Computational Syntax and Formal Semantics

2018, © 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). Please read assigned sections in advance of the sessions, attempt the exercises, and be prepared to ask and answer questions on the material covered.

Contents

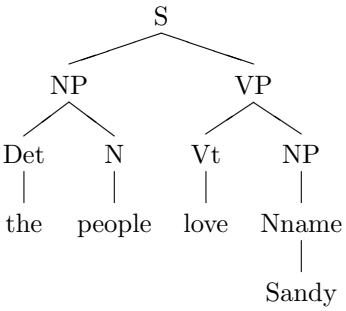
1 (Context-free) Phrase Structure Grammar

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:

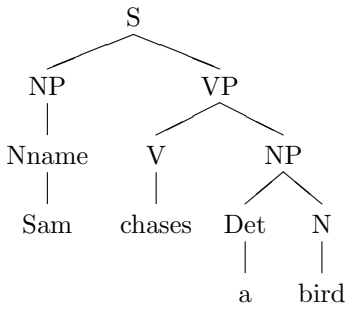
```
S => NP VP => Nname (Sam) VP => Nname (Sam) V (chases) NP
=> Nname (Sam) V (chases) Det (a) N (bird)
```

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.

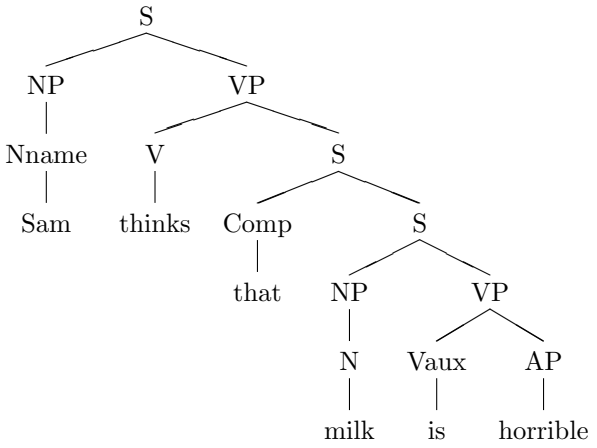
(1) a Sam thinks that milk is horrible
    b All linguists draw trees

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 → 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 → NP:[num=N, per=P] VP:[num=N, per=P]
VP:[num=N, per=P] → 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 3.8 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

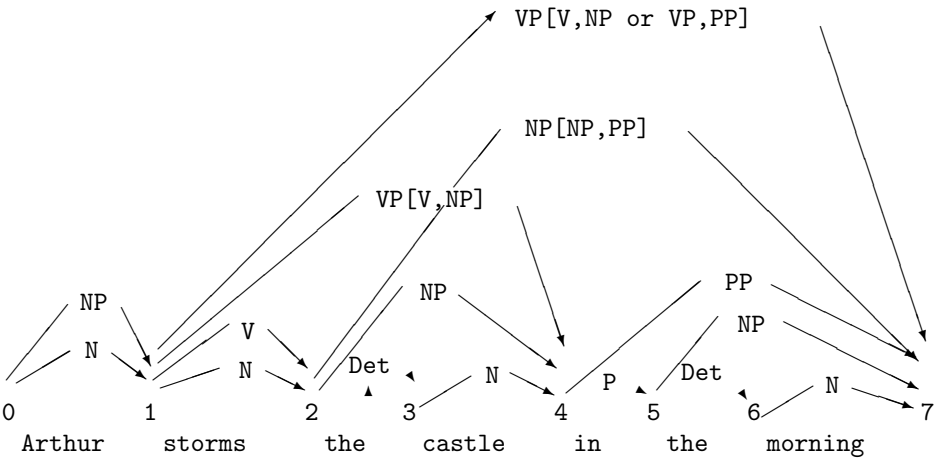


Figure 1: Chart Data Structure

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).

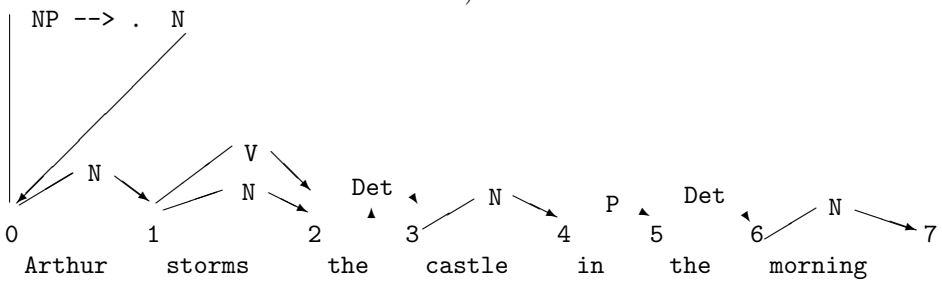
Exercise
Using this rule and the grammar above, see if you can simulate the behaviour of an active chart on a couple of the sentences we considered before. Assume that the fundamental rule is applied to every pair of adjacent active-inactive edge pairs and that active edges are added to the chart bottom-up 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).

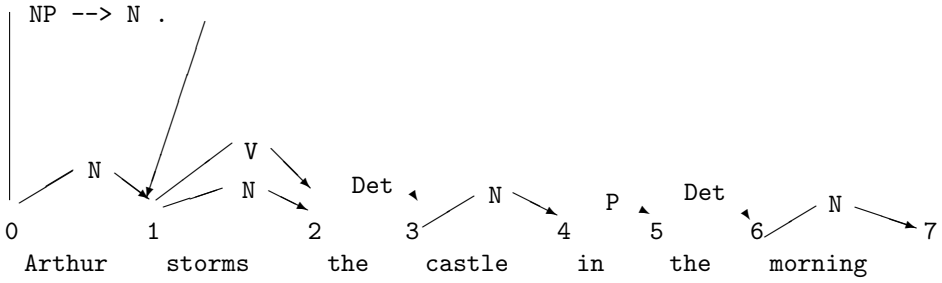
In describing charts we will use a pseudo-code notation where words in italics are variables. We use a Lisp/Prolog-style notation for lists, as follows:

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

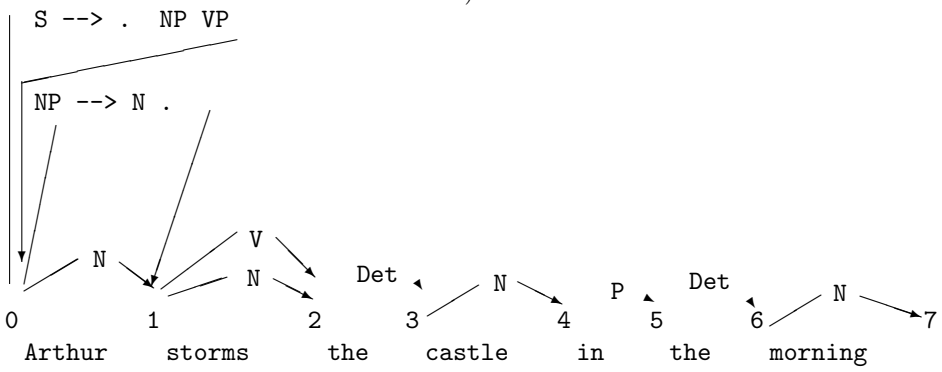
1)



2)



3)



4)

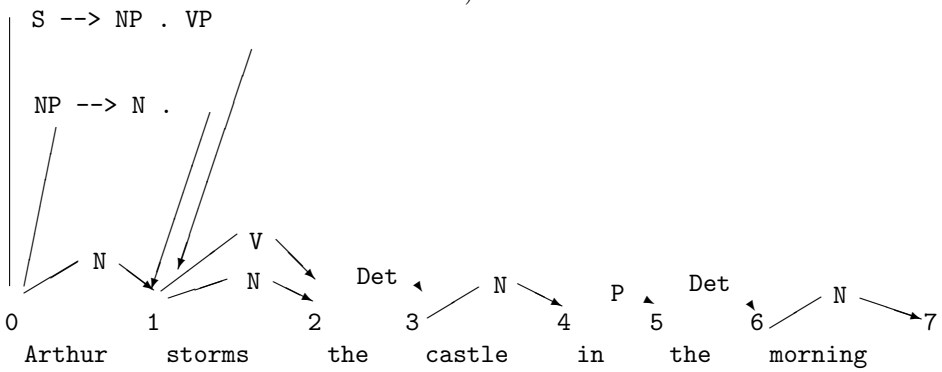


Figure 2: Active Chart Parsing

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 → 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 → 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 $a^n b^n c^n$:

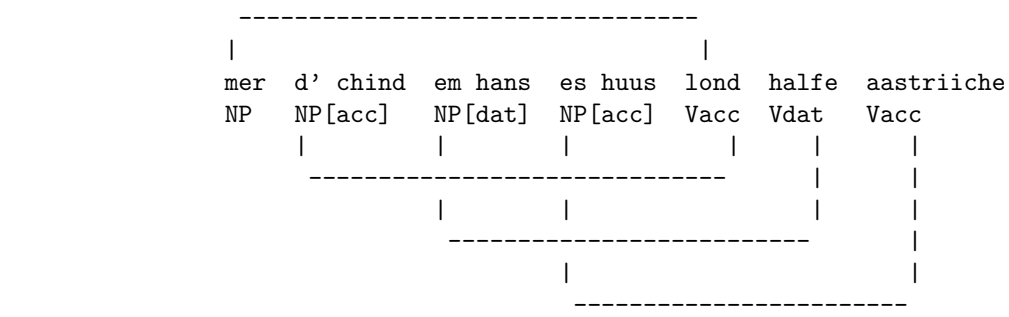
S:[s=H|T] → S1:[s=H|T] S2:[s=H|T] S3:[s=H|T]
S1:[s=H|T] → a S1:[s=T]
S1:[s=[]] → a

(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 Model-theoretic Semantics

The particular approach to truth-conditional semantics we will utilise is known as model-theoretic semantics because it represents the world as a mathematical abstraction made up of sets and relates linguistic expressions to this *model*. This is an external theory of meaning *par excellence* because every type of linguistic expression must pick out something in the model. For example, proper nouns refer to objects, so they will pick out entities in the model. (Proof theory is really derivative on model theory in that the ultimate justification of a *syntactic* manipulation of a formula is that it always yields a new formula true in such a model.)

4.1 An Example

Whilst Noam Chomsky's major achievement was to suggest that the syntax of natural languages could be treated analogously to the syntax of formal languages, so Richard Montague's contribution was to propose that not only the syntax but also the semantics of natural language could be treated in this way. In his article entitled 'English as a Formal Language', Montague made this very explicit, writing: 'I reject the contention that an important theoretical difference exists between formal and natural languages' (compare computational linguist Martin Kay's equally famous remark about that 'machine translation is nothing more than high-level compiling', though not so true for modern statistical, neural machine translation).

As a first introduction to an interpreted language, we will provide a syntax and semantics for an arithmetical language.

```
a) Exp --> Int
b) Exp --> Exp Op Exp
c) Stat --> Exp = Exp
d) Int(eger): 1,2,...9,...17...
```

```
e) Op(erator): +, -
```

Notice that this grammar generates a bracket-less language. We can provide a straightforward interpretation for this language by firstly defining the meaning of each symbol of the language and secondly stating how these basic 'meanings' combine in (syntactically permissable) expressions and statements. Lets assume that the interpretation of integers is as the familiar base ten number system, so that *7* is **7**, *19*, **19**, and so on. (Just to make clear the difference between the symbol and its interpretation we will use bold face for the interpretation of a symbol and italics for a symbol of some language.) The interpretation of the operators and equality sign is also the familiar one, but if we are going to characterise the meaning of expressions and statements in terms of these more basic meanings we will need to define them in a manner which makes the way they combine with integers and other expressions clear. We will define them as (mathematical) functions which each take two arguments and give back a value. By function, we mean a relation between two sets, the domain and range, where the domain is the set of possible arguments and the range the set of possible values. For some functions, it is possible to simply list the domain and range and show the mappings between them. We cannot characterise + properly in this fashion because its domain and range will be infinite (as the set of integers is infinite), but we can show a fragment of + as a table of this sort.

Domain	Range	Domain	Range
<0, 0>	0	<11, 1>	12
<0, 1>	1	<11, 2>	13
<1 ,0>	1	<11, 3>	14
<1, 1>	2	<11, 4>	15
<1, 2>	3	<11, 5>	16
...	...	...	...

The domain of + is a set of ordered pairs, written between angle brackets, the range the set of integers. Ordering the arguments for + is not very important, but it is for −. (You might like to construct a similar table for − to convince yourself of this point.) = is a rather different kind of function whose range is very small, consisting just of the set {F,T} which we will interpret as 'false' and 'true' respectively. The table for = would also be infinite, but we show a

	Domain	Range	Domain	Range
	<0, 0>	T	<1, 0>	F
	<1, 1>	T	<0, 1>	F
fragment:	<2, 2>	T	<1, 2>	F
	<3, 3>	T	<0, 2>	F
	<4, 4>	T	<2, 3>	F
	...	...	...	...

Functions like = which yield truth-values are sometimes called characteristic or Boolean functions (after the logician George Boole). There is a close relationship between the concept of a function and sets because we can always represent a function in terms of sets and mappings between them (although we cannot

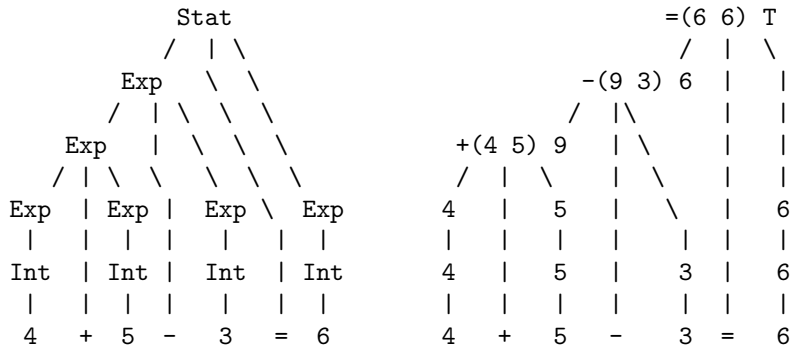
always exhaustively list the members of these sets).

Now that we have defined the meaning of all the symbols in our language, of its vocabulary, we can define how they combine semantically. We do this by adding a semantic component to each of the syntactic rules in the grammar. The result is shown below:

a) Exp $\rightarrow$ Int : Int$'$
b) Exp $\rightarrow$ Exp Op Exp : Op$'$(Exp$'_1$, Exp$'_2$)
c) Stat $\rightarrow$ Exp = Exp : $='$(Exp$'_1$, Exp$'_2$)

Each rule now has two parts delimited by a colon. The second is the semantic part. The primes are used to indicate 'the semantic value of' some category, so the category Int has values in 1,2,.. whilst Int$'$ has values in **1, 2,...**. The semantic operation associated with rules a) and b) is function-argument application, which is notated $F(A_1, \ldots A_n)$. The value returned by the function applied to the particular arguments which occur in some expression is the semantic value of that expression. Where the same category labels occur twice on the right hand side of some rule, we use subscripted numbers to pick them out uniquely, by linear order, for the semantic part of the rule.

Applying this interpretation of our language to some actual expressions and statements should make the mechanics of the system clearer. Below we show one of the syntactic structures assigned to $4 + 5 - 3 = 6$ and give the corresponding semantic interpretation where each symbol has been replaced by its interpretation and each node of the tree by the interpretations derived from applying the semantic rule associated with each syntactic rule to the semantic values associated with the daughter categories.



Rule a) just states that an integer can be an expression and that the semantic value of that expression is the semantic value of the integer. Accordingly, we have substituted the semantic values of the integers which occur in our example for the corresponding categories in the syntactic tree diagram. Rule b) is used in to form an expression from 4, + and 5. The associated semantic operation is

function-argument application, so we apply the semantic value of the operator + to the semantic value of the arguments, 4 and 5. The same syntactic rule is used again, so we perform another function-argument application using the result of the previous application as one of the arguments. Finally, c) is used, so we apply = to 6 and 6, yielding 'T' or 'true'. You might like to draw the other tree that can be assigned to this example according to the grammar and work through its semantic interpretation. Does it yield a true or false statement?

It may seem that we have introduced a large amount of machinery and associated notation to solve a very simple problem. Nevertheless, this apparently simple and familiar arithmetical language, for which we have now given a syntax and semantics, shares some similarities with natural language and serves well to illustrate the approach that we will take. Firstly, there are an infinite number of expressions and statements in this language, yet for each one our semantic rules provide an interpretation which can be built up unit-by-unit from the interpretation of each symbol, and each expression in turn. This interpretation proceeds hand-in-hand with the application of syntactic rules, because each syntactic rule is paired with a corresponding semantic operation. Therefore, it is guaranteed that every syntactically permissable expression and statement will receive an interpretation. Furthermore, our grammar consists of only three rules; yet this, together with a lexicon describing the interpretation of the basic symbols, is enough to describe completely this infinite language. This expressive power derives from recursion. Notice that the semantic rules 'inherit' this recursive property from their syntactic counterparts simply by virtue of being paired with them. Secondly, this language is highly ambiguous – consider the number of different interpretations for $4 + 5 - 2 + 3 - 1 = 6 - 4 + 9 - 6$ – but the grammar captures this ambiguity because for each distinct syntactic tree diagram which can be generated, the rules of semantic interpretation will yield a distinct analysis often with different final values.

4.2 Exercises

1) Can you think of any arithmetical expressions and statements which cannot be made given the grammar which do not require further symbols? How would you modify the grammar syntactically and semantically to accommodate them?

2) What is the relationship between brackets and tree structure? Can you describe informally a semantic interpretation scheme for a bracketed variant of arithmetical language in section 4.1 which does not require reference to tree diagrams or syntactic rules?

3) The interpretation we have provided for the bracket-less arithmetical language corresponds to one which we are all familiar with but is not the only possible one. Find an interpretation which makes the statements in a) true and those in b) false. Define a new grammar and lexicon which incorporates this interpretation.

```
a)                  b)
4 + 1 = 4             3 - 2 = 1
4 - 1 = 4             4 - 6 = 8
5 + 3 = 1             7 + 4 = 10
9 + 2 = 6             5 + 2 = 3
6 - 2 = 5             3 - 4 = 2
```

4) Provide a grammar for a language compatible with the examples illustrated below. Interpret the integers in the usual way, but choose an interpretation for the new symbols chosen from $+$, $-$, and $*$ (multiply). Give the interpretation that your grammar assigns to each example and demonstrate how it is obtained.

```
a) @ 2 3
b) #  2 @ 3 4
c) ^ 4 @ 2 #  6 8
d) #  @ 4 ^ 3 2 5
e) @ #  ^ 2 3 7 9
```

5 Denotation and Truth

Truth-conditional semantics attempts to capture the notion of meaning by specifying the way linguistic exressions are 'linked' to the world. For example, we argued that the semantic value of a sentence is (ultimately) a proposition which is true or false (of some state of affairs in some world). What then are the semantic values of other linguistic expressions, such as NPs, VPs, and so forth? If we are going to account for semantic productivity we must show how the semantic values of words are combined to produce phrases, which are in turn combined to produce propositions. It is not enough to just specify the semantic value of sentences.

One obvious place to start is with proper names, like *Kim* or *Sandy* because the meaning of a proper name seems to be intimately connected to the entity it picks out in the world (ie. the entity it refers to, eg. kim1, a particular unique entity named *Kim*). So now we have the semantic values of proper names and propositions but we still need to know the semantic values of verbs before we can construct the meaning of even the simplest propositions. So what is the 'link' between verbs and the world? Intransitive verbs combine with proper names to form propositions – intransitive verbs pick out properties of entities. But how can we describe a 'property' in terms of a semantic theory which attempts to reduce all meaning to the external, referential aspect of meaning? One answer is to say that the semantic value of an intransitive verb is the set of entities which have that property in a particular world. For example, the semantic value of *snore* might be {kim1, fido1}. Actually, we will say that a set like this is the

semantic value of a predicate like snore1, a particular sense of *snore*. Now we are in a position to say specify the meaning of (9) in a compositional fashion.

(9) Kim snores

First find the referent of *Kim* and then check to see whether that entity, say kim1, is in the set of entities denoted by snore1. Now we have specified the truth-conditions of the proposition conveyed by (9) (ignoring any possible ambiguities concerning the referent of *Kim* and the sense of *snore*).

Developing a truth-conditional semantics is a question of working out the appropriate 'links' between all the different types of linguistic expression and the world in such a way that they combine together to build propositions. To distinguish this extended notion of reference from its more general use, we call this relation **denotation**. Thus the denotation of an intransitive verb will be a set of entities and of a proper name, an entity.

At this point we should consider more carefully what sentences denote. So far we have assumed that the semantic value of a sentence is a proposition and that propositions are true or false. But what is the link with the world? How is this to be described in external, referential terms? One answer is to say that sentences denote their truth-value (ie. true or false) in a particular world, since this is the semantic value of a proposition. So we add the 'entities' true and false to the world and let sentences denote these 'entities'. However, there is an immediate problem with this idea – all true (false) sentences will mean the same thing, because truth-conditional semantics claims in effect that denotation exhausts the non-pragmatic aspects of meaning. This appears to be a problem because *Mr. Cameron is prime minister* and *Mr. Obama is president* are both (now) false but don't mean the same thing.

At this point you might feel that it is time to give up truth-conditional semantics, because we started out by saying that the whole idea was to explain the internal aspect of meaning in terms of the external, referential part. In fact things are not so bad because it is possible to deal with those aspects of meaning that cannot be reduced to reference in model-theoretic, truth-conditional semantics based on an intensional 'possible worlds' logic. The bad news is though that such logics use higher-order constructs in ways which are difficult to reduce to first-order terms for the purposes of automated theorem proving. More on this later. For the moment we will continue to develop a purely 'extensional' semantics to see how far we can get.

5.1 Propositional Logic

Consider a sentence such as (10) made up of two simple sentences conjoined by *and*.

(10) Kim snores and Sandy smiles

Lets assume that *Kim snores* and *Sandy smiles* convey true propositions – then what is the truth-value of the proposition conveyed by the complex sentence in (10)?

Propositional logic (PL) addresses the meaning of sentence connectives by ignoring the internal structure of sentences / propositions entirely. In PL we would represent (10) and many other English sentences as $p \wedge q$ where p and q stand in for arbitrary propositions. Now we can characterise the meaning of *and* in terms of a truth-table for $\wedge$ which exhausts the set of logical possibilities for the truth-values of the conjoined propositions (p, q) and specifies the truth of the complex proposition as a function of the truth-values for the simple propositions, as below:

p	q	p ∧ q
t	t	t
t	f	f
f	t	f
f	f	f

Can you write down the truth-tables for $\vee$, $\neg$ and $\Rightarrow$? (I use $\Rightarrow$ to denote logical implication.)

The semantics of PL must specify the truth-conditions for the truth or falsity of well-formed formulas in terms of the truth or falsity of the simple unanalysed propositions represented by sentential variables and the truth-conditions for each connective. We have already seen what the truth-conditions for *and* are; we gloss this as below, where α and β are metavariables standing for any sentential variable:
$\alpha \wedge \beta$ is true iff both α and β are true.
Notice that is a statement in the meta language we are using to describe the semantics of PL. We could state the semantics of each sentence connective in this way. This semantic model-theoretic interpretation licences entailments or rules of valid inference like $p, q \models p \wedge q$ – can you see why? The truth-table characterisations of the logical constants guarantee that such entailments must hold, given any valid model of a PL language.

Each connective combines with two propositions to form a new complex proposition. Therefore we can represent the general form of the semantics of a connective as a function which takes two arguments and yields a result. In the case of sentential connectives this function will be what is called a truth function because both arguments and result will be truth-values. The precise function will vary depending on which connective we are discussing. Similarly, the negation operator can be described as a truth function which takes one argument. Truth-tables define these functions. However, we still need some way of incorporating these semantic rules into a grammar. This will ensure that we can interpret any well-formed formula that the grammar generates. The following CF rules generate a bracketless variant of PL, in which Var represents any propositional variable (p,q,r) and Scon any 2-place connective ($\wedge, \vee, \Rightarrow$). and the semantic rules have the interpretation given in the previous section; that is, primes de-

note the 'semantic value of' and $[F, A_1, \ldots A_n]$ function-argument application:

1) S → Var : Var′
2) S → S Scon S : [Con′,S′,S′]
3) S → Neg S : [Neg′,S′]

We can read the semantic interpretation of a formula of bracketless PL off its structural description. To see how this works it is probably best to imagine that you are applying the semantic rules to the syntax tree working from the 'deepest' point in the tree up to its root node. The interpretation of the simple propositions depends on the truth-value that we (arbitrarily) assign them to get the process started. This process is analogous to choosing a model of the world in which to interpret the formula (ie. assuming some state of affairs). The interpretation of the connectives, however, remains the same in any model we care to imagine; for this reason the connectives are often called **logical constants**.

5.2 English Fragment 1

Our first semantically interpreted fragment of English (F1) is going to consist of the natural language 'equivalents' of the sentential connectives of PL and basic sentences constructed from proper names and transitive or intransitive verbs; for example (11) is one sentence in F1.

(11) Kim smiles and Sandy likes Fido

We will use the semantics of the connectives developed for PL and then consider how good this semantics is for the English words *and*, *or*, and so forth. We will develop a semantics of simple sentences along the lines sketched above.

5.2.1 Lexicon for F1

The lexicon stores syntactic and semantic information about each word (in that order). The semantic interpretation of a proper name is the particular entity it refers to. Semantic interpretations are written with numbers to indicate which sense of the predicate or referent in the model is intended – since we are not dealing with reference or word meaning at the moment this will always be 1.

```
Kim : Name : kim1
Sandy : Name : sandy1
Fido : Name : fido1
Felix : Name : felix1

and : Conj : and
or : Conj : or
it-is-not-the-case-that : Neg : not
```

```
snores : Vintrans : snore1
smiles : Vintrans : smile1
likes : Vtrans : like1
loves : Vtrans : love1
```

5.2.2 Grammar for F1

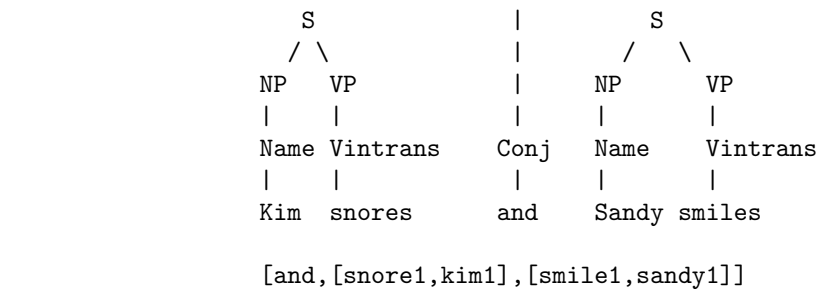
The syntactic rules are only marginally more complex than those for PL. The interpretation of the semantic part of the rules is identical to that described for PL:

1) S → NP VP : [VP′,NP′]
2) S → S Conj S : [Conj′,S′,S′]
3) S → Neg S : [Neg′,S′]
4) VP → Vtrans NP : [V′,NP′]
5) VP → Vintrans : V′
6) NP → Name : Name′

5.2.3 Some Examples

Unless we construct a model in which to interpret F1 we cannot illustrate the workings of the semantic rules directly in terms of truth-values. But we can also think of the rules as a specification of how to build up a 'logical form' for a sentence. The logical form of the sentences in F1 look rather like formulas in PL except that we are using prefix notation and using 'and', 'or' and 'not' instead of the less computer friendly ∧, ∨ and ¬. The example below shows how the logical form is built up in tandem with the construction of the syntax tree:

```
a)
     S                          S
    / \                        /   \
  NP   VP                    NP      VP
  |    |                     |       |
  Name Vintrans    Conj      Name    Vintrans
  |    |           |         |       |
  Kim  snores      and       Sandy smiles

  [snore1,kim1]              [smile1,sandy1]

b)
                     S
          /          |   \
```



One difference between the sentences of PL, the logical forms of F1 and F1 itself is the absence of brackets in F1. This means that many of the (infinite) grammatical sentences of F1 are ambiguous; for example, the sentence in (12a) has the two logical forms shown in b) and c).

(12) a It-is-not-the-case-that Kim snores and Sandy smiles
not,[and,[snore1,kim1 ,[smile1,sandy1]]]
and,[not,[snore1,kim1],[smile1,sandy1]]

The interpretation in b) means that neither Kim snores nor Sandy smiles, whilst that in c) means that Kim doesn't snore but Sandy does smile. The different interpretations are a direct consequence of the syntactic ambiguity and the corresponding different order in which the semantic rules are applied. (Incidentally *it-is-not-the-case-that* is not a word of English, but *not* is rather different from the sentential connective of PL – so I'm cheating for now)

5.3 A Model for F1

Model-theoretic, truth-conditional semantics interprets linguistic expressions with respect to some model. Thus the truth or falsity of a proposition is calculated with respect to a particular model and a truth-conditional semantics consists of a set of general procedures for calculating the truth or falsity of any proposition in any model (ie. the truth-conditions for the proposition).

A model is represented in terms of sets. It is defined as a domain of entities (abbreviated E) and a function (F) which supplies the denotations of the vocabulary of the language being interpreted in that model. Here is one model in which we can interpret sentences of F1:

```
E {kim1, sandy1, fido1, felix1}

F (snore1) {kim1, fido1}

F (smile1) {sandy1, felix1}
```

```
F (like1) {<kim1, sandy1> <sandy1, felix1>}

F (love1) {<fido1, sandy1> <fido1, kim1>}
```

In this model, Kim and Fido snore, Sandy and Felix smile, Kim likes Sandy and Sandy likes Felix, and Fido loves Sandy and Kim. Now we are in a position to interpret a sentence relative to this model:

```
[and,[snore1,kim1],[smile1,sandy1]]
t    t             t
```

I illustrate the interpretation by indicating the truth-values which result from each function-argument application under the logical form for the sentence. We can now see more clearly what it means to say that snore1 is a function which we apply to the argument kim1 (analogous to the description of *and* ($\wedge$) as a truth function above). snore1 is a function from entities to truth-values – a characteristic function. We can define it by exhaustively listing its range of inputs and corresponding outputs in this model:

```
kim1       -->   t
fido1      -->   t
sandy1   -->   f
felix1     -->   f
```

The set-theoretic operation used to compute this function is just to check whether the argument is a member of the denotation of the relevant intransitive verb and return T if it is and F if it isn't.

6 Entailment and Possible Models

We can define entailment, contradiction, synonymy, and so forth in terms of the notion of possible models for a language. Lets start with contradictory propositions. The proposition conveyed by (13) is a contradiction in F1, because it is impossible to construct a model for F1 which would make it true.

(13) a Kim snores and it-is-not-the-case-that Kim snores
and,[snore1,kim1 ,[not,[snore1,kim1]]]

The reason for this is that whatever model you construct for F1, *Kim snores* must either come out true or false. Now from the truth-tables for *it-is-not-the-case-that* and *and* the truth-value for (13) will always be false, because a proposition containing *and* is only ever true if both the coordinated simple

propositions are true. However, this can never be the case when we coordinate a proposition and its negation because of the truth-table for negation. In F1, the semantic 'translation' of the connectives, unlike say the denotation of *snores*, remains the same in every possible model. The opposite of a contradictory proposition is one which is logically true; that is, true in every possible model. An example in F1 is *Kim snores or it-is-not-the-case-that Kim snores*. See if you can explain why using the same kind of reasoning that I used to explain the contradictoriness of (13).

One proposition A entails another proposition B if and only if in every possible model in which A is true, B is true as well. For example, in F1 (14a) entails (14b) because it is impossible to make up a model for F1 in which you can make a) come out true and b) come out false (unless you change the semantic rules). Try it!

(14) a Kim snores and Sandy smiles
b Kim snores

Can you think of any other entailment relations in F1?

Finally, two propositions are synonymous, or more accurately logically equivalent, if they are both true in exactly the same set of models. For example (15a) and b) are logically equivalent in F1 because it is impossible to make up a model in which one is true and the other false (unless you change the semantic rules). Try it!

(15) a a) Kim snores and Sandy smiles
b b) Sandy smiles and Kim snores

Synonymy goes beyond logical equivalence (in PL and F1) because sometimes it involves equivalence of meaning between words or words and phrases. However, to account for this we need to say something more about word meaning.

6.1 Exercises

1) Using the grammar for PL above and choosing an assignment of t-values to propositional variables generate some formulas and calculate their t-values

2) Construct the syntax trees and associated logical forms for the following sentences of F1:

a) It-is-not-the-case-that Fido likes Felix
b) Felix loves Kim or Felix loves Sandy
c) Felix loves Kim or Kim loves Sandy and Felix loves Fido
d) Kim snores and Sandy loves Felix or it-is-not-the-case-that Felix smiles

3) Construct a different model in which to interpret the sentences of 2) illustrate how the interpretation proceeds by writing truth-values under the logical forms

you have associated with each sentence.

4) Can you think of any problems with using the PL semantics of sentential connectives to describe the semantics of *and* and *or* in English? Think about the following examples:

a) The lone ranger rode off into the sunset and he mounted his horse
b) Either we'll go to the pub or we'll go to the disco or we'll do both

5) What sort of a function will we need to associate with a transitive verb? Write down the function for love1 (as I did for snore1). What is the general set-theoretic operation which underlies this function? (See next section if in doubt.)

7 First Order Logic (FOL)

In this section, we will describe FOL, also often called (First-order) Predicate Logic / Calculus (FOPC) in more detail.

FOL extends PL by incorporating an analysis of propositions and of the existential and universal quantifiers. We have already informally introduced the analysis of one-place predicates like *snores* in the fragment F1. The semantics of FOL is just the semantics of F1 augmented with an account of the meaning of quantifiers, and free and bound variables. The interpretation of the connectives and of predicates and their arguments will remain the same, though we will now consider predicates with more than one argument.

7.1 FOL syntax

The FOL lexicon contains symbols of the following types:

Individual Constants such as a,b, kim1, cat10, etc, indicated by lower case early letters of the alphabet, and possibly a number

Individual Variables such as w,x,y,z, indicated by lowercase late usually single letters of the alphabet (or uppercase letters in Prolog-like implementations)

Predicates such as snore1, love1, *P,Q,R,S* indicated by usually non-singular letters possiby ending with a number, or a single italicised capital

Connectives $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, (**and, or, if, iff** in implementations)

Negation $\neg$, **not** in implementations

Quantifiers $\exists$, $\forall$ (**exists, forall** in implementations)

Functions such as F_1, F_2, mother1-of, etc, indicated by pred-of or F_n

Brackets ()

We will describe the syntax of FOL using a CFG to generate well-formed formulas:

1) S → Pred(Term)
2) S → Pred(Term,Term)
3) S → Pred(Term,Term,Term)
4) S → (S Conn S)
5) S → Quan Var S
6) S → Neg S
7) Term → Const
8) Term → Var
9) Term → Fun(Term)

(Note that we only allow 1/2/3-place predicates, although this could be generalised by replacing rules 1,2,and 3 with a rule schema using Kleene plus: S → Pred(Term$^+$).) Can you write a variant grammar replacing infix connectives with prefix connectives and using square brackets around quantifiers, the hat operator etc to generate Prolog-style list syntax (see egs. in (16e,g))? – the point is there can be more than one syntax for FOL.

7.2 FOL semantics

A model for FOL consists of entities, denoted by entity constants, and properties or relations, denoted by predicate or function symbols. One-place predicates denote properties and others denote relations. Functions denote relations which yield an entity rather than a truth-value. Terms denote entities. Sentences or well-formed formulas denote truth-values. Some examples of well-formed formulas and terms are given in (16). See if you can say which is which and also name the atomic components of each one. It might help to draw the derivations.

(16) a mother1(sandy1)
b (mother1(sandy1) ∧ snore1(kim1))
c mother1-of(kim1)
d $\exists\, x(P(x) \Rightarrow\ Q(x))$
e `[exists,x^[and,[snore1,x],[smile1,x]]]`
f ∀ x ∃ y (mother1(x) ⇒ equal(mother1-of(y),x))
g `[exists,y^[forall,x^[if[snore1,x],[love1,x,y]]]]`

To model relations set-theoretically we need to introduce a notation for ordered pairs, triples and ultimately n-ary relations. `<sandy1 kim1>` is an ordered pair which might be in the set denoted by love1. Notice that the two-place predicate love1 is not a symmetric relation so it does not follow (unfortunately) that `<kim1`

`sandy1>` will also be a member of this set. So a formula $P(t_1, t_2, t_n)$ will be true in a model M iff the valuation function, F(P) yields a set containing the ordered entities denoted by terms, t_1, t_2, t_n. Otherwise a model for FOL is identical to the model we developed for F1.

FOL allows us to make general statements about entities using quantifiers. The interpretation of the existential quantifier is that there must be at least one entity (in the model) which can be substituted for the bound variable 'x' in e.g. (16d) to produce a true proposition. The interpretation of the universal quantifier ($\forall$) is that every entity (in the model) can be substituted for the variable bound by the quantifier (eg. 'x' in (16f)) to yield a true proposition. A variable is free in a formula if it is not bound by a quantifier. Vacuous quantifiers can also be generated whose variables do not occur inside the formula within their scope. Can you derive formulas with free variables and vacuous quantifiers using the grammar above?

To interpret a formula containing a variable bound by a quantifier (in a model) we need a **value assignment function** which assigns values to variables in formulas. For FOL, we want a function which assigns an entity to variables over entities. Now we can talk about the truth of a proposition in a model given some value assignment to its variables. For example, if we assume the model below (where F (A) {m b} just specifies the members of the domain of A which yield a value of t(rue) so implicitly F(A(g)) $\rightarrow$ f):

```
E {m b g}    F (A) {m b}    F (B) {<b g>}
```

then the value assignment function will assign one of the entities from the model to a variable. Thus the set of possible assignments to 'x' in (17a) are shown in b,c,d).

(17) a B(x g)
b B(m g)
c B(b g)
d B(g g)

Of these, only c) will yield a true proposition in this model. When variables are bound by an existential or universal quantifier, this imposes extra conditions on the interpretation of the formula, which are given below:

Existential Quantifier: The formula $\exists\ \alpha$ is true iff for some value assignment to the variable bound by $\exists$ in α, the resulting variable free formula is true.

Universal Quantifier: The formula $\forall\ \alpha$ is true iff for every value assignment to the variable bound by $\forall$ in α, the resulting variable free formula is true.

Thus the process of computing the truth-value of a formula with a variable bound by an existential quantifier is a question of mechanically substituting

each entity (in the model) for the variable until you find one which makes the formula true, whilst in the case of one bound by a universal quantifier, every possible substitution must come out true. So the process of interpreting the following formula in the model given above, can be illustrated as below:

$\forall\ x\ B(x\ g)$

```
B(m g) --> f
B(b g) --> t   --> f
B(f g) --> f
```

A well-formed formula may contain more than one quantifier in which case the interpretation of the formula will vary depending on the relative scope of the quantifiers (just as we saw that the interpretation of sentences of F1 varied depending on the relative scope of *and*. For example, if we are interpreting the formula in (18a), then we must first choose a value assignment for the variable 'x' bound by the existential quantifier and calculate the truth of the universally quantified sub-formula with respect to this value assignment, because the universal quantifier binding 'y' is 'inside' (ie. in the scope of) the existential.

(18) a $\exists\ x \forall\ y\ B(x\ y)$
b $\forall\ y \exists\ x\ B(x\ y)$

On the other hand, in b), we must first choose an assignment for 'y' and then calculate the truth of the existentially quantified sub-formula with respect to each possible value assignment to 'y'. The interpretation of a) in the model above proceeds as illustrated below:

```
x/b               x/g               x/m

B(b b) --> f       B(g b) --> f       B(m b) --> f
B(b g) --> t       B(g g) --> f       B(m g) --> f
B(b m) --> f       B(g m) --> f       B(m m) --> f
```

We try to find one assignment to 'x' such that that one entity stands in the B relation to every entity in the model. None of the three possible assignments to 'x' yield a complete set of true propositions when we try all possible value assignments to 'y'; therefore, a) is false in the model above. However, when we compute b) we fix the assignment to the universally quantified variable first and then vary the assignment to the existentially quantified one:

```
y/b

B(b b) --> f
B(g b) --> f
B(m b) --> f
```

For a universally quantified formula to be true in a model it must be true for every possible value assignment. For an existentially quantified formula to be true in a model it must be true for one value assignment. We have assigned 'b' to 'y', but this fails to yield a true formula under any assignment of a value to 'x' so the formula must be false in this model. On the other hand if one of these formulas had been true, then we would need to continue and compute all the other possible assignments to 'y' until we found another value which did not yield a true formula under any assignment to 'x'. Thus the relative scope of the quantifiers affects which assignment is made first and therefore which is fixed with respect to the other assignment.

We can think of a quantifier as a function which is applied to the result of applying the variable it binds to the sub-formula in its scope. Applying the variable to the sub-formula means performing all the possible substitutions of entities for the variable licensed by the model and computing the truth-value of the resulting propositions. This will yield a set of truth-values. Quantifiers are then, functions from sets of truth-values to a truth-value. For example, applying the variable function to a term such as B(x f) might yield truth-values for the propositions B(f f), B(m f) and B(b f), say {t t f}. Now applying the quantifier function to {t t f} will yield a truth-value for the whole formula – 't' if the quantifier is existential, 'f' if it is universal.

(This is a fairly informal account of the model-theoretic semantics of FOL. Cann *Formal Semantics* and the other textbooks go into more detail.)

7.3 Proof Theory

So far, we have been developing a model-theoretic version of truth-conditional semantics in which we interpret linguistic expressions 'with respect to' or 'relative to' or just 'in' an abstract set-theoretically defined model of the world. We have seen that it is possible to characterise judgements of synonymy, contradictoriness, relations of entailment, and so forth, in terms of the possible models for a language. However, this only works if each model is complete, in the sense that it represents every state of affairs, and we have access to every possible and complete model. (If you can't see why, read the definition of entailment again in section 6)

We would like our semantic theory to not only characterise the semantics of a language correctly (competence) but also to shed light on the process of language comprehension (performance). However, if language users do inference in

a model-theoretic fashion, they would need to carry around the 'whole actual world' (and all the other possible variations on it) 'inside their heads'. This sounds unlikely, because most of us are aware that there are big gaps in our knowledge. One answer to this is to say the competence theory characterises the ideal (omnipotent) language user and that we all operate with partial information and therefore make wrong inferences occasionally. Clearly, doing semantics by machine we cannot hope to model the whole world so we will in practice make inferences (i.e. generate useful entailments) by applying proof-theoretic rules in a goal-directed fashion (i.e. by doing something analogous to automated theorem proving).

Logics, such as PL and FOL, were invented by philosophers to study the **form** of valid argumentation, independently of its content. So philosophers have looked for rules which define valid ways of reasoning in terms of the syntax of logical expressions (regardless of their semantic content). For example, two such rules for PL are shown in (19).

(19) a And-elimination: $p \wedge\ q\ p$
b Modus Ponens: $p \Rightarrow\ q, p\ q$

Each of these rules has some premises and a conclusion (written after the () entails metasymbol). These rules are valid because if the premises are true, the conclusion is guaranteed to be true as well, regardless of the semantic content of the propositional variables p and q. The rules work because of the semantics of the connectives, but given this it is possible to perform inferences using proof-theory 'mechanically'. Proof theory may well be a better way to approach the psychology of inference (and is often a better way to perform inferences mechanically by computer). For now, it is important to recognise how such rules are justified as rules of *valid* entailment in terms of reasoning about possible models for the logics and English fragments we are looking at.

Here is an outline of an axiomatic proof theory for FOL. For any well-formed formulas, ψ, ϕ, φ, the following rules of inference hold:

Modus Ponens: $\psi, \psi\ \Rightarrow\ \phi\ \ \phi$
And-introduction: $\psi, \phi\ \ \psi\ \wedge\ \phi$
And-elimination: $\psi\ \wedge\ \phi\ \ \psi$
Or-introduction: $\psi\ \psi \vee\ \phi$
Or-elimination: $\psi \vee\ \phi, \neg\phi\ \ \psi$
Universal-introduction: $\psi(x)\ \ \forall x\ \psi(x)$
(any free variable is implicitly universally quantified)
Universal-elimination: $\forall x\ \psi(x)\ \psi(t/x)$
(where 't' is any term substituted for all occurrences of 'x' in ψ)

and the following logical equivalences:

De Morgan: $\neg(\psi\ \wedge\ \phi) \Leftrightarrow\ \neg\psi\ \vee\ \neg\phi$

De Morgan: $\neg(\psi \vee \phi) \Leftrightarrow \neg\psi \wedge \neg\phi$
De Morgan: $\forall x\ \neg\psi \Leftrightarrow \neg\exists x\ \psi$
De Morgan: $\neg\forall x\ \psi \Leftrightarrow \exists x\ \neg\psi$
Distributivity: $\psi \wedge (\phi \vee \varphi) \Leftrightarrow (\psi \wedge \phi) \vee (\psi \wedge \varphi)$
Distributivity: $\psi \vee (\phi \wedge \varphi) \Leftrightarrow (\psi \vee \phi) \wedge (\psi \vee \varphi)$
Contraposition: $\psi \Rightarrow \phi \Leftrightarrow \neg\phi \Rightarrow \neg\psi$
Contraposition: $\psi \Leftrightarrow \phi \Leftrightarrow \psi \Rightarrow \phi \wedge \phi \Rightarrow \psi$

Rules of inference and logical equivalences allow purely syntactic manipulation of formulas to derive valid conclusions (proofs).

Can you reformulate the syllogism in (20) in FOL and show that it is valid?

(20) a All men are mortal
b Socrates is a man
c Socrates is mortal

Soundness: if $\Gamma \vdash \psi$ then $\Gamma \models \psi$

Completeness: if $\Gamma \models \psi$ then $\Gamma \vdash \psi$

Decidability: no for FOL
That is, FOL proof theory is sound because every proposition which is entailed by a FOL language is also in any model of that language, and it is complete because every fact in any model of a FOL language is also an entailment (see e.g. Ramsay, A. *Formal Methods in Artificial Intelligence*, CUP, 1988 for proofs and further discussion)

From proof theory to theorem proving involves control principles to avoid non-termination, 'irrelevant' inferences etc. How can one of the rules of inference given above lead to non-termination?

7.4 Automated Theorem Proving

So far, we have not considered the computational implementation of a truth-conditional approach to semantics. We cannot transport model theory (or proof theory) onto a machine directly, because of the problems connected with completeness of the model and with obtaining just those inferences we want (as opposed to an infinite number of mostly useless ones). For example, in model-theoretic semantics we can characterise the truth of *Kim doesn't love Sandy* relative to some model on the basis of the denotation of love1 not containing the ordered pair <Kim1 Sandy1>. However, this only works if the model is *complete* in the sense that it represents every fact (and non-fact) about the world. It is not practical to implement complete, closed models. For this reason, automated theorem proving takes place in the context of a database of known facts represented as formulas in some logic (ie. a partial model). Furthermore, it is not possible to apply proof-theoretic rules in an unconstrained way to such

a database.

Exercises

What would happen if you freely applied the rule of $\wedge$-Introduction (p, q $\models$ p $\wedge$ q). (easy)

The techniques of automated theorem proving provide ways of applying a subset of proof-theoretic rules in a constrained and goal-directed way. Mostly, such techniques are restricted to a subset of FOL, so it follows that a computational implementation of NL semantics will need to (at least) restrict itself to FOL analyses.

Our approach will look something like this: NL Semantics $\rightarrow$ [Lambda-reduction] $\rightarrow$ FOL $\rightarrow$ [Skolemisation, etc.] $\rightarrow$ Clausal Form $\rightarrow$ Query/Assert in Database.

Forward chaining is a technique for doing goal-directed inference. It is a technique for implementing Modus (Ponendo) Ponens (MPP) or implication elimination (p $\rightarrow$ q, p $\models$ q) which doesn't result in making many 'unnecessary' inferences. For example, if we store (21a) in our database and apply MPP and Universal Elimination, we will infer as many formulas like (21b) as there are men in the database and instantiate (21a) to as many useless formulas like c) as there are individuals (who are not men) in the database.

(21) a (all (x) (if (man1 x) (snore1 x)))
b (snore1 Kim1)
c (if (man1 felix1) (snore1 felix1))
d (man1 Kim1)

Forward chaining is a technique which dispenses with the step of Universal Elimination and inferring formulas like c). Instead, formulas like a) have no effect until an assertion like d) is added to the database and only then is b) inferred. To do this formulas like a) must be represented in implicit-quantifier form:

(if (man1 _X) (snore1 _X))

Now we replace UE with unification (i.e. term matching, see section on unification in parsing handout and J&M, ch15 and ch18) and state the forward chaining rule as follows: from p$'$ and p $\rightarrow$ q infer q$'$ where p$'$ unifies with p and q$'$ is the result of making the same substitution(s) in q. Thus we have converted UE + MPP into unification and search in a database of formulas expressed in a notational variant of FOL.

Most of the time forward chaining is still too undirected to be efficient. The effect of adding any universally quantified statement to the database will be to also add all the valid inferences which follow from it (by UE + MPP) in the database. Backward chaining is an alternative technique which performs these inferences at 'query time' rather than 'assertion time'. That is, inferences are only performed when they are needed.

For example, if we query the database with (22a) and the database contains b)

and c), then backward chaining will attempt to unify a) with all the variable-free, ground propositions in the database and, if this fails, with all the consequents of implications like b). The latter will succeed, but before we can answer 'yes', it is necessary to prove the truth of the antecedent with identical variable substitutions. This can be done by unifying the resulting variable-free proposition with the identical formula c).

(22) a (snore1 Kim1)
b (if (man1 _X) (snore1 _X)
c (man1 Kim1)

We need to distinguish queries from assertions because the variables in the implicitly-quantified queries behave more like existential than universal variables. So the alternative query (23) means 'is there a value for _X which results in a true proposition?'

(23) (snore1 _Y)

In this case backward chaining will produce the sub-query (man1 _Y) which will unify with the ground proposition, again signifying success with _Y=Kim1. Intuitively, querying with a variable-free formula is like a yes/no-question, whilst one containing variables is like a wh-question.

We can describe backward chaining more precisely now as follows: Given Query: q' where $p \rightarrow q$ is in the database and q' and q unify with substitutions t, then Query: p^t (ie. the result of making the same substitutions in p) and if this unifies with p'' with substitutions t' then $t \sqcap t'$ is the answer.

There is much more to theorem proving than this; eg. negation as failure, skolemisation, etc (see e.g. J&M and references therein or Blackburn and Bos, 2005).

8 An extended FOL-like English fragment, F2

Our second fragment (F2) extends and builds on F1 to include sentences such as those in (24).

(24) a Every man smiles
b Every man likes some woman
c No man smiles
d Every man_i loves $himself_i$
e He loves Sandy
f A man_i likes Fido and he_i likes Sandy (too)
g Sandy gives Kim a dog

8.1 Verb Complementation

We have already seen how to represent intransitive verbs / one-place predicates in F1. Now we can add transitive, ditransitive etc. (25) and capture (some of) the semantics of an extended fragment of English, F2, in terms of FOL.

(25) a Kim loves Sandy
b love1(kim1, sandy1)
c Kim gave Sandy Fido
d give1(kim1, sandy1, fido1)

(Note that we are ignoring tense/aspect.) Can you construct models which will make (25b,d) true? Can you 'translate' the examples in (26) into FOL formulas?

(26) a Fido is on Sandy
b Sandy needs a computer
c Sandy thinks Kim owns a computer

Can you construct models again? What problems do these examples raise? The extensional semantics of complementation commits us to the existence of 'a computer' in order to assign a logical form to (26b) – can you see why? Verbs like *need* are often called intensional verbs because they require an account of sense/intension to capture their truth-conditions and entailments properly. (26c) raises similar but even more difficult problems concerning so-called propositional attitude verbs (see section 11.1).

8.2 Quantifiers and pronouns

We will also try to extend our truth-conditional semantics of English to cover English quantifers, such as *every* and *some*, and pronouns, such as *he/him* and *she/her*.

We will treat the English quantifiers *every* and *some* as analogous to the universal and existential quantifiers of FOL, respectively. The meaning of other English 'quantifiers', such as *no*, *a*, *the*, and so forth, will hopefully be reducible to the meaning of these two 'basic' quantifiers. Pronouns will be treated analogously to bound variables in FOL. F2 also includes nouns, such as *man*, *woman*, and so forth. Unlike proper names, nouns do not denote entities but rather properties of entities. Therefore, their meaning is the same as that of an intransitive verb, such as *snore*.

If we consider the meaning of the two sentences in (27a) and c), it should be clear that we can't capture their meaning by translating them into the FOL expressions in b) and d) respectively.

(27) a Every man snores
b $\forall\ x\ snore1(x)$
c Some woman smiles
d $\exists\ x\ smile1(x)$

The problem is that the FOL expressions will be true of any entity in the model who snores or smiles. They aren't restricted in the way the English sentences are to apply to just men or just women, respectively. Obviously, we have failed to include the meaning of the nouns in our translation. The question is how to combine the noun denotations and verb denotations correctly to arrive at the correct truth-conditions for sentences of this type? a) has the logical form of an if-then conditional statement. It says that for any entity if that entity is a man then that entity snores. On the other hand, c) says that there exists at least one entity who is both a woman and smiles, so it has the logical form of a conjunction of propositions. Therefore, the correct translations of these two sentences are given in (28a,b) and (28c,d), respectively.

(28) a $\forall$ x man1(x) $\Rightarrow$ snore1(x)
b `[forall,X^[if,[man1,X],[snore1,X]]]`
c $\exists$ x woman1(x) $\wedge$ smile1(x)
d `[exists,X^[and,[woman1,X],[smile1,X]]]`

We want our grammar of F2 to associate these logical forms with these sentences. To achieve this in a compositional fashion we will need to associate a 'template' logical form for the entire sentence with each of the different quantifiers – otherwise we won't be able to capture the different ways in which the NP and VP are combined semantically, depending on which quantifier is chosen. In (29) I give a 'translation' of *every*.

(29) a $\forall$ x P(x) $\Rightarrow$ Q(x)
b `[forall,X^[if,[P,X],[Q,X]]]`

'P' and 'Q' are to be interpreted as (one-place) predicate variables. The process of combining the meaning of the quantifier with that of the noun and then that of VP is now one of substituting the meaning of the noun for 'P' and the meaning of the VP for 'Q'. The templates for the other quantifiers are shown in the lexicon for F2 below. However, note that FOL does not include predicate variables, so I am cheating in order to try to construct a compositional FOL-like treatment of F2. (We'll return to this problem in section 11.1.)

Many pronouns in English sentences pick out (at least) one entity in the universe of discourse, so one way to treat them semantically is to translate them into existentially-quantified variables ranging over the entities in the model with appropriate gender constraints, etc. For example, (30a) might translate as b).

(30) a He loves belinda
b $\exists$ x male1(x) $\wedge$ love1(x sandy1)

But this analysis ignores the fact that the referent of a pronoun is normally determined anaphorically or indexically; that is, from the linguistic or extralinguistic context, respectively. However, it provides a reasonable first approximation of the semantic part of the meaning of a pronoun (ie. the part which is independent of the context of utterance). In other examples, such as (31a), it seems more appropriate to have the variable translating the pronoun bound by the universal quantifier, as in b).

(31) a Every man$_i$ thinks that he$_i$ snores
b $\forall$ x man1(x) $\Rightarrow$ think1(x snore1(x)))
c `[forall,X^[if,[man1,X],[think1,X,[snore1,X]]]]`

This is not the only possible interpretation of examples like (31a), but when the pronoun is interpreted as being anaphorically linked with the subject NP, it does not pick out one entity but rather the set of entities who are men. This is exactly what the translation as a universally-quantified bound variable predicts. We'll leave examples like (31a) out of F2 because of the problems with propositional attitude verbs, but examples like (32a) are in F2, and here translating the pronoun as a bound variable in the scope of the existential quantifier seems to capture their truth-conditions correctly.

(32) a A man$_i$ likes Sandy and he$_i$ likes felix (too)
b $\exists$ x man1(x) $\Rightarrow$ like1(x sandy1) $\wedge$ like1(x felix1)
c `[exists,X^[and,[and,[man1,X],[like1,X,sandy1]],[like1,X,felix1]]]`

However, we still have a problem because the scope of the quantifier can only be determined when we have decided whether the pronoun is anaphoric and coreferential with a quantified NP antecedent, so the 'translation' of a pronoun as an 'externally' bound variable only works for F2 with subscripts indicating coreference, as in (32a). Note also the nested prefix 'and's in (32c) – can you see how these get produced given the lexicon and grammar for F2 given in the next sections?

8.2.1 Lexicon for F2

The lexicon for F2 is the same as F1 with the addition of some determiners, nouns and pronouns, and a ditransitive verb (so that we have one three-place predicate). The complete lexicon is given below:

```
Kim : Name : kim1
Fred : Name : fred1
Sandy : Name : sandy1
```

```
Fido : Name : fido1
Felix : Name : felix1

he_x : PN : X
he : PN : [exists,X^[and,[male1,X],[P,X]]]
her_x : PN : X
her : PN : [exists,X^[and,[female1,X],[P,X]]]
himself_x : PN : X
herself_x : PN : X

and : Conj : and
or : Conj : or
it-is-not-the-case-that : Neg : not

snores : Vintrans : snore1
smiles : Vintrans : smile1
likes : Vtrans : like1
loves : Vtrans : love1
gives : Vditrans : give1

a : Det : [exists,X^[and,[P,X],[Q,X]]]
no : Det : [not,[exists,X^[and,[P,X],[Q,X]]]]
some : Det : [exists,X^[and,[P,X],[Q,X]]]
every : Det : [forall,X^[if,[P,X],[Q,X]]]

man : N : man1
woman : N : woman1
dog : N : dog1
cat : N : cat1
```

8.2.2 Grammar for F2

The syntactic rules of F2 are the same as those for F1 with the addition of three further rules, shown below:

7) VP $\rightarrow$ Vditrans NP NP : [V′,NP′,NP′]
8) NP $\rightarrow$ Det N : [Det′,N′]
9) NP $\rightarrow$ PN : PN′

Rule 7) allows us to cover sentences containing ditransitive verbs, and says that semantically the denotation of the verb represents a function which takes the denotations of both NP objects as arguments. Rule 8) introduces NPs containing a determiner and a noun and rule 9) pronouns. The semantics of 9) is straightforward and identical to that for the rule which introduces proper

names. Determiners are treated semantically as functions which take nouns as arguments. The complete new grammar is shown below:

1) S → NP VP : [NP′,VP′]
2) S → S Conj S : [Conj′,S′,S′]
3) S → Neg S : [Neg′,S′]
4) VP → Vtrans NP : [V′,NP′]
5) VP → Vintrns : V′
6) NP → Name : Name′
7) VP → Vditrans NP NP : [V′,NP′,NP′]
8) NP → Det N : [Det′,N′]
9) NP → PN : PN′

There is one other change to this grammar involving rule 1). The subject NP is now treated as a function which takes the denotation of the VP as its argument. This is because we are treating the semantics of quantifiers as the semantic template into which the meaning of the rest of the sentence slots. (In fact, this means that there is a problem with treating non-subject NPs as arguments of verbs. However, we will ignore this until later when we consider the process building up a logical form through a series of function-argument applications in more detail.)

8.3 Logical Form

There is a difference between the way we have treated the semantics of the two artificial languages (logics) PL and FOL and the way we have treated the semantics of our two English fragments F1 and F2. For the former, we have given semantic rules which work out the truth of formulas relative to some model directly because the semantic rules are interpreted as instructions to check the model in various ways. For example, applying the predicate snore1 to the entity kim1 is defined in FOL as a semantic operation which reduces to seeing whether kim1 is a member of the set denoted by snore1. On the other hand, in specifying the semantics of F1 and F2 we have translated sentences of F1 and F2 into formulas which we have called logical forms or function-argument structures which themselves are equivalent to (ie. just notational variants of) formulas in FOL.

One way to think about this is to imagine that we are giving the semantics of English indirectly by first translating English into logic forms and then interpreting the resulting logical expressions in some model. The advantage of translating into FOL (or some similar logic) is firstly, that this representation is unambiguous (because of the brackets and the substitution of predicates and variables or constants for words) and secondly, that the semantics of, say, FOL is relatively clear-cut (compared to the semantics of English).

8.4 Scope Ambiguities

We saw that formulas of FOL containing more than one quantifier have different truth-conditional interpretations depending on the order or relative scope of these quantifiers. However, our grammar for F2 only produces one of these orderings for analogous sentences of F2. For example, (33a) only receives the interpretation (33b,c).

(33) a Every man loves some woman
b $\forall$ x man1(x) $\Rightarrow$ $\exists$ y woman1(y) $\wedge$ love1(x y))
c `[forall,X^[if,[man1,X],[exists,Y^[and,[woman1,Y],[love1,X,Y]]]]]`

can you verify this is the interpretation yielded by the rules above? This is the interpretation which is true provided for each man there is at least one woman (who may be different in each case) who he loves. However, to get the 'every man loves Marilyn Monroe' sort of reading we would need to reverse the order of the quantifiers so that the existential has (so-called) wide scope. Incidentally, if you find this reading a little forced for (33a), there are other examples where it seems more natural, as (34) illustrates.

(34) a Every man loves a woman
b Every man loves one woman
c One language is spoken by everyone (here)
d A student guide took every prospective candidate round the lab

In (34a) and b) it is easier to get the wide scope existential reading where there is just one woman, but most people still feel that this is not the preferred interpretation. In c), where the existential *one* occurs before *everyone* in the surface form of the sentence, the existential wide scope reading seems more readily available, whilst in d) it is definitely preferred. We adopt the existential wide scope reading of d) very readily because of our general knowledge about the likely routine for showing prospective new students aound – that one existing student takes them on a tour.

Further evidence that lexical semantic and general knowledge affects the quantifier scoping that we choose comes from the examples in (35).

(35) a There was a name tag near every door
b A flag was hanging from every window

In these examples, the existential *a* precedes the universal *every* in the surface realisation of these examples, yet the universal is given wide scope, and we naturally assume there is more than one flag or name tag. Presumably, we reach these conclusions on the basis of our knowledge about the relative size of flags and windows, name tags and doors, their function, and so forth.

Scope ambiguities of this type are not restricted to quantifiers. There are scope

ambiguities in F1 concerning the relative scope of the negation, conjunction and disjunction operators. These also interact with quantifiers to create further ambiguities. For example, (36a) is ambiguous between b) and c) depending on the relative scope of *not* and *every*.

(36) a Everyone doesn't snore
b $\forall$ x $\neg$ snore1(x)
c $\neg\forall$ x snore1(x)

c) is compatible with some people snoring, whilst b) is not. In addition, there are scope ambiguities in connection with the interpretation of examples containing pronouns. In (37a) the preferred reading is the one in which *he* is treated as co-referential with *man*, but in (37b) it seems more natural to assume *he* does not have an antecedent within the sentence.

(37) a A man likes Sandy and he likes Felix (too)
b Every man likes Sandy and he likes Felix (too)

We can describe this difference in terms of whether the variable which translates *he* is in the scope of the quantifier translating *a* or *every*. Once again many factors seem to influence the calculation of pronominal reference. For example, stressing *he* in a) prevents the otherwise preferred interpretation, whilst in (38a) lexical semantics and general world knowledge play a role in calculating the antecedent of *their*.

(38) a The men allowed the women to join their club
b The men allowed the women to found their club

Unlike the scope ambiguities between connectives and negation in F1, scope ambiguities involving quantifiers are not the result of syntactic ambiguity. There is no syntactic reason to believe that these examples are syntactically ambiguous, but nevertheless they are semantically ambiguous. This is a problem for the theory we have been developing because it predicts that sentences should only be semantically ambiguous if they are syntactically or lexically ambiguous. Thus we can produce two logical forms for (39a) and b) because we can associate *bank* with two concepts (financial institution and river side) and because PPs can function adverbially or adjectivally (attaching to NP or VP).

(39) a Kim sat by the bank
b Kim hit the woman with the umbrella

We have tried to treat pronouns as lexically ambiguous, at the cost of subscripting F2, but it it is even harder to see how to treat quantifier scope as a lexical ambiguity. A lot of research has gone into this problem and there are several proposals as to how to produce sets of logical forms from a lexically and syntactically unambiguous sentence. The technical details of these proposals don't matter. What is important is that they weaken the claim that syntax de-

termines interpretation and undermine our initial proposal that syntactic and semantic rules are paired one-to-one in the grammar.

8.5 Exercises

1) Interpret the following expressions of FOL in the model given underneath, show the process of interpretation by displaying the possible substitutions of entities for variables and then the contribution of each quantifier.

(40) a $\forall\ x\ A(x)$
b $\exists\ x\ (A(x) \Rightarrow\ B(x\ b))$
c $\forall\ x\ (A(x) \Rightarrow\ C(a\ x\ e))$
d $\forall\ x\ \exists\ y\ (A(x) \Rightarrow\ B(a\ x\ y))$

```
Model:

I {a, b, c, d, e}

F (A) {a, b, c}

F (B) {<a b> <a c>}

F (C) {<a d e> <b d e> <c d e>}
```

2) Think of English sentences which the formulas in question 1) could be used to represent or 'translate'.

3) Say whether the following sentences are part of F2. If they are draw their syntax trees and give their logical forms. If not change them so that they retain the same meaning but are in F2 and then do the same. If any examples are ambiguous give all the possibilities:
a) Every woman loves Kim and he loves Felix
b) It-is-not-the-case-that a dog snores and every cat smiles
c) Fred gives some woman a dog
d) Fred loves Sandy and likes Felix
e) Every man likes Sandy and Sandy likes her dog
f) No woman loves every man and every dog

4) Construct a model for F2 which makes all of the sentences of 3) true under at least one interpretation. Indicate which interpretation you are assuming by marking the appropriate logical form with an asterisk in your answer to question 3).

5) Can you find sentences which the grammar for F2 assigns incorrect logical forms or not enough logical forms as a result of scope ambiguities? Write them

down and write down one other appropriate logical form not produced by the grammar.

9 Syntax-Directed Compositional Translation

In the previous sections, I cheated a bit over the processes involved in building up the logical form of sentences of F2 in tandem with performing a syntactic analysis. For F1, it was possible to characterise this process as one of function-argument application for all of the rules of the grammar. However, this doesn't work for F2, mainly because we included more complex NPs involving quantifiers. In this section, we won't extend the English fragment much, but will clarify the semantic rules which lie behind F2 in more detail.

9.1 The Typed Lambda Calculus

The Lambda Calculus (LC) is another artificial language like PL and FOL. (For historical reasons its more often called a calculus rather than a logic – the word 'calculus' emphasises the proof-theoretic aspect of the language, but like FOL it (now) has a complete model-theoretic semantics). The variety we will look at is typed because variables range over particular syntactic categories or types of the language; for example one-place predicates or entity constants.

LC is an extension of FOL to include the lambda (λ) operator. This operator operates syntactically rather like a quantifier in FOL; for example, (41a) is a well-formed formula of LC in which the lambda operator binds the variable 'x'.

(41) a $\lambda\ x\ A(x) \wedge\ B(x\ a)$
b $\lambda\ x\ [A(x) \wedge\ B(x\ a)](b)$
c $A(b) \wedge\ B(b\ a)$

However, the interpretation of lambda expressions is rather different to that of expressions containing quantifiers. The lambda operator is a function forming device which can be used to compose existing functions to build one new composite, complex function. So (41a) should be read as 'the property of being simultaneously A and being in the B relation to a'. We can find constants to substitute for 'x' and thereby obtain a well-formed formula of FOL again. The process of doing this is called lambda reduction (or, less obviously but more conventionally, beta reduction). In b) we have written the same formula but indicating the scope of the lambda operator with square brackets. The constant in brackets outside the lambda expression is interpreted as the argument to the function denoted by the lambda expression, and the process of applying the function to its argument is beta reduction. This results in the formula in c). The expressions in b) and c) are logically equivalent (truth-conditionally synonymous) because of the semantics of LC. Another way of saying this in terms

of proof theory is to say that c) can be validly inferred from b).

To understand the semantics of beta reduction in model-theoretic terms we need to revise slightly the way we represent predicates. We have said that a one-place predicate such as snore1 denotes a set of entities and is a characteristic function from entities (domain) to truth-values (range). One way of writing this function down is as a lambda expression, as in (42a).

(42) a λ x [snore1(x)]
b λ x [snore1(x)] (kim1)
c snore1(kim1)

Now both b) and c) represent the application of that function to the entity kim1. However, in order to be able to extract over n-place predicates to form new 'n minus 1'-place predicates we must interpret two and three -place predicates as functions from entities to functions from entities to truth-values and as functions from entities to functions from entities to functions from individuals to truth-values. (Got that?!) For example, we have thought of two-place predicates like love1 as a function which takes two individuals as arguments and yields a truth-value. However, we can also break it down into two functions, the first of which takes one entity as an argument and yields a new function which is exactly like the function associated with a one-place predicate. Thus, the domain of this function will be the set of entities and the range will be a set of (characteristic) functions. Assuming the following model, we can represent love1 as shown underneath:

```
I {kim1, sandy1, fido1, felix1}

F (love1) {<kim1 sandy1> <felix1 sandy1> <fido1 felix1>
           <felix1 fido1>}

love1:       1               2

kim1        -->    kim1      --> f
                   sandy1   --> f
                   fido1     --> f
                   felix1    --> f

sandy1    -->    kim1      --> t
                   sandy1   --> f
                   fido1     --> f
                   felix1    --> f

felix1      -->    kim1      --> f
                   sandy1   --> t
                   fido1     --> t
```

```
                    felix1    --> f

fido1       -->     kim1      --> f
                    sandy1  --> f
                    fido1     --> f
                    felix1    --> t
```

The first function takes us from one entity to a new function from entities to truth-values. When we supply the second entity, the second function yields a truth-value. Using this technique, we can now see how the lambda operator is able to create new one-place predicates out of two-place predicates where one argument is fixed. So, for example, the lambda expression in (43a) is one way of expressing or referring to the function which appears to the right of the arrow after kim1 above (assuming that it is interpreted in this model). Note that we assume that we always combine with the object NP before the subject NP.

(43) a λ x [love1(x kim1)]
b λ x [love1(x kim1)] (sandy1)
c love1(sandy1 kim1)

So far we have used the lambda operator to create new functions which abstract over entities (ie. the variable bound by the lambda operator has ranged over the entities in the model). However, lambda abstraction can be applied to any category or type in the logical language; applying it to entities gives Second Order Logic, applying it to one-place predicates, Third Order Logic, two-place predicates Fourth Order Logic and so forth. We will want to apply lambda abstraction to some predicates in the new version of the grammar for F2, so the logical forms we will be constructing will be formulas of (at least) the third order typed lambda calculus.

Quite rapidly lambda expressions get hard to manipulate syntactically; for example, the formula in (44a) is equivalent to b) by beta reduction.

(44) a $\lambda\ x[A(x\ a) \vee\ B(x) \Rightarrow\ \lambda\ y\ [C(y\ b) \wedge\ \lambda\ z\ [D(x\ z) \wedge E(y\ c)](d)](e)](f)]$
b $A(fa) \vee\ B(f) \Rightarrow\ C(eb) \wedge\ D(fd) \wedge\ E(ec)$

In Prolog-style notation lambda bound variables are represented as `X^`$[\psi]$ and pulled to the front of formulas, as in (45).

(45) a `[X^[Y^[Z^[if,[or,[A,X,a],[B,X]],[and,[C,Y,b],[and,[D,X,Z],[E,Y,c]]]],d],e],f]`
b `[if,[or,[A,f,a],[B,f]],[and,[C,e,b],[and[D,f,d],[E,e,c]]]]`

9.2 Beta Reduction

Whenever possible, our strategy will be to use LC as a way of specifying the meaning of sub-expressions but reduce these via beta reduction to well-formed formulas of FOL for specifying the semantics of propositions. This is because we want to make use of these formulas with automated theorem provers and these are (mostly) restricted to (a subset of) first-order logic.

Beta reduction applied to a 'beta-redex' (a formula containing an argument to a lambda term) such as $\lambda\, x[P(x) \wedge Q(x)](a)$ is defined as the substitution of 'a' for all occurrences of 'x' in $P(x) \wedge Q(x)$. It is necessary to be careful about variable names, since it is possible to accidentally bind or substitute for variables with identical names in different formulas when performing reduction. For instance, reducing $\lambda P\ [\exists x\ P(x)]$ with $\lambda\ x\ R(x)$ should produce $\exists x\ \lambda x1\ [R(x1)](x)$ and not $\exists x\ \lambda x\ [R(x)](x)$ in which further lambda reduction would fail because all the variables would be bound by the existential quantifier. This problem can be avoided by assigning distinct names to variables in separate terms before reduction.

9.3 Types

We use a typed version of LC to avoid logical paradoxes and to keep the notion of a function within the bounds of standard set theory. For instance, in untyped LC it is possible to construct terms which denote things like 'the set of all sets which are not members of themselves' – $\lambda x\ \neg Member(x\ x)$ – and to ask questions like 'is this set a member of itself?' – $(\lambda x\ \neg Member(x\ x) \lambda x\ \neg Member(x\ x))$. If it is, then it is a set which is a member of itself and if it isn't, then it is a set which is a not a member of itself, and so should already be part of its own denotation. Typing prevents such paradoxical results.

An extensional type system can be built up from the primitives 'e' for entity and 't' for truth-value. So an entity constant or variable is of type 'e' and a proposition of type 't'. The type of everything else follows from this. For example, a one-place predicate is of type `<e t>` or `e --> t` because it is a function from an entity to a truth-value; a two-place predicate is a function from an entity to a function from an entity to a truth value; ie. `<e <e t>>` or `(e --> (e --> t))`. So if we are being careful, we should explicitly type all the lambda bound variables in a LC formula rather than relying on loose conventions like 'x','y' or 'X', 'Y' are entity variables and 'P' and 'Q' are x-place predicate variables. However, it all gets a bit verbose, as (46) indicates, so a lot of the time we'll suppress the types BUT it is important to check for typing, at least implicitly, when constructing typed LC formulas.

(46) $\lambda P_{<et>} \lambda x_e P(x)$

9.4 Rule-to-rule Translation

Armed with typed LC, we can now specify more precisely the manner in which logical forms are built up compositionally in tandem with the application of syntactic rules in F2. For example, consider the analysis of *Kim snores* again. The syntactic analysis of this sentence requires the following three rules of F2:

1) S $\rightarrow$ NP VP : [NP′,VP′]

2) VP $\rightarrow$ V : `X^`[V′,X]

3) NP $\rightarrow$ Name : `P^`[P,Name′]

These rules are similar to those used in the grammars of F1/2 except that we have specified the semantic part of the rules in more detail using the notation of LC. The denotation of *snores* is a set of entities (written snore1). When we apply rule 2 to *snores* to obtain the partial syntactic analysis shown in a) below, we also instantiate the variable in the LC formula paired with this rule to form the semantic interpretation of *snores*, as in b):

```
a)      VP                    b) X^[snore1,X]
        |
        V

      snores
```

This lambda expression is an instruction to form a function over the denotation of snore1 from entities to truth-values. The denotation of *Kim* is kim1, so the analysis of the subject NP proceeds in a similar way to produce:

```
a)     NP                    b) P^[P,kim1]
       |
       Name
       |
       Kim
```

However, the semantic rule associated with NPs is an instruction to form a function which abstracts over the set of properties predicated of kim1 (ie. the set of functions representing one-place predicates). This more complex account of the denotation of an NP is needed so that NPs containing either proper names or determiners (esp. quantifiers) and nouns will end up with the same denotation and can therefore be combined with predicates using the same semantic rules. Thus in F1, where all NPs are proper names, the denotation of an NP is always an entity constant. Therefore, the semantic rule combining NPs and VPs is function-argument application, where the function associated with the VP is a characteristic function which checks for membership of the argument in the denotation of the verb and returns a truth-value. However, in F2 this won't work because NPs like *every man* cannot denote entities. Instead, we make NPs denote functions from characteristic functions to truth-values. The only difference between proper name NPs and quantified NPs will be whether the lambda abstracted set of properties is predicated of an entity constant or (bound) variable.

Since this treatment of NPs is more complex, we'll try to make things clearer with a concrete example. Assuming the model given below, the denotation of the NP *Kim* will be the function shown underneath:

```
I {kim1 sandy1}

F (snore1) {kim1}

F (love1) {<kim1, sandy1>}
```

```
F (like1) {<sandy1 kim1>}

NP(kim1)

{
snore1:
                            --> t

love1:
               kim1         --> f
               sandy1     --> t

like1:
               kim1         --> f
               sandy1     --> f
                       }
```

In other words, the denotation of the NP *Kim* contains the set of possible properties which can be predicated of kim1 in this model, such as snoring, liking himself, loving Sandy etc., and the lambda operator creates a function from these VP (one-place predicate) denotations to truth-values.

Combining the denotation of the NP and VP above using rule 1) produces the syntactic and semantic analysis shown in a) and b) below:

```
a)          S               b) [P^[P,kim1],X^[snore1,X]]
        /       \
    NP            VP            [X^[snore1,X],kim1]
    |             |
    Name          V             [snore1,kim1]
    |             |
    Kim           snores
```

The formulas in b) are logically equivalent, the third is the result of applying beta reduction (twice) to the first. Here beta reduction is like 'merging' the two lambda expressions by matching the (typed) variables 'P' and 'X' to expressions of the appropriate type. In fact, what we have done here is compose two functions – one abstracting over predicates, the other over entities – to produce a formula equivalent to the one we got in F1 for *Kim snores* by treating kim1 as an argument of the one-place predicate snore1.

This may seem like a lot of extra complexity for nothing, but this analysis is essential when we move on to sentences like *Every man snores*. To analyse this sentence, we need the other NP rule given below:

4) NP → Det N : [Det′,N′]

This rule applies the denotation of the determiner to that of the noun. In section 7.2, we anticipated the use of LC by describing the meaning of *every* as the 'template' for a logical form in (47a).

(47) a `[forall,X^[if,[P,X],[Q,X]]]`
b `P^[Q^[forall,X^[if,[P,X],[Q,X]]]]`

The correct way to write this in LC is as a lambda abstraction over two one-place predicates, as in (47b). So the analysis of *Every man* will come out like this:

```
a)    NP     b) [P^[Q^[forall,X^[if,[P,X],[Q,X]]]],X^[man1,X]]
    /   \

  Det     N     Q^[forall,X^[if,[X1^[man1,X1],X],[Q,X]]]
  |       |
 Every   man    Q^[forall,X^[if,[man1,X],[Q,X]]]
```

By two applications of beta reduction, we obtain the third formula. Now when we analyse *snores* we get:

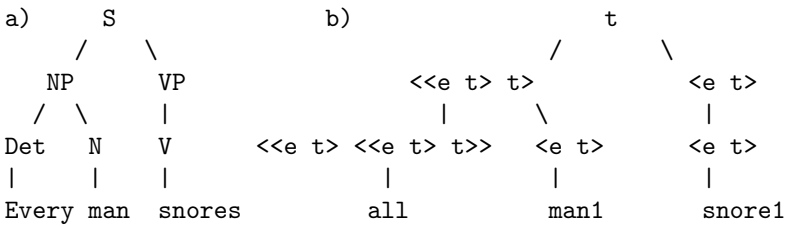
```
a)      S         b) [Q^[forall,X^[if,[man1,X],[Q,X]]],Y^[snore1,Y]]
     /    \

  NP         VP       [forall,X^[if,[man1,X],[snore1,X]]]
 /  \        |
Det   N      V
|     |      |
Every man    snores
```

Using the apparatus of LC, we are able to get the different logical forms for quantified and unquantified NPs, but at the same time achieve this without needing different semantic rules for combining different types of NPs with VPs. This is essential if we are going to have an account of semantic productivity. In effect, using LC and raising the type of NPs allows us to maintain compositionality but provides more flexibility in the way we semantically combine the meanings of words and phrases.

9.5 Types revisited

We can now give a more precise account of the relationship between syntactic and semantic rules and categories in the grammar. We can associate with each syntactic category of the grammar, a semantic type which specifies the type of

object denoted by expressions of that category and also specifies how it combines with other types. For example, we have said that proper names denote entities (i.e. any discrete object – car1, table1, kim1 etc.). We can write this `<e>`. Sentences denote propositions (ie truth-values) written `<t>`, intransitive verbs denote functions from entities to truth-values `<e t>`, NPs denote functions from functions from entities to truth-values to truth-values `<<e t> t>`. Given this new notation we could represent the semantic analysis of *Every man snores* given above slightly more abstractly as below:



The type notation illustrates how the generic functions apply to arguments or compose with each other to yield new functions, and ultimately a truth-value. Once we know the semantic type of every syntactic category, we know how to combine them to form the semantic types of larger categories. Of course, when interpreting an actual sentence, we replace the semantic types with the functions derived from the denotations of the words in the sentence in the model in which we are performing the interpretation (ie. we replace types with tokens).

9.6 English Fragment 2 Redone

We will now rewrite the grammar for F2, making use of LC to specify more precisely the form of the semantic rules and giving the semantic type of each syntactic category in the grammar.

Lexicon

The only change to the lexicon concerns the semantics of the quantifiers, which are now written as double lambda abstractions over two one-place predicates – the noun and VP denotations, respectively, and of the pronouns, where we now write the 'externally' bound version as a lambda abstracted entity variable.

```
Kim : Name : kim1
Fred : Name : fred1
Sandy : Name : sandy1
Fido : Name : fido1
Felix : Name : felix1

he_x : PN : P^[X^[and,[male1,X],[P,X]]]
he : PN : P^[exists,X^[and,[male1,X],[P,X]]]
her_x : PN : P^[X^[and,[female1,X],[P,X]]]
her : PN : P^[exists,X^[and,[female1,X],[P,X]]]
himself_x : PN : P^[X^[and,[male1,X],[P,X]]]
herself_x : PN : P^[X^[and,[female1,X],[P,X]]]

and : Conj : and
or : Conj : or
```

```
it-is-not-the-case-that : Neg : not

snores : Vintrans : snore1
smiles : Vintrans : smile1
likes : Vtrans : like1
loves : Vtrans : love1
gives : Vditrans : give1

a : Det : P^[Q^[exists,X^[and,[P,X],[Q,X]]]]
no : Det : P^[Q^[not,[exists,X^[and,[P,X],[Q,X]]]]]
some : Det : P^[Q^[exists,X^[and,[P,X],[Q,X]]]]
every : Det : P^[Q^[forall,X^[if,[P,X],[Q,X]]]]

man : N : man1
woman : N : woman1
dog : N : dog1
cat : N : cat1
```

Grammar for F2

1) S → NP VP : [NP′,VP′]
2) S → S Conj S : [Conj′,S′,S′]
3) S → Neg S : [Neg′,S′]
4) VP → Vtrans NP : [NP′,`Y^[X^`[V′,X,Y]]]
5) VP → Vintrans : `X^`[V′,X]
6) NP → Name : `P^`[P Name′]
7) VP → Vditrans NP NP : [NP'_2,[NP'_1,`Z^[Y^[X^`[V′,X,Y,Z]]]]]
8) NP → Det N : [Det′,N′]
9) NP → PN : `P^`[PN′]

The changes from the old version just involve the semantics of the VP rules. These are now written as lambda expressions. For example, the semantic rule associated with 5) says that the meaning of an intransitive verb is a lambda abstraction over whatever the denotation of that intransitive verb is to form a function from entities to truth-values. The semantic rule for 4) says that the denotation of the transitive verb should be the argument to a function from entities to a second function from entities to truth-values. Since the semantic type of the NP will be a lambda expression denoting a function from VP type functions to further functions, the resulting formula is quite complex. However, after beta reductions it reduces to the appropriate FOL formula.

9.7 Pronouns and Quantifier Scoping Revisited

The LC treatment of quantifiers in F2 does not resolve the problem of how to get wide-scope existential readings of examples like *Every man loves one woman*. Can you show this? Nor does treating pronouns as ambiguous between existientially-bound for diectic uses and lambda bound for coreferential uses get us that far. Concentrating again on the coreferential cases like *A man$_i$ likes Sandy and he$_i$ likes Felix (too)*, the rules of F2 will not actually reduce the lambda bound variable translating *he* so that it is bound by the existential associated with *A*. Can you prove this to yourself? The problem is that the semantics of the first conjunct gets 'closed off' before the second is interpreted. For this reason (and others), several semantic theories (DRT, Dynamic Semantics: see eg. Bos and Blackburn references) have explored variant logics which allow the scope of quantifiers to be kept open as the interpretation of a sentence is built up incrementally.

Since we've spent some time looking at the problem scope ambiguities raise for syntax-directed semantic interpretation, you might wonder why we are explaining in more detail exactly how syntactic and semantic rules are paired together. However, if we gave up organising the grammar in this fashion we would be in danger of losing our account of semantic productivity. We want to show how the meaning of words and phrases are combined using general rules. The apparatus of LC is likely to be essential to this enterprise because it provides a very flexible way of specifying the denotations and modes of combination of linguistic expressions. Therefore, we have a chance of coming up with a general rule which states how **all** NPs and VPs combine, say; and not one rule for NPs containing proper names, another for quantified NPs, another for NPs with pronouns, another for NPs with relative clauses, and so forth. It may be that ultimately, we want to make these rules sensitive to more than just the syntactic analysis of these linguistic expressions, but we will still want a compositional account of the meaning of words, phrase, clauses, and sentences; even if the semantic rules are sensitive to, say, the presence of specific words or intonation, and aspects of the wider (non-)linguistic context too.

9.8 Exercises

1) Write down the logically equivalent first order formulas corresponding to the following lambda expressions by performing as many beta reductions as are necessary:
a) $\lambda x\ [M(x) \wedge\ L(x\ b)](m)$
b) $\lambda x\ [\lambda y\ [L(x\ y) \vee\ H(y\ b)](m)](f)$
c) $\lambda x\ [\forall y\ M(y) \Rightarrow\ L(x\ y)](f)$

2) Convert the following formulas to equivalent lambda expressions by abstracting over the entity or predicate listed after them (ie. perform the opposite of beta reduction on them):

a) $M(m) \wedge\ L(mb)$ (abstract over 'm')
b) $\forall x\ M(x) \Rightarrow\ L(x\ b)$ (abstract over 'M')
c) $\exists x\ M(x) \wedge\ L(x\ b)$ (abstract over 'M' and 'b')

3) If we have a model M containing a set of entities E and the model contains the following set { x : x snore1} (ie. the set of entities which snore), then we can represent the same set as a function from entities to truth-values. Write this function as a lambda expression.

4) I defined love1 (a two-place predicate) as a function from entities to a function from entities to truth-values. This can be expressed as a lambda expression – λ x [λ y [love1(x y)]] – and, given a model, as two functions mapping from the entities in the model to more entities and finally a truth-value. Construct a model which includes denotations for like1 and give1. Then define these functions by exhaustively listing the possibilities in your model.

5) It would be nice to get rid of *it-is-not-the-case-that* from our English fragment (since it isn't English!) and replace it with *not* or *n't*. To do this we need to extend the fragment in several ways to cover examples like:

Kim doesn't snore
Kim doesn't love Sandy

We need an entry for *doesn't* and new syntactic rules for analysing VPs containing this word. Lets assume the following entry and rule:

doesn't : Aux : `P^[X^`[not,[P,X]]]

and the following PS rule:

VP $\rightarrow$ Aux VP : Aux$'(VP')$

see if you can work out how this works and check that you understand by adding the entry and rule to the grammar for F2 above and parsing these sentences.

10 Discourse Processing

The problem with the truth-conditional semantics plus theorem proving approach to language understanding is that we have no notion of the goals of the speaker/user, what speech acts s/he is attempting; for example, asserting, requesting, asking. The system will simply assume that all declarative sentences are assertions and all interrogative sentences are requests for information.

Exercises

Can you construct a situation / dialogue in which this would lead to a system misinterpreting some input?

Below I survey some ways of augmenting deduction (entailment) with techniques able to handle some of the phenomena described in the Intro to Linguistics handout.

10.1 Abductive Inference

A form of plausible inference which often looks like the 'structural opposite of MPP' and involves reasoning from consequent to antecedent, as in (48).

(48) a (if (drunk x) (not (walk-straight x)))
b (not (walk-straight john))
c (drunk john)

This is not deduction and the truth of the 'conclusion' c) is not guaranteed by the truth of the premises a) and b). – having a fever may also result in not walking straight. Abduction is often an inference about causation to find explanations for events. Various forms of abductive inference have been proposed as a way of recognising speaker intentions or goals and modelling the defeasible (i.e. default or plausible rather than guaranteed) nature of these inferences. For example, the discourse in (49a) can be interpreted using abductive inferences.

(49) a Kim can open the safe. He knows the combination.
b (can Kim1 (open safe1))
c (know he (combination-of x c))
d (if (can x state) (know x (cause (act x) state)))
e (if (combination-of x c) (cause (dial z x c) (open x)))
f (plausibly-if (and (know x p) (if p q)) (know x q))
g (know Kim1 (cause (act Kim1) (open safe1)))
h (know he (cause (dial z x c) (open x)))
i (know Kim1 (cause (dial Kim1 safe1 c) (open safe1)))

I have given a FOL-like representation (in which variables are implicitly universally quantified) of how to do this. b) and c) are the formulas associated with the two sentences (by parsing / semantic interpretation). d), e) and f) are background knowledge. g) is inferred from b) and d) by MPP / forward chaining. h) is inferred from c), e) and f), where e) is first 'embedded in' f) by replacing the propositional variables p and q by e). i) is derived by matching g) and h) and represents recognition of the elaboration relation, because 'dial' is more specific than 'act' and Kim1 more specific than 'he'. The details of the matching process are complex, and must involve more than first-order unification of (individual) variables. A 'side-effect' is that the antecedent of the pronoun is found, as is the link between *the combination* and safe1.

Exercises

Can you see any disadvantages to this technique as a general approach to dis-

course understanding? (hard)
see Blythe J., Hobbs, J. *et al.*, Implementing weighted abduction in Markov logic, Int. Wkshp on Computational Semantics, 2011
aclweb.org/anthology-new/W/W11/W11-0107.pdf
for a recent attempt to build a full discourse interpretation system based on abduction with weights inferred from data using machine learning.

11 Inadequacies of Formal Semantics

11.1 Intensionality

Embedded propositions cause problems for our semantics. For instance, *believe* takes a sentential complement, as in *Kim believes (that) Sandy loves Fred* which we might 'translate' into:

believe1(Kim1, love1(Sandy1 Fred1))

However, there is a problem with this analysis because it is quite possible for the two examples in (50) to correctly characterise Kim's beliefs.

(50) a Kim believes Sandy loves Fred
b Kim doesn't believe Miss UK loves Fred

If Sandy is Miss UK, then we must assume that Kim is unaware of this, otherwise he would modify his beliefs appropriately. The translation of *Miss UK* will be something like ∃ x miss-uk1(x) . . . , so in a world where Sandy is Miss UK, the referent of this NP and *Sandy* will be Sandy1 and we will end up saying that Kim both does and doesn't believe that Sandy1 loves Fred. The way out of this problem is to say that Kim believes the proposition *Miss UK loves Fred* and not its denotation (ie. its truth-value t/f) in the actual world. Or in other words, that Kim's belief is about the sense of the embedded sentence. This example shows then that not all meaning can be reduced to the extended notion of reference that we have called denotation.

One way of approaching this problem is to switch from an extensional logic whose model-theoretic interpretation concerns only actual states of affairs to an intensional logic whose model-theoretic semantics is characterised in terms of 'possible worlds'. In this approach, the extension of an expression can vary depending on which world we are considering, so in the actual world the extension of *Miss UK* may be Sandy1 but in another possible world it may be different. We will define the intension of an expression as the set of extensions that it has in every possible world. Names are usually treated as having the same extension in every possible world. Since the extensions of *Sandy* and *Miss UK* will differ between possible worlds, the intensions or propositions conveyed by these two embedded sentences will also differ. The intension of a proposition will be the set of truth-value assignments defined by evaluating its truth in every possible world. Thus, if Kim's beliefs are about the intensions of the

embedded sentences, they will be logical (rather than contradictory) but possibly inaccurate in the actual world. All verbs which take embedded sentences as arguments denote some relation between an individual and (the intension of) a proposition.

This analysis of embedded propositions predicts that examples such as *Kim believes Miss UK is not a feminist* should be ambiguous. It could convey a proposition telling us that Kim believes that Sandy (who she knows is Miss UK) is not a feminist or one telling us that Kim believes that whoever Miss UK is she is not a feminist. These two alternative logical forms can be notated as:

a) $\exists$ x Miss-uk1(x) $\wedge$ Believe1(Kim1 ^[$\neg$ Feminist(x)]
b) Believe1(Kim1 ^[$\exists$ x Miss-uk1(x) $\wedge$ $\neg$ Feminist(x)])

The 'hat' sign indicates that what is believed is the intension (rather than extension) of the formula in square brackets. If the existential variable is in the scope of the intension operator, we get the so-called *de dicto* reading where Kim believes that whoever Miss UK is, she is not a feminist. If the existential has wide scope, then we get the so-called *de re* reading where Kim believes a particular proposition about Sandy (or whoever happens to be the extension of *Miss UK* in the actual world).

However, one consequence of this analysis of *believe* is that if Kim believes any necessarily true propositions, then Kim must believe every necessarily true proposition. For example, if (51a) is true, then b) must be true (if we assume that *bachelor* just means 'unmarried man').

(51) a Kim believes it is raining or it isn't raining
b Kim believes all bachelors are unmarried men

However, this seems wrong because it is not a consequence of Kim's beliefs about rain that he believes things about bachelors. To see why we get into this bind, it is necessary to think about the denotation of a proposition in possible world semantics. The intension of a proposition is just a set of truth-value assignments derived by evaluating its truth in every possible world. However, every necessarily true proposition will be true in all worlds, so their intensions will be identical.

It seems that the notion of intension defined by possible world semantics is not adequate to capture the sense (as opposed to reference) of a linguistic expression. In any case, a direct interpretation of possible world semantics in a computational context would be difficult. (See Cann's book in references for more on possible worlds, intensionality, etc.)

Exercises

Can you see why? – How many possible worlds are there? How many facts are there in one world?

Instead we will ignore these problems and do theorem proving with the FOL formulas for these examples. This works (to some extent) because we are

keeping propositions distinct by utilising their syntactic form – believe1 (Kim1 love1(Kim1 Sandy1)) is structurally distinct from ∃ x miss-uk1(x) ∧ believe1(Kim1 (love1 Kim1 x)). Why is this not really adequate?

11.2 Word Meaning

So far we have said very little about word meaning. However, we have made a couple of assumptions in order to get our account of sentence meaning off the ground. The first assumption is that word meaning divides into a number of distinct senses (much as in a conventional dictionary). So we have written Snore1 as the translation of the first (main) sense of *snore*. In cases of lexical ambiguity, we can create further senses – Bank1, Bank2, etc. This approach is fine providing that we agree that word senses are really distinct in this way. However, many words exhibit polysemy; that is, they have a variety of closely related meanings which shade off into each other. Consider, for example, the meaning of *strike* in (52).

(52) a Kim struck Sandy
b Kim struck a match
c Kim struck a bargain
d Kim struck Sandy as ridiculous

Different dictionaries will make different decisions on how to 'divide up' the meaning of *strike* into distinct senses, underlining the rather arbitrary nature of this activity. Truth-conditional semantics offers no insights into this issue, so we will not dwell on it further. However, it is important to remember that when we discuss word meaning, we are discussing word sense meaning (or equivalently, the meaning of a predicate or lexical item).

The second assumption is central to the principle of compositionality and the account of semantic productivity which is incorporated into truth-conditional semantics. The contribution of word meaning to sentence meaning must take the form of an invariant contribution to the truth-conditions of each sentence. Otherwise, compositionality will be undermined, because the rules which combine the meanings of words, phrases and clauses to form sentence meanings do not modify the units which they combine. Of course, this does not prevent selection of an appropriate sense in a particular context, but it does imply that word meaning is not unpredictably altered by context. Some linguists argue that word meaning is partially determined by the context of occurrence – idioms and metaphors usually figure largely in this type of debate. For example, in (53) it is implausible to argue that productive semantic rules combining the meanings of the individual words produce the most likely interpretation.

(53) a Truth-conditional semantics is dead wood
b Kim is rapidly sliding into a moral cesspool

How serious a problem you think this is will depend a) on how common and central to language you feel idiomatic and metaphorical usage to be and b) on whether you believe a theory of idioms and metaphors can be derived from a theory of 'literal' meaning.

Formal semanticists largely ignored word meaning except to point out that in formulas we need to replace a word form or lemma by an appropriate word sense (usually denoted as bold face lemma prime, lemma-number, etc (*loved*, **love′** / **love1**). But we also need to know what follows from a word sense, and this is usually encoded (if at all) in terms of (FOL) meaning postulates:

(54) a $\forall x, y$ love′(x, y) $\rightarrow$ like′(x, y)
b $\forall x, y$ love′(x, y) $\rightarrow \neg$ hate′(x, y)
c $\neg\forall x, y$ desire′(x, y) $\rightarrow$ love′(x, y)

Although this is conceptually and representationally straightforward enough, there are at least three major issues:

1. How to get this information?
2. How to ensure it is consistent?
3. How to choose the right sense?

Johann Bos, perhaps the only major computational formal semanticist(!) as part of his development of Boxer (see section 13) solves 1) by pulling lexical facts from WordNet (nouns) and VerbNet – these are manually created databases (derived in part from dictionaries) which are certainly not complete and probably inconsistent. The information they contain is specific to senses of the words defined, so is only applicable in context to a word sense, so Bos simply assumes the most frequent sense (sense 1, given Wordnet) is appropriate. If the background theory built via WordNet/VerbNet is overall inconsistent, because the data is inconsistent, the algorithm for extracting relevant meaning postulates doesn't work perfectly, or a word sense is wrong, then the theorem prover cannot be used or will produce useless inferences.

There has been a lot of computational work on learning word meaning from text using distributional models of meaning (see Turney and Pantel, 2010 or Baroni *et al.* 2012a in references for a review) These models represent words by their contexts of occurrence using approaches which are extensions of techniques used from information retrieval and document clustering, where a document is represented as a bag-of-words and retrieved via keywords indexed to documents, a word-document matrix is reduced so that documents are clustered, or document similarity is computed using the distribution of words (or clustered 'topics' or latent semantic dimensions) in each document.

12 Generalized Categorial Grammars

12.1 Categorial Grammar

Classic (AB) categorial grammar consists of atomic categories of the form: N, NP, S, etc., and functor categories of the form S/N, (S\ NP)/NP, etc. constructed by combining atomic categories with slash and backslash with the functor leftmost and the 'outer' argument rightmost (see Wood, 1993 for a textbook introduction to CG and its generalisations).

Functors and arguments are combined by directional rules of (function-argument) application as in Figure 4 below. CGs of this form are weakly equivalent to CFGs but assign a fully binary branching structure. So ditransitive verb complements, whose categories will be ((S\ NP)/PP)/NP, will be assigned a different structure than in a standard CF PSG approach. Figure 3 shows the CG derivation for a simple example. One feature of CG is that syntax and semantics can be

Kim	loves	Sandy
NP	(S\NP)/NP	NP
kim'	λ y,x [love$'$(x y)]	sandy$'$
	---------- FA	
	S\NP	
	λ x [love$'$(x sandy$'$)]	
---------- BA		
S		
love$'$(kim$'$ sandy$'$)		

Figure 3: CG Derivation for *Kim loves Sandy*

more closely associated than in a standard 'rule-to-rule' framework as function application in the syntax can correspond to function application (beta reduction) in the lambda calculus (regardless of directionality). This framework is 'radically lexical' since now there are just two rules of syntactic combination (FA,BA) and one rule of semantic application. Everything else must be captured in terms of the lexical categories. For example, modifiers cannot be dealt with in terms of separate rules and instead must be characterised lexically as functor arguments which yield categories of the same type (X/X, X\X) e.g. N/N or (S\ NP)/(S\NP) – can you see what classes of word these categories would be appropriate for?

12.2 Generalized Categorial Grammar

The main interest in exploring CGs is that various extensions of classic AB CG (with just function application) have been proposed in recent years. These deal well with phenomena like non-constituent coordination and mostly extend the generative capacity of the grammar to 'mild context-sensitivity' / indexed

Forward Application:	
X/Y Y ⇒ X	λ y [X(y)] (y) ⇒ X(y)
Backward Application:	
Y X\Y ⇒ X	λ y [X(y)] (y) ⇒ X(y)
Forward Composition:	
X/Y Y/Z ⇒ X/Z	λ y [X(y)] λ z [Y(z)] ⇒ λ z [X(Y(z))]
Backward Composition:	
Y\Z X\Y ⇒ X\Z	λ z [Y(z)] λ y [X(y)] ⇒ λ z [X(Y(z))]
(Generalized Weak) Permutation:	
$(X\|Y_1)\ldots\|Y_n \Rightarrow (X\|Y_n)\|Y_1\ldots$	$\lambda\ y_n\ldots,y_1\ [X(y_1\ldots,y_n)] \Rightarrow \lambda\ y_1,y_n\ldots\ [X(y_1\ldots,y_n)]$
Type Raising:	
(X ⇒ T/(T\X)	a ⇒ λ T [T a]
(X ⇒ T\(T/X)	a ⇒ λ T [T a]

Figure 4: GCG Rule Schemata

languages. The specific extension I will outline adds rules of composition, permutation and type raising to AB CG as in Figure 4. These license derivations involving non-standard constituency such as Figure 5. Each of the rule schema come with a corresponding semantic operation defined in terms of the lambda calculus, illustrated in the sample derivations. What semantic type is being associated with NPs in the derivations shown below? How does typing work in this framework (i.e. given the X,Y and T labels in the rule schemata, how do we know what types are being combined in specific derivations and that these types are compatible)? Neither function composition nor permutation change the semantics associated with a given sentence, rather they introduce 'spurious' ambiguity in that they allow the same semantics to be recovered in different ways. This can be exploited to deal with non-constituent coordination (Figure 7), unbounded dependencies (Figure 6), and the relationship between intonation, focus and semantics. (See Wood, 1993 or Steedman, 1996, 2000, 2012 for fuller treatments of closely related approaches. CCG is like my GCG, but without permutation.)

There are polynomial parsing algorithms (n^6) for some types of generalized CGs of this form (so long as rules such as type raising are constrained to apply finitely. Because of the 'spurious' ambiguity of GCGs some effort has been devoted to defining parsing algorithms which only find a single derivation in the equivalence class of derivations defining the same logical form. Steedman (esp. 2000) argues

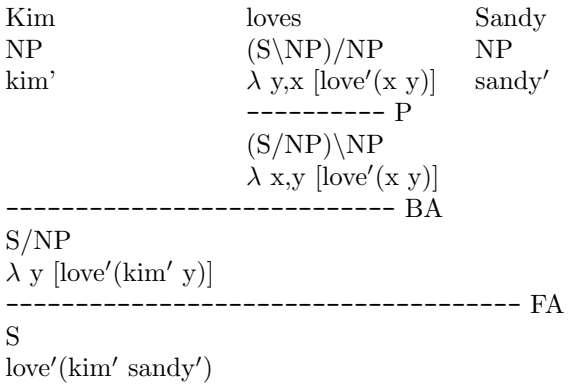


Figure 5: GCG Derivation for *Kim loves Sandy*

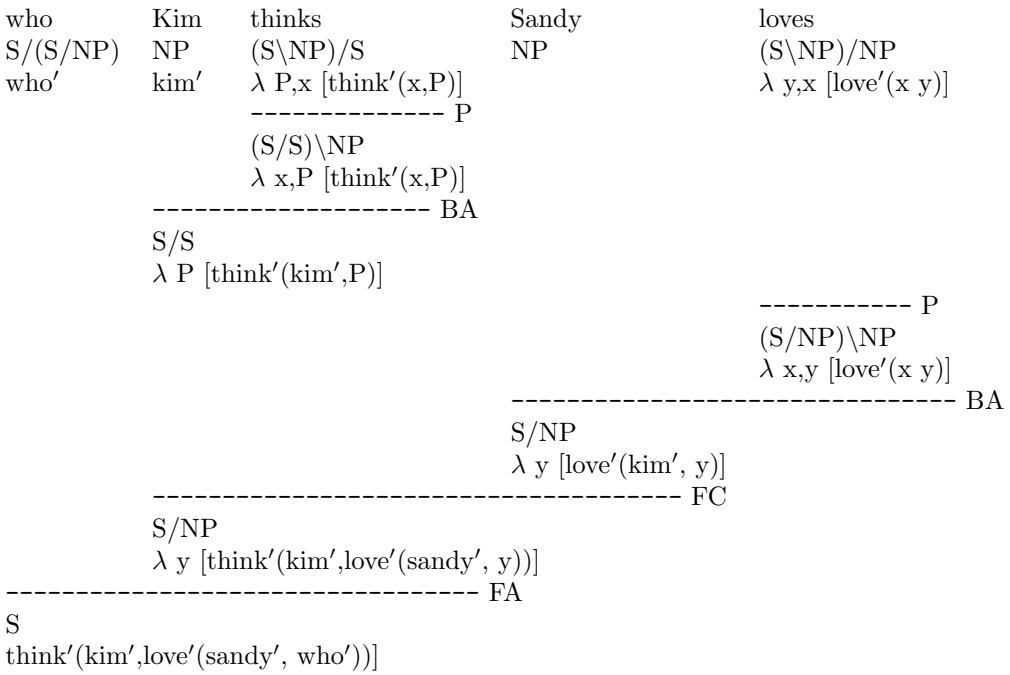


Figure 6: GCG Derivation for *who Kim thinks Sandy loves*

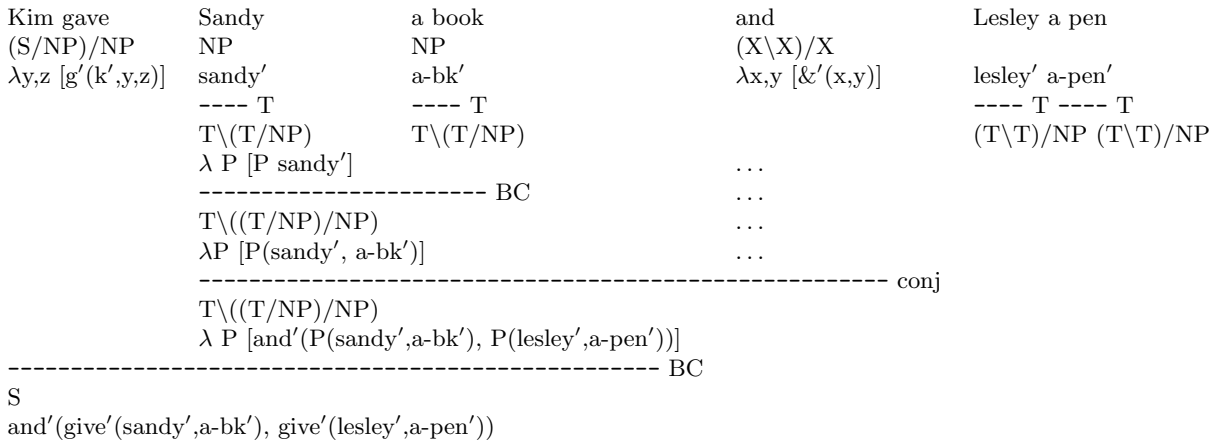


Figure 7: GCG Derivation for *Kim gave Sandy a book and Lesley a pen*

instead that the ambiguity is not spurious at all but rather correlates with different prosodies conveying different information structure (give-new, theme-rheme, focus – see Discourse Processing (R216) module or a linguistics textbook from the Intro to Linguistics handout reading list).

Bayer (1996) draws on another tradition in GCG research which emphasises the connection between CG and substructural or resource logics (see e.g. Carpenter, 1997; Morrill, 1994). This tradition has concentrated on demonstrating that the rules of GCGs can be shown to implement sound deductive systems, rather than on implementation via unification operations. Thus the slash operator is a form of (linear) implication: from X/Y, infer an X given a Y.

From this perspective, it makes sense to introduce rules (of inference) like $\wedge$-elimination (AE): given X $\wedge$ Y, infer Y, and $\vee$-introduction (OI): given Y, infer X $\vee$ Y. Bayer defines his GCG category set as the closure of the atomic category under the operators: / \, $\wedge$, and $\vee$. He assigns *and* the usual polymorphic category (X\X)/X and *be* the category (S\NP)/(NP $\vee$ AP). This along with the rule of OI is enough to license coordinations of unlike categories when the verb allows different complement types, as in Fig 8 This approach can be generalised to featural mismatches 'within' the same cateogory simply by allowing disjunctive feature values and aplying OI and AE to these values (e.g. CASE: **acc** $\vee$ **dat**) (feature neutralisation in German).

Although the use of unrestricted disjunction operators with complex unification-based feature systems is known to lead to computational inefficiency, if not intractability, it is not clear that this move would be so problematic in the context of the 'logicist' approach to GCG, as the features would be restricted to finite-valued morphosyntactic ones.

The sample derivations above illustrate (succinctly!) how GCG/CCG can handle constructions such as unbounded dependencies and non-constituent coordi-

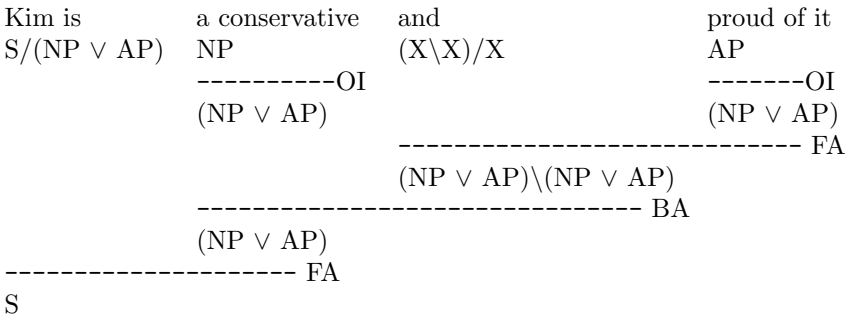


Figure 8: GCG Derivation for *Kim is a conservative and proud of it*

nation, both syntactically and semantically, which would be problematic for a CFG + LC approach like that introduced in section 9 above.

12.3 Exercises

1. Work out another derivation for *Kim loves Sandy* to that shown in Fig 5 and show that it yields the same semantic interpretation

2. Suppose we assign NPs semantic type `<<e t> t>` so that we can handle quantifiers properly, but continue to assign verbs types like `<e t>`, `<e <e t>>` etc. Can you show that the derivation of the interpretation of *Kim snores* using the syntactic and semantic definition of BA given in Figure 4 still works using appropriate lambda expressions for the NP and V(P) (i.e. S\NP)?

3. Work out the interpretation for *Every man snores* by assigning an appropriate semantic type and lambda expression to *every*.

4. There are two derivations for *Every man loves some woman*, the more complex involving P, BA and then FA. Can you see how to make this second derivation less 'spurious' by building the more marked interpretation in which *some* outscopes *every* using this derivation?

5. For F2 (sections 8), we argued that *every* and *some* should be treated like the quantifiers ∀ and ∃ in FOL. We could plausibly do the same for *each*, *all*, *a* an *one*, but what about other 'quantifiers' like *most* or *more than one third (of)*? Read about Generalized Quantifiers in the Blackburn and Bos references, or on the web in Wikipedia or the Stanford Encyclopedia of Philosophy, and see if you can work out how to assign a formula and interpretation to *Most men snore* (harder).

13 (Neo-)Davidsonian Semantics

We've seen that some quite simple constructions (e.g. those involving adjectives or adverbs) create problems for FOL representations of natural language semantics. The obvious interpretation of an adverb is that it modifies a verbal predicate or proposition, but this isn't possible in FOL. We've extended FOL with LC but so far only used LC as a means to compositionally construct FOL semantics for sentences in a syntax-guided fashion. We want to avoid higher-order logics such as modal, intensional or possible world semantics in computational semantics to keep theorem proving / inference tractable. One way to do so is to reify events (and worlds) in FOL:

(55) a Kim kissed Sandy passionately
b passionate1(kiss1(kim1, sandy1))
c $\exists$ e kiss1(e, kim1, sandy1) $\wedge$ passionate1(e)

(56) a Possibly Kim kissed Sandy
b possible1(kiss1(kim1, sandy1))
c $\exists$ e kiss1(e, kim1, sandy1)) $\wedge$ possible1(e)

Davidson was the first to suggest that we could replace the b) semantics with the c) semantics by reifying events, i.e. including event individuals/entities in the corresponding FOL model. We will write them, e, $e1$, etc to indicate that events are of a different *sort* to other entities. A *sort* being just like a *type* but applying only to the space of individuals/entities in the FOL model. States like *Kim weighs too much* are also reified, so some prefer to talk about 'eventualities' rather than events. Actually, this move doesn't quite work for *possibly* because there is a difference in meaning between b) and c) below – can you see it?

(57) a Possibly every unicorn is white
b possible1($\forall$ x unicorn1(x) $\rightarrow$ white1(x))
c ($\forall$ x unicorn1(x) $\rightarrow$ possible1(white1(x)))

(The problem is very similar to that discussed for propositional attitude verbs and related to the *de re /de dicto*, sense vs. reference distinctions – see section 11.1).

Parsons took this a stage further by proposing that arguments to predicates become binary relations between event variables and entities:

(58) a Kim kissed Sandy passionately
b $\exists$ e kiss1(e) $\wedge$ agent(e,kim1) $\wedge$ patient(e,sandy1) $\wedge$ passionate1(e)
c $\exists$ e kiss1(e) $\wedge$ arg1(e,kim1) $\wedge$ arg2(e,sandy1) $\wedge$ passionate1(e)

The problem with relations like 'agent' and 'patient' is determining exactly what they entail which is constant across all verbs (e.g. is kim1 the agent in *Kim enjoys films*?), so we generally prefer to use more semantically-neutral relations, as in c). The advantage of this neo-Davidsonian, Parsons-style representation is that it makes it easy to handle argument optionality. For example, the nominalization of (58i)s:

(59) a Kim's / The kissing of Sandy (was passionate).
b ∃ e kiss1(e) ∧ arg1(e,kim1) ∧ arg2(e,sandy1) ∧ passionate1(e)
c ∃ e kiss1(e) ∧ arg2(e,sandy1) ∧ passionate1(e)

and we don't have to specify the agent, so c) is a reasonable semantics for this case. Some other advantages of this representation are that we can handle tense more naturally, and PP adjectival and adverbial modifiers look more similar:

(60) a ∃ e kiss1(e) ∧ arg1(e,kim1) ∧ arg2(e,sandy1) ∧ passionate1(e) ∧ past(e)
b ∃ e,x kiss1(e) ∧ arg2(e,sandy1) ∧ passionate1(e) ∧ past(e) ∧ in1(e,x) ∧ bar(x)
c ∃ e,x kiss1(e) ∧ arg1(e,kim1) ∧ arg2(e,y) ∧ passionate1(e) ∧ past(e) ∧ in1(y,x) ∧ bar(x) ∧ person1(y)

13.1 Exercises

1. Can you provide English sentences that match the semantics of the examples in (60?) Can you work out how to build these representations compositionally for sentences using CCG + LC? (hard)

2. Assign an event-based semantics to the following examples:

   (61) a Most men snore loudly or snuffle.
   b Each man (who) I met snored.
   c Kim's examination was hard.
   d Few students at Cambridge think they are geniuses.

3. Suppose we want to capture the semantics of *probably* or *may* properly using a similar approach. Can you work out how to reify worlds in FOL and give a semantics to these words in sentences like *Kim may snore* or *Probably Kim snores*? (See Blackburn & Bos book/papers or http://plato.stanford.edu/entries/possible-worlds/semantics-extensionality.html)

13.2 Wide-coverage Event-based Semantics for CCG

Bos *et al.* (2004) show how to derive FOL neo-Davidsonian representations from CCG derivations using the lambda calculus by assigning lambda functions to complex CCG categories (e.g. (S\NP)/NP λ P,y,x [P(x y)]) and defining decomposed function application schemes to associate with combinatory and type changing rules. (The decomposition operator, (@), circumvents some of the complexities of using the lambda calculus for syntax-directed semantic composition.). The paper is the first to show that it is possible to derive a logical semantic representation compositionally from a wide-coverage state-of-the-art parser applied to real data, and to evaluate the well-formedness of the representions produced. However, the resulting semantics doesn't handle scope underspecification or integrate with generalized quantifiers, it introduces argument relations like *agent* and *patient* which lack a coherent semantics, and it doesn't handle 'construction-specific semantics' (e.g. a noun compound such as *steel warehouse* can mean warehouse *for* steel or warehouse *of* steel, so the N/N + N forward application (FA) rule needs to be sensitive to whether it is forming a compound or combining an adjective and noun, because for the compound an adequate semantics will introduce an additional underspecified relation: steel(x) $\wedge$ warehouse(y) $\wedge$ R(x,y)).

13.3 Boxer

Bos (2005, 2008) has developed the approach to obtaining a wide-coverage FOL semantics from CCG to support reasoning. Firstly, he uses (underspecified) Discourse Representation Theory, (u)DRT, as his semantic representation. This is a neo-Davidsonian FOL variant historically developed to handle anaphora better, rather than to support underspecification of sets of well-formed FOL formulas; e.g. in (62a) and (62b), the pronouns function semantically like bound variables within the scope of *every* and *a*:

(62) a Every farmer who owns a donkey beats it.
b Every farmer owns a donkey. He beats it.
c every(x, farmer(x), some(y, donkey(y), own(x y), beat(x y)))

That is the (simplified) semantics of these examples is captured by (62c). For (62b) it is fairly easy to see that syntax-guided translation of sentences into FOL will lead to problems as the translation of the first sentence will 'close off' the scope of the quantifiers before the pronouns are reached. Something similar happens in (62a), at least in classical Montague-style semantics (as in Cann's *Formal Semantics* book). Bos & Blackburn (2004, *Working with DRT*) discuss (u)DRT and pronouns in detail.

Although, uDRT provides a technical solution that allows predications to be inserted into an implictly conjunctive semantic representation within the scope

of quantifiers, this doesn't really solve the problem of choosing the right antecedent for a pronoun. So Bos (2008) extends Boxer with a simple anaphora resolution system and Bos (2005) extends it with meaning postulates for lexical entailments derived from WordNet (see section 11.2).

At this point, Boxer is able to output a resolved semantics for quite a large fragment of English. This can (often) be converted to FOL / Horn Clauses and fed to a theorem prover to perform inference, and to a (minimal) model builder to check for consistency between meaning postulates and Boxer's output. Bos' papers give examples of inferences that are supported by the system and discuss where the system makes mistakes. The inferences mostly involve comparatively simple hyponymy or synonymy relations and the mistakes mostly involve discourse interpretation (pronouns, presuppositions). The off-the-shelf theorem proving technology that he uses also means that generalized quantifiers can't be handled unless they translate into FOL quantifiers. Nevertheless, the coverage of real data is unprecedented and impressive.

13.3.1 Exercise

If you want to explore what Boxer can and can't do, try running some examples through the on-line demo. You can use the sentences from previous exercises and examples from this handout, if you need inspiration for things to try out (e.g. the 'donkey' sentences and variants like *Every farmer who owns a donkey beats it. It hurts.*

Boxer: http://svn.ask.it.usyd.edu.au/trac/candc/wiki/boxer
(Including on-line demo, and see references below)

14 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 performance will require better treebanks (or grammars) and thus better representations of the task across a variety of languages as well as better (deep?) machine learning.

The best way (so far) to do semantics is to specify the 'translations' of sentences into a logic with a model theory and proof theory. We have considered the semantics of a small fragment of English (which we have only dealt with partially, ignoring e.g. tense/aspect). We can construct these translations compositionally so that the meaning of a sentence is a function of the meaning of its constituents. We can implement such a compositional semantics for CFG or CCG (possibly extended with unification / syntactic features).

Tractable computation in this framework requires theorem proving but extend-

ing the approach to interpreting even simple discourses requires more than deduction – perhaps something like weighted abduction or some other more data-driven, statistical approach. Similarly dealing with word meaning by manually specifying meaning postulates looks like a Sysphean (impossible) task, so more data-driven approaches that infer similarity from text will be useful. The big prize will come with integration word embeddings into a more logical framework capable of modelling entailment as well a plausible inference.

15 Supplementary and Background Reading

Syntax / Parsing

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

Artificial Intelligence: A Modern Approach, S. Russell and P. Norvig, Pearson, 3rd ed., 2010/2017 (esp. ch. 23).

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

Computational Semantics

Blackburn and Bos *Representation and Inference for Natural Language* and *Working with Discourse Representation Theory* (see `http://www.let.rug.nl/bos/comsem/`.

Parsons, T., *Events in the Semantics of English*, MIT Press, 1990

Word Meaning

Turney P. & P. Pantel, From frequency to meaning: vector space models of semantics, JAIR, 37, 141–188, 2010

arxiv.org/pdf/1003/1141

Baroni, M., Bernardi, Zamparelli, Frege in Space: A program for compositional distributional semantics, 2012

clic.cimec.unitn.it/composes/materials/frege-in-space.pdf

CCG / GCG References

Bayer, S. 'The coordination of unlike categories', *Language* 72.3, 579–616, 1996.

Carpenter, R. *Type-Logical Semantics*, MIT Press, 1997.

Morrill, G. *Type-logical Grammar*, Kluwer, 1994.

Steedman, M. *Surface Structure and Interpretation*, MIT Press, 1996.

Steedman, M. *The Syntactic Process*, MIT Press, 2000.

Steedman, M. *Taking Scope*, MIT Press, 2012.

Wood, M. *Categorial Grammar*, Routledge, 1993

Boxer References

Wide-Coverage Semantic Representations from a CCG Parser. Johan Bos, Stephen Clark, Mark Steedman, James R. Curran and Julia Hockenmaier. Proceedings of COLING-04, pp.1240-1246, Geneva, Switzerland, 2004.
http://www.aclweb.org/anthology/C/C04/C04-1180.pdf
Wide Coverage Semantic Analysis with Boxer, Johan Bos, 2008, STEP
Towards Wide Coverage Semantic Interpretation, Johan Bos, 2005, IWCS-6
http://www.let.rug.nl/bos/publications.html