Statistical PoS Tagging and Syntactic Parsing

2017, © Ted Briscoe (ejb@cl.cam.ac.uk) GS18, Computer Lab

Abstract

This handout is not intended to replace textbooks. Many things are only covered sparsely. If you don't follow and/or can't do the exercises interspersed in the text, then read some of Jurafsky and Martin (J&M, see references in text to 2008, 2nd edition, not online 3rd edition).

Contents

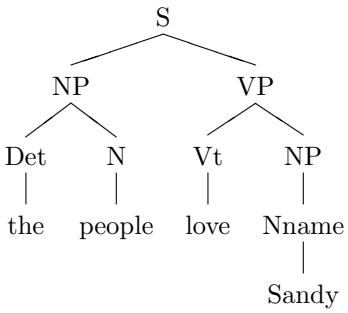
1 (Context-free) Phrase Structure Grammar

A generative grammar is a finite set of rules which define the (infinite) set of grammatical sentences of some language.

Here are some example rules for English:

a) S → NP VP
b) NP → Det N
c) NP → Nname
d) VP → Vt NP

These rules assign the sentence *The people love Sandy* the same analysis and phrase structure tree that was proposed in the Intro. to Linguistics handout, repeated below and followed by the corresponding labelled bracketing.




```
S(NP((Det The) (N people)) VP((Vt love) NP(N Sandy)))
```

Exercises

Write down the rules needed to generate this sentence from 'top to bottom'. What's missing? (easy)

The aim of a specific generative grammar is to provide a set of rules which generate (or more abstractly license, predict. etc.) all the phrase structure trees which correspond to grammatical sentences of, say, English. That is, generate all and only the word sequences which a linguist would consider correct and complete sentences considered in isolation, along with a description of their syntactic structure (phrase structure). The rules also incorporate claims about English constituent structure.

One way to formalise the grammar we have introduced above is to treat it as a context-free phrase structure grammar (CF PSG) in which each rule conforms to the following format:

$$\text{Mother} \rightarrow \text{Daughter}_1\ \text{Daughter}_2\ \ldots\ \text{Daughter}_n$$

and the syntactic categories in rules are treated as atomic symbols – non-terminal symbols being clausal and phrasal categories, terminal symbols lexical categories.

CF rules encode (immediate) dominance and (immediate) precedence relations between (non-) terminal categories of the grammar. All grammars have a designated root or start symbol (see e.g. Jurafsky and Martin, ch12) for more details on CF PSGs). To make the grammar complete, we also need a lexicon in which we pair words (preterminals) with their lexical categories.

If we do formalise such rules this way, we are claiming that CF PSGs provide an appropriate (meta)theory of grammars for human languages, and thus that (all) syntactic rules for any human language can be expressed as CF PSGs.

Grammar 1 (G1) illustrates a simple CF PSG for a small fragment of English.

Grammar 1

Rules		Lexicon			
a.	S --> NP VP.	Sam	: Nname.	plays	: V.
b.	VP --> V.	Kim	: Nname.	chases	: V.
c.	VP --> V NP.	Felix	: Nname.	sings	: V.
d.	VP --> V PP.	Tweety	: Nname.	the	: Det.
e.	VP --> V NP NP.	cat	: N.	miaows	: V.
f.	NP --> Nname.	bird	: N.	a	: Det.
g.	NP --> Det N.	park	: N.	in	: P.
h.	PP --> P NP.	ball	: N.	with	: P.

Exercises

Find 10 sentences this grammar generates. Is the set of grammatical sentences generated by G1 finite? Are they all grammatical? Can you make any changes to G1 which would stop some of the ungrammatical sentences being generated? (easy)

Give a formal definition of a CF grammar (as a quintuple) and the same for a regular (right/left linear) grammar. State the additional restrictions on the right hand side of regular grammar rules over CF rules. (Hard, unless you have done formal language theory, or read some of Jurafsky and Martin (ch12) by now.)

1.1 Derivations

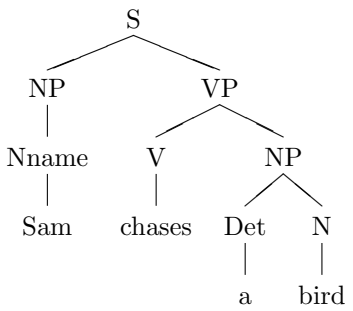
Given a CF PSG and lexicon, we can determine whether a sentence is or is not generated by attempting to construct a derivation (tree) for it. To construct a leftmost derivation, we start with the root symbol of the grammar and always rewrite (expand) the leftmost non-terminal category in the current sentential form (sequence of terminal and non-terminal categories resulting from each expansion) according to the rules in the grammar. A leftmost derivation for *Sam chases a bird* using the grammar and lexicon above is given below, where the words (preterminal categories) are given in brackets in the sentential forms:

```
S => NP VP => Nname (Sam) VP => Nname (Sam) V (chases) NP
=> Nname (Sam) V (chases) Det (a) N (bird)
```

In a rightmost derivation, we start with the root symbol but always rewrite the rightmost symbol of the sentential form. The corresponding rightmost derivation for *Sam chases a bird* is given below:

```
S => NP VP => NP V (chases) NP => NP V (chases) Det (a) N (bird)
=> Nname (Sam) V (chases) Det (a) N (bird)
```

Although the sequence of rule applications is different, the same set of rules appears in both derivations. Furthermore, the derivations are unique in that there are no alternative rewrites at any point in either which yield a derivation for the example. Constructing the derivation tree from a left/right-most derivation is straightforward – we simply represent successive rewrites by drawing lines from the mother (rewritten) symbol to the daughter (expanded) symbol(s). Both the derivations above correspond to the derivation / phrase structure tree below:



CF PSG is a so-called declarative formalism because it does not matter which order we apply rules in, we will always assign the same phrase structure tree(s) to any given sentence. Thus the rules encode the facts about grammar independently of their method of application in parsing, generation, etc.

1.2 Ambiguity

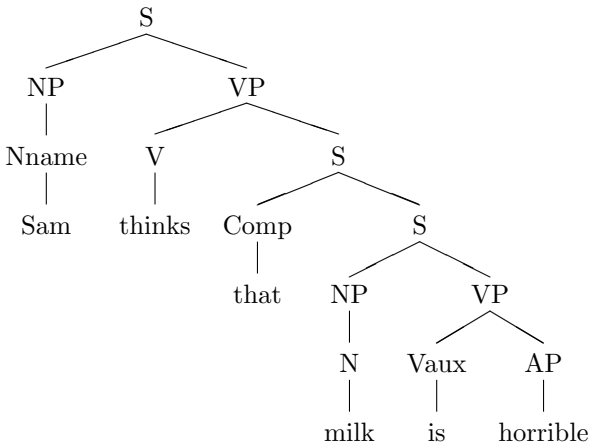
A sentence is said to be 'ambiguous' when it can be assigned two (or more) distinct semantic interpretations. A distinction is often made between two types of ambiguity, 'structural' and 'lexical'. Broadly speaking, a sentence will be said to be structurally ambiguous if a syntactic analysis of it results in the assignment of two (or more) distinct constituent structures to it, where each distinct structure corresponds to one of the possible interpretations. A sentence will be said to be lexically ambiguous, on the other hand, if it contains some ambiguous lexical item but all the distinct interpretations receive the same constituent structure. (1a) is an example of structural ambiguity and (1b) an example of lexical ambiguity.

(1) a Sam thinks that milk is horrible
    b All linguists draw trees

In (1a) the word *that* is ambiguous between a determiner reading and a complementiser reading. (A complementiser is a word that introduces a subordinate clause, traditionally called a subordinating conjunction.) The most common complementiser is *that*, but other examples include *whether* and *if*, as in (2).

(2) a Sam wonders whether Felix is in the garden
b They want to know if Felix is in the garden

As a determiner it forms an NP constituent with the noun *milk* but as a complementiser it forms a constituent with the clause *milk is horrible*. The tree below shows the complementiser analysis where it is all milk that Sam considers to be horrible.



Another way of saying a sentence is structurally ambiguous (given a grammar) is to say that it has two or more left/right-most derivations.

Exercises

Can you construct the phrase structure tree for the other reading? With (1b) we will end up with the same phrase structure tree but different senses of the noun *trees*. Can you construct it and add the rules and words to G1 needed to generate these examples, ensuring that (1a) is assigned two analyses corresponding to the structural ambiguity? (easy)

1.3 Inadequacies of CF PSG

CF PSGs have some strengths as a theory of grammar; for instance, they seem to account for hierarchical constituency quite well and by using recursion we can capture the syntactic productivity of human language. There are infinitely many grammatical sentences of English; consider, for example, the sentence in (3).

(3) Sam saw the man in the park with the telescope on the monument.

We can introduce the recursive CF rule below

VP $\rightarrow$ VP PP

which will assign a phrase structure tree to this example which corresponds to the reading in which Sam saw the man whilst Sam was in the park using the telescope standing on the monument.

Exercises

Draw the tree by adding this rule to G1 along with some appropriate lexical entries. What rule(s) would we need to add if we wanted to capture the reading in which the telescope is mounted on a monument to be found in the park where the man is? (medium)

If you have explored G1, you will have realised that it is difficult in some cases to generate all and only the grammatical sentences. For instance, there is nothing to stop us generating the examples in (4).

(4) a *Sam chases in the park.
b *Sam sings the cat.
c *Sam chases the cat the bird.

We could introduce different types of verb category (eg. Vintrans, Vtrans, Vppwith, etc.) and specify each VP rule more carefully, as we did with names and common nouns, but this would lead to many more rules if we are not careful. For example, imagine what will happen if we attempt to capture person-number agreement between subject and verb in our grammar to block examples like (5)

(5) a *The cat chase the bird.
b *The cats chases the bird.
c *I is clever.
d *You am clever.

If we add number and person to the categories in G1, we will end up with the much larger CF PSG, Grammar 2 (G2).

```
Grammar 2

1. S --> NPsg1 VPsg1
2. S --> NPsg2 VPsg2
3. S --> NPsg3 VPsg3
4. S --> NPpl1 VPpl1
5. S --> NPpl2 VPpl2
6. S --> NPpl3 VPpl3
7. VPsg1 --> Vsg1
...
13. VPsg1 --> Vsg1 NP
```

```
...
19. VPsg1 --> Vsg1 PP
...
25. VPsg1 --> Vsg1 NP NP
...
31. NPsg3 --> Nname
32. NPsg3 --> Detsg Nsg
33. NPpl3 --> Detpl Npl
34. PP --> P NPsg1
...
39. PP --> P NPpl3
```

We appear to be failing to directly capture a simple rule – 'the N(P) and V(P) in an S agree in num(ber) and per(son)'.

Exercises

What would happen if we also tried to subcategorise verbs into intransitive, transitive and ditransitive to avoid generating examples like **I smiled the ball to him* and combined this with the approach to agreement outlined above? How big would the new grammar be if you did this systematically? (medium)

Thinking back to the Intro. to Linguistics handout and the exercise on English auxiliary verbs, can you see how to formalise your analysis in CF PSG? How successful would this be? (hard)

1.4 Unification and Features

CFGs utilise atomic symbols which match if they are identical, unification-based phrase structure grammars (UB PSGs) utilise complex categories which match by unification. They provide us with a means to express the person-number agreement rule of English and many others more elegantly and concisely. Assume that syntactic categories are annotated with sets of feature attribute-value pairs, then we can factor out information about number and person from information about which type of category we are dealing with:
NP:[num=sg, per=3]
V:[num=pl, per=3]

where the possible values for the attribute num are sg/pl and for per 1/2/3. As well as being able to specify values for features we will also allow features to take variable values (represented as capital letters) which can be bound within a unification-based PS rule. The rules below express per-num agreement:

S $\rightarrow$ NP:[num=N, per=P] VP:[num=N, per=P]
VP:[num=N, per=P] $\rightarrow$ V:[num=N, per=P] NP:[]

Mostly, words in the lexicon will have fully specified categories but categories in rules often contain variable values, so they generalise across subcategories. Unification can be used as the operation to match categories during the construction of a phrase structure tree; that is, two categories will match if for any given attribute they do not have distinct values. The resultant category, if unification succeeds, is the one obtained by taking the union of the attributes, substituting values for variables and rebinding variables. Some examples are given below; the first two columns contain the categories to be unified and the third column contains the result of the unification.

```
a.  | NP:[per=3, num=N]    |  NP:[per=P, num=pl] |  NP:[per=3, num=pl]
b.  | NP:[per=2, num=sg]   |  NP:[per=2, num=N]  |  NP:[per=2, num=sg]
c.  | NP:[per=P, num=N]    |  NP:[per=3, num=N]  |  NP:[per=3, num=N]
d.  | NP:[per=1, num=sg]   |  NP:[per=2, num=sg] |  FAIL
    |                      |                     |
e.  |      N:[]            |         N:[]        |     N:[]
    |                      |                     |
f.  | V:[val=intrans,      |  V:[val=intrans,    |  V:[val=intrans,
    |    per=3, num=sg]    |     per=P, num=N]   |     per=3, num=sg]
g.  | VP:[in=F, out=F]     |  VP:[in=G, out=H]   |  VP:[in=I, out=I]
h.  | NP:[per=1]           |  NP:[num=pl]        |  NP:[per=1, num=pl]
```

A category consists of a category name (the functor, eg. X:) and a set of features enclosed in square brackets after the functor. Features are made up of attributes (eg. per) and values/variables (eg. 1 or P) separated by = and delimited from each other by commas.

Unification can be defined in terms of subsumption.

If two categories, A and B, unify, this yields a new category, C, defined as the smallest (most general) category subsumed by A and B, otherwise fail.

Category A subsumes category B (i.e. B is more specific than A) iff (if and only if):
1) every attribute in A is in B;
2) every attribute=value pair in A is in B;
3) every attribute=variable pair in A is either in B or B has a legal value for this attribute;
4) every attribute sharing a variable value in A, shares a (variable) value in B.

The notation I have used for categories is similar to that used for Prolog terms. However, Prolog uses fixed-arity term unification in which unifiable categories must explicitly have the same set of attributes – given this 'stronger' definition of unification case h. above would lead to FAIL because the two argument categories don't explicitly contain the attributes num or per with variable values. The advantage of 'relaxing' the definition of unification in this way is that it

is notationally less verbose when categories have lots of attributes. (Jurafsky and Martin, ch15 give a more detailed introduction to unification of 'feature structures', using a slightly extended notation.)

Grammar 3 (G3) is a small UB PSG generative grammar; see if you can work out what structural descriptions it assigns to some examples and why it fails to assign any to sentences which violate per-num agreement or verb val(ence) constraints (verb subcategorisation).

```
Grammar 3

S:[] --> NP:[per=P, num=N] VP:[per=P, num=N]
VP:[per=P, num=N] --> V:[per=P, num=N, val=intrans]
VP:[per=P, num=N] --> V:[per=P, num=N, val=trans] NP:[]
VP:[per=P, num=N] --> V:[per=P, num=N, val=ditrans] NP:[] NP:[]
NP:[per=P, num=N, pronom=yes] --> N:[per=P, num=N, pronom=yes]
NP:[per=P, num=N, pronom=no] --> Det:[num=N] N:[per=P, num=N, pronom=no]

Sam N:[per=3, num=sg, pronom=yes]
I N:[per=1, num=sg, pronom=yes]
you N:[per=2, pronom=yes]
she  N:[per=3, num=sg, pronom=yes]
we  N:[per=1, num=pl, pronom=yes]
they N:[per=3, num=pl, pronom=yes]
cat N:[per=3, num=sg, pronom=no]
cats N:[per=3, num=pl, pronom=no]
sheep N:[per=3, pronom=no]
laughs V:[per=3, num=sg, val=intrans]
laugh V:[per=1, num=sg, val=intrans]
laugh V:[per=2, num=sg, val=intrans]
laugh V:[num=pl, val=intrans]
chases V:[per=3, num=sg, val=trans]
chase V:[per=1, num=sg, val=trans]
chase V:[per=2, num=sg, val=trans]
chase V:[num=pl, val=trans]
gives V:[per=3, num=sg, val=ditrans]
give V:[per=1, num=sg, val=ditrans]
give V:[per=2, num=sg, val=ditrans]
give V:[num=pl, val=ditrans]
the Det:[]
a Det:[num=sg]
those Det:[num=pl]
```

Exercises

Can you define precisely how to do a left/right-most derivation in UB PSG? Is

UB PSG a declarative formalism? (i.e. does it make any difference whether we choose to do a left- or right- most derivation to the results) (easy)

Try adding the analogues of some of the other rules and/or lexical entries developed for the CF PSGs, G1 and G2 and the exercises above (e.g. names, PPs) to the UB PSG, G3. (medium)

How could we add the case=nom/acc distinction to G3 in order to block examples like **Sam chases we*? (easy)

Think again about the analysis of auxiliary verbs – how does UB PSG help make it simpler and more effective? Can you think of any remaining problems which can't be handled elegantly in the UB PSG formalism introduced? How could we make rules even simpler if we labelled head daughters or had a convention about how to select the head daughter in a rule? (hard)

You can read more about such issues in J&M:chs 3, 12

2 Parsing

2.1 Recognition vs. Parsing

A recognizer given an input and a grammar decides whether that input is generated by the grammar or not – it returns yes/no.

A parser given an input and a grammar returns one or more derivations for the input according to the grammar, if the input is generated by the grammar (and an error message or partial derivation otherwise). To be 'correct' with respect to a grammar, a parser should return all and only the phrase structure trees associated by the grammar with a sentence.

Some important questions: are either of these tasks computable for natural language (NL)? If NL is CF, yes, if NL were RE, at the 'top' of Chomsky hierarchy, maybe not (see section 4 below). How efficiently?

Decision procedure – recogniser / parser guaranteed to terminate in finite number of steps

Parsing complexity – number of steps to return structures as function of length of input (in tokens n) – linear eg. 3n, polynomial eg. n^3 exponential eg. 3^n But constant factors (big-0 notation) e.g. length of the grammar, $\mid G \mid$, can dominate.

2.2 Local & Global Ambiguity

Parsing is a procedure which involves applying grammar rules to an input in some sequence.

Left-to-right / Right-to-left – parsing proceeds (usually) left-to-right (much as

people – *look out there's a fall...*); this means that the context for a parsing decision does not consist of the entire input.

Parsers not only need to deal with global syntactic ambiguity (more than one analysis), but also local syntactic ambiguity (temporary indeterminacies created by limited nature of the parsing 'window')

Visiting aunts can be boring
The farmer killed the duckling in the barn

Who does ...
Who does Kim want ...
Who does Kim want to kiss ...
Who does Kim want to kiss (Sandy / 0)

The cotton clothing is made of grows in Mississipi
Without her contributions to the fund would be inadequate
They told the girl the boy seduced the story

These latter examples are known as 'garden paths' in the psychological literature. They raise the issue of whether humans parse syntactically (somewhat) independently of other information (eg. intonation, semantics, etc.).

Exercise

Can you draw partial phrase structure trees which show the two different syntactic analyses up to the point of the resolution of the ambiguity in (some of) these examples? (easy)

2.3 Parsing Algorithms

Top-down / Bottom-up – parsing can proceed top-down, hypothesise an S node and try to find rules to connect it to the input, or bottom-up, starting from the first (or last) word of the input try to find rules which successively combine constituents until you find one S spanning them all.

Search & Non-determinism – parsing involves search at a number of levels, successively search grammar for a rule containing a category which matches the current item, search for combinations of rules which allow a successful parse, find all the combinations of rules which produce successful parses.

Breadth-first / Depth-first / Backtracking or Lookahead – differing search strategies, pursue all options in parallel, follow one 'path' through the search space & backtrack if necessary, use (limited) lookahead into right context to make correct choice.

2.4 Shift-Reduce Parsing for CF PSG

Depth-first, no backtracking:
Stack (PDS) + Input Buffer (Queue)
Shift next lexical item in Input Buffer onto top of stack (with lexical syntactic category/ies)
Reduce one or more constituents in top two cells of Stack whenever they match a rule of the grammar (i.e. replace right hand side (RHS) of rule with LHS in stack)

```
Grammar                    Lexicon
S --> NP VP                Kim, Hannah... : Nname
NP --> Nname               the, a.... : Det
NP --> Det N               boy, girl... : N
VP --> Vintrans            runs, smiles... : Vintrans
VP --> Vtrans NP           loves, hits... : Vtrans
VP --> VP PP               in, on, with... : P
PP --> P NP
```

```
Stack                                |  Input Buffer

---                                  | Kim hits the girl

(Kim, Nname)                         | hits the girl

(Kim, NP)                            | hits the girl

(Kim, NP) (hits, Vtrns)              | the girl

(Kim, NP) (hits, Vtrns) (the, Det)   | girl

(Kim, NP) (hits, Vtrns) (the, Det) (girl, N)

(Kim, NP) (hits, Vtrns) (the girl, NP)

(Kim, NP) (hits the girl, VP)

(Kim hits the girl, S)
```

Exercises

1) (all easy)
Is this algorithm bottom-up or top-down?
How much use of the left/right context are we making?
What are the conditions for success/failure with this parser? What would happen if we collapsed the distinction between Vtrans/Vintrans?

What more would we need to do if we wanted to output the phrase structure? Is this algorithm guaranteed to terminate?
What would happen if the grammar contained a rule such as NP $\rightarrow e$, where e is the empty symbol (no input)?

2) Try some other examples: (easy)

The boy hit the girl with the shovel
The boy hit the girl with the shovel in the park

3) Can you design a similar top-down algorithm and parse the previous two egs? Do you notice anything about the rule VP $\rightarrow$ VP PP? (medium)

4) Shift / Reduce Parsing (breadth-first) – remove the Vtrans/Vintrans distinction from the grammar and define a similar breadth-first parser capable of correctly parsing our first e.g. What modifications do you need to make to the parser to make this possible? Now add the rule NP $\rightarrow$ NP PP and parse the same egs. What do you notice about the behaviour of the parser? How efficient is it? (medium)

2.5 CFG Worst-case Ambiguity

In the worst case, the number of analyses of a string of length n can be exponential, approx., 3^n (which is why any parser which enumerates all possible analyses will have exponential complexity).

A simple example is noun-noun compounding in English. The noun compounds in (6) all consist of a sequence of English nouns. The bracketings indicate which noun (compound) modifies which other noun (compound).

(6) a winter holiday
b winter holiday resort ((winter holiday) resort)
c toy coffee machine (toy (coffee machine))
d water meter attachment screw ((w m) (a s))
e airport long-term car park courtesy vehicle pick-up point
f (((airport (long-term (car park))) (courtesy vehicle)) (pick-up point))

Since only adjacent noun (compounds) can be in a modification relation the constraints on noun compounding can be expressed with one doubly recursive CF rule: N $\rightarrow$ N N – which licences all the binary branching trees with nodes labelled 'N' over the number of nouns in the compound. (Actually, strictly speaking we need to allow for some Nom/N1 constituents, eg. *long-term*.)

The number of such binary branching trees is defined by the Catalan series, $C(n)$:

$$
\begin{aligned}
C(n) &= \left(\begin{array}{c} 2n \\ n \end{array} \right) - \left(\begin{array}{c} 2n \\ n-1 \end{array} \right) \\
&= \tfrac{2n!}{n!(2n-n)!} - \tfrac{2n!}{(n-1)!(2n-(n-1))!} \\
C(4) &= \tfrac{8!}{4!4!} - \tfrac{8!}{3!5!} \\
&= 70 - 56 = 14
\end{aligned}
$$

(where $n! = n(n-1)\ldots 1$ ie. factorial)
The number of trees for a n noun compound is given by $C(n-1)$. Here is a table for n [3-8] with $(n-1)!$ shown for comparison:

n	3	4	5	6	7	8
$C(n-1)$	2	5	14	42	132	469
$(n-1)!$	2	6	24	120	720	5040

So (6e) has 469 analyses, but this represents a great deal of syntactic constraint over the 'free word order' case where any noun (compound) can modify any other (non-)contiguous noun (compound) – this is equivalent to $C(n-1)(n-1)!$ interpretations!

PP attachment is another example of worst case CFG ambiguity which follows the Catalan series – see Church and Patil, 1982 *Computational Linguistics*, 8, ' Coping with syntactic ambiguity or how to put the block in the box on the table' for more details.

2.6 Chart Parsing

Well-formed Substring Tables (WFSTs) – a way of avoiding duplicated effort in non-deterministic parsing. All efficient algorithms use WFSTs in some way, often referred to as parse forests as they represent parse trees efficiently. The basic insight is that there is no point in looking for and parsing the same phrase in the input repeatedly just because it features more than once in various different derivations (recall the breadth-first version of the shift / reduce parser).

The standard technique used in NLP to implement this idea is known as the 'chart'. So I will describe this. You might like to think though, how the same idea could be used with the 'stack' in a breadth-first shift / reduce parser. The CYK (often CKY) algorithm, which is popular for statistical constituency parsers is a special case of chart parsing (see J&M:ch13).

A chart is a labelled, directed acyclic graph consisting of vertices and edges. The bottom of the graph contains the words of the input to be parsed. Vertices mark the boundaries of each word and of the complete input (the convention is 0 to n vertices). Edges span vertices and are labelled with syntactic categories. Edges which contain other edges are also labelled with the contained edges; i.e. VP[V,NP] labels a VP edge containing a V and NP edge, as illustrated

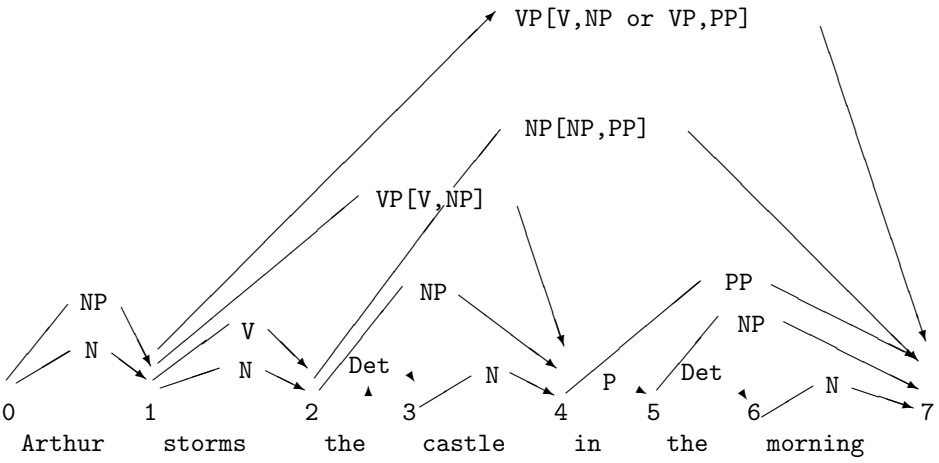


Figure 1: Chart Data Structure

informally in Figure 1 (where the containment information is not given for the NP edges).

Exercise (easy)

This chart is incomplete. Can you add the S edge which represents the most plausible reading?

The chart contains several edges which intuitively are not part of the final analysis (e.g. N, 1-2; NP, 2-7). In order to make this explicit and provide a way of recovering the phrase structure from the chart we need to annotate edges with the information about which other edges they contain (e.g. NP, 2-7 contains NP, 2-4 + PP 4-7). The chart inplements the WFST idea though, because it is possible for more than one edge to contain the same 'lower' edge (e.g. NP 2-4 is contained by NP 2-7 and by VP 1-4). The chart can represent global as well as local ambiguity, since we could add further S and VP edges which contained NP 2-7 to get the semantically unlikely reading in which the PP is attached to the N (*castle*).

Exercise (easy)

Can you add these edges and then annotate all the edges in the chart to show which other edges they subsume using the grammar from the previous section?

The chart data structure is a method for sharing partial subanalyses between alternative partial or complete analyses. It is neutral between parsing algorithms (and grammar formalisms).

Exercises

Can you define a breadth-first algorithm which builds a chart using the grammar above? (hard)

Can you show that it is more efficient than the shift / reduce, breadth-first

stack-based parser you defined above? (hard)

Active Charts – WFSTs stop duplication of successful subparses but don't prevent the same hypothesis being tried many times over if it fails, because nothing gets put in the WFST. Active charts avoid this inefficiency by augmenting the chart to contain active (incomplete) and inactive (complete) edges.

Active edges encode hypotheses about the input to be tested. Inactive edges encode completed (sub)analyses. To add active edges, we augment the labels on edges from categories to 'dotted rules' and allow single cyclic arcs in the chart to represent active edges which (so far) haven't subsumed any other edges. The active charts in Figure 2 contain inactive lexical edges and add active edges representing the hypotheses about the input. As parsing proceeds some of these hypotheses are proved correct and the dot moves on through the RHS of the rules encoded in the active edges.

So far, we have considered the chart as a data structure, but now we need to explain how parsing proceeds in terms of active and inactive edges. **The fundamental rule** of chart parsing is: if the chart contains an active edge from i to j and an inactive edge from j to k (where $i \leq j < k$) then add a new edge from i to k if the category after the dot in the active edge matches the category of the active edge. The new edge should have the same label as the active edge with the dot 'advanced one' (where A $\rightarrow$ B C . is equivalent to A).

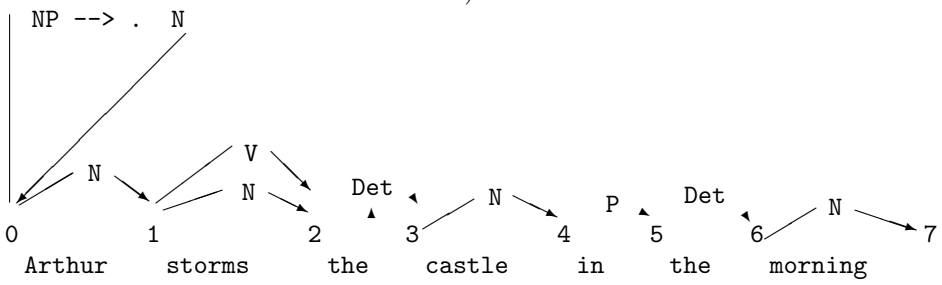
Exercise
Using this rule and the grammar above, see if you can simulate the behaviour of an active chart on a couple of the sentences we considered before. Assume that the fundamental rule is applied to every pair of adjacent active-inactive edge pairs and that active edges are added to the chart bottom-up keyed off the left-most RHS category. Is this the most efficient way of applying the fundamental rule and adding active edges? (easy)

Different parsing algorithms / search strategies correspond to different strategies for adding active edges and different orders of application of the fundamental rule to active-inactive pairs. Chart parsers usually employ an **Agenda** which is a list of edges. When a new edge is created by an application of the fundamental rule, it is put on the agenda. When we get further edges to work on, if we treat the agenda like a stack (last in, first out) we get depth-first search, if we treat it like a queue (last in, last out) we get breadth-first search. Depending on how we add active edges to the chart, we can get top-down, bottom-up or a mixed parsing strategy. For instance if we index rules by their leftmost RHS category and add active edges to the chart 'bottom-up', whenever we build one of these leftmost daughters, and then parse depth-first from this new edge, we will get a 'left-corner' parser (which has some characteristics of both bottom-up and top-down parsers).

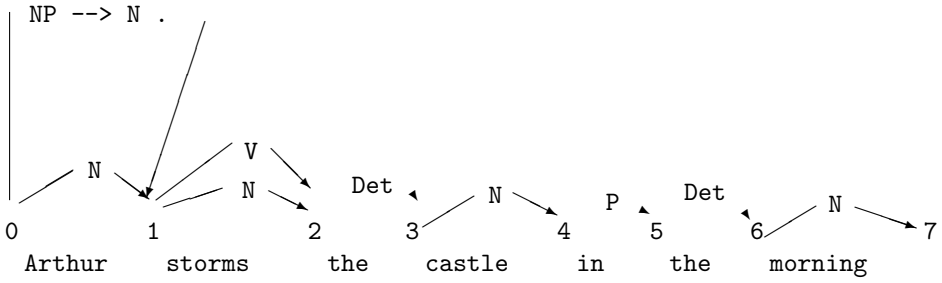
In describing charts we will use a pseudo-code notation where words in italics are variables. We use a Lisp/Prolog-style notation for lists, as follows:

`[]`	empty list
`[x | y ]`	first element of list is x, rest of list is y

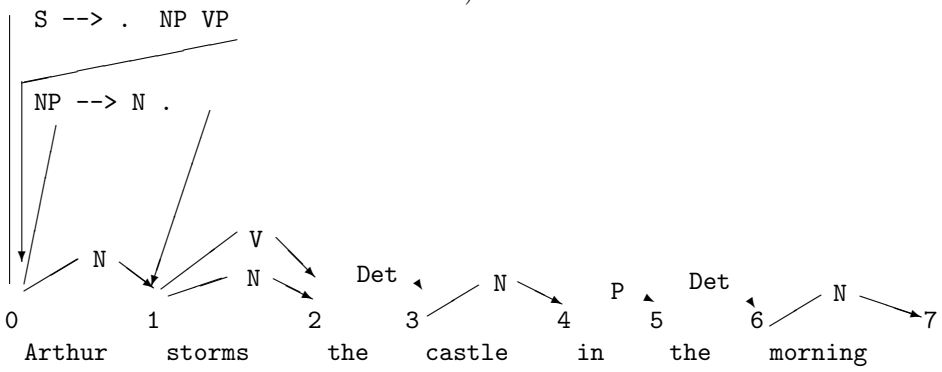
1)



2)



3)



4)

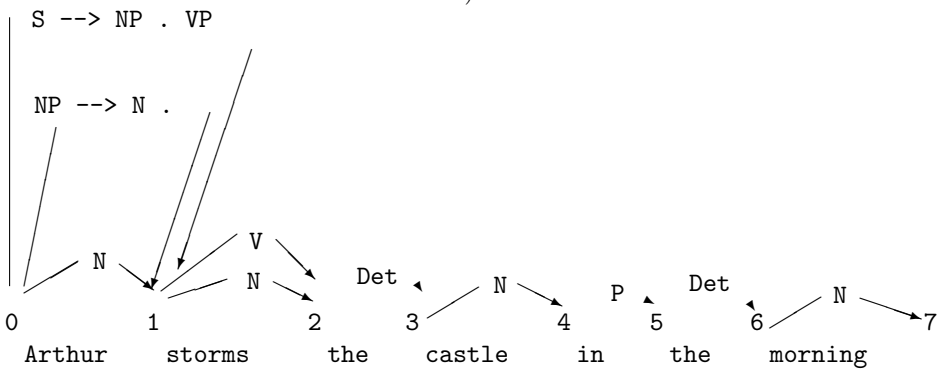


Figure 2: Active Chart Parsing

An edge has an identifier and connects vertices. DaughtersFound will usually be represented in terms of a list of other edges representing the analysis of those daughters. DaughtersSought is a list of categories. An edge is complete if DaughtersSought is empty.

[*id*,*left_vertex*,*right_vertex*,*mother_cat* ,*daughters_found*,*daughters_sought*]

To specify a particular instantiation of a chart framework, we need to state:

(i) a regime for creating new edges
(ii) a way of combining two old edges to form new edge
(iii) access to an 'agenda' of edges created by (i) or (ii) that are waiting for further processing when they are entered into the chart.

A bottom up chart parser

One particular chart based algorithm can be specified as follows: it proceeds bottom up, one word at a time.

New edges:
whenever there is a complete edge put in the chart of form:
[*id*,*from*,*to*,*category*,*found*,`[]`]
then
for each rule in the grammar of form *lhs* `->` *rhs*,
where *category* is the first member of *rhs*,
put a new edge on the agenda of form:
[*newid*,*from*,*from*,*lhs*,`[]`,*rhs*]

(Not all rules in the grammar meeting this criterion will lead to a complete parse. This step of the procedure can be made sensitive to information precomputed from the grammar so as to only select rules which are, say, compatible with the next word in the input, or alternatively, compatible with the next category sought of at least one incomplete edge ending at the point where the current word starts.

Combine Edges:
whenever a new edge is put into the chart of the form:
[*id1*,*b*,*c*,*cat1*,*found1*,`[]`]
then
for each edge in the chart of form:
[*id2*,*a*,*b*,*cat2*,*found2*,`[`*cat1* `|` *rest_sought* `]`]
create a new edge:
[*id3*,*a*,*c*,*cat2*,**append**(*found2*,*id1*),*rest_sought*]

whenever a new edge is put into the chart of the form:
[*id1*,*a*,*b*,*cat1*,*found1*,`[` *cat2* `|` *rest_sought* `]`]
then

for each edge in the chart of form:
[*id2*,*b*,*c*,*cat2*,*found2*,`[]`]
  create a new edge
  [*id3*,*a*,*c*,*cat1*,**append**(*found1*,*id2*),*rest_sought*]

The first part of Combine Edges is triggered by the addition of a complete edge to the chart, and produces a new edge for each incomplete edge ending where the complete edge begins which can combine with it. These incomplete edges are already in the chart. The new edges are put on the agenda.

The second part is triggered by the addition of an incomplete edge to the chart, and produces a new edge for each complete edge beginning where the incomplete edge ends. These new edges are put on the agenda.

Looking at things from the point of view of a complete edge, the first part of Combine Edges ensures that it is combined with whatever is already in the chart that it can be combined with, whereas the second part ensures that it will be combined with any future incomplete edges entering the chart. Thus no opportunity for combination will be missed, at whatever stage of parsing it arises.

All we have to do now to specify a complete chart parsing procedure is to define access to the agenda: we can choose to treat the agenda as a stack (last in, first out) in which case the general search strategy will be depth first; or as a queue (first in, first out) in which case we will search hypotheses breadth first. We could also have more complex heuristics ordering edges on the agenda according to some weighting function: this would mean that the highest scoring hypotheses were explored first, independently of the order in which they were generated.

We also have to embed the procedure in some kind of top level driver, so as to start off the process, and check for complete analyses when the process is ended. The final program, then, might have the following structure:

Until no more words:
  create new edges for next word
  do New Edges for these edges.
  Until agenda is empty:
    pop next edge off agenda and put in chart
    do New Edges
    do Combine Edges
Check for complete edges of desired category spanning start to finish

Given the following grammar the algorithm would proceed as follows on the input sentence 'they can fish'.

```
S   --> NP VP
NP  --> they | fish
VP  --> Aux VP
```

```
VP  --> Vi
VP  --> Vt NP
Aux --> can
Vi  --> fish
Vt  --> can
```

Operation	Chart	Agenda
new word edge		e1(1,2,NP,[they],[])
pop	e1	
new edge		e2(1,1,S,[],[NP,VP])
pop	+e2	
combine e1 e2		e3(1,2,S,[e1],[VP])
pop	+e3	
new word edges		e4(2,3,Aux,[can],[]),
		e5(2,3,Vt,[can],[])
pop	+e4	
new edge		e6(2,2,VP,[],[Aux,VP]),
		e5
pop	+e6	
combine e4 e6		e7(2,3,VP,[e4],[VP]),
		e5
pop	+e7	e5
pop	+e5	
new edge		e8(2,2,VP,[],[Vt,NP])
pop	+e8	
combine e5 e8		e9(2,3,VP,[e5],[NP])
pop	+e9	
new word edges		e10(3,4,Vi,[fish],[])
		e11(3,4,NP,[fish],[])
pop	+e10	e11
new edge		e12(3,3,VP,[],[Vi]),
		e11
pop	+e12	e11
combine e12 e10		e13(3,4,VP,[e10],[]),
		e11
pop	+e13	e11
combine e7 e13		e14(2,4,VP,[e4,e13],[]),
		e11
pop	+e14	e11
combine e3 e14		e15(1,4,S,[e1,e14],[]),
		e11
pop	+e15	e11
pop	+e11	
new edge		e16(3,3,S,[],[NP,VP])
combine e9 e11		e17(2,4,VP,[e5,e11],[])
		e16
pop	+e17	e16
combine e3 e17		e18(1,4,S,[e1,e17],[])
		e16

```
pop             +e18              e16
pop             +e16
combine e16 e11                   e19(3,4,S,[e11],[VP])
pop             +e19
```

At this point no more processing can take place. Inspecting the chart we find that we have two complete edges spanning the input, of the desired category, S. By recursively tracing through their contained edges we can recover the syntactic structure of the analyses implicit in the chart. Notice that as well as sharing some complete subconstituents (edge 1), the final analyses were built up using some of the same partial constituents (edge 3).

Exercise (easy)

Construct a similar table for the input 'fish fish' using the same grammar.

See J&M:ch13 for more on parsing and CFGs.

3 Parsing Performance and Complexity

3.1 Chart Parsing and UB-PSGs

Extending chart parsing to unification-based grammars is easier if we define a context-free 'backbone' from which we can trigger edge creation and combination operations. In the UB-PSG formalism this is trivial since we can simply use the functors (S,NP,VP,etc) in the categories in a UB PSG (see Linguistic Description). However, some approaches such as HPSG, the LKB, etc treat categories as undifferentiated bundles of features and, when this is combined with the use of a list-valued feature system, we need to be careful to ensure that rule invocation and edge combination are keyed off attributes in categories whose values will be instantiated at the relevant point in parsing. Basically, this means defining a proper subset of features, typically not including ones like GAP which handle unbounded dependencies (see next handout), whose values are instantiated before parse time (see Gazdar and Mellish for further discussion).

3.2 Unification and Non-determinism

When categories in edges are unified during parsing a copy must be taken to ensure that the original category and edge is available for use in other (sub)analyses. This is because unification is implemented as a destructive operation. However, copying categories can create a substantial (constant) parsing overhead, affecting efficiency considerably, particularly when categories are large, make heavy use of list/category-valued features, and so forth. There is a substantial literature on schemes for more efficient parsing involving structure sharing, delayed evaluation, and more selective memorisation than is implicit in active chart parsing (see Gazdar and Mellish, J&M:ch14). Most efficient parsers

use a mixture of precompilation techniques and quasi-destructive unification operations, but the precise details are very dependent on the type of unification grammar and even style of the grammar writer.

3.3 Subsumption Checking

Context-free chart parsing requires a recursion check to ensure that the same recursive rule is not invoked ad infinitum leading to non-termination. For example, a bottom-up parser invoking rules from leftmost daughter categories will loop if the grammar contains a rule such as N1 $\rightarrow$ N1 Srel. (Such behaviour can also be caused by indirect recursive loops in the grammar.) In chart parsing it is straightforward to implement a check to see if an active edge corresponding to such a rule has already been introduced into the chart at a given vertex. However, in the unification-based case this needs to be generalised to a subsumption check (ie. a check that the putative new active edge is more general than the existing one).

3.4 Packing

The standard active chart parsing algorithm does not include so-called packing where the Found list on an edge can contain disjunctions of conjunctions of contained edge identifiers, obviating the need to introduce further edges identical apart from the containment relations they encode. (Recall, we looked at examples of such edges in the parsing of noun compounds with a rule like N $\rightarrow$ N N.) Packing is implicit in tabular parsing algorithms such as CYK and Earley (see J&M:chs 14,15) which are closely related to chart parsing. It is straightforward to extend the algorithm given above in the context-free case by including a check for existing edges spanning the same vertices and resulting in the same category during combine edges. However, in the unification-based case this check must once again be a check for subsumption rather than identity. If neither category subsumes the other packing can still be achieved if categories share the same backbone category by in effect delaying the binding of attributes to specific values in superordinate edges. Packing is essential for polynomial worst-case complexity (though does not alone guarantee this in the unification-based case)

3.5 Rule Invocation

The efficiency of rule invocation can significantly affect parser performance particularly when grammars are large. (A typical wide-coverage unification-based grammar which does not utilise a radically lexicalist approach will contain around 1k PS rules.) Active edges can be added to a chart using a number of strategies – bottom-up, left corner etc and the grammar can be preindexed to compute which subset of rules can potentially be applied in different parse contexts.

Some researchers (e.g. Tomita, 1987, An efficient augmented context-free parsing algorithm *Computational Linguistics*, 13) have advocated a generalised version of LR parsing in which LR grammar precompilation techniques are combined with a chart-like graph-structured stack so that LR tables with shift-reduce and reduce-reduce ambiguities can be parsed efficiently. (You may have come across LR parsing algorithms in the context of parsing and compiling programming languages.) LR tables, in this case, are non-deterministic finite-state automata which specify contextually appropriate parse actions given a parse state and lookahead item. LR techniques will yield, in principle, the most effective rule invocation strategies that can be automatically found for a given grammar. However, the LR table itself can be exponentially larger than the grammar using standard precompilation techniques, the preprocessing can be expensive, and the informativeness of the CF backbone derived from a unification-based grammar can critically affect performance.

3.6 Worst vs. Average Case Complexity

In parsing complexity results, grammar size is treated as a constant factor and assumed to be non-significant. But if the grammar gets big enough it may be that the overhead of searching for appropriate rules etc. is so great that for 'normal' inputs (i.e. sentences of, say, up to 60 words) this is a real potential source of inefficiency. Sensible rule invocation strategies with an active chart parser (or most other methods using WFSTs) gives (worst-case) polynomial complexity as a function of length of input when using an ambiguous CF grammar (e.g. n^3).

For UB PSG wide-coverage grammars of natural language, the evidence is that worst-case parsing results are less important than empirically-derived average case performance results on typical input. Carroll (1994, ' Relating complexity to practical performance in parsing with wide-coverage unification grammars' *Proc. of ACL*) demonstrates that a worst-case exponential LR parser tuned to a specific grammar outperforms a worst-case polynomial variant of the same parser, yielding approximately quadratic average complexity in the length of the input. Properly optimised chart-like parsers often seem to produce average case approx. quadratic (n^2) behaviour in experiments.

It appears that worst-case results do not dominate because a) NL grammars are predominantly binary branching (i.e. few rules like: VP $\rightarrow$ V NP PP), and b) there are very few rules which license both left and right recursion (i.e. few rules like: N $\rightarrow$ N N). Instead optimisations to deal with quite specific properties of individual grammars seem to be more important (eg. the type of unification used and how it is optimised in the face of non-determinism).

See J&M:ch15 for a related discussion of unification and parsing.

3.7 Formal Language Theory

There is a hierarchy of classes of formal languages and associated grammars known as **The Chomsky Hierarchy** (Chomsky worked in fml lg theory / theoretical comp. sci as well as linguistics in the '50s and early '60s).

The weakest class is the class of regular languages which can be generated by finite-state automata (FSAs). An example is $a^n b^m$ (i.e. any string of n *a*s followed by m *b*s is 'grammatical'). Can you define a FSA which generates this language? There is an equivalent 'production rule' notation for defining regular grammars in which rewrite rules are restricted to be left/right linear with at most a single non-terminal on the left/right of the arrow.

Exercise (easy)

What language does the following right linear grammar generate?

```
S --> a S
S --> a B
B --> b (B)
```

Context-free grammars and the associated class of CF languages properly include the regular grammars / languages and others like $a^n b^n$ (i.e. balanced strings of *a*s followed by *b*s: ab, aabb, aaabbb, *abb, etc.) which are CF but not regular. A CF rule can have any number of (non-)terminal daughters.

Exercises (easy)

What language does the following CF grammar generate?

```
S --> a S b
S --> a b
```

Now try to write a regular grammar for this language! (impossible)

CFGs are equivalent to push-down automata (PDAs i.e. FSAs equipped with a push-down store or stack). CFLs exhibit nested dependencies. Indexed (I) languages / grammars properly include the CF grammars / languages and also others like $a^n b^n c^n$ (i.e. balanced strings of *a*s, *b*s and *c*s: abc, aabbcc, *abbcc, etc.) which is not CF – it exhibits so-called cross-serial dependencies in which the first a, b and c are codependent, then the second set are all codependent, and so on. I grammars are equivalent to nested PDAs (NPDAs) in which each cell of the push down store can itself be a PDA. After this there are context-sensitive (CS) grammars / languages. An example of a CS, non I, language is $\{a^n, b^n, c^n\}$ (i.e. balanced strings of *a*s, *b*s and *c*s occurring in any order: ababcc, aabbcccab, etc) exhibiting arbitrarily intersecting dependencies (the so-called MIX lgs). Finally comes the largest category of all, the recursively-enumerable

(RE) grammars / languages, which includes all languages whose description is shorter than the set of strings in the language. These can be generated using unrestricted rewrite rules in which one or more non-terminals can be rewritten as any (non-)terminal sequence. These grammars are equivalent to Turing Machines (TMs). So we have the following inclusion hierarchy:

Reg < CF < I < CS < RE

See J&M:ch16 for some more on formal language theory.

3.8 Human Lg and the Chomsky Hierarchy

The evidence that human lgs are not regular languages and thus captured by FSAs / regular grammars is based on the grammaticality of centre/self-embedding constructions like (7)

(7) a The students the police arrested laughed
b ??The students the reporters the police arrested interviewed laughed
c ?The rumour that the students the police arrested laughed spread quickly

These examples require non-regular rules of the form N1 → N1 Srel. However, the depth to which such embedding is grammatical is clearly questionable. The standard story is that they are grammatical to any depth but only acceptable in use to depth 1, or occasionally 2, due to human working memory limitations. Centre-embedding seems more acceptable than self-embedding to the same depth ((7b) vs. (7c)). Perhaps a stronger argument for using at least CF equivalent formalisms is that they naturally capture the hierarchical nature of constituent structure – an argument based on so-called strong generative capacity (the descriptions paired with strings / sentences) as opposed to weak generative capacity (just the strings / sentences).

Our UB PSG Grammar 4 for the fragment of English we have covered is CF equivalent in the sense that we could automatically compile it into a finite set of CF rules. All we need to do is expand out each attribute in each rule with each of its finite set of possible values and generate new rules for all possible attribute-value combinations – we may end up with a vast set of rules, but as long as it is finite, it is still a CFG. (Actually we allowed regular expressions in the right hand side of rule schemata, but these still only license CF languages with phrase structure trees that could have been generated by *some* CFG.)

It would be a strong (and useful) result if all human languages turned out to be CF. However, there is evidence that human lgs, in general, are not CF and some evidence that certain (rarish) constructions of English aren't either. Basically cross-serial as well as nested dependencies seem to crop up, but not arbitrarily intersecting dependencies (MIX lgs). So the consensus is human languages fall

into the class of indexed languages, a proper superset of the CF languages and proper subset of the context-sensitive languages.

An indexed grammar can be constructed out of a CF grammar 'backbone' with categories employing at least one attribute with an open-ended value (i.e. the category set is no longer finite.)

UB PSG for the indexed lg $a^n b^n c^n$:

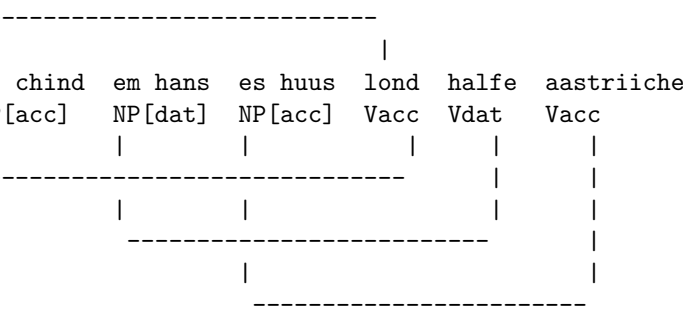
S:[s=H|T] → S1:[s=H|T] S2:[s=H|T] S3:[s=H|T]
S1:[s=H|T] → a S1:[s=T]
S1:[s=[]] → a

(and ditto for S2 and S3, where value of s=h|t is a (Lisp/Prolog) list consisting of a 'head' element and a 'tail' of further elements; equivalent to a push down store / stack).

So far we have only utilised attributes with a finite set of possible values in our UB PSG for F2. However, it is a straightforward extension of UB PSG to allow attributes which take open-ended list-values of this kind. Once we do this we can no longer compile from UB PSG into a CFG with even equivalent weak generative capacity.

Cross-serial dependencies occur in Swiss German of the form NP^n VP^n:

mer d' chind em hans es huus lond halfe aastriiche
we the children-ACC Hans-DAT the house-ACC let help paint
we let the children help Hans paint the house



So UB PSGs have the expressive power to capture (all?) rules of human syntax, but we have also seen that FSA/Ts and CFGs may potentially be useful for certain natural language processing (NLP) tasks.

Exercises

1) What do the examples in (8) tell us about where English syntactic rules lie on the Chomsky hierarchy? (medium)

(8) a If Kim comes, (then) I'm leaving
b If Kim comes, (then) if there's a fight, (then) I'm leaving
c *If Kim comes, a fight

2) Construct a case for natural languages being finite-state based on the data introduced above. (Hint: centre-embedding and cross-serial dependencies are very complex / rare beyond depth one.) How might strong generative capacity / the need to represent the form:meaning mapping affect this argument? (hard)

3) Write a grammar which generates cross-serial dependencies in UB PSG. See if you can extend it to generate MIX languages (Hint: think about lexical/unary rules.) (hard)

4 Statistical Techniques for NLP

4.1 Probabilistic Disambiguation Motivated

Since parsers typically return more than one syntactic analysis and associated logical form for a given input, due to the high degree of ambiguity in natural language, how do we resolve these ambiguities and obtain the correct analysis? The traditional answer in NLP was that such ambiguities should be resolved in a knowledge-based fashion in a domain- and application-dependent way. Winograd's SHRDLU remains a classic example of such a system (eg. Winograd, 1972). In this system, a user could control a simulated robot operating in the 'blocks world' and able to understand commands like *Put the blue pyramid in the red box on the table*. The resolution of such PP attachment ambiguities was performed via domain-specific knowledge – eg. objects cannot be on top of pyramids – and examination of the context – is there a blue pyramid in a red box in the current context? This approach led to some impressive systems but not to commercially-viable NLP technology. For example, NL database access systems worked quite well when tuned to a particular database domain but needed considerable expert effort for effective porting to a new domain. Other potential applications, such as document summarisation for information extraction from the Web, are not domain-specific. The problem is that there is just too much fine-grained knowledge required to resolve every potential ambiguity. Also the degree and nature of the ambiguity is much greater and much more perverse than we typically recognise (except via long and bitter experience of trying to build automatic parsers!) – see Abney (1996a) 'Statistical methods and linguistics' (http://www.sfs.nphil.uni-tuebingen.de/ abney/) for a good discussion.

4.2 Treebanks

Treebanks are collections of annotated corpora where linguists have manually or semi-automatically assigned the correct analysis from some grammatical frame-

work to each sentence. Such treebanks are resource intensive to create and prone to errors. Nevertheless, there was a big effort to create some from the mid '80s to '90s. The most popular is the Penn Treebank, Wall Street Journal dataset (hereafter WSJ treebank), which contains about 1M words (40K sentences) of annotated financial newspaper text. However, there are others, such as Susanne, a genre/topic-balanced subset of annotated texts from the Brown Corpus. Treebanks are annotated with PoS and punctuation tags and with labelled bracketings which utilise around a dozen non-terminal categories (e.g. S, NP, VP) and some functional and relational features (e.g. NP-SUBJ, PP-TMP, NP-ADV). Here is the first sentence of WSJ treebank:

```
(S (NP (NP Pierre Vinken)
       ,
       (NP (NP 61 years)
           (ADJP old))
       ,)
    will
    (VP join
        (NP the board)
        (PP as
           (NP a nonexecutive director))
        (ADVP (NP Nov 29))))
```

(See Manning & Schultze 1999:412f (henceforth M&S) for discussion, more examples, etc. or J&M:3rd ed; chs 12-14)

4.3 Stochastic PoS Tagging

One area where statistical techniques have been successfully used in NLP is PoS and punctuation tagging. The task is to select the correct PoS or lexical syntactic category for a word in context (see Linguistic Description handout) and similarly label punctuation marks (though these are usually unambiguous). Rules for doing this can be stated (largely) in terms of the immediate lexical context so a tagger can be formalised as a finite-state automaton (FSA). However, the FSA will be non-deterministic and will often output many possible PoS tag paths for a given input. Hidden Markov Models (HMMs) are probabilistic versions of FSAs which encode two types of probability: *Transition probabilities* between states representing the conditional probability that one, two or n PoS tags will be followed by tag_k eg. $P(tag_k \mid tag_i, tag_j)$ and *Lexical/Emission Probabilities* that, given a tag, the model will output a particular word form, $P(word_j \mid tag_k)$. Such models are known as *language models* because they assign a probability to any sequence of words generated by the model and the total probability of all such sequences for the language is 1. They differ from Ngrams / MMs used as language models for word prediction in speech recogni-

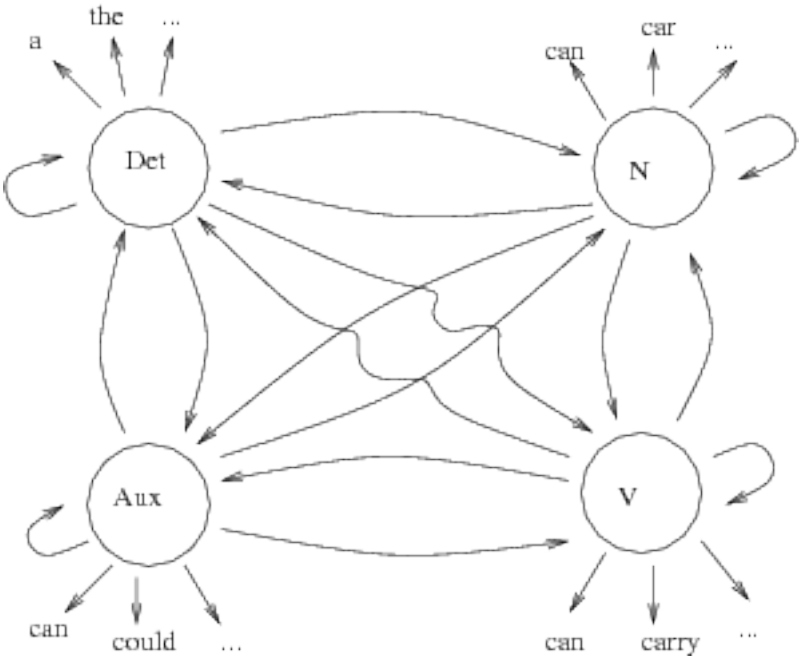


Figure 3: Simple Ergodic 1st Order HMM Pos Tagger

tion systems in that there is an additional hidden layer or state sequence, the PoS and punctuation tags. (See Charniak, 1993, M&S, ch9, J&M ch6-8 for more on Ngrams, HMMs and PoS tagging and M&S for brief revision of probability theory).

Figure 3 shows a transition diagram for a simplified bigram PoS tagger in which states are labelled with PoS tags and transitions will be labelled with probabilities, such that the probabilities of all transitions from a given state must sum to 1 and all lexical outputs from a given state sum to 1. The model is ergodic because any state can be reached from any other state. A full model will also require START and END states for sentence boundaries (which will also be reachable from any other state). Finding the most probable path or state sequence requires taking the product of the lexical and transition probabilities for each possible path through the HMM, so if we have an observed sequence of words like *I can can a can* we find all the sequences of 'hidden' PoS tags that could have generated that sentence and compute the probability of each such sequence: $P(N \mid START) \times P(I \mid N) \times P(Aux \mid N) \times P(can \mid Aux) \ldots$ In general, there will be an exponential number of such paths bounded by the length of the sentence and the size of the HMM. However, the Viterbi algorithm is a method of incrementally pruning submaximal paths which guarantees finding the maximal one in linear time.

The Viterbi algorithm is based on the observation that there is a locality constraint on computing the maximally probable derivation / state path dependent on the order of the HMM. Given a bigram / 1st order model, it is only necessary to remember the maximal subpath from $word_1$ to each possible tag for $word_3$ because any maximal extended subpath to $word_4$ must incorporate one of these subpaths. Thus at $word_3$ it is safe to prune the non-maximal paths because they cannot play any further role in the maximal path / derivation for the input. However, if we want to do probabilistic thresholding of the kind discussed below then we need to use the forward-backward algorithm which is quadratic in the length of the input and state space.

In PoS tagging we are interested in finding likely PoS tag sequences given an input word sequence (ie specific paths through the HMM). This is different from the speech recognition (Ngram) application of (typically) predicting the next word/phoneme where we are not interested in particular derivation paths. Once we have moved to the probabilistic setting it makes sense to use the most liberal model given a specific tagset which does not rule out any possible combination of tags (e.g. Det-V-Det...). This can be defined automatically as the ergodic nth-order model over the PoS tagset in which every state is accessible from every other state (see Figure 3 above).

Exercises (medium)

How is the probability of a word sequence calculated?
How is the probability of the next word or tag, given a partial word sequence, calculated?
What independence assumptions are built into the model?

4.3.1 Parameter Estimation

Transition probabilities and lexical probabilities can be estimated from a corpus which has been manually annotated with the correct PoS symbols. A standard method of parameter estimation is to maximise the probability of the training corpus relative to the model adopted (maximum likelihood estimation, MLE, see section 5.1 for details). The parameters are estimated from annotated training data, in which each word is manually associated with its contextually correct PoS tag, so that the probability of each sentence of the training data is as high as possible given the model. The probability of a sentence, $P(S)$, is the sum of the probabilities of the different state paths through the HMM which could have generated the sentence. This probability will be maximised if the transition probabilities (TP) and lexical probabilities (LP) are estimated by their relative frequency in the annotated training data; e.g:

$$\text{LP} = P(can \mid \ Aux) = \frac{freq(can{:}Aux)}{freq(Aux)}$$

$$\text{TP} = P(Aux \mid \ N) = \frac{freq(N{+}Aux)}{freq(N)}$$

Given 1M+ words of training data such systems perform on 'similar' test material with around 95-7% accuracy, defined as the ratio of correct PoS symbols assigned to all symbols assigned (i.e. precision=recall=accuracy in forced choice mode). It turns out that lexical probabilities alone yield 90% accuracy – the fact that *can* is most frequently a modal auxiliary is really the most important piece of information in our toy example. However, lexical information is the most difficult to acquire accurately and comprehensively.

Exercises (hard)

Why is this the case?
What might we do to tag a sentence containing a word (form) or requiring a word:PoS pair that didn't occur in the training data? Maximum likelihood estimation will assign zero probability to such events. It amounts to the statistical assumption that the training data is 'representative'. Can you see why? (We'll return to the problem of unseen events when we consider probabilistic approaches to parsing.)

Since some words are unambiguous typically these results mean less than 95% accuracy at disambiguation when a word:tag pair has been seen in the training data, and this can drop to as low as 70% for unseen words (and zero for unseen word:tag pairs), unless sophisticated smoothing and/or unknown word handling techniques are used.

There are certain constructions which are known to cause many tagger errors. N/A N sequences such as *good breakfast* or *coffee machine* are the most common since so many open-class words can function as adjectives and nouns. But many kinds of long-distance dependencies which fall outside the range of bigram or trigram sequences also cause problems; for instance, a conjunction such as *and* is typically preceded and followed by constituents of the same type – where these are lexical they are in range of a trigram / 2nd-order HMM, but when phrasal or clausal they will be out of range. There are many potentially useful rules, such as any finite clause can only contain one finite verb, which are beyond the range of Ngram-style rules, prompting other approaches (see Linguistic Description handout).

Nevertheless, the basic technique works well enough to be useful, especially when combined with a good approach to unknown words. These make use of word features such as suffixes, capitalisation, and so forth, to create an initial probability distribution for candidate PoS tags for different types of unknown words. One problem with HMM taggers built entirely from annotated data is that rare words may only be seen with a subset of their possible tags. These will not be treated by special unknown word handling components and will always be tagged incorrectly when they occur in test data and require an unseen tag. One possibility which has been explored is to smooth the lexicon and remove words seen only once or add extra tags predicted in a general dictionary.

One way tagging can be used is to reduce lexical ambiguity before parsing – then it is typically best to return all tags and associated probabilities which are within some probabilistic threshold or ratio of each other to maximise precision

against recall and avoid filtering out the correct tag – the error rate can be reduced to about 0.1% by allowing a mean of 1.3 tags/word.

Unsupervised training of PoS taggers on unannotated text using estimation maximisation (EM) / the forward-backward algorithm does not work well because the system is given no information about which are the linguistically likely tags (see J&M:307 and ch7). An HMM is a stochastic generator of a language in which the derivations are hidden through the use of output / lexical probabilities conditioned on PoS tags, so maximising the probability of a corpus of observations using an ergodic model places no linguistic constraint on how the probability mass gets distributed in derivation space. However, some types of semi-supervised training where general linguistically-motivated constraints on the model are used in place of annotated training data have achieved comparable levels of accuracy (see Cutting *et al.*, 'A practical part-of-speech tagger' ANLP, 1992). Thus accurate PoS taggers for English and increasingly other well-studied European languages, Japanese etc are available, but for many languages accurate annotated training data in quantity remains unavailable.

4.4 Stochastic Regular Grammar

HMM PoS taggers are typically based on ergodic FSAs (i.e. every state is reachable from every other state). This means that no tag sequence is ruled out and all derivations (state sequences) for a given word sequence pass through the same number of states (as there are words in the input). This has led many to claim that PoS tagging is qualitively different from 'grammar-based approaches'. However, unsurprisingly for any HMM model there is a corresponding stochastic regular grammar. For example, for the 'bigram', 1st-order HMM tagging model, the corresponding regular grammar is of the form:

$p,S^1 \rightarrow tag^1\ (S^2)$
$p,S^2 \rightarrow tag^2\ (S^3)$
...
$p,S^n \rightarrow tag^n\ (S^{n+1})$

$p,tag^1 \rightarrow w1 \mid w2 \mid w3 \ldots$

where all possible sequences of tags in the tagset are licensed, and the probabilities of the S rules correspond to transition probabilities, and those of the unary tag rules to output or lexical probabilities. Thus, there is a very close connection between a generative grammar and a language model – the latter being essentially a probabilistic / stochastic version of the former. (In fact, the probabilities assigned to word sequences by the two models will not always be identical – see M&S, 1999; Charniak, 1993). This close connection suggests that if we want to use statistical techniques to rank syntactic analyses, stochastic / probabilistic versions of CFG or UB PSG grammars may also be useful.

Exercise

Is there any difference between a stochastic and a probabilistic model?

5 Stochastic Context-free Grammar

A SCFG is a CFG (see Linguistic Description handout, J&M:326f) with a probability associated with each rule such that the probabilities of all rules expanding a given non-terminal category sum to one and, thus, the probability of all the (infinite) possible derivations of the grammar (and sentences of the language) also sum to one (see e.g. J&M:ch14; or M&S:ch11 for more detailed definitions, more formulas, some standard algorithms, and so forth).

A CFG consists of a finite set of productions or rewrite rules of the form $A \rightarrow \alpha$, where $A \in NT$ and $\alpha \in (NT \cup T)^*$. A SCFG is a CFG with a probability associated with each rule such that the sum of the probabilities of all distinct expansions of any given 'mother' category on the LHS of a rule, A, is one:

$$\forall A \in NT, \sum P(A \rightarrow \alpha) = 1$$

The probability of a given derivation is the product of the probability of the rules used in the derivation:

$$P(D) = \prod_{R \in D} P(R)$$

and the probability of a sentence is the sum of the probabilities of the derivations of that sentence:

$$P(S) = \sum_{D \in S} P(D)$$

(analogously to the HMM/regular grammar case).

Exercise

What independence assumptions are being made? (easy)

We want to use a language model capable of capturing hierarchical (constituent) structure – it is essential to recovering logical form – but there are some problems with SCFGs which don't occur with PoS taggers. Unlike the case of Ngram models / HMMs, there is nothing in the definition of SCFGs which ensures that all derivations for a given sentence will utilise the same number of rules (rules can have different numbers of daughters). This means that the standard formulation of a SCFG favours shorter over longer derivations (and shorter over longer sentences). SCFGs do not model lexical probabilities in any way so they are insensitive to the words occuring in the input. Why might this be problematic? SCFGs predict that the probability of a phrase being expanded in a certain way is a global (i.e. not context sensitive) property of that phrase; eg., this predicts (erroneously) that pronouns will be as frequent in subject as in object position (assuming pronouns are expanded by the same rule, NP $\rightarrow$ Npro).

Exercises

Show that *He kissed the girl* has the same probability as *The girl kissed him* in a simple SCFG (easy)

Show that the derivation for *He saw the girl with the telescope* in which the PP is an argument of the verb is very likely to have higher probability than one in which the PP is a nominal or verbal modifier (medium)

There are generalisations of the Viterbi algorithm to SCFGs which allow efficient computation of the most likely derivation of a given input. Perhaps the easiest wat to see this is to note the equivalence between a CFG and a (recursive) transition network (see e.g. Gazdar and Mellish). Once a grammar has been transformed into this representation probabilities on transitions represent the probability of expanding a given mother category with a specific rule, and a Viterbi-like algorithm can be defined. There are also versions which exploit chart-like parse forest data structures to obtain parsers for SCFGs capable of ranking derivations efficiently.

5.1 Maximum Likelihood Estimation (MLE)

In any probabilistic (language) model we need to estimate the parameters (individual probabilities) of the model from data (given that no standard distribution is likely to provide a good estimate). Thus in the case of a SCFG, we want to estimate probabilities for each rule of the grammar. The standard estimation technique is MLE. Suppose we have a training corpus of sentences associated with (hopefully correct) derivations and we are trying to estimate the probability of NP $\rightarrow$ Npro, then we proceed by counting the total number of NP 'events' (i.e. rules expanding NP) in the training data and estimate this probability as the ratio of times the NP is rewritten as a pronoun over the number of times NP is rewritten as anything:

$$P(NP \rightarrow Npro) = \frac{f(NP \rightarrow Npro)}{f(NP \rightarrow \alpha)}$$

(where f denotes frequency) and so on for each rule of the grammar, i.e. more generally:

$$P(A \rightarrow \alpha) = \frac{f(A \rightarrow \alpha)}{f(A)}$$

MLE will assign the training corpus the highest probability possible given the model, hence the name (see e.g. Abney 'Stochastic attribute-value grammars' *Computational Linguistics* 23.4, 1997 for a careful and accessible discusion of MLE for SCFGs and more generally M&S J&M, or Charniak, 1993 on MLE). The justification for using MLE is the assumption that the training data is a representative or 'good' sample of the phenomenon being modelled. However, MLE won't work well for natural language because most distributions (whether of words, constructions, subcategorisation frames, or whatever) appear to be 'Zipfian'.

Words, for instance, have a highly-skewed, hyperbolic distribution. That is, the distribution approximates a power law of the form: $\frac{1}{RF^a}$ where $1 < a < 2$ and RF is the rank frequency of a word. So, for example, *the*, the most common

word with a rank frequency of one will have a probability approximated by $\frac{1}{1^a}$, where a is around 1.3, and so forth. (A 'doubly exponential' distribution of this form will approximate to a straight line when plotted on log-log scales with probability plotted against ranked frequency (see M&S:23f).

So given Zipf's Law, there will be a small proportion of very frequent constructions and a large tail of very infrequent ones. This so-called data sparsity problem means that there will always be a significant number of unseen events (unused rules, words not seen with a specific rule or complementation pattern, etc) for realistic grammars and (annotated) training corpora. MLE assigns these unseen events zero probability. Sometimes this is referred to as 'overfitting' the training data, because the model will not generalise well to (unseen) test data.

5.2 Parameter Smoothing – 'Add1'

Parameter estimation is a vast topic and fundamental to successful statistical NLP. The simplest way to modify MLE to deal with unseen events is to use Laplace's Law or the 'Add1' technique. We revise MLE to add one (phantom) observation to every frequency count required to obtain an MLE for a given model so that unseen events each have a frequency estimate of one; for the SCFG case this means we use the following general schema:

$$P(A \rightarrow \alpha) = \frac{f(A \rightarrow \alpha)+1}{\sum_1^N (f(A \rightarrow \beta))+N}$$

where A is any NT, α and β are arbitrary RHSs of CF rules, and N is the number of expansions of A (i.e. the number of phantom observations)

This looks a little ad hoc as presented in procedural form, but actually is equivalent to assuming a uniform 'least informative' prior on events (rules) and applying Bayes' Theorem. Note that it is not the case that all unseen events have the same probability in this scheme – rather the probability of an event depends on the frequency with which the context in which we could, in principle, see it itself occurs.

Exercises

Can you see why?
How does this alter the probability of a given unseen event?

Bayes' Theorem takes the following form:

$$P(H \mid D) = \frac{P(H)P(D \mid H)}{P(D)} \tag{1}$$

Here, the term $P(H)$ is the *prior probability* of the hypothesis, $P(D \mid H)$ is the *likelihood probability* of the hypothesis given the data, and $P(H \mid D)$ is the *posterior probability*. When estimating parameters in a Bayesian framework, we try to find some hypothesis that maximises the posterior probability. If we assume the prior is a uniform distribution on rules (hypotheses, H), then

we minimally augment MLE by ensuring that every parameter (event) receives some probability. In the limit, the posteriori probability will be dominated by the likelihood probability, computed by MLE, no matter how the prior is set, but for most practical applications and training datasets, the prior will be significant because of data sparsity.

The Add1 approach to smoothing has severe limitations; for example, given Zipfian distributions with long tails of infrequent events, it tends to assign far too much of the probability mass to the unseen events. Thus, there are variants of Add1 which involve adding 1/2 or 1/4 or whatever (see M&S, ch6) – I'll refer to all such schemes that add a uniform low probability to unseen events generically as 'AddN'. AddN is better than plain MLE and simple to compute. We will return to the issue of smoothing again in section 5.7

5.3 Parse Disambiguation Using SCFG

In speech recognition, language models are typically used to predict the most likely next word form in the input. That is why Ngrams are popular because they emphasise the word sequence, and neither the shape nor ranking of the derivations (i.e. state sequences) is of direct interest. In NLP applications, the rankings and shapes of the derivations are critical. In this section, we consider how good SCFGs are for probabilistically distinguishing distinct derivations with distinct associated compositional semantics, and how good they are at assigning a plausible probabilistic ranking to sentences / derivations.

Grammar A below is an example of a stochastic CFG, in which each production is associated with a probability and the probabilities of all rules expanding a given non-terminal category sum to one.

Grammar A

```
1)   S' --> S       (1.0)
2)   S --> NP VP    (1.0)
3)   VP --> Vt NP   (.4)
4)   VP --> Vi      (.6)
5)   NP --> ProNP   (.4)
6)   NP --> Det N   (.3)
7)   NP --> NP PP   (.3)
8)   N --> N N      (.3)
9)   PP --> P NP    (1.0)
10)  N --> N0       (.7)
```

The probability of a particular parse is the product of the probabilities of each rule used in the derivation, regardless of the order of application of the rules. Thus the probability of parses a) and b) in Figure 4 is 0.0336 in both cases (though pronouns are far more frequent as subjects than objects). The probability of parse c) or d) must also be identical (0.09), because the same rule is applied twice in each case. Similarly, the probability of d) and e) is also identical

(0.09) because the same two rules are each used twice. However, these derivations are natural treatments of noun compounding and prepositional phrase (PP) attachment in English, and the different derivations correlate with different interpretations. For example, d) would be an appropriate analysis for *toy coffee grinder*, whilst c) would be appropriate for *cat food tin*, and each of e) and f) yields one of the two possible interpretations of *the man in the park with the telescope*. We want to keep these structural configurations probabilistically distinct in case there are structurally conditioned differences in their frequency of occurrence; as would be predicted, for example, by the theory of parsing strategies (see J&M:ch14.10) which would claim that, eg. PPs are attached low, by default. These considerations suggest that we need a technique which allows use of both a more adequate grammatical formalism than CFG and a more context dependent probabilistic model.

If we add rules like VP $\rightarrow$ VP PP and VP $\rightarrow$ V NP PP to our grammar, we create another problem. Now parsing a sequence like *hit the man with the umbrella*, the SCFG will always prefer the instrumental argument reading for the PP as this derivation covers this sequence with one rule, as opposed to either the adjectival or adverbial PP readings which each require derivations involving an extra rule. We can avoid the bias in favour of shortest derivations by taking the geometric mean rather than product of the probabilities in the derivation (but this is unmotivated by probability theory) or by 'padding' shorter derivations with the expected probability of a rule to normalise them with respect to the longest derivation in a competing set for a given sentence. If the longest derivation for a given sentence contains, say, n rules, we pad the shorter derivations of $n - m$ rules, with $n - m$ mean probabilities for the 'missing' rules in these derivations to normalise their length, and then rank in terms of the products of these normalised derivations. (Unfortunately, this may not work well because of the underlying 'Zipfian' distribution of rules, which means that there will be a great deal of variance around the mean.)

5.4 Stochastic 'Expanded Daughter' CFG

One way to improve on the 'standard' SCFG model is to take some account of the context of a rule expansion. The simplest and most obvious way to do this is to condition rule probabilities on the 'ancestor' rule in the derivation (i.e. the rule whose daughter categories are being expanded:

$$p(B \rightarrow \alpha \mid B, A \rightarrow \ldots B \ldots)$$

This can be distinguished from standard SCFG where, the probability of a rule is conditioned only by the probability of the mother category. Using a scheme like this we can, e.g., distinguish expansions of NP $\rightarrow$ Npro in subject and object position, because the 'ancestor' rules will be S $\rightarrow$ NP VP and VP $\rightarrow$ V NP ..., respectively. In fact, given the possibility of identical RHS categories (eg. VP $\rightarrow$ NP NP) we should, for completeness, condition rules on the specific RHS category being expanded by, e.g., by a number encoding position

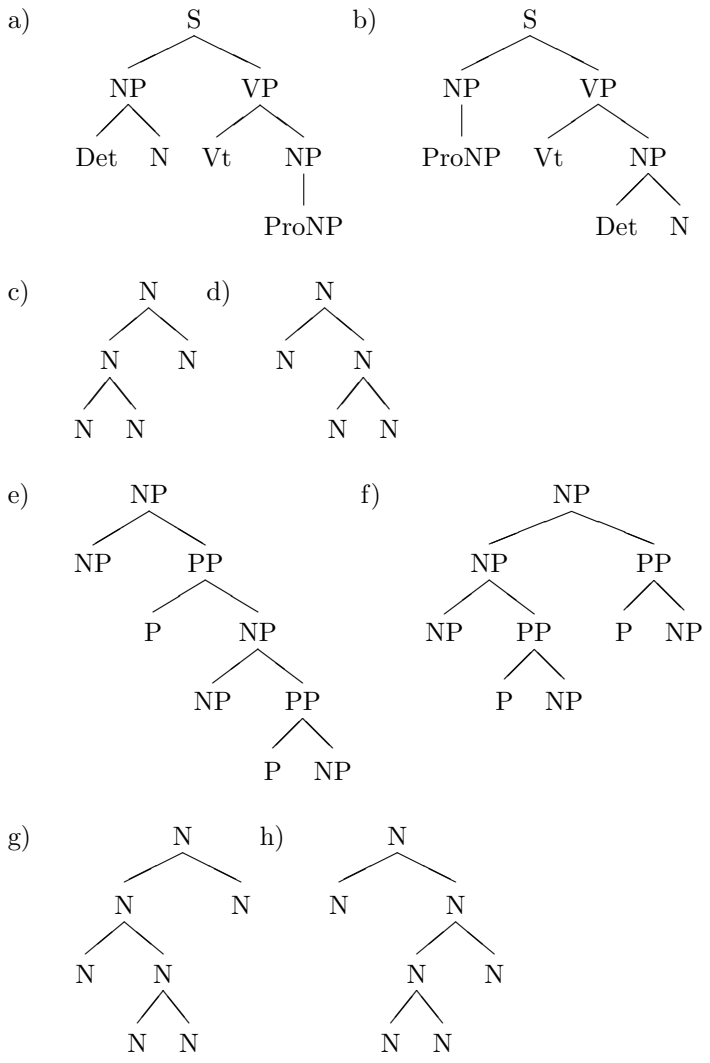


Figure 4: CFG Derivations

on the RHS of the ancestor rule:

$P(B \rightarrow \alpha \mid B, A \rightarrow \ldots B \ldots, RHSNo)$

Manning and Carpenter ('Left-corner probabilistic chart parsing' IWPT, 1997) demonstrate how a model of this type can be efficiently integrated with a left-corner chart parser. I will concentrate instead on considering how successfully such a model distinguishes derivations. Although the model is an improvement on standard SCFG, it is easy enough to provide examples whose derivations will be indistinguishable. E.g. 3 word noun compounds, as in Figure 4 c) and d), will be distinguishable, because left- vs right- branching structure will correlate with the second application of N $\rightarrow$ N N being an expansion of the first or second daughter in the first application, i.e., the probabilities being compared will be:

$p(N \rightarrow N\ N \mid N, N \rightarrow N\ N, 2)$ vs.
$p(N \rightarrow N\ N \mid N, N \rightarrow N\ N, 1)$

However, 4 word compounds with structures like g) or h) in Figure 4 will not be distinguishable because they will both have a probability:

$p(N \rightarrow N\ N \mid ?, ?, ?) \times$
$p(N \rightarrow N\ N \mid N, N \rightarrow N\ N, 2) \times$
$p(N \rightarrow N\ N \mid N, N \rightarrow N\ N, 1)$

The order in which the first and second N daughters are expanded will not affect the probability of the derivation.

Exercise

Similarly, for PP attachment ambiguities with three PPs, it is easy to construct distinct derivations in SCFG Grammar A (see § 5.3) which will be indistinguishable given this probabilistic model – can you find a case?

It is an empirical question how well such a model would do on a realistic disambiguation task – as far as I know, no such model has been tested properly to date. (Though closely related models, such as the Generalized LR probabilistic parser (RASP), perform better than SCFGs). The relationship between SCFG and SEDCFG is analogous to that between a 1st and 2nd order HMM. We could continue to condition on more ancestor rules in the derivation at the cost of increasing the number of parameters in the model. However, this will not resolve the fundamental problem with all such 'structural' or 'unlexicalised' models – they are insensitive to the words in the sentence.

5.5 Evaluation

The advent of robust and relatively accurate statistical parsing, created a need for evaluation measures. (Previously parsers were primarily evaluated in terms of their grammatical coverage and not the proportion of sentences from a dataset

parsed and/or the number of analyses returned per sentence.)

The approach which became the de facto standard is called PARSEVAL and was designed to abstract away to some extent from system differences to create a standard (see J&M:ch13, M&S:432f). It involves 3 measures of the fit between a gold standard unlabelled bracketing and one returned by a parser: precision, recall and crossing rate. Precision is the ratio (converted to a percentage) of correctly located brackets in the system analysis to all brackets in the system analysis. Recall is the ratio (converted to a percentage) of correctly located brackets in the system analysis to all brackets in the gold standard. Crossing rate is the number of system bracketings which 'cross' or conflict with rather than supplement those in the gold standard; e.g. (the (very old) man) does not cross (the very old man) but does cross (the very (old man)). The mean of these figures is calculated for all sentences of the test set. Most researchers working with the WSJ treebank also report labelled recall and precision rates, where in addition each bracket must have the same label as its counterpart in the gold standard. This latter move makes the task harder but also more treebank specific. The state-of-the-art is labelled precision and recall rates of around 92% with a mean crossing rate of less than 1 bracket / sentence for the WSJ test set (section 23). Usually, an overall F_1 score is reported which is the harmonic mean of labelled precision and recall.

Recently, an alternative or supplementary evaluation in terms of the labelled lexical dependencies or grammatical relations between pairs of words has become more popular; e.g. *The girl who saw Kim waved* (subj saw who), (obj saw Kim), (subj waved girl) etc. The advantage of this approach is that it measures more directly the effectiveness of parsing for recovery of semantic information rather than more arbitrary tree topology. The best reported rate training and testing on WSJ involve F_1 scores of between 88% and 92%, but there is less agreement on the precise set of relations to measure, so less easy comparison of systems, as yet. Depending on the scheme chosen, dependency-based evaluation is maybe 5% harder than PARSEVAL-style evaluation on WSJ trees (as the latter do not contain all the information needed to extract lexical dependencies e.g. some argument-adjunct distinctions, unbounded dependencies, etc.) The biggest distinction between the two dominant GR / dependency evaluation schemes in use is whether they encode 'understood' relations where dependents can have more than one head (as directed (acyclic) graphs, Stanford Dependencies, Grammatical Relations) or whether they stick to one head per dependent (the CoNLL03 sharted task scheme).

Exercise

Define precision, recall in terms of false / true positives and false / true negatives and F_1 in terms of precision and recall (see e.g. M&S, 268-9 or J&M:ch13)

5.6 Stochastic Lexicalised Grammars

A number of people have pointed out the lack of lexical information in models like SCFG and SEDCFG. There isn't a very obvious way of lexicalising CFGs unlike HMMs. One way that one might try to take account of the words would be by replacing the terminal categories in CF rules with preterminals (i.e. words or lemmatised words); for example. the rule VP $\rightarrow$ V NP might be replaced by a set of rules VP $\rightarrow$ eat| love| ... NP, but this won't help much because if we are interested in modelling likely direct objects of these specific transitive verbs (e.g. *eat sweets*, *love Madonna* etc) then the probabilities of NP $\rightarrow$ sweets| Madonna|... will be quite independent of the probabilities of the VP rules, given a standard SCFG. This won't be true of the higher-order model sketched above in which the NP rule probabilities will be conditioned on the VP rules. However, it is trivial to show that many lexical co-occurrences of this form extend across even larger stretches of a CF derivation; for example, in *children eat lots of sweets / broccoli* there are dependencies between *eat* and *sweets / broccoli* and between *children* and *eat sweets / broccoli*.

A variety of techniques have been proposed which involve passing a subset of word forms or lemmas up through the entire derivation tree and conditioning rule probabilities on the presence of specific word forms. For example, passing up the head lemma from the head daughter to mother in each rule would provide one of many such possible schemes. (A good way to implement this in a UB PSG would be to define a feature LEX=lemma which was bound between the head daughter and mother of each rule and to condition the probability of each rule on the value of this feature in addition to the mother category (SLCFG) or mother and 'ancestor' rule (SLEDCFG).) Suppose we are computing the probability of *John hit the man with the umbrella*, this would reduce for the SLCFG case to considering the probabilities of the following event sequences which distinguish the adjectival from adverbial PP reading:

$p(NP \rightarrow NP\ PP \mid NP, man)$ vs.
$p(VP \rightarrow VP\ PP \mid VP, hit)$

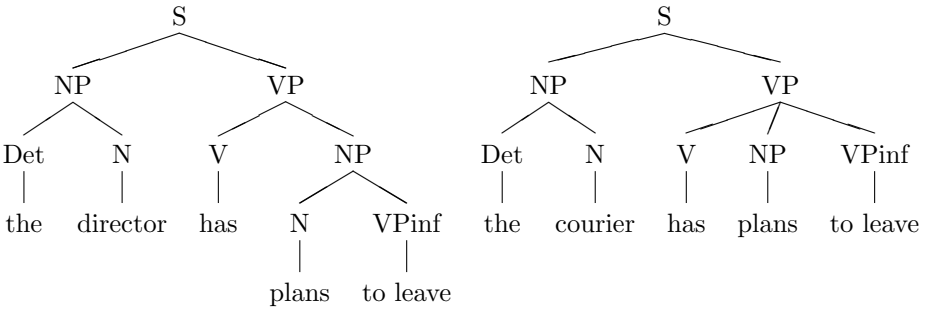
However, most researchers have argued that to effectively resolve PP attachments, it is necessary to consider the head lemmas of each daughter category in the rule:

$p(NP \rightarrow NP\ PP \mid NP, man, with)$ vs.
$p(VP \rightarrow VP\ PP \mid VP, hit, with)$

and even the head lemma of the PP complement

$p(NP \rightarrow NP\ PP \mid NP, man, with, umbrella)$ vs.
$p(VP \rightarrow VP\ PP \mid VP, hit, with, umbrella)$

The pattern of propagation of word forms, lemmas or a LEX feature now doesn't really look anything like that relating to syntactic heads. A possible method



of implementing this approach generally (short of propagating every word form 'upwards' through the derivation would be to propagate a predefined proper subset of lexical categories – in this case everything except Det.

The class of models considered above is somewhat ad hoc because it doesn't seem to be based on any well-defined linguistic concept or representation. In fact, careful consideration of the lexicalisation issue suggests that the appropriate way to distinguish derivations is to distinguish logical forms (after all what we are trying to do is recover the correct logical form!). For example, only a few lexicalised models proposed so far in the literature are able to distinguish the two readings of *Bill has plans to leave* shown informally in logical form / pred-arg notation below:

exists-more-than-one(x) have(bill, x), leave(bill, x), plan(x)
exists(x), have(bill, x), plan(x, leave(bill))

What is crucial about this example is that in one case *leave* and *plan* denote one-place predicates and in the other two-place ones, and in both cases *Bill* is the understood subject, but these differences don't show up in (typical) syntactic derivations: The semantic nature of the syntactic subject and its semantic 'fit' with each predicate-argument structure is what distinguishes the readings, compare: *The motorcycle courier had plans to leave* and *The company director had plans to leave*. A workable scheme of this kind would need to define the probability of a logical form in terms of the probability of constituent predicate-argument cooccurrences, such as leave(bill, ?), leave(?, plan) etc above. One reason why such models have not been more widely explored is because very few researchers have access to wide-coverage parsing systems capable of returning logical forms and grammars / parsers of this kind cannot be induced or trained from existing large treebanks (though PropBank has changed this to some extent – see work in ACL archive on 'semantic parsing').

5.7 'Back-off' Smoothing

All the models discussed above, with the possible exception of unlexicalised SCFG, will be faced with a significant data sparseness problem when estimat-

ing parameters. This is one reason there are many more probabilistic models definable than properly implemented and tested. To effectively estimate parameters for such models will need far more annotated data and more sophisticated approaches to smoothing.

One very general but fairly simple technique which has been used with some success is the technique of *back-off* estimation from a more context-sensitive model to a less context-sensitive one when a parameter of the more sophisticated model cannot be directly estimated using MLE. Backing-off is closely related to linear interpolation of models: e.g. if we want smooth estimates of trigrams based on bigrams and unigrams in Ngram language modelling, we estimate each trigram as:

$P(x_n \mid x_{n-1}, x_{n-2}) = \lambda_1 P_1(x_n) + \lambda_2 P_2(x_n \mid x_{n-1}) + \lambda_3 P_3((x_n \mid x_{n-1}, x_{n-2})$
where $0 < \lambda_i < 1$ and $\sum_i = 1$

The conditions on the weights combining estimates from different models (λ) ensure that the resulting estimates form a new probability distribution. The weights themselves can be estimated from the data using so-called deleted interpolation (see M&S:218f; J&M:ch5). This is a form of cross validation in which the data is divided into sections and firstly the parameters of each individual model (trigram, bigram, unigram) are estimated using MLE and then the weights are estimated from a held out section of the data. (Estimating the weights in such a way as to maximise the probability of the (held out) data requires an estimation maximisation (EM) algorithm (in this case the forward-backward algorithm) because which model generated a given piece of data is 'hidden'. To get good results the weights need to be conditioned on the specific parameters of the individual models, but to keep the number of weights tractable and not create new sparse data problems, equivalence classes of similar parameters (e.g. parameters of similar frequency) are formed.

The general technique can be used whenever, the events being conditioned on in the back-off model are a subset of those from the most sophisticated one, and the scheme can be applied recursively to back off as many times as needed. The general form of such a model is to think of the weights as functions of the possible parameters or history, h, forming the final estimate:

$P(x \mid h) = \sum_i^k \lambda_i(h) P_i(x \mid h)$

Intuitively, it is clear that if we have a good trigram estimate this should have a high weight or perhaps even be used exclusively in order to produce a useful model.

In this approach, models are consulted in order of specificity and backing-off only takes place when no reliable estimate is available at the current 'level'. For example, the estimate for an Ngram allowed to (recursively) back off through progressively shorter histories is:

$P(x_n \mid x_1 \ldots x_{n-1}) =$
$(1 - d_{x_1 \ldots x_n}) \frac{f(x_1 \ldots x_n)}{f(x_1 \ldots x_{n-1})}$ if $f(x_1 \ldots x_n) > k$
$\alpha_{x_1 \ldots x_{n-1}} P(x_n \mid \ldots x_{n-1})$ otherwise

where k is usually 0 or 1 and the discount function, d, represents the amount of the probability mass reserved for backed-off estimates, and α is a normalising coefficient which ensures this reserved mass is distributed among Ngrams estimated by backing off. This is a special case of linear interpolation where the weight functions over histories, $\lambda_i(h)$ take on values of 0 or 1.

This estimation technique could be used, for instance, to back-off from a SEDCFG model to a SCFG one when estimates of rules in specific contexts are unavailable in the training data:

SEDCFG(R) = $(B \rightarrow \beta \mid B, A \rightarrow \ldots B \ldots, RHSNo)$
SCFG(R) = $(B \rightarrow \beta \mid B)$
(where R = $B \rightarrow \beta$, etc)

$p(SEDCFG(R)) = 1 - d_{SEDCFG(R)} \frac{f(SEDCFG(R))}{f(SCFG(R))}$
where f(SEDCFG(R) ≥ 1, otherwise
$\alpha_{SCFG(R)} p(SCFG(R))$

This approach can be generalised to three levels or more, if rules are also conditioned on lexical items, and can be combined with AddN smoothing to deal with entirely unseen rules, etc.

There are a variety of methods for deriving the discount function d which defines how much of the probability mass is discounted from the observed events and redistributed amongst the unobserved ones. Absolute discounting is the simplest useful method in which all non-zero counts are discounted by a small constant, δ, and the total reserved frequency (TRF) redistributed over the unseen events by setting α to $\frac{TRF}{N_1}$ (N_1, the number of events seen just once) and dividing this between the unseen events N_0. To redistribute the reserved frequency non-uniformly, α can be multiplied by the probability of the backed-off estimate for each unseen event. Note that the overall probability mass assigned to all events will now sum to less than one, so strictly speaking renormalisation should be done when this process is completed. However, this should not affect the probabilistic ranking of events and composite events.

M&S (1999, ch6) discuss how the back-off scheme relates to other related techniques, such as deleted interpolation and linear interpolation of models, and discuss other approaches to discounting. See also the discussion of Collins' parser in J&M:ch14.

5.8 Stochastic Unification/Constraint-based Grammars

As with lexicalised grammars a number of researchers have defined probabilistic variants of UB PSGs. To be a genuine stochastic grammar, derivations must

be conditioned on the possible values of every feature which is defined in the grammar. This greatly increases the parameter space (though probably not as much as lexicalisation does). Therefore, it is not very surprising that there have been few attempts to test SUBGs by estimating their parameters from data.

Abney ('Stochastic attribute-value grammars' *Computational Linguistics* 23.4, 1997) is an excellent discussion of the problems of defining a stochastic version of a UB PSG as well as offering a very clear account of the rationale of MLE estimation for SCFGs. The problem of assigning probabilities to CF backbone rules in a UB PSG can be seen by imagining that the SCFG Grammar A of section 5.3 is the backbone and has been augmented with features for PER/NUM agreement. Suppose that we have an annotated corpus of sentences with correct derivations assigned according to Grammar A and we use MLE to assign probabilities to the CF backbone. The values of the PER/NUM features will be irrelevant so we will simply count how many times rules like S $\rightarrow$ NP VP have applied to expand S in this training corpus, and so forth. In each of the training derivations the values of these features will be consistent since otherwise the derivation would have failed. However, the resulting probabilised grammar cannot be a language model because it will not assign a probability of 1 to the language that it generates (i.e. to all possible derivations it licenses). To see this consider the probability of a sentence which violates PER/NUM agreement. It should be zero because the sentence is not generated by the grammar, but it won't be because it *will* be generated by the CF backbone (and thus its probability will simply be the sum of the products of the rules in each possible derivation). Therefore, the model is assigning a significant proportion of the probability mass to strings which it classifies as ungrammatical. It follows that it is assigning a probability of less than one to those which are grammatical.

One response to this problem is that it is irrelevant if the task is to rank derivations. The effect of unification failure during parsing of a sentence which violates PER/NUM agreement will be to remove that derivation from consideration. The remaining complete derivations can now be ranked according to the partial SCFG model of the UB PSG. Abney shows that this may not work. Not only will MLE applied to such a partial model result in a 'deficient' language model (i.e. one where the probability mass assigned to the language is less than one), it will also not be MLE and may not rank the derivations in a manner consistent with the relative frequency of constructions in the training corpus. It is not clear that Abney's theoretical examples are a significant factor in practice – as I pointed out in the last section, partial derivation-based models have outperformed SCFGs. Nevertheless, they are a cause for concern if the goal is to achieve good performance based on a good understanding of why we have achieved it.

Abney's proposed solution is to produce a genuine language model in three stages. In the first stage, a partial and deficient model is produced by using MLE to assign probabilities to the CF backbone as above. In the second, this grammar is used to generate a new representative sample of the language. Abney presents an argument, based on sampling theory, that if the CF backbone

probabilities are used to randomly sample the language, the relative frequency of constructions in the new sample should approximate their true frequency, partly because unification failure will prevent generation of sentences outside the language. Thirdly, a correct probabilistic language model is constructed from the new sample by finding the optimal model for distinguishing the derivations in the new sample which assigns maximum likelihood to the new training corpus. For the third stage, Abney proposes a maximum entropy (see next term) approach which automatically finds the best model with the fewest parameters which effectively discriminate the sample derivations. If this latter step models derivations to the level of feature values it will be computationally expensive and it is not clear that it would be feasible in practice.

In conclusion, then, stochastic UB PSGs remain more of a (well-defined) theoretical possibility than a practical reality. Recent statistical parsing work has ignored such features, tried to cpature their value indirectly through lexicalisation, or handled them in a relatively ad hoc way (see the C&C parser, next handout).

5.9 Supertagging

One practical approach to stochastic disambiguation with lexicalist grammars is supertagging – tagging with a superset of PoS tags. In lexicalist grammar, words are associated with lexical categories which encode syntactic valency / subcategorisation. We can treat abbreviations of these as 'supertags' Srinivas *Nat Lg Eng.* 6.2, 2000) created a large corpus of sentences annotated with the correct derivation by manually selecting the right analysis from the set returned by a parser for a small amount of data and by automatically transforming the WSJ treebank for the rest. He used a mixture of simple structural rules and a trigram / 2nd order HMM model over supertags to remove contextually inappropriate candidate supertags assigned by the lexicon. The trigram model uses both supertag transition probabilities, $P(T_3 \mid T_2, T_1)$ and output $P(W_j \mid T_i)$ (See section 4.3 for more details of such models.)

The accuracy of the supertagger itself is better than 92% word/supertag assignment rising to better than 97% if probabilistic thresholding is used and an average of 3 supertags per word are accepted. However, the main novelty of the approach is that, given the disambiguated supertags, a full or partial parse can be reconstructed fast because there is very little residual ambiguity left after supertag disambiguation wrt the full grammar, though not all the supertag sequences are necessarily within coverage of the full grammar, so in many cases only a partial parse can be returned. The figures for the recovery of lexical dependencies or grammatical relations using this approach exceed 83% for precision and recall.

This was the first work which developed a system capable of utilising a full grammar, of recovering analyses fast, and of gracefully degrading to recovering partial analyses where the input is extragrammatical. It also contrasts with the approach described in section 4.3 in that lexical subcategorisation information is

utilised from the start of the parsing process but is deployed in a soft statistical fashion allowing more graceful degradation where it is inaccurate.

5.9.1 A Simplified Example

To give a flavour of this approach. Here is a simple example utilising a UB PSG like Grammar 5: *John saw Sandy*.

```
John (PN-0.9 | Adj-0.1) saw (N-0.4 | Vtrans-0.6) Mary (PN-1)

                          ---------------------------------- HC1
                          VP
--------------------------- HS
S
```

Assuming the above assignments of supertag categories to the words, the supertagger should be able to associate probabilities with the lexical assignments so that the unlikely interpretation where a *John saw* is a type of cutting instrument and *John saw Mary* is a proper name denoting a female person closely associated with such an instrument (analogous to *Typhoid Mary*) is ruled out. Next the two rule schemata (Head-Specifier rule and Head-Complement-1 rule) can be applied to the most probable asignments to construct the derivation.

5.10 Treebank Parsers

Collins, Charniak and others built robust statistical parsers which acquire both their linguistic and statistical knowledge from the WSJ Penn Treebank. These parsers make choices between constructing alternative subanalyses (which are represented as a very ambiguous and large CFG extracted from the treebank) in terms of conditional probability distributions over CF rules utilising many aspects of the parse context, including the words in the input (see section 5.4 etc. above). Models of this type containing between 100K-1M parameters have proved very accurate at selecting the right analysis for test data from the same WSJ treebank (around 88% F_1 score using PARSEVAL, see section 5.5 above). However, they are somewhat domain dependent since they utilise a lot of lexical information from WSJ, and they are limited to producing CF-like analyses. The former means that accuracy degrades by as much as 5% when they are deployed on test data not drawn from WSJ, the latter that some aspects of predicate-argument structure are not (standardly) recovered, (though various researchers have proposed second stage extensions of these parsers to recover the empty node structure which represents unbounded dependencies and control relations in the WSJ treebank).

(See J&M:ch13 or M&S 451f for further discussion of lexicalised treebank parsers of this type, in particular those of Collins and Charniak.)

5.11 Language vs. Parse Decision Models

So far, we have mostly considered stochastic models which are clearly generative language models, in the sense that they define a probability distribution over grammatical sentences. However, since we are interested in ranking derivations/trees it is also possible to define probabilistic models which differentiate derivations but which do not assign a probability distribution to possible sentences. For example, imagine a model in which parameters are defined by acquiring the minimal information needed to effectively discriminate the correct from competing derivations found in a training corpus in which the correct derivation or tree was manually annotated. In such an approach, the probability of a sentence like *John hit the man with an umbrella* is irrelevant, but the difference between the adjectival and adverbial PP reading can be defined solely in terms of the two competing rules / subtrees VP $\rightarrow$ VP PP vs. NP $\rightarrow$ NP PP and however much extra information is needed, according to the model, to reliably distinguish when an adjectival or adverbial reading is correct.

One popular way of achieving this is to use a maximum entropy or log-linear model in which we define a large set of boolean-valued features over trees, such as 'does the PoS tag on the head daughter of mother(x) equal(y)?' and combining them to define histories, H, to compute the probability of the next parser action, a (or subconstituent of a tree):

$$p(a \mid H) = \frac{1}{Z(H)} e^{\lambda_1(a,H)f_1(a,H)+\ldots+\lambda_m(a,H)f_m(a,H)}$$

where λ_i are weights that indicate the importance of each feature, f_i, to the value of the probability, and $Z(H)$ is the partition function or normalising term of the given history, H so probabilities over all a sum to 1. The advantage of such models is that more or less arbitrary features of the context (history) can be factored into a decision, the contexts can be 'smoothed' by including all subfeatures of a given boolean combination, and no assumption of feature independence is made. The disadvantage is that it is expensive to compute the weights (using an iterative EM-like algorithm), and often difficult to compute the partition functions.

Charniak (2000, A maximum entropy inspired parser' ACL) developed a parser which retains many of the advantages of the approach but avoids the inefficiency and complexity. He noted that an equivalent effect can be achieved by decomposing a conditional probability in the following manner:

$$p(a \mid b, c, d) = p(a \mid b) \frac{p(a \mid b,c)}{p(a \mid b)} \frac{p(a \mid b,c,d)}{p(a \mid b,c)}$$

sequentially conditioning on increasing portions of the context (history). He then relaxes the requirement to condition on every subcomponent of the context, using only the intuitively most relevant, and ignores the normalisation step on the basis that the estimated probabilities will not depart much from 1. Using this method, Charniak develops a treebank parser with a F_1 PARSEVAL score of 90.1%, which remains the highest score on section 23 of the WSJ using this evaluation and a single pass parser.

Collins and colleagues (e.g. 'Discriminative Reranking for Natural Language Parsing', *Computational Linguistics*, 31(1):25-69, 2005) have developed a two stage approach to parsing in which an initial set of n-ranked derivations is reranked to improve the baseline performance obtained with the initial pass. This turns the problem into one of (binary) classification of the n analyses into (in)correct. Many techniques developed in the field of machine learning can be applied to this task. The reranking function is based on a vector of binary-valued features (as above) which can be arbitrarily complex patterns over (sub)trees. The learning task is to learn vector weights which minimise the error rate on training data (e.g. find the maximum margin hyperplane as in SVMs). The main practical disadvantage of the approach is that it is very inefficient at run-time because it is necessary to enumerate the n-best analyses for quite high n (approx. 1K) in order to ensure the correct tree is present, and then to derive their vector descriptions by matching the usuallly very large number (approx. 500k) of features to them. Nevertheless, the best current score on the standard WSJ PARSEVAL evaluation of 91.3% F_1 score (McCloskey *et al.* 'Effective self-training' NAACL 2006) was achieved with a reranking approach. Taskar *et al.* (2004, 'Max-Margin Parsing' ACL) develop a system which applies the discriminative classification approach to parse forests using dynamic programming to improve the run-time efficiency of these techniques. However, this places limits on the features that can be used by the classifier.

6 Dependency Parsing

In recent years, deterministic linear-time dependency tree parsing has become the most popular practical parsing (family of) algorithm(s). Probably because there are dependency treebanks for a number of languages, fast approximate parsing has many applications (particularly with web-scale data), and the output representation is easy to understand and abstracts away from many of the intricacies of syntactic theories (which also makes it easier to create new treebanks). J&M:online:ch14 gives a fuller introduction and more references.

6.1 Greedy Transition-based Parsing

Basically, this is deterministic shift-reduce parsing (see section 2.4) with the reduce operation split into LEFTARC and RIGHTARC for left/right dependencies respectively (eg: *Kim kissed Sandy* LEFTARC Kim ← kissed; RIGHTARC kissed *rightarrow* Sandy). This modification is enough to ensure that the algorithm can be used to produce a dependency tree with one head per daughter.

Innovation has focussed on developing a statistical procedure to decide between competing reduce and shift-reduce actions depending on the parse context (ie. contents of the Stack and Input Buffer). There are numerous ways to do this by applying various supervised machine learning techniques to a configuration of the parser, defined as a stack and buffer state and a correct action derived

from the training data (the dependency treebank). For example, actions can be treated as classes in a multiclass classifier (such as an SVM or Perceptron, see work by Nivre, Zhang, etc in ACL Archive) or can be learnt as an optimal policy (Reinforcement or Imitation Learning, see work by eg Vlachos in ACL Archive).

Commonly used features are wordforms, lemmas, PoS, PoS 'bigrams' of head + dependent of current word and words in left context (stack) and right context (buffer), dependency relation type (subj, obj, modifier, etc) , direction (left/right), and distance between head and dependent. Often these are combined in a prespecified way so that at training time all instances of a feature type that eg combines head lemma with dependent PoS are extracted. Such dependency parsers can end up with very large number of feature instances (low millions trained on existing treebanks), which in practice limits the amount of context that can be exploited in choosing the best parse action.

Standard evaluation is in terms of labelled attachement score (LAS) and unlabelled attachment score (UAS) calculating precision, recall and F-measure overall or on a per label (dependency type) basis reporting macro-averaged or micro-averaged performance. What is the difference and why might it matter?

7 Neural Parsing

Most deep learning models, multilayer neural network models with non-linear connections, treat text as a sequential series of words. Such so-called end-to-end models have not achieved state-of-the-art performance on parsing problems (whether constituency or dependency). However, models that use word or more recently character embeddings and eschew feature engineering but retain a parsing (decoding) algorithm but make parsing decisions using a deep learning model appear to be both smaller and more efficient and increasingly more accurate than their feature engineered counterparts.

Most of the recent work has been done with stack-based shift-reduce or graph-based dependency parsers and Manning and colleagues have consistently managed to just outperform others in terms of parse accuracy, achieving over 95% UAS parsing accuracy on the English WSJ-derived dependency treebank this year (Dozat and Manning, Deep Biaffine Attention for Neural Dependency Parsing, ICLR2017) with a model that also performs overall very well across the full set of treebanks for typologically diverse languages.

8 Conclusions

Parsing with or without supplementary PoS tagging remains a very actively researched area of computational linguistics with increasingly practical applications. In the next few years we can expect to see incremental advances using better techniques derived from deep learning. However, a step change in per-

formance will require better treebanks (or grammars) and thus better representations of the task across a variety of languages.

Supplementary Reading

Jurafsky, D. and Martin, J. *Speech and Language Processing*, Prentice-Hall 2000 / 2008 / 2017. (References in text to 2nd edition)

A recent paper or two from the ACL archive or arXiv on consituency or dependency parsing – eg in 2017 Dozat and Manning and/or Liu and Zhang, in 2015 Durrett and Klein,...

Parsing / Formal Lg Theory:
Natural Language Processing in Lisp/Prolog, G. Gazdar and C. Mellish, Addison-Wesley, London (1989).
Natural Language Understanding, (Second Edition), J. Allen, Benjamin/Cummings (ed. Cummings) (UK distributors Addison-Wesley), (1995).
Artificial Intelligence: A Modern Approach, S. Russell and P. Norvig, Prentice-Hall, (1995) (paperback) (esp. chs. 7&9).
A First Course in Formal Language Theory, V. Rayward-Smith, Blackwell, 1983.

Statistical NLP
Foundations of Statistical Natural Language Processing, C. Manning and H. Schultze, MIT Press, 1999.
Beyond Grammar: An experience-based theory of language, Bod, R. CUP, 1998.
Statistical Language Learning, Charniak, E. MIT Press, 1993