *trousers* is grammatically always plural in English (we say *My trousers are grey* not *My trousers is grey*) but semantically can be singular or plural: in *I'm wearing my green trousers* we mean one pair of trousers whereas in *All my trousers are in the wash* we mean more than one pair. An English syntax tree will always contain *trousers*+[p]. If this is simply mapped to a French syntax tree the result will always be *pantalon*+[p]. However, *je porte mes pantalons verts* means that I am wearing multiple pairs of green trousers! Arabic has THREE values for number: singular, dual and plural. Translating *Karim's brothers are coming to see him* into Arabic requires knowledge of whether Karim has two brothers or more than two, since the exact word for brothers will be different.
- Different languages, even within the same language family, use different morphological and syntactical structures to convey the 'same' meaning. The normal way of saying *I like dancing* in Spanish or Modern Greek demands the use of a quite different sentence structure. The literal translation of the Spanish *me gusta bailar* is 'me it-pleases to-dance'; the

literal translation of the Greek *mou aresi na horevo* is 'of-me it-pleases to I-dance'. In such cases translation methods based on syntax trees would need to be able to make quite radical changes to the trees, rather than simply mapping one into the other.

- Idioms, metaphors and other non-literal uses of language mean that the best translation will often use totally different lexemes as well as different syntactical structures. The literal meaning of the Greek sentence *ta ékana thálassa* is something like 'them I-made sea' or less literally 'I made them into a sea'. A appropriate English translation is 'I made a mess'.
- Other problems arise in translating connected passages, such as resolving anaphora – this will be discussed later in the module.

The consequence of these problems is that although most modern MT systems have syntax analysis and syntax tree manipulation at their core, the bulk of the processing carried out often consists of applying a large number of more-or-less *ad hoc* rules to deal with special cases. High quality MT systems have been built up in this way over many years.

Such systems are still far from perfect. Measuring the quality of translations is difficult, since only for very simple sentences will there be a single 'correct' translation. The National Insitute of Standards and Technology in the USA has carried out automated tests since 2001.[7] These rely on determining what proportion of sequences of N words (N-grams) in the translation are also present in a reference set of translations generated by expert human translators. In 2005, the best systems achieved around 50% of 4-grams matching the human-generated translations. This does not mean that the other 50% were necessarily wrong, but does show that MT is still a considerable distance from high quality human translation.

8 Parsing Algorithms

I have concentrated so far on grammars rather than algorithms for processing them.

A full grammar for a NL clearly requires a large number of rules, plus a complex lexicon. The number of syntax rules can be reduced by noting that some sentences appear to be 'transformations' of other sentences. A good example is the passive in English.

Consider these sentences:

The dog chased the cat. (Active)
The cat was chased by the dog. (Passive)
The man puts the cat on the mat. (Active)
The cat is put on the mat by the man. (Passive)
**The man put the cat*. (Active)
**The cat was put by the man*. (Passive)

The passive sentence of each pair is predictable from the active. Also if the active version is valid, so is the passive; if the active is invalid, so is the passive. One way of handling such sentences is to generate the active sentence from the 'normal' grammar, then transform this sentence into the passive. We need a different kind of grammar – a 'transformational' grammar – to handle this approach. An alternative is to apply transformational rules to the GRAMMAR. (These are then 'meta-rules' because they are rules for generating new rules.)

One algorithm for parsing using PSGs is to apply the same approach we used to expand a grammar 'by hand' but do it strictly top-down, left-to-right. However this must be accompanied by the ability to back-track, sometimes extensively. Consider the productions we used to handle verb complements:

```
VP → verb VerbComps
VerbComps → Ø | NP | PP | NP PP
```

Given the verb complements *the box on the table*, straightforward top-down, left-to-right processing will first parse *the box* as the NP alternative. Parsing will then fail, as the sentence has not ended. The algorithm will must then back-track, starting again at *the*. Treating the verb complement as a PP will fail, after which *the box* will be parsed again as the first part of

[7] http://www.nist.gov/speech/tests/mt/

the NP+PP complements. Thus given the sentence *I put the small red boxes on the table*, *the small red boxes* will be parsed twice, each parse probably including the morphological processing of *boxes*. In some cases, this can be avoided by clever re-writing of the grammar. For example:

```
VP → verb VerbComps
VerbComps → Ø | NP After_NP | PP
After_NP → Ø | PP
```

However, this kind of 'trick' makes the grammar more obscure (and hence error-prone).

A better alternative is to use a 'chart parser'. This operates top-down, but saves every successful sub-parse (in a 'chart'). So if *the small red boxes* was parsed as NP, the parse sub-tree would be saved and would then be re-used when trying to parse *the small red boxes on the table* as NP+PP. 'Garden path' sentences like those described earlier (in the handout on Phones and Phonemes) can still cause problems. In the following two sentences, the status of *killed* cannot be determined until the word after *bush* is reached:

The lion killed yesterday afternoon in the open bush and was seen today.
The lion killed yesterday afternoon in the open bush was seen today.

Another alternative is to abandon top-down parsing and use a bottom-up approach. The table which follows shows in outline how this might be done for the sentence *I put the small red boxes on the table*. The first stage is to categorize the words in the sentence. Then categories are gradually merged until S is reached. The italicized entries show where the wrong choice was made the first time, thus triggering back-tracking.

Process:	*I*	*put*	*the*	*small*	*red*	*boxes*	*on*	*the*	*table*
lexicon entries	prn	verb	det	adj	adj	noun	prep	det	noun
SNP	prn	verb	det	SNP			prep	det	SNP
NP	NP	verb	NP				prep	NP	
PP	NP	verb	NP				PP		
VerbComps	*NP*	*verb*	*VerbComps*				*PP*		
VP	*NP*	*VP*					*PP*		
S	*S*						*PP*		
(Backtrack)	NP	verb	NP				PP		
VerbComps	NP	verb	VerbComps						
VP	NP	VP							
S	S								

Note that the first step in bottom-up parsing is to identify the category of words. This process, often known as 'tagging', has been the subject of considerable research which will not be discussed here.

Like chart parsing, bottom-up parsing keeps successful sub-parses, so that although back-tracking may still be needed it need not go so far back. Bottom-up parsing has particular attractions for parsing NLs where the order of constituents within sentences is more flexible. However, there are still sentences which will cause deep back-tracking with bottom-up parsers. Consider:

Time flies like an arrow.

Syntactically there are three possible parses for this sentence. In the first, *time* is a verb, *flies* a noun (so the sentence is of the same form as *Answer questions like an expert*). In the second, *time* is a noun, *flies* a verb (so the sentence is similar to *The aeroplane flies like an arrow*). In the third, *time flies* is a noun phrase, *like* a verb (so the sentence is similar to *Fruit flies like a banana*). The simple morphology of English makes such lexical ambiguity more common than in most other Indo-European languages. When significant lexical ambiguity is possible, bottom-up parsing loses many of its attractions.

Real parsing systems frequently use a combination of a variety of approaches, e.g. combining bottom-up tagging methods to identify the categories of words and pattern-matching to identify idioms with top-down parsing using grammars.

Appropriate parsing methods (and their corresponding grammars) have been the subject of extensive research in NLP, and a variety of methods have been proposed, some quite dif-

ferent from those appropriate for PSGs. However, there is widespread agreement that much of the complexity of NLs must be handled via the lexicon, not the grammar.

Exercises

1. Classify each word in the following sentences as either a noun (N), verb (V), adjective (A), preposition (P), determiner (D), pronoun (PRN) or auxiliary (AUX).

   a) *Some people like cats.*
   b) *Europeans peopled America.*
   c) *Careful owners wash their cars.*
   d) *Down fills the best duvets.*
   e) *She might drive down my street.*
   f) *The man with a wooden leg ate my hamburger.*
   g) *No-one saw her.*
   h) *You should put paint on the sound wood.*
   i) *I heard a wooden sound.*
   j) *The bell sounds for tea.*
   k) *I have painted the outside of my house.*
   l) *I put the tub of red geraniums outside my house.*

2. Identify ALL the NPs, PPs and VPs in the sentences in Exercise 1. Allow a single noun or pronoun to form a noun phrase.

3. a) The simplest possible sentences in English are formed from one or two plural nouns plus one verb, e.g. *otters swim* or *otters eat fish.* Write a grammar which generates ONLY sentences of this form. Your lexicon should contain the words *eat*, *fish*, *otters*, *swim* and *anglers*. Does your grammar generate any syntactically invalid sentences? Does it generate any semantically invalid sentences?

   b) Extend your grammar to include the auxiliaries *can*, *do*, *may* and *will*, i.e. allow sentences of the form *otters may eat fish* as well as *otters eat fish.*

   c) [More difficult] Extend your grammar to allow negative and interrogative sentences, i.e. sentences of the form *otters will not eat fish* or *do otters eat fish?* How could forms such as *don't* or *won't* be handled? What about negative questions, i.e. sentences of the form *Don't otters eat fish?* or *Do otters not eat fish?*

4. Write a simple grammar which handles ONLY sentences of the form 'subject + verb + location.' The verb should always be in the third person; pronouns should be ignored. (Hence the only agreement which is required is in number.) Some sentences which the grammar should handle are:

   The dog is in the basket.
   A dog sits on the floor.
   Dogs sit on floors.
   The baskets are on the floor.

5. Consider the following fragment of a grammar for Japanese. (Note that `topic` and `locn` are constants, `C` is a variable, and `S` is the start symbol.)

   `S` → `PartP(topic) VP | VP`

   `VP` → `PartP(locn) verb`

   `PartP(C)` → `noun part(C)`

   `part(topic)` → *wa*
   `part(locn)` → *ni*

   `noun` → *honsha* | *sensei*
   `verb` → *iru*

   [*ni* ≈ *in*, *at, to*; *honsha* = office; *sensei* = teacher; *iru* ≈ *is*.]

   a) Draw the syntax tree based on this grammar for the sentence *sensei wa honsha ni iru.*

b) Give two further sentences generated by this grammar, one semantically meaningful, the other not.

6. For English, the generalized 'Indo-European' grammar given earlier can be reduced to the following:

```
S → NP(P,N,nom) VP(P,N)

NP(P,N,C) → prn(P,N,C)
NP(3,N,_) → det(N) SNP(N)
NP(3,p,_) → SNP(p)
SNP(N) → noun(N) | adj SNP(N)

VP(P,N) → verb(P,N) verbComps

verbComps → Ø | NP(_,_,acc) | PP | NP(_,_,acc) PP

PP → prep NP(_,_,acc)

det(N)  →    {det,N}
adj  →       {adj}
prn(P,N,C) → {prn,P,N,C}
noun(N)  →   {noun,N}
prep  →      {prep}
verb(P,N)  → {verb,P,N}
```

a) The grammar involves three variables: `P`(erson), `N`(umber) and `C`(ase). Assume the values of these will be:

Person: `1` = First, `2` = Second, `3` = Third
Number: `s` = singular, `p` = plural
Case: `nom` = nominative (subject), `acc` = accusative (object)

The value _ can be used when any of the values are acceptable. For example, *you* is both singular and plural and can be used as the subject (nominative) as in *You saw him* and as the object (accusative) as in *He saw you*. Thus a suitable lexical entry for *you* might be:

you `: prn,2,_,_`

In some cases, we may need more than one entry. For example, to ensure that it agrees with *I*, *we*, *you* and *they* but not *he*, *she* or *it*, *give* requires:

give `: verb,1,_`
give `: verb,2,_`
give `: verb,3,p`

Write out a suitable lexicon for the grammar. Include the following words:

I, me, we, us, you, he, him, she, her, it, they, them, the, man, men, woman, women, bottle, bottles, give, gives, gave, to.

b) Which of the following sentences are accepted by the grammar with your lexicon? For those that are accepted, draw the resulting syntax trees.

i) *We give the bottle to the man.*
ii) *He gives me the bottle.*
iii) *They give the bottle.*
iv) *I gave the bottle to me.*

c) Extend the grammar and lexicon given above to allow for agreement in verb and verb complements.

d) Verbs such as *think* or *know* allow a further kind of verb complement: *that* followed by a sentence (e.g. *I think that they gave bottles to the women*). The word *that* can also be omitted (e.g. *I think they gave bottles to the women*). Such complements can be nested (e.g. *I think they know that I gave bottles to the women*). Extend the grammar and lexicon to allow such sentences.

7. A feature of NL not covered so far is 'co-ordination'. Consider sentences such as:

   Men and women gave bottles to us.
   They gave bottles to the men and women.
   They gave bottles to the man and cans to the woman.

   One hypothesis to account for such sentences is that wherever a nonterminal occurs in the sentence it can be replaced by two co-ordinated nonterminals of the same type, with appropriate adjustments to the variables.

   For example, if we add the production:

   `noun(p)` → `noun(N1)` *and* `noun(N2)`

   the grammar generates/accepts sentences such as *The old man and woman sleep* or *I gave bottles to the man and woman*. Write appropriate productions for THREE other nonterminals for which this works. Are there any nonterminals for which it does not work?

8. Here is a possible grammar and lexicon for some simple French sentences, written in the notation used in this handout. (Gender has been ignored, all words being masculine where relevant.)

```
F_S → F_NP(P,N,nom) F_VP(P,N)

F_NP(P,N,C) → F_NP1(P,N,C)
F_NP(3,N,_) → F_NP2(N)

F_NP1(P,N,C) → f_prn(P,N,C)

F_NP2(N) → f_det(N), f_noun(N)

F_VP(P,N) → F_NP1(_,_,acc), f_verb(P,N)
F_VP(P,N) → f_verb(P,N), F_NP2(_)

f_prn(P,N,C) → {f_prn,P,N,C,_}.
f_det(N)     → {f_det,N,_}.
f_noun(N)    → {f_noun,N,_}.
f_verb(P,N)  → {f_verb,P,N,_}.
```

PRONOUNS: `f_prn,PERSON,NUMBER,CASE,ENGLISH`

```
je :    f_prn,1,s,nom,'I'
me :    f_prn,1,s,acc,me
nous :  f_prn,1,p,nom,we
nous :  f_prn,1,p,acc,us
tu :    f_prn,2,s,nom,you
te :    f_prn,2,s,acc,you
vous :  f_prn,2,p,_,you
il :    f_prn,3,s,nom,he
le :    f_prn,3,s,acc,him
ils :   f_prn,3,p,nom,they
les :   f_prn,3,p,acc,them
```

DETERMINERS: `f_det,NUMBER,ENGLISH`

```
le :    f_det,s,the
les :   f_det,p,the
```

VERBS: `f_verb,PERSON,NUMBER,ENGLISH`

```
vois :    f_verb,1,s,see
vois :    f_verb,2,s,see
voit :    f_verb,3,s,sees
voyons :  f_verb,1,p,see
voyez :   f_verb,2,p,see
voient :  f_verb,3,p,see
```

NOUNS: `f_noun,NUMBER,ENGLISH`

```
chat :    f_noun,s,cat
chats :   f_noun,p,cats
```

a) Classify the following French sentences as valid or invalid according to the above grammar and lexicon. In each case, make sure you understand EXACTLY how the grammar does or does not generate the sentence.

i) *je vois le chat*
ii) *nous voient le chat*
iii) *le chat voit me*
iv) *le chat me voit*
v) *les chats voient le chat*
vi) *le chat le voit*
vii) *vous vous voyez*
viii) *tu tu vois*

b) Give six further French sentences which are valid according to the above grammar and lexicon.

9. a) Make sure you understand how to alter the French grammar given in Exercise 8 to include the parse tree as a variable. You don't need to re-write the whole grammar.

b) Work out what parse tree would be generated by the grammar for the sentence *le chat me voit*. What would be the problem(s) in translating this sentence to English using the English words stored in the lexicon?

Appendix: Coding CFGs in Prolog

CFGs are easy to code in Prolog, once the basic idea is grasped. Each production can be represented by a rule in which every non-terminal symbol becomes a predicate with two arguments: an input and an output sequence, both composed of terminals. Sequences can be represented in Prolog as lists. Consider recognition first. Each predicate should REMOVE the appropriate terminal symbol(s) from the front of the input sequence to form the output sequence. Thus the production:

```
S → NP VP
```

can be written in Prolog as:

```
s(Sin,Sout):- np(Sin,S1), vp(S1,Sout).
```

np/2 must take whatever it is given in Sin and remove an NP from the front to form S1. vp/2 then takes this sequence and removes a VP from it to form Sout. If Sin is a list representing a valid sentence, then Sout should be the empty list. Thus if Sin = [this,man,killed,the,cat], S1 should be [killed,the,cat] (since np/2 should remove [this,man]), and Sout should be ø (since vp/2 should remove [killed,the,cat]). Note that the upper-case symbols used in the grammar must start with a lower-case letter when they become Prolog predicates.

A recognizer for the complete grammar given on Page 4 can be coded as follows:

```
s(Sin,Sout):- np(Sin,S1), vp(S1,Sout).

np(Sin,Sout):- det(Sin,S1), SNP(S1,Sout) ; SNP(Sin,Sout).
snp(Sin,Sout):- noun(Sin,Sout) ; adj(Sin,S1), snp(S1,Sout).

vp(Sin,Sout):- verb(Sin,S1), verbComps(S1,Sout).
verbComps(Sin,Sout):- Sin = Sout ; np(Sin,Sout) ; pp(Sin,Sout) ;
                      np(Sin,S1), pp(S1,Sout).

pp(Sin,Sout):- prep(Sin,S1), np(S1,Sout).
```

Notice how the empty symbol is dealt with.

The pre-terminal symbols could be mapped into Prolog in a similar way, e.g.:

```
det(Sin,Sout):- Sin = [the|Sout] ; Sin = [this|Sout] ; ...
```

However, this is both tedious and difficult to update with new words. A better approach is to store words in a lexicon. For the present, lex/2 will hold a word and the pre-terminal symbol from which it can be generated. Here words will be stored as SINGLE symbols, rather than as a list of letters as might be necessary in morphological processing. (In Prolog, the two formats can easily be inter-converted when necessary.)

```
lex(the,det).                    lex(cat,noun).
lex(this,det).                   lex(man,noun).
lex(that,det).                   lex(table,noun).
lex(my,det).                     lex(on,prep).
lex(black,adj).                  lex(killed,verb).
lex(young,adj).                  lex(put,verb).
lex(happy,adj).                  lex(slept,verb).
```

The pre-terminal symbols can then be mapped into:

```
det(Sin,Sout):-  Sin = [Word|Sout], lex(Word,det).
adj(Sin,Sout):-  Sin = [Word|Sout], lex(Word,adj).
noun(Sin,Sout):- Sin = [Word|Sout], lex(Word,noun).
prep(Sin,Sout):- Sin = [Word|Sout], lex(Word,prep).
verb(Sin,Sout):- Sin = [Word|Sout], lex(Word,verb).
```

Using the sentence recognizer s/2 is achieved by a queries such as:

```
?- s([this,young,cat,slept],[]).
yes

?- s([this,my,cat,slept],[]).
no
```

In principle, we do not need to write a separate generator, since the code is reversible:

```
?- s(Sin,[]).
Sin = [the,cat,killed] ;
Sin = [the,cat,killed,the,cat] ;
Sin = [the,cat,killed,the,man] ;
Sin = [the,cat,killed,the,table] ;
Sin = [the,cat,killed,the,black,cat] ;
...
```

However, the grammar allows INFINITE recursion via the production `SNP → adj SNP`, so that at some stage we shall start getting output such as:

```
Sin = [the,cat,killed,the,black,black,black,black,cat] ;
```

Hence either the generator has to be written to prevent this or the grammar needs to be simplified to remove infinite recursion before being used for generation.

Other kinds of query are also possible, e.g.:

```
?- s([the,Noun1,Verb,the,Noun2],[]).
Noun1 = cat, Verb = killed, Noun2 = cat
```

The conversion of a CFG production to a Prolog clause is highly predictable. A production such as:

```
A → B C .. Y Z
```

should become:

```
a(Sin,Sout):- b(Sin,S1), c(S1,S2), .., y(S24,S25), z(S25,Sout).
```

The names of the variable arguments are, of course, arbitrary. Most Prolog systems have been extended to automate this process. The special symbol `-->` is recognized by the input routines, and any properly formatted clause containing this symbol is expanded appropriately.

Thus a production of the form:

```
A → B C .. Y Z
```

can be input in Prolog as:

```
a --> b, c, .., y, z.
```

It will then be expanded to a form similar to the rule above (although meaningful variable names will not usually be generated). Two further conventions are needed to cope with a rule such as:

```
det(Sin,Sout):- Sin = [Word|Sout], lex(Word,det).
```

After the `-->` symbol, `Sin = [Word|Sout]` can be input simply as `[Word]`, and will be expanded correctly. Predicates such as `lex`/2 which must NOT have `Sin`/`Sout` type arguments added to them must be enclosed in curly brackets `{}`. Thus if we input:

```
det --> [Word], {lex(Word,det)}.
```

the system will expand this to some equivalent of the required rule. As a further example, consider what happens if we input:

```
a --> b, c, [d], e, {f}, g.
```

The system should expand this to the equivalent of:[8]

```
a(Sin,Sout):- b(Sin,S1),   c(S1,S2),   S2   =   [d|S3],   e(S3,S4),
f,   g(S4,Sout).
```

The entire recognizer given above can thus be input as:

```
s --> np, vp.

np --> det, snp ; snp.
```

[8] 'Equivalent of' because many Prolog systems will produce a different but equivalent expansion of `[d]`. Open Prolog, for example, would expand this to `'C'(S2,d,S3)`, which then succeeds if `S2 = [d|S3]`.

```
snp --> noun ; adj, snp.

vp --> verb, verbComps.
verbComps --> [] ; np ; pp ; np, pp.

pp --> prep, np.

det -->  [Word], {lex(Word,det)}.
adj -->  [Word], {lex(Word,adj)}.
noun --> [Word], {lex(Word,noun)}.
prep --> [Word], {lex(Word,prep)}.
verb --> [Word], {lex(Word,verb)}.
```

It is important to note that it will be stored internally in its expanded form.

(Here I have retained semicolons to separate alternatives in order to emphasize the similarity between the original symbolic grammar and the Prolog version. However, debugging code is easier when separate clauses are used, so when entering Prolog grammars I recommend avoiding semicolons.)

Adding variables to enforce agreement or to generate trees is equally straightforward. Thus the grammar for a noun phrase given on Page 9:

```
NP(np(T1,T2),N) → det(T1,N) SNP(T2,N)
NP(np(T),N) → SNP(T,N)
SNP(snp(T),N) → noun(T,N)
SNP(snp(T1,T2),N) → adj(T1) SNP(T2,N)
```

can be written in Prolog as:

```
np(np(T1,T2),N) --> det(T1,N), snp(T2,N).
np(np(T),N) --> snp(T,N).
snp(snp(T),N) --> noun(T,N).
snp(snp(T1,T2),N) --> adj(T1), snp(T2,N).
```

After defining the remaining productions and the lexicon, we can input the query:

```
?- np(T,N,[these,happy,young,cats],[]).
T = np(det(these),snp(adj(happy),snp(adj(young),snp(noun(cats)))))
N = p
```