Notes on computational linguistics

E. Stabler

UCLA, Winter 2003
(under revision)

Contents

Linguistics 185a/209a: Computational linguistics I

Lecture 12-2TR in Bunche 3170 Prof. Ed Stabler
Office: Campbell 3103F Office Hours: 2-3T, by appt, or stop by
x50634 stabler@ucla.edu

TA: Ying Lin Discussion: TBA

Prerequisites: Linguistics 180/208, Linguistics 120b, 165b

Contents: What kind of computational device could use a system like a human language? This class will explore the computational properties of devices that could compute morphological and synactic analyses, and recognize semantic entailment relations among sentences. Among other things, we will explore

(1) how to define a range of grammatical analyses in grammars G that are expressive enough for human languages

(2) how to calculate whether a sequence of gestures, sounds, or characters $s \in L(G)$ (various ways!)

(3) how to calculate and represent the structures d of expressions $s \in L(G)$ (various ways!)
(importantly, we see that $size(d) < size(s)$, for natural size measures)

(4) how to calculate morpheme sequences from standard written (or spoken) text

(5) how to calculate entailment relations among structures

(6) how phonological/orthographic, syntactic, semantic analyses can be integrated

(7) depending on time and interest, maybe some special topics:
- how to distribute probability measures over (the possibly infinitely many) structures of $L(G)$, and how to calculate the most probable structure d of ambiguous $s \in L(G)$
- how to handle a language that is "open-ended:" new words, new constructions all the time
- how to handle various kinds of context-dependence in the inference system
- how to handle temporal relations in the language and in inference
- how to calculate certain "discourse" relations
- tools for studying large collections of texts

Readings: course notes distributed during the quarter from the class web page, supplemented occasionally with selected readings from other sources.

Requirements and grades: Grades will be based entirely on problem sets given on a regular basis (roughly weekly) throughout the quarter. Some of these problem sets will be Prolog programming exercises; some will be exercises in formal grammar. Some will be challenging, others will be easy.
Graduate students are expected to do the problem sets and an additional squib on a short term project or study.

Computing Resources: We will use SWI Prolog, which is small and available for free for MSWindows, Linux/Unix, and MacOSX from `http://www.swi-prolog.org/`
Tree display software will be based on tcl/tk, which is available for free from `http://www.scriptics.com/`

The best models of human language processing are based on the programmatic hypothesis that human language processes are (at least, in large part) computational. That is, the hypothesis is that understanding or producing a coherent utterance typically involves changes of neural state that can be regarded as a calculation, as the steps in some kind of derivation.

We could try to understand what is going on by attempting to map out the neural responses to linguistic stimulation, as has been done for example in the visual system of the frog (Lettvin et al., 1959, e.g.). Unfortunately, the careful in vitro single cell recording that is required for this kind of investigation of human neural activity is impractical and unethical (except perhaps in some unusual cases where surgery is happening anyway, as in the studies of Ojemann?).

Another way to study language use is to consider how human language processing problems could possibly be solved by any sort of system. Designing and even building computational systems with properties similar to the human language user not only avoids the ethical issues (the devices we build appear to be much too simple for any kind of "robot rights" to kick in), but also, it allows us to begin with systems that are simplified in various respects. That this is an appropriate initial focus will be seen from the fact that many problems are quite clear and difficult well before we get to any of the subtle nuances of human language use.

So these lecture notes briefly review some of the basic work on how human language processing problems could possibly be solved by any sort of system, rather than trying to model in detail the resources that humans have available for language processing. Roughly, the problems we would like to understand include these:

perception: given an utterance, compute its meaning(s), in context. This involves recognition of syntactic properties (subject, verb, object), semantic properties (e.g. entailment relations, in context), and pragmatic properties (assertion, question,...).

production: given some (perhaps only vaguely) intended syntactic, semantic, and pragmatic properties, create an utterance that has them.

acquisition: given some experience in a community of language users, compute a representation of the language that is similar enough to others that perception/production is reliably consistent across speakers

Note that the main focus of this text is "computational linguistics" in this rather scientific sense, as opposed to "natural language processing" in the sense of building commercially viable tools for language analysis or information retrieval, or "corpus linguistics" in the sense of studying the properties of collections of texts with available tools. Computational linguistics overlaps to some extent with these other interests, but the goals here are really quite different.

The notes are very significantly changed from earlier versions, and so the contributions of the class participants were enormously valuable. Thanks especially to Dan Albro, Leston Buell, Heidi Fleischhacker, Alexander Kaiser, Greg Kobele, Alex MacBride, and Jason Riggle. Ed Keenan provided many helpful suggestions and inspiration during this work.

No doubt, many typographical errors and infelicities of other sorts remain. I hope to continue revising and improving these notes, so comments are welcome!

stabler@ucla.edu

1 Setting the stage: logics, prolog, theories

1.1 Summary

(1) We will use the programming language *prolog* to describe our language processing methods.

Prolog is a logic.

(2) We propose, following Montague and many others:

Each human language is a logic.

(3) We also propose:

a. Standard sentence recognizers can be naturally represented as logics.
(A "recognizer" is something that tells whether an input is a sentence or not.
Abstracting away from the "control" aspect of the problem, we see a recognizer as taking the input as an axiom, and deducing the category of the input (if any).
We can implement the deduction relation in this logic in the logic of prolog. Then prolog acts as a "metalanguage" for calculating proofs in the "object language" of the grammar.)

b. Standard sentence parsers can be naturally represented as logics.
(A "parser" is something that outputs a structural representation for each input that is a sentence, and otherwise it tells us the input is not a sentence.
As a logic, we see a parser as taking the input as an axiom from which it deduces the structure of the input (if any).)

All of the standard language processing methods can be properly understood from this very simple perspective.[1]

(4) What is a logic? A logic has three parts:

i. a language (a set of expressions) that has

ii. a "derives" relation $\vdash$ defined for it (a syntactic relation on expressions), and

iii. a semantics: expressions of the language have meanings.

a. The meaning of an expression is usually specified with a "model" that contains a semantic valuation fuction that is often written with double brackets. So instead of writing

semantic_value(`socrates_is_mortal`)=true

we write

$[\![$`socrates_is_mortal`$]\!] = 1$

b. Once the meanings are given, we can usually define an "entails" relation $\models$, so that for any set of expressions Γ and any expression A, $\Gamma \models A$ means that every model that makes all sentences in Γ true also makes A true.
And we expect the derives relation $\vdash$ should correspond to the $\models$ relation in some way: for example, the logic might be sound and complete in the sense that, given any set of axioms Γ we might be able to derive all and only the expressions that are entailed by Γ.

So a logic has three parts: it's (i) a language, with (ii) a derives relation $\vdash$, and with (iii) meanings.

[1] Cf. Shieber, Schabes, and Pereira (1993), Sikkel and Nijholt (1997). Recent work develops this perspective in the light of resource logical treatments of language (Moortgat, 1996, for example), and seems to be leading towards a deeper and more principled understanding of parsing and other linguistic processes. More on these ideas later.

(5) **Notation: Sequences** are written in various ways:

$$abc$$
$$\langle a, b, c \rangle$$
$$a, b, c$$
$$[\mathsf{a}, \mathsf{b}, \mathsf{c}]$$

The programming language prolog requires the last format; otherwise, I try to choose the notation to minimize confusion.

Similarly, the empty sequence is sometimes represented ϵ, but the prolog notation is [].

A **stack** is a sequence too, but with limitations on how we can access its elements: elements can only be read or written on the "top" of the sequence. We adopt the convention that the top of a stack is on the left, it is the "front" of the sequence.

(6) **Notation:** Context free grammars are commonly written in the familiar rewrite notation, which we will use extensively in these notes:

S → NP VP	
NP → D N	VP → V NP
NP → N	VP → V
N → students	V → sang
N → songs	V → knew
D → some	
D → all	

These grammars are sometimes written in the more succinct Backus-Naur Form (BNF) notation:

S ::= NP VP	
NP ::= D N | N	VP ::= V NP | V
N ::= students | songs	V ::= sang | knew
D ::= some | all	

The categories on the left side of the `::=` are expanded as indicated on the right, where the vertical bar separates alternative expansions. (Sometimes in BNF, angle brackets or italics are used to distinguish category from terminal symbols, rather than the capitalization that we have used here.) This kind of BNF notation is often used by logicians, and we will use it in the following chapter.

1.2 Propositional prolog

The programming language *prolog* is based on a theorem prover for a subset of first order logic. A pure prolog "program" is a *theory*, that is, a finite set of sentences. An execution of the program is an attempt to prove some theorem from the theory. (Sometimes we introduce "impurities" to do things like produce outputs.) I prefer to introduce prolog from this pure perspective, and introduce the respects in which it acts like a programming language later.

(Notation) Let $\{a-z\} = \{a, b, c, \ldots, z\}$
Let $\{a-zA-Z0-9\_\} = \{a, b, c, \ldots, z, A, B, \ldots, Z, 0, 1, \ldots, 9, \_\}$
For any set S, let S^* be the set of all strings of elements of S.
For any set S, let S^+ be the set of all non-empty strings of elements of S.
For any sets S, T, let ST be the set of all strings st for $s \in S, t \in T$.

language:

atomic formulas	$p = \{a-z\}\{a-zA-Z0-9\_\}^* \mid' \{a-zA-Z0-9\_\ \ @\#\$\%*()\}^{*'}$
conjunctions	$C ::= \epsilon. \mid p, C$
goals	$G ::= ?\text{-}C$
definite clauses	$D ::= p\text{:-}C$

(Notation) Definite clauses $p\text{:-}q_1, \ldots, q_n, \epsilon.$ are written $p\text{:-}q_1, \ldots, q_n$.
And definite clauses $p\text{:-}\epsilon.$ are written p.
The consequent p of a definite clause is the **head**, the antecedent is the **body**.

(Notation) The goal $?\text{-}\epsilon.$ is written $\square$. This is the contradiction.

(Notation) Parentheses can be added: $((p\text{:-}q)).$ is just the same as $p\text{:-}q$.

inference: $G, \Gamma \vdash G$ [axiom] for any set of definite clauses Γ and any goal G

$$\frac{G, \Gamma \vdash (?\text{-}p, C)}{G, \Gamma \vdash (?\text{-}q_1, \ldots, q_n, C)} \quad \text{if } (p\text{:-}q_1, \ldots, q_n) \in \Gamma$$

semantics: a model $\mathcal{M} = \langle 2, [\![\cdot]\!] \rangle$ where $2 = \{0, 1\}$ and $[\![\cdot]\!]$ is a valuation of atomic formulas that extends compositionally to the whole language:

$[\![\text{p}]\!] \in 2$, for atomic formulas p
$[\![\text{A, B}]\!] = \min\{[\![\text{A}]\!], [\![\text{B}]\!]\}$
$[\![\text{B:-A}]\!] = \begin{cases} 1 & \text{if } [\![\text{A}]\!] \leq [\![\text{B}]\!] \\ 0 & \text{otherwise} \end{cases}$
$[\![?\text{-A}]\!] = 1 - [\![\text{A}]\!]$
$[\![\epsilon]\!] = 1$

metatheory: For any goals G, A and any definite clause theory Γ,

Soundness: $G, \Gamma \vdash A$ only if $G, \Gamma \vDash A$,

Completeness: $G, \Gamma \vdash A$ if $G, \Gamma \vDash A$

So we can establish whether C follows from Γ with a "reductio" argument by **deciding:** $(?\text{-}C, \Gamma \vdash ?\text{-}\epsilon.)$

(Terminology) A problem is **decidable** iff there is an algorithm which computes the answer.

decidable:

(For arbitrary string s and CFG G, $s \in L(G)$)
(For formulas F, G of propositional logic, $F \vdash G$)
(For conjunction C def.clauses Γ of propositional prolog, $(?\text{-}C, \Gamma \vdash ?\text{-}\epsilon.)$

undecidable:

(For arbitrary program P, P halts)
(For formulas F, G of first order predicate logic, $F \vdash G$)
(For conjunction C def.clauses Γ of predicate prolog prolog, $(?\text{-}C, \Gamma \vdash ?\text{-}\epsilon.)$

Warning: many problems we want to decide are undecidable,
and many of the decidable ones are intractable.

This is one of the things that makes computational linguistics important: it is often not at all clear how to characterize what people are doing in a way that makes sense of the fact that they actually succeed in doing it!

(7) A Prolog theory is a sequence of definite clauses, clauses of the form p or p:-$q_1, \ldots q_n$, for $n \geq 0$.
A definite clause says something definite and positive. No definite clause let's you say something like

a. Either Socrates is mortal or Socrates is not mortal.

Nor can a definite clause say anything like

b. X is even or X is odd if X is an integer.

Disjunctions of the sort we see here are not definite, and there is no way to express them with definite clauses. There is another kind of disjunction that can be expressed though. We *can* express the proposition

c. Socrates is human if either Socrates is a man or Socrates is a woman.

This last proposition can be expressed in definite clauses because it says exactly the same thing as the two definite propositions:

d. Socrates is human if Socrates is a man

e. Socrates is human if Socrates is a woman.

So the set of these two definite propositions expresses the same thing as c. Notice that no set of definite claims expresses the same thing as a, or the same thing as b.

Prolog proof method: depth-first, backtracking. In applications of the inference rule, it can happen that more than one axiom can be used. When this happens, prolog chooses the first one first, and then tries to complete the proof with the result.
If the proof fails, prolog will back up to the most recent choice, and try the next option, and so on.

Given sequence of definite clauses Γ and goal $G = RHS$
if $(RHS = ?\text{-}p, C)$
  if (there is any untried clause $(p :- q_1, \ldots, q_n) \in \Gamma$)
    choose the first and set $RHS = ?\text{-}q_1, \ldots, q_n, C$
  else if (any choice was made earlier)
    go back to most recent choice
  else fail
else succeed

(8) **Pitfall 1: Prolog's proof method is not an algorithm, hence not a decision method.** This is the case because the search for a proof can fail to terminate. There are cases where $G, \Gamma \vdash A$ and $G, \Gamma \models A$, but prolog will not find the proof because it gets caught in "infinite recursion."

Infinite recursion can occur when, in the course of proving some goal, we reach a point where we are attempting to establish that same goal. Consider how prolog would try to prove p given the axioms:

```
p :- p.
p.
```

Prolog will use the first axiom first, each time it tries to prove p, and this procedure will never terminate.

We have the same problem with

```
p :- p, q.
p.
q.
```

And also with

```
p :- q, p.
p.
q.
```

This problem is sometimes called the "left recursion" problem, but these examples show that the problem results whenever a proof of some goal involves proving that same goal. We will consider this problem more carefully when it arises in parsing.

Prolog was designed this way for these simple practical reasons:

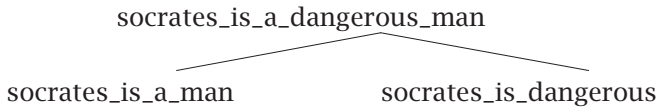
(i) it is fairly easy to choose problems for which Prolog does terminate, and

(ii) the method described above allows very fast execution!

(9) **Example:** Let Γ be the following sequence of definite clauses:

```
socrates_is_a_man.
socrates_is_dangerous.

socrates_is_mortal :- socrates_is_a_man.

socrates_is_a_dangerous_man :-
        socrates_is_a_man,
        socrates_is_dangerous.
```

Clearly, we can prove `socrates_is_mortal` and `socrates_is_a_dangerous_man`.
The proof can be depicted with a tree like this:



(10) **Example:** A context free grammar

$$G = \langle \Sigma, N, \rightarrow, \mathsf{S} \rangle,$$

where

1. Σ, N are finite, nonempty sets,
2. S is some symbol in N,
3. the binary relation $(\rightarrow) \subseteq N \times (\Sigma \cup N)^*$ is also finite (i.e. it has finitely many pairs),

For example,

ip → dp i1	i1 → i0 vp	i0 → will
dp → d1	d1 → d0 np	d0 → the
np → n1	n1 → n0	n0 → idea
	n1 → n0 cp	
vp → v1	v1 → v0	v0 → suffice
cp → c1	c1 → c0 ip	c0 → that

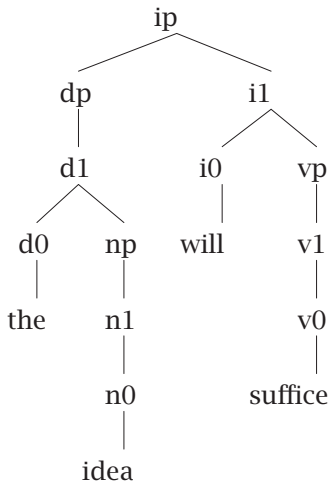
Intuitively, if ip is to be read as "there is an ip," and similarly for the other categories, then the rewrite arrow cannot be interpreted as implies, since there are alternative derivations. That is, the rules (n1 → n0) and (n1 → n0 cp) signify that a given constituent can be expanded either one way or the other.

In fact, we get an appropriate logical reading of the grammar if we treat the rewrite arrow as meaning "if." With that reading, we can also express the grammar as a prolog theory.

```
/*
* file:  th2.pl
*/
   ip :- dp, i1.      i1 :- i0, vp.      i0 :- will.        will.
   dp :- d1.          d1 :- d0, np.      d0 :- the.         the.
   np :- n1.          n1 :- n0.          n0 :- idea.        idea.
                      n1 :- n0, cp.
   vp :- v1.          v1 :- v0.          v0 :- suffice.     suffice.
   cp :- c1.          c1 :- c0, ip.      c0 :- that.        that.
```

In this theory, the proposition `idea` can be read as saying that this word is in the language, and `ip :- dp, i1` says that `ip` is in the language if `dp` and `i1` are. The proposition `ip` follows from this theory. After loading this set of axioms, we can prove `?- ip`. Finding a proof corresponds exactly to finding a derivation from the grammar.

In fact, there are infinitely many proofs of `?- ip`. The first proof that prolog finds can be depicted with a tree like this:



1.3 Using prolog

Here is a session log, where I put the things I typed in **bold**:

```
1%pl
Welcome to SWI-Prolog (Version 4.0.11)
Copyright (c) 1990-2000 University of Amsterdam.
Copy policy:  GPL-2 (see www.gnu.org)
For help, use ?- help(Topic).  or ?- apropos(Word).

?-  write('hello world').
hello world
Yes
?-  halt.
2%emacs test.pl
3%cat test.pl
% OK, a test
p :- q,r.
r :- s.
q :- t.
s.
t.

4%pl
Welcome to SWI-Prolog (Version 4.0.11)
Copyright (c) 1990-2000 University of Amsterdam.
Copy policy:  GPL-2 (see www.gnu.org)

For help, use ?- help(Topic).  or ?- apropos(Word).

?-  [test].
% test compiled 0.00 sec, 1,180 bytes Yes
?-  listing.
```

```
% Foreign:  rl_add_history/1
p :-
     q,
     r.
% Foreign:  rl_read_init_file/1
q :-
     t.
r :-
     s.
s.
t.
Yes
?-  p.
Yes
?-  q.
Yes
?-  z.
ERROR: Undefined procedure:  z/0
?-  halt.
5%
```

1.4 Some distinctions of human languages

Even if we regard human languages as logics, it is easy to see that they differ in some fundamental respects from logics like the propositional calculus. Let's quickly review some of the most basic properties of human languages, many of which we will discuss later:

1. To a first approximation, the physical structure of an utterance can be regarded as a sequence of perceivable gestures in time. We will call the basic elements of these sequences *perceptual atoms*.
2. Utterances have syntactic and semantic structure whose atoms are often not perceptual atoms, but perceptual complexes.
3. Properties of atoms are remembered; properties of complexes may be calculated or remembered.
   At any time, the number of remembered atomic properties (perceptual, syntactic, semantic) is finite.
4. A sequence of perceptual atoms that is a semantic or syntactic atom in one context may not be one in another context.
   For example, in its idiomatic use, *keep tabs on* is semantically atomic, but it has literal uses as well in which it is semantically complex.
5. In every human language, the sets of perceptual, syntactic, and semantic atoms may overlap, but they are not identical.
6. Every human language is open-ended: ordinary language use involves learning new expressions all the time.
7. In every human language, the interpretation of many utterances is context dependent.
   For example, *it is here* is only true or false relative to an interpretation of the relevant context of utterance.
8. Every language has expressions denoting properties, relations, relations among properties and relations, quantifiers and Boolean operations. Some of the relations involve "events" and their participants.
   "Agents" of an event tend to be mentioned first
9. In every human language, utterances can be informative.
   Humans can understand (and learn from) sentences about individuals and properties that we knew nothing of before.
   So, for one thing, declarative sentences do not mean simply true or false.

10. Call a truth-valuable expression containing at most one relation-denoting semantic atom a "simple predication." In no human language are the simple predications logically independent, in the sense that the truth values of one are independent of the others.

    For example, since it is part of the meaning of the predicate *red* that red objects are also in the extension of *colored*, the truth value of *a is red* is not independent of the truth value of *a is colored*.

    In propositional prolog, the interpretation of each atomic formula is independent of the others. The importance of this property has been discussed by Wittgenstein and many other philosophers (Wittgenstein, 1922; Pears, 1981; Demopoulos and Bell, 1993).

11. Language users can recognize (some) entailment relations among expressions.

    Every language includes hyponyms and hypernyms.

    Call an expression *analytic* if it is true simply in virtue of its meaning. Every language includes analytic expressions.

    Perfect synonymy is rare; and perfect (non-trivial) definitions of lexical items are rare.

12. In all languages: Frequent words tend to be short. The most frequent words are grammatical formatives. The most frequent words tend to denote in high types.

    Related facts: affixes and intonation features tend to denote in high types.

13. Quantifiers that are semantic atoms are monotonic.

14. Relation-denoting expressions that are semantic atoms may require argument-denoting expressions to occur with them, but they never require more than 3.

15. Roughly speaking, specifying for each relation-denoting expression the arguments that are required is simpler than specifying for each argument-denoting expression the relations it can be associated with. So we say: verbs "select" their arguments.

16. At least some human languages seem to have truth predicates that apply to expressions in the same language.

    But the semantic paradoxes considered by Russell, Tarski and others show that they cannot apply to all and only the true expressions. – See for example the papers in (Blackburn and Simmons, 1999).

17. Human languages have patterns that are not (appropriately) described by finite state grammars (FSGs), or by context free grammars (CFGs). But the patterns can all be described by multiple context free grammars (MCFGs) and other similar formalisms (a certain simple kind of "minimalist grammars," MGs, and multi-component tree-adjoining grammars, MC-TAGs).

    This last idea has been related to Chomsky's "subjacency" and the "shortest move constraint."

    As we will see, where these problems can be defined reasonably well, only certain kinds of devices can solve them. And in general, of course, the mechanisms of memory access determine what kinds of patterns can distinguish the elements of a language, what kinds of problems can be solved.

Propositional prolog lacks most of these properties. We move to a slightly more human-like logic with respect to properties 7 and 8 in the next section. Many of the other properties mentioned here will be discussed later.

1.5 Predicate Prolog

Predicate prolog allows predicates that take one or more arguments, and it also gives a first glimpse of expressions that depend on "context" for their interpretation. For example, we could have a 1-place predicate `mathematician` which may be true of various individuals that have different names, as in the following axioms:

```
/*
* file:  people.pl
*/
mathematician(frege).
mathematician(hilbert).
mathematician(turing).
mathematician(montague).

linguist(frege).
linguist(montague).
linguist(chomsky).
linguist(bresnan).
president(bush).
president(clinton).
sax_player(clinton).
piano_player(montague).
```

And using an initial uppercase letter or underscore to distinguish variables, we have expressions like $human(X)$ that have a truth value only relative to a "context" – an assignment of an individual to the variable. In prolog, the variables of each definite clause are implicitly bound by universal quantifiers:

```
human(X) :- mathematician(X).
human(X) :- linguist(X).
human(X) :- sax_player(X).
human(X) :- piano_player(X).

sum(0,X,X).
sum(s(X),Y,s(Z)) :- sum(X,Y,Z).

self_identical(X).
socrates_is_mortal.
```

In this theory, we see 9 different **predicates.** Like 0-place predicates (=propositions), these predicates all begin with lower case letters.

Besides predicates, a theory may contain **terms**, where a term is a variable, a name, or a function expression that combines with some appropriate number of terms. **Variables** are distinguished by beginning with an uppercase letter or an underscore. The theory above contains only the one variable `X`. Each axiom has all of its variables "universally bound." So for example, the axiom `self_identical(X)` says: for all `X`, `X` is identical to itself. And the axiom before that one says: for all `X`, `X` is human if `X` is a piano player.

In this theory we see nine different **names**, which are either numerals or else begin with lower case letters: `frege, hilbert, turing, montague, chomsky, bresnan, bush, clinton, 0`. A name can be regarded as a 0-place function symbol. That is, it takes 0 arguments to yield an individual as its value.

In this theory we have one function symbol that takes arguments: the function symbol `s` appears in the two axioms for `sum`. These are Peano's famous axioms for the sum relation. The first of these axioms says that, for all `X`, the sum of `0` and `X` is `X`. The second says that, for all `X, Y` and `Z`, the sum of the successor of `X` and `Y` is the successor of `Z` where `Z` is the sum of `X` and `Y`. So the symbol `s` stands for the **successor** function. This is the function which just adds one to its argument. So the successor of `0` is `1`, `s(0)=1`, `s(s(0))=2`, …. In this way, the successor function symbol takes 1 argument to yield an individual as its value. With this interpretation of the symbol `s`, the two axioms for `sum` are correct. Remarkably, they are also the only axioms we need to

compute sums on the natural numbers, as we will see.

From these axioms, prolog can refute `?- human(montague).` This cannot be refuted using the proof rule shown above, though, since no axiom has `human(montague)` as its head. The essence of prolog is what it does here: it **unifies** the goal `?- human(montague)` with the head of the axiom, `human(X) :- mathematician(X).` We need to see exactly how this works.

Two expressions unify with each other just in case there is a substitution of terms for variables that makes them identical. To unify `human(montague)` and `human(X)` we substitute the term `montague` for the variable `X`. We will represent this substitution by the expression $\{X \mapsto montague\}$. Letting $\theta = \{X \mapsto montague\}$, and writing the substitution in "postfix" notation – <u>after</u> the expression it applies to, we have

$$\texttt{human(X)}\theta = \texttt{human(montague)}\theta = \texttt{human(montague).}$$

Notice that the substitution θ has no effect on the term `human(montague)` since this term has no occurrences of the variable `X`.

We can replace more than one variable at once. For example, we can replace `X` by `s(Y)` and replace `Y` by `Z`. Letting $\theta = \{X \mapsto s(Y), Y \mapsto Z\}$, we have:

$$\texttt{sum(X,Y,Y)}\theta = \texttt{sum(s(Y),Z,Z).}$$

Notice that the `Y` in the first term has not been replaced by `Z`. This is because all the elements of the substitution are always applied *simultaneously*, not one after another.

After a little practice, it is not hard to get the knack of finding substitutions that make two expressions identical, if there is one. These substitutions are called (most general) **unifiers**, and the step of finding and applying them is called (term) **unification.**[2]

To describe unification we need two preliminaries. In the first place, we need to be able to recognize subexpressions. Consider the formula:

`whats_it(s(Y,r(Z,g(Var))),func(g(Argument),X),W).`

The subexpression beginning with `whats_it` is the whole expression.
The subexpression beginning with `s` is `s(Y,r(Z,g(Var)))`.
The subexpression beginning with `Argument` is `Argument`.
No subexpression begins with a parenthesis or comma.

The second preliminary we need is called **composition** of substitutions – we need to be able to build up substitutions by, in effect, applying one to another. Remember that substitutions are specified by expressions of the form

$$\{V_1 \mapsto t_1, \ldots, V_n \mapsto t_n\}$$

where the $\texttt{V}_\texttt{i}$ are *distinct* variables and the $\texttt{t}_\texttt{i}$ are terms ($\texttt{t}_\texttt{i} \neq \texttt{V}_\texttt{i}$) which are substituted for those variables.

Definition 1 The **composition** of substitutions η, θ is defined as follows: Suppose

$$\eta = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$$

$$\theta = \{Y_1 \mapsto s_1, \ldots, Y_m \mapsto s_m\}.$$

The composition of η and θ, $\eta\theta$, is

$$\begin{aligned}\eta\theta = \quad & \{\texttt{X}_1 \mapsto (\texttt{t}_1\theta), \ldots, \texttt{X}_\texttt{n} \mapsto (\texttt{t}_\texttt{n}\theta), \texttt{Y}_1 \mapsto \texttt{s}_1, \ldots, \texttt{Y}_\texttt{m} \mapsto \texttt{s}_\texttt{m}\} - \\ & (\{\texttt{Y}_\texttt{i} \mapsto \texttt{s}_\texttt{i} \mid \texttt{Y}_\texttt{i} \in \{\texttt{X}_1, \ldots, \texttt{X}_\texttt{n}\}\} \cup \{\texttt{X}_\texttt{i} \mapsto \texttt{t}_\texttt{i}\theta \mid \texttt{X}_\texttt{i} = \texttt{t}_\texttt{i}\theta\}).\end{aligned}$$

[2] So-called "unification grammars" involve a related, but slightly more elaborate notion of unification (Pollard and Sag, 1987; Shieber, 1992; Pollard and Sag, 1994).

Earlier predicate logic theorem proving methods like the one in Davis and Putnam (1960) were significantly improved by the discovery in Robinson (1965) that term unification provided the needed matching method for exploiting the insight from the doctoral thesis of Herbrand (1930) that proofs can be sought in a syntactic domain defined by the language of the theory.

That is, to compute $\eta\theta$, first apply θ to the terms of η and then add θ itself, and finally remove any of the θ variables that are also η variables and remove any substitutions of variables for themselves. Clearly, with this definition, every composition of substitutions will itself satisfy the conditions for being a substitution. Furthermore, since the composition $\eta\theta$ just applies θ to η, $A(\eta\theta) = (A\eta)\theta$ for any expression A.

Example 1. Let

$$\eta = \{X_1 \mapsto Y_1, X_2 \mapsto Y_2\}$$
$$\theta = \{Y_1 \mapsto a_1, Y_2 \mapsto a_2\}.$$

Then

$$\eta\theta = \{X_1 \mapsto a_1, X_2 \mapsto a_2, Y_1 \mapsto a_1, Y_2 \mapsto a_2\}.$$

And, on the other hand,

$$\theta\eta = \{Y_1 \mapsto a_1, Y_2 \mapsto a_2, X_1 \mapsto Y_1, X_2 \mapsto Y_2\}$$

Since $\eta\theta \neq \theta\eta$, we see that composition is thus not commutative, although it is associative.

Example 2. Let

$$\eta = \{X_1 \mapsto Y_1\}$$
$$\theta = \{Y_1 \mapsto X_1\}.$$

Then although neither η nor θ is empty, $\eta\theta = \theta$ and $\theta\eta = \eta$.

Example 3. Let

$$\eta = \{\}$$
$$\theta = \{Y_1 \mapsto X_1\}.$$

Then $\eta\theta = \theta\eta = \theta$.

This empty substitution $\eta = \{\}$ is called an "identity element" for the composition operation.[3]

Now we are ready to present a procedure for unifying two expressions E and F to produce a (most general) unifier $mgu(E, F)$.

UNIFICATION ALGORITHM:

1. Put $k = 0$ and $\sigma_0 = \{\}$.
2. If $E\sigma_k = F\sigma_k$, stop. σ_k is a mgu of S. Otherwise, compare $E\sigma_k$ and $F\sigma_k$ from left to right to find the first symbol at which they differ. Select the subexpression E' of E that begins with that symbol, and the subexpression F' of F that begins with that symbol.
3. If one of E', F' is a variable V and one is a term t, and if V does not occur as a (strict) subconstituent of t, put $\sigma_{k+1} = \sigma_k\{V \mapsto t\}$, increment k to $k + 1$, and return to step 2. Otherwise stop, S is not unifiable.

The algorithm produces a most general unifier which is unique up to a renaming of variables, otherwise it terminates and returns the judgment that the expressions are not unifiable.[4]

Now we are ready to define predicate prolog. All clauses and goals are universally closed, so the language, inference method, and semantics are fairly simple.

[3] In algebra, when we have a binary associative operation on a set with an identity element, we say we have a **monoid**. So the set of substitutions, the operation of composition, and the empty substitution form a monoid. Another monoid we will discuss below is given by the set of sequences of words, the operation of appending these sequences, and the empty sequence.

[4] Our presentation of the unification algorithm is based on Lloyd (1987), and this result about the algorithm is established as Lloyd's Theorem 4.3. The result also appears in the classic source, Robinson (1965).

(11) **Predicate prolog**

language: **atoms** $a = \{a-z\}\{a-zA-Z0-9\_\}^* | '\{a-zA-Z0-9\_\ \ @\#\$\%*()\}^{*'}$
variables $v = \{A-Z\_\}\{a-zA-Z0-9\_\}^*$
terms $T ::= v \mid a \mid a(S) \mid 0-9^+$
sequence of terms $S ::= T \mid T, S$ (n.b. one or more terms in the sequence)
predications $p ::= a \mid a(S)$
conjunctions $C ::= \epsilon. \mid p, C$
goals $G ::= ?\text{-}C$
definite clauses $D ::= p\text{:-}C$

(Notation) We write f^n for an atom f that forms a term with a sequence of n terms as arguments. A term f^0 is an individual constant.

We write p^n for an atom p that forms a predication with a sequence of n terms as arguments. A 0-place predication p^0 is a proposition.

inference: $G, \Gamma \vdash G$ [axiom] for any set of definite clauses Γ and any goal G

$$\frac{G, \Gamma \vdash (?\text{-}p, C)}{G, \Gamma \vdash (?\text{-}q_1, \ldots, q_n, C)\theta} \quad \text{if } (q\text{:-}q_1, \ldots, q_n) \in \Gamma, mgu(p, q) = \theta$$

N.B. For maximal generality, we avoid confusing any variables in the goal with variables in an axiom with the following policy: every time a definite clause is selected from Γ, rename all of its variables with variables never used before in the proof. (Standard implemented prolog assigns these new variables numerical names, like `_1295`.)

semantics: a model $\mathcal{M} = \langle E, 2, [\![\cdot]\!] \rangle$, where
$[\![\mathrm{f^n}]\!] : E^n \to E$. When $n = 0$, $[\![\mathrm{f}^0]\!] \in E$.
$[\![\mathrm{p^n}]\!] : E^n \to 2$. When $n = 0$, $[\![\mathrm{p}^0]\!] \in 2$.
$[\![\mathrm{v}]\!] : [V \to E] \to E$, such that for $s : V \to E$, $[\![\mathrm{v}]\!](s) = s(v)$.
$[\![\mathrm{f^n}(\mathrm{t}_1, \ldots \mathrm{t_n})]\!] : [V \to E] \to E$, where for $s : V \to E$, $[\![\mathrm{f^n}(\mathrm{t}_1, \ldots, \mathrm{t_n})]\!](s) = [\![\mathrm{f^n}]\!]([\![\mathrm{t}_1]\!](s), \ldots, [\![\mathrm{t_n}]\!](s))$
$[\![\mathrm{p^n}(\mathrm{t}_1, \ldots, \mathrm{t_n})]\!] : [V \to E] \to 2$, where for $s : V \to E$, $[\![\mathrm{p^n}(\mathrm{t}_1, \ldots, \mathrm{t_n})]\!](s) = [\![\mathrm{p^n}]\!]([\![\mathrm{t}_1]\!](s), \ldots, [\![\mathrm{t_n}]\!](s))$
$[\![\mathrm{A, B}]\!] : [V \to E] \to 2$, where for $s : V \to E$, $[\![\mathrm{A, B}]\!](s) = \min\{[\![\mathrm{A}]\!](s), [\![\mathrm{B}]\!](s))\}$.
$[\![\epsilon]\!] : [V \to E] \to 2$, where for $s : V \to E$, $[\![\epsilon]\!](s) = 1$
$$[\![\mathrm{B\text{:-}A}]\!] = \begin{cases} 0 & \text{if } \exists s \in [V \to E],\ [\![\mathrm{A}]\!](s) = 1 \text{ and } [\![\mathrm{B}]\!](s) = 0 \\ 1 & \text{otherwise} \end{cases}$$
$[\![?\text{-A}]\!] = 1 - \min\{[\![\mathrm{A}]\!](s) \mid s : V \to E\}$

metatheory: $G, \Gamma \vdash A$ iff $G, \Gamma \vDash A$, for any goals G, A and any definite clause theory Γ.

Loading the axioms of `people.pl`, displayed on page 13 above, prolog will use these rules to establish consequences of the theory. We can ask prolog to prove that Frege is a mathematician by typing `mathematician(frege).` at the prolog prompt. Prolog will respond with `yes`. We can also use a variable to ask prolog what things X are mathematicians (or computers). If the loaded axioms do not entail that any X is a mathematician (or computer), prolog will say: no. If something can be proven to be a mathematician (or computer), prolog will show what it is. And after receiving one answer, we can ask for other answers by typing a semi-colon:

```
| ?- mathematician(X).

X = frege ? ;

X = hilbert ? ;

X = turing ? ;

X = montague ? ;

no
| ?- mathematician(fido).

no
```

Prolog establishes these claims just by finding substitutions of terms for the variable `X` which make the goal identical to an axiom. So, in effect, a variable in a goal is existentially quantified. The goal `?-p(X)` is, in effect, a request to prove that there is some `X` such that `p(X)`.[5]

```
| ?- mathematician(X),linguist(X),piano_player(X).

X = montague ? ;

no
| ?-
```

We can display a proof as follows, this time showing the clause and bindings used at each step:

goal		theory		workspace	
`?-human(X)`	,	Γ	$\vdash$	`?-human(X)`	
`?-human(X)`	,	`human(X'):-mathematician(X')`	$\vdash$	`?-mathematician(X)`	$\{X' \mapsto X\}$
`?-human(X)`	,	`mathematician(frege)`	$\vdash$	$\square$	$\{X \mapsto$ `frege`$\}$

[5] More precisely, the prolog proof is a refutation of $\neg(\forall X)\ p(X)$, and this is equivalent to an existential claim: $\neg(\forall X)\ p(X) \equiv (\exists X)\neg p(X)$.

1.6 The logic of sequences

Time imposes a sequential structure on the words and syntactic constituents of a spoken sentence. Sequences are sometimes treated as functions from initial segments of the set of natural numbers. Such functions have the right mathematical properties, but this is a clunky approach that works just because the natural numbers are linearly ordered. What are the essential properties of sequences? We can postpone this question, since what we need for present purposes is not a deep understanding of time and sequence, but a way of calculating certain basic relations among sequences and their elements – a sort of arithmetic.[6]

We will represent a string of words like

the cat is on the mat

as a *sequence* or *list* of words:

```
[the,cat,is,on,the,mat]
```

We would like to be able to prove that some sequences of words are good sentences, others are not. To begin with a simple idea, though, suppose that we want to prove that the sequence shown above contains the word `cat`. We can define the membership relation between sequences and their elements in the following way:[7]

```
member(X,[X|L]).
member(X,[_|L]) :- member(X,L).
```

The vertical bar notation is used in these axioms to separate the first element of a sequence from the remainder. So the first axiom says that X is a member of any sequence which has X followed by any remainder L. The second axiom says that X is a member of any sequence which is Y followed by the remainder L if X is a member of L.

With these axioms, we can prove:

```
| ?- member(cat,[the,cat,is,on,the,mat]).

yes
```

There are exactly 6 proofs that something is a member of this sequence, with 6 different bindings of X:

```
| ?- member(X,[the,cat,is,on,the,mat]).

X = the ? ;

X = cat ? ;

X = is ? ;

X = on ? ;

X = the ? ;

X = mat ? ;

no
| ?-
```

The *member* predicate is so important, it is "built in" to SWI prolog – The two axioms shown above are already there.

Another basic "built in" predicate for sequences is *length*, and this one uses the "built in" predicate *is* for arithmetic expressions. First, notice that you can do some simple arithmetic using *is* in the following way:

[6] A formal equivalence between the logic of sequences and concatenation, on the one hand, and arithmetic on the other, has been observed by Hermes (1938), Quine (1946), Corcoran, Frank, and Maloney (1974). See the footnote 9, below.

The stage was later set for understanding the connection between arithmetic and theories of trees (and tree-like structures) in language by Rabin (1969); see Cornell and Rogers (1999) for an overview.

[7] The underscore _ by itself is a special variable, called the "anonymous variable" because no two occurrences of this symbol represent the same variable. It is good form to use this symbol for any variable that occurs just once in a clause; if you don't, Prolog will warn you about your "singleton variables."

```
?- X is 2+3.

X = 5

Yes
?- X is 2^3.

X = 8

Yes
```

Using this predicate, length is defined this way:

```
length([],0).
length([_|L],N) :- length(L,N0), N is N0+1.
```

The first axiom says that the empty sequence has length 0. The second axiom says that any list has length N if the result of removing the first element of the list has length N0 and N is N0+1. Since these axioms are already "built in" we can use them immediately with goals like this:

```
?- length([a,b,1,2],N).

N = 4

Yes
?- length([a,b,[1,2]],N).

N = 3

Yes
```

We can do a lot with these basic ingredients, but first we should understand what's going on.

This standard approach to sequences or lists may seem odd at first. The empty list is a named by [], and non-empty lists are represented as the denotations of the period (often pronounced "cons" for "constructor"), which is a binary function symbol. A list with one element is denoted by cons of that element and the empty list. For example, `.(a,[])` denotes the sequence which has just the one element `a`. And `.(b,.(a,[]))` denotes the sequence with first element `b` and second element `a`. For convenience, we use `[b,a]` as an alternative notation for the clumsier `.(b,.(a,[]))`, and we use `[A|B]` as an alternative notation for `.(A,B)`.

Examples:

If we apply {X↦frege,Y↦[]} to the list [X|Y], we get the list [frege].

If we apply {X↦frege,Y↦[hilbert]} to the list [X|Y], we get the list [frege,hilbert].

[X|Y] and [frege,hilbert] match after the substitution {X↦frege,Y↦[hilbert]}.

Using this notation, we presented an axiomatization of the `member` relation. Another basic thing we need to be able to do is to put two sequences together, "concatenating" or "appending" them. We can accordingly define a 3-place relation we call `append` with the following two simple axioms:[8]

```
append([],L,L).
append([E|L0],L1,[E|L2]) :- append(L0,L1,L2).
```

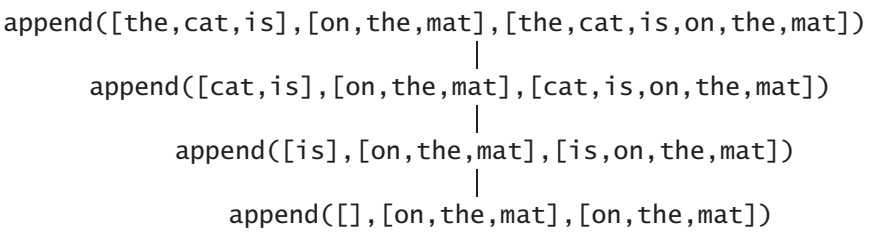
The first axiom says that, for all L, the result of appending the empty list and L is L. The second axiom says that, for all E, L0, L1, and L2, the result of appending [E|L0] with L1 is [E|L2] if the result of appending

[8] These axioms are "built in" to SWI-prolog – they are already there, and the system will not let you redefine this relation. However, you could define the same relation with a different name like myappend.

L0 and L1 is L2. These two axioms entail that there is a list L which is the result of appending [the,cat,is] with [on,the,mat].[9] Prolog can prove this fact:

```
| ?- append([the,cat,is],[on,the,mat],L).

L = [the,cat,is,on,the,mat] ? ;

no
```

The proof of the goal,
append([the,cat,is],[on,the,mat],[the,cat,is,on,the,mat])
can be depicted by the following proof tree:



This axiomatization of append behaves nicely on a wide range of problems. It correctly rejects

```
| ?- append([the,cat,is],[on,the,mat],[]).

no
| ?- append([the,cat,is],[on,the,mat],[the,cat,is,on,the]).

no
```

We can also use it to split a list:

```
| ?- append(L0,L1,[the,cat,is,on,the,mat]).

L0 = []
L1 = [the,cat,is,on,the,mat] ? ;

L0 = [the]
L1 = [cat,is,on,the,mat] ? ;

L0 = [the,cat]
L1 = [is,on,the,mat] ? ;

L0 = [the,cat,is]
L1 = [on,the,mat] ? ;

L0 = [the,cat,is,on]
L1 = [the,mat] ? ;

L0 = [the,cat,is,on,the]
L1 = [mat] ? ;

L0 = [the,cat,is,on,the,mat]
L1 = [] ? ;

no
```

[9] Using the successor function s and 0 to represent the numbers, so that s(0)=1, s(s(0))=2,..., notice how similar the definition of append is to the following formulation of Peano's axioms for the sum relation:

```
sum(0,N,N).
sum(s(N0),N1,s(N2)) :- sum(N0,N1,N2).
```

Each of these solutions represents a different proof, a proof that could be diagrammed like the one discussed above.[10]

Pitfall 2: Infinite terms

Step 3 of the unification algorithm involves checking to see if a variable occurs in a term. This is called the "occurs check." It is easy to find examples to show that this check is necessary. Consider the prolog axiom which says that for any number N, the successor of N is greater than N:

```
greaterthan(s(N),N).
```

Now suppose we try to prove that some number is greater than itself. In prolog, this would be an attempt to prove the goal:

```
| ?- greaterthan(N,N).
```

To establish this goal, prolog will select the axiom `greaterthan(s(N),N)` and rename the variables to get something like $E = \texttt{greaterthan}(\texttt{s}(\texttt{A1}),\texttt{A1})$. We will then try to unify this with the goal $F = \texttt{greaterthan}(\texttt{N},\texttt{N})$, but these are *not* unifiable. Let's see why this is so. Comparing these expressions, we find that they differ at the expressions `s(A1)` and `N`, so $\sigma_1 = \{\texttt{N} \mapsto \texttt{s}(\texttt{A1})\}$. Now we compare $E\sigma_1 = \texttt{greaterthan}(\texttt{s}(\texttt{A1}),\texttt{s}(\texttt{A1}))$ with $F\sigma_1 = \texttt{greaterthan}(\texttt{s}(\texttt{A1}),\texttt{A1})$. These differ at the expressions `s(A1)` and `A1`. But `A1` occurs in `s(A1)` and so the algorithm stops with failure. We do *not* attempt to apply the substitution $\{\texttt{A1} \mapsto \texttt{s}(\texttt{A1})\}$.

For the sake of efficiency, though, **standard implementations of prolog do not apply the occurs check.** This only causes trouble in certain cases, like the one described above. If you give prolog the axiom above and the goal `?- greaterthan(N,N).`, then prolog will start printing out something horrible like:

```
 N = s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s
(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s
(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s
(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s
...
```

If you ever get something like this, try to stop it by typing control-C once or twice. It should be clear by now why prolog prints this out.

Like the problem with left recursion, the designers of prolog could have eliminated this problem, but instead they chose to let the users of prolog avoid it so that (when the users are appropriately careful to avoid the pitfalls) the proofs can be computed very quickly.

[10]Try diagramming one or two of these proofs on a dull evening.

Exercises

(1) A propositional representation of a grammar is presented in (10), and the first proof of `?-ip` that prolog will find is shown in tree form on page 10. Draw the tree that depicts the second proof prolog would find.

(2) For each of the following pairs of literals, present a most general unifier (if there is one):

a. `human(X)  human(bob)`

b. `loves(X,mary)  loves(bob,Y)`

c. `[cat,mouse]  [Z|L]`

d. `[cat,mouse,fish]  [Z|L]`

e. `[cat,mouse,fish]  [dog|L]`

f. `member(X,[cat,mouse,fish])  member(Z,[Z|L])`

(3) You may remember from (3) on page 2 the hypothesis that perception aims to compute small representations.[11] I am interested in the size of things. We have seen how to calculate the length of a list, but the elements of a list can have various sizes too. For example, we might want to say that the following two structures have different sizes, even though they are sequences of the same length:

```
                                    [a,b]
[akjjdfkpodsaijfospdafpodsa,aposdifjodsahfpodsaihfpoad]
```

SWI prolog has a built in predicate that let's you take apart an atom into its individual characters:

```
?- atom_chars(a,L).

L = [a]

Yes
?- atom_chars(akjjdfkpodsaijfospdafpods,L).

L = [a, k, j, j, d, f, k, p, o|...]

Yes
```

Notice that SWI prolog puts in an ellipsis when it is showing you long lists, but the whole list is there, as we can see by checking the lengths of each one:

```
?- atom_chars(a,L),length(L,N).

L = [a]
N = 1

Yes
?- atom_chars(akjjdfkpodsaijfospdafpods,L),length(L,N).

L = [a, k, j, j, d, f, k, p, o|...]
N = 25

Yes
```

So we can define a predicate that relates an atom to its length as follows:

```
atom_length(Atom,Length) :- atom_chars(Atom,Chars),length(Chars,Length).
```

Define a predicate *sum_lengths* that relates a sequence of atoms to the sum of the lengths of each atom in the sequence, so that, for example,

[11] It is sometimes proposed that learning in general is the discovery of small representations (Chater and Vitányi, 2002). This may also be related to some of the various general tendencies towards economy (or "optimality") of expression in language.

```
?- sum_lengths([],N).

N = 0

Yes
?- sum_lengths([a,akjjdfkpodsaijfospdafpods],N).

N = 26

Yes
```

Extra credit: The number of characters in a string is not a very good measure of its size, since it matters whether the elements of the string are taken from a 26 symbol alphabet like {a-z} or a 10 symbol alphabet like {0-9} or a two symbol alphabet like {0,1}.

The most common size measures are given in terms of two symbol alphabets: we consider how many symbols are needed for a binary encoding, how many "bits" are needed.

Now suppose that we want to represent a sequence of letters or numbers. Let's consider sequences of the digits 0-9 first. A naive idea is this: to code up a number like 52 in binary notation, simply represent each digit in binary notation. Since 5 is 101 and 2 is 10, we would write 10110 for 52. This is obviously not a good strategy, since there is no indication of the boundaries between the 5 and the 2. The same sequence would be the code for 26.

Instead, we could just express 52 in base 2, which happens to be 110100. While this is possible, it is a rather inefficient code, because there are actually infinitely many binary representations of 52:

$$110100, 0110100, 00110100, 000110100, , \ldots$$

Adding any number of preceding zeroes has no effect! A better code would not be so wasteful.

Here is a better idea. We will represent the numbers with binary sequences as follows:

decimal number	0	1	2	3	4	5	6	…
binary sequence	ϵ	0	1	00	01	10	11	…

Now here is the prolog exercise:

i. Write a prolog predicate e(N,L) that relates each decimal number N to its binary sequence representation L.

ii. Write a prolog predicate elength(N,Length) that relates each decimal number N to the length of its binary sequence representation.

iii. We saw above that the length of the (smallest – no preceding zeroes) binary representation of 52 is 6. Use the definition you just wrote to have prolog compute the length of the binary sequence encoding for 52.

3 more exercises with sequences – easy, medium, challenging!

(4) Define a predicate *countOccurrences(E,L,Count)* that will take a list L, an element E, and return the Count of the number of times E occurs in L, in decimal notation.

Test your predicate by making sure you get the right response to these tests:

```
?- countOccurrences(s,[m,i,s,s,i,s,s,i,p,p,i],N).

N = 4 ;

No
?- countOccurrences(x,[m,i,s,s,i,s,s,i,p,p,i],N).

N = 0 ;

No
```

(5) Define a predicate *reduplicated(L)* that will be provable just in case list L can be divided in half – i.e. into two lists of the same length – where the first and second halves are the same.

Test your predicate by making sure you get the right response to these tests:

```
?- reduplicated([w,u,l,o]).

No
?- reduplicated([w,u,l,o,w,u,l,o]).

Yes
```

This might remind you of "reduplication" in human languages. For example, in Bambara, an African language spoken by about 3 million people in Mali and nearby countries, we find an especially simple kind of "reduplication" structure, which we see in complex words like this:

wulu	*'dog'*	*wulo o wulo*	*'whichever dog'*
malo	*'rice'*	*malo o malo*	*'whichever rice'*
**malo o wulu*	*NEVER!*		
malonyinina	*'someone who looks for rice'*	*malonyinina o malonyinina*	*'whoever looks for rice'*

(6) Define a predicate *palindrome(L)* that will be provable just in case when you look at the characters in the atoms of list L, L is equal to its reverse.

Test your predicate by making sure you get the right response to these tests:

```
?- palindrome([wuloowulo]).

No
?- palindrome([hannah]).

Yes
?- palindrome([mary]).

No
?- palindrome([a,man,a,plan,a,canal,panama]).

Yes
```

Two harder extra credit problems for the go-getters

(7) *More extra credit, part 1.* The previous extra credit problem can be solved in lots of ways. Here is a simple way to do part a of that problem: to encode N, we count up to N with our binary sequences. But since the front of the list is the easiest to access, we use count with the order most-significant-digit to least-significant-digit, and then reverse the result. Here is a prolog program that does this:

```
e(N,L) :-
        countupReverse(N,R),
        reverse(R,L).

countupReverse(0,[]).
countupReverse(N,L) :-
        N>0,
        N1 is N-1,
        countupReverse(N1,L1),
        addone(L1,L).

addone([],[0]).
addone([0|R],[1|R]).
addone([1|R0],[0|R]) :- addone(R0,R).
```

This is good, but suppose that we want to communicate two numbers in sequence. For this purpose, our binary representations are still no good, because you cannot tell where one number ends and the next one begins.

One way to solve this problem is to decide, in advance, that every number will be represented by a certain number of bits – say 7. This is what is done in standard ascii codes for example. But blocks of n bits limit you in advance to encoding no more than 2^n elements, and they are inefficient if some symbols are more common than others.

For many purposes, a better strategy is to use a coding scheme where no symbol (represented by a sequence of bits) is the prefix of any other one. That means, we would never get confused about where one symbol ends and the next one begins. One extremely simple way to encode numbers in this way is this. To represent a number like 5, we put in front of [1,0] an (unambiguous) representation of the length n of [1,0] – namely, we use n 1's followed by a 0. So then, to represent 5, we use [1,1,0,1,0]. The first 3 bits indicate that the number we have encoded is two bits long.

So in this notation, we can unambiguously determine what sequence of numbers is represented by

$$[1,0,0,1,0,1,1,1,0,1,1].$$

This is a binary code for the number sequence $[1,2,6]$. *Define a predicate e1(NumberSequence,BinaryCode) that transforms any sequence of numbers into this binary code.* (We will improve on this code later.)

(8) *More extra credit, part 2 (hard!).* While the definition of $e(N,L)$ given above works, it involves counting from 0=[] all the way up to the number you want. Can you find a simpler way?

<u>*Hint:*</u> The empty sequence ϵ represents 0, and any other sequence of binary digits $[a_n, a_{n-1}, \ldots, a_0]$ represents

$$\sum_{i=0}^{n} (a_i + 1)2^i.$$

So for example, [1,0] represents $(0+1)2^0 + (1+1)2^1 = 1 + 4 = 5$. Equivalently, $[a_n, a_{n-1}, \ldots, a_0]$ represents

$$2^{n+1} - 1 + \sum_{i=0}^{n} a_i 2^i.$$

So for example, [1,0] represents $2^{1+1} - 1 + (0 \cdot 2^0) + (1 \cdot 2^1) = 4 - 1 + 0 + 2 = 5$.

(Believe it or not, some students in the class already almost figured this out, instead of using a simple counting strategy like the one I used in the definition of e above.)

2 Recognition: first idea

(1) We noticed in example (10) on page 9 that the way prolog finds a proof corresponds exactly to a simple way of finding a derivation from a context free grammar. In fact, the two are the same if we represent the grammar with definite clauses:

```
/*
* file:  th2.pl
*/
   ip :- dp, i1.      i1 :- i0, vp.      i0 :- will.         will.
   dp :- d1.          d1 :- d0, np.      d0 :- the.          the.
   np :- n1.          n1 :- n0.          n0 :- idea.         idea.
                      n1 :- n0, cp.
   vp :- v1.          v1 :- v0.          v0 :- suffice.      suffice.
   cp :- c1.          c1 :- c0, ip.      c0 :- that.         that.
```

Often we want to know more than simply whether there is some derivation of a category though.

For example, rather than asking simply, "Can an ip be derived" using the goal ?-ip, we might want to know whether the sequence *the idea will suffice* is an ip.

(2) In effect, what we want is to ask whether there is a certain kind of proof of ?-ip, namely a proof where the "lexical axioms" *the,idea,will,suffice* are used exactly once each, in order.

(3) Resource logics allow control over the number of times an axiom can be used, and in what order, and so they are well suited for reasoning about language (Moortgat, 1996; Roorda, 1991; Girard, Lafont, and Taylor, 1989). Intuitively, these logics allow us to think of our formulas as resources that can get used up in the course of a proof.

In a sense, these logics are simpler than standard logics, since they simply lack the "structural rules" that we see in standard logics, rules that allow the axioms to be arbitrarily reordered or repeated.

(4) So we will first define a prolog provability predicate, and then we will modify that definition so that it is "resource sensitive" with respect to the lexical axioms.

2.1 A provability predicate

(5) Given a theory Γ (the "object theory"), can we define a theory Γ' (a "metatheory") that contains a predicate $provable(A)$ such that:

$$(?\text{-}C), \Gamma \vdash ?\text{-}A \quad \text{iff} \quad (?\text{-}\texttt{provable}(C)), \Gamma' \vdash (?\text{-}\texttt{provable}(A))$$

Notice that the expressions C, A are used in the statement on the left side of this biconditional, while they are mentioned in the goal on the right side of the biconditional.

(This distinction is sometimes marked in logic books with corner quotes, but in prolog we rely on context to signal the distinction.[12])

(6) Recall that the propositional prolog logic is given as follows, with just one inference rule:

$G, \Gamma \vdash G$ [axiom] for any set of definite clauses Γ and any goal G

$$\frac{G, \Gamma \vdash (?\text{-}p, C)}{G, \Gamma \vdash (?\text{-}q_1, \ldots, q_n, C)} \quad \text{if } (p\text{:-}q_1, \ldots, q_n) \in \Gamma$$

(7) In SWI-Prolog, let's represent an object theory using definite clauses of the form:

```
p :~ [q,r].
q :~ [].
r :~ [].
```

So then, given a prolog theory Γ, we will change it to a theory Γ' with a provability predicate for Γ, just by changing `:-` to `:~` and by defining the infix `?~` provability predicate:

```
/*
 * provable.pl
 */
:- op(1200,xfx,:~).   % this is our object language "if"
:- op(400,fx,?~).     % metalanguage provability predicate

(?~ []).
(?~ Goals0) :- infer(Goals0,Goals), (?~ Goals).

infer([A|C], DC) :- (A :~ D), append(D,C,DC).   % ll

p :~ [q,r].
q :~ [].
r :~ [].
```

[12] On corner quotes, and why they allow us to say things that simple quotes do not, see Quine (1951a, §6).

There are other tricky things that come up with provability predicates, especially in theories Γ that define provability in Γ. These are explored in the work on provability and "reflection principles" initiated by Löb, Gödel and others. Good introductions to this work can be found in (Boolos and Jeffrey, 1980; Boolos, 1979).

(8) Then we can have a session like this:

```
1 ?- [provable].
provable compiled, 0.00 sec, 1,432 bytes.

Yes
2 ?- (?- [p]).

Yes
3 ?- trace,(?- [p]).
   Call:  (  7) ?-[p] ?
   Call:  (  8) infer([p], _L154) ?
   Call:  (  9) p:-_L168 ?
   Exit:  (  9) p:-[q, r] ?
   Call:  (  9) append([q, r], [], _L154) ?
   Call:  ( 10) append([r], [], _G296) ?
   Call:  ( 11) append([], [], _G299) ?
   Exit:  ( 11) append([], [], []) ?
   Exit:  ( 10) append([r], [], [r]) ?
   Exit:  (  9) append([q, r], [], [q, r]) ?
   Exit:  (  8) infer([p], [q, r]) ?
   Call:  (  8) ?-[q, r] ?
   Call:  (  9) infer([q, r], _L165) ?
   Call:  ( 10) q:-_L179 ?
   Exit:  ( 10) q:-[] ?
   Call:  ( 10) append([], [r], _L165) ?
   Exit:  ( 10) append([], [r], [r]) ?
   Exit:  (  9) infer([q, r], [r]) ?
   Call:  (  9) ?-[r] ?
   Call:  ( 10) infer([r], _L176) ?
   Call:  ( 11) r:-_L190 ?
   Exit:  ( 11) r:-[] ?
   Call:  ( 11) append([], [], _L176) ?
   Exit:  ( 11) append([], [], []) ?
   Exit:  ( 10) infer([r], []) ?
   Call:  ( 10) ?-[] ?
   Exit:  ( 10) ?-[] ?
   Exit:  (  9) ?-[r] ?
   Exit:  (  8) ?-[q, r] ?
   Exit:  (  7) ?-[p] ?

Yes
```

2.2 A recognition predicate

(9) Now, we want to model recognizing that a string can be derived from `ip` in a grammar as finding a proof of `ip` that uses the lexical axioms in that string exactly once each, in order.

To do this, we will separate the lexical rules Σ from the rest of our theory Γ that includes the grammar rules. Σ is just the vocabulary of the grammar.

(10) The following proof system does what we want:

$G, \Gamma, S \vdash G$ [axiom] for definite clauses Γ, goal G, $S \subseteq \Sigma^*$

$$\frac{G, \Gamma, S \vdash (?\text{-}p, C)}{G, \Gamma, S \vdash (?\text{-}q_1, \ldots, q_n, C)} \quad \text{if } (p\text{:-}q_1, \ldots, q_n) \in \Gamma$$

$$\frac{G, \Gamma, wS \vdash (?\text{-}w, C)}{G, \Gamma, S \vdash (?\text{-}C)}\,[scan]$$

(11) We can implement this in SWI-prolog as follows:

```
/*
 * file: recognize.pl
 */
:- op(1200,xfx,:~).   % this is our object language "if"
:- op(1100,xfx,?~).   % metalanguage provability predicate

[] ?~ [].
(S0 ?~ Goals0) :- infer(S0,Goals0,S,Goals), (S ?~ Goals).

infer(S,[A|C], S,DC) :- (A :~ D), append(D,C,DC).   % ll
infer([A|S],[A|C], S,C).                            % scan

ip :~ [dp, i1].     i1 :~ [i0, vp].     i0 :~ [will].
dp :~ [d1].         d1 :~ [d0, np].     d0 :~ [the].
np :~ [n1].         n1 :~ [n0].         n0 :~ [idea].
                    n1 :~ [n0, cp].
vp :~ [v1].         v1 :~ [v0].         v0 :~ [suffice].
cp :~ [c1].         c1 :~ [c0, ip].     c0 :~ [that].
```

(12) Then we can have a session like this:

```
2% swiprolog
Welcome to SWI-Prolog (Version 3.2.9)
Copyright (c) 1993-1999 University of Amsterdam.  All rights reserved.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- [recognize].
recognize compiled, 0.01 sec, 3,128 bytes.

Yes
2 ?- [the,idea,will,suffice] ?~ [ip].

Yes
3 ?- [idea,the,will] ?~ [ip].

No
4 ?- [idea] ?~ [Cat].

Cat = np ;

Cat = n1 ;

Cat = n0 ;

Cat = idea ;

No
5 ?- [will,suffice] ?~ [Cat].

Cat = i1 ;

No
6 ?- [will,suffice,will,suffice] ?~ [C,D].

C = i1
D = i1 ;

No
7 ?- S ?~ [ip].

S = [the, idea, will, suffice] ;

S = [the, idea, will, v0] ;

S = [the, idea, will, v1]

Yes
```

(13) The execution of the recognition device defined by `provable` can be depicted like this, where Γ is the grammar:

goal		theory		resources		workspace
?-ip	,	Γ	,	[the,idea,will,suffice]	⊢	?-ip
?-ip	,	Γ	,	[the,idea,will,suffice]	⊢	?-dp,i1
?-ip	,	Γ	,	[the,idea,will,suffice]	⊢	?-d0,np,i1
?-ip	,	Γ	,	[the,idea,will,suffice]	⊢	?-the,np,i1
?-ip	,	Γ	,	[idea,will,suffice]	⊢	?-np,i1
?-ip	,	Γ	,	[idea,will,suffice]	⊢	?-n1,i1
?-ip	,	Γ	,	[idea,will,suffice]	⊢	?-n0,i1
?-ip	,	Γ	,	[idea,will,suffice]	⊢	?-idea,i1
?-ip	,	Γ	,	[will,suffice]	⊢	?-i1
?-ip	,	Γ	,	[will,suffice]	⊢	?-i0,vp
?-ip	,	Γ	,	[will,suffice]	⊢	?-will,vp
?-ip	,	Γ	,	[suffice]	⊢	?-vp
?-ip	,	Γ	,	[suffice]	⊢	?-v1
?-ip	,	Γ	,	[suffice]	⊢	?-v0
?-ip	,	Γ	,	[suffice]	⊢	?-suffice
?-ip	,	Γ	,	[]	⊢	□

(14) This is a standard top-down, left-to-right, backtracking context free recognizer.

2.3 Finite state recognizers

(15) A subset of the context free grammars have rules that are only of the following forms, where `word` is a lexical item and `p,r` are categories:

```
p :~ [word,r].
p :~ [].
```

These grammars "branch only on the right" – they are "right linear."

(16) Right linear grammars are finite state in the following sense:

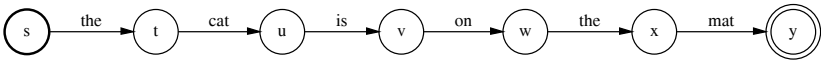
> there is a finite bound k such that every sentence generated by a finite state grammar can be recognized or rejected with a sequence ("stack") in the "workspace" of length no greater than k.

(17) Right linear grammars can be regarded as presentations of finite state transition graphs, where the empty productions indicate the final states.
For example, the following grammar generates $\{0,1\}^*$:

```
s :~ [0,s].
s :~ [1,s].
s :~ [].
```

Another example

```
s :~ [the,t].
t :~ [cat,u].
u :~ [is,v].
v :~ [on,w].
w :~ [the,x].
x :~ [mat,y].
y :~ [].
```



(18) These properties will be carefully established in standard texts on formal language theory and the theory of computatation (Moll, Arbib, and Kfoury, 1988; Lewis and Papadimitriou, 1981; Hopcroft and Ullman, 1979; Salomaa, 1973), but the basic ideas here are simple.
Finite state grammars like this are sometimes used to represent the set of lexical sequences that most closely fit with an acoustic input.
These grammars are also used to model parts of OT phonology (Ellison, 1994; Eisner, 1997; Frank and Satta, 1998).

Exercises

(1) Type `recognize.pl` from page 29 into your own editor, except call it `newrecognize.pl`, and replace the grammar with a a right-branching finite state grammar with start category `input` that accepts all possible word sequences that might plausibly be confused for an acoustic presentation of:

mares eat oats

For example, *"mares" see dotes*,[13] *mayors seat dotes*, *mairs seed oat's*[14],...(If this task is not possible, explain why not, and write a grammar that comes as close as you can to this goal.)

Design this grammar so that it has the start category `input`, and so that it does not use any of the categories that I used in the grammar in (11) of §2.2 above.

(2) Check your grammar by making sure that prolog can use it to accept some of the possibilities, and to reject impossibilities:

For example, you should get something like this:

```
1 ?- [newrecognize].
newrecognize compiled, 0.00 sec, 2,780 bytes.

Yes
2 ?- [mayors,seat,dotes] ?~ [input].

Yes
3 ?- [do,it] ?~ [input].

No
```

(3) *Extra credit. (This one is not too hard – you should try it.)*

a. Modify the example grammar given in (11) of §2.2 above so that it accepts `mares eat oats` as an `ip`. (Leave the syntax as unchanged as possible in this step.

b. Now suppose that when we hear an utterance of "mares eat oats", the resources available to be recognized are not `[mares,eat,oats]`, but rather any one of the strings given by your finite state machine. Provide a new definition of `provable` which, instead of using resources from a particular string S, uses any string that is accepted by the finite state grammar designed above.

(Hint: Instead of using a list of resources, use a state of the finite state machine as a representation of the resources available.)

[13] The first word of this sentence is "the." But the first and last words of the previous sentence are not the same! The first is a determiner, while the last is a proper noun, a quotation name of a word. The similar point applies to the first word in the example: it is also a proper noun.

[14] The OED says *mair* is a "northern form of *more*, and *night<u>mare</u>*."

(4) There are quite a few language resources available online. One of them is Roger Mitton's (1992) phonetic dictionary in the Oxford Text Archive. It has 70645 words of various kinds with phonetic transcriptions of British English. The beginning of the listing looks like this:

```
'neath              niT                 T-$                 1
'shun               SVn                 W-$                 1
'twas               tw0z                Gf$                 1
'tween              twin                Pu$,T-$             1
'tween-decks        'twin-deks          Pu$                 2
'twere              tw3R                Gf$                 1
'twill              twIl                Gf$                 1
'twixt              twIkst              T-$                 1
'twould             twUd                Gf$                 1
'un                 @n                  Qx$                 1
A                   eI                  Ki$                 1
A's                 eIz                 Kj$                 1
A-bombs             'eI-b0mz            Kj$                 2
A-level             'eI-levl            K6%                 3
A-levels            'eI-levlz           Kj%                 3
AA                  ,eI'eI              Y>%                 2
ABC                 ,eI,bi'si           Y>%                 3
```

The second column is a phonetic transcription of the word spelled in the first column. (Columns 3 and 4 contain syntactic category, number of syllables.)

The phonetic transcription has notations for 43 sounds. My guesses on the translation:

Mitton	IPA	example	Mitton	IPA	example
i	i	bead	N	ŋ	sing
I	ɪ	bid	T	θ	thin
e	ɛ	bed	D	ð	then
&	æ	bad	S	ʃ	shed
A	a	bard	Z	ʒ	beige
0(zero)	ɒ	cod	O	ɛə	cord
U	ʊ	good	u	u	food
p	p		t	t	
k	k		b	b	
d	d		g	g	
V	ʌ		m	m	
n	n		f	f	
v	v		s	s	
z	z		3	ɜ	bird
r	r		l	l	
w	w		h	h	
j	j		@	ə	about
eI	eɪ	day	@U	oʊ	go
aI	aɪ	eye	aU	aʊ	cow
oI	oɪ	boy	I@	ɪə	beer
e@	ɛə	bare	U@	ʊə	tour
R	ɹ	far			

The phonetic entries also mark primary stress with an apostrophe, and secondary stress with an comma. Word boundaries in compound forms are indicated with a +, unless they are spelled with a hyphen or space, in which case the phonetic entries do the same.

bookclub above board air-raid

a. Mitton's dictionary is organized by spelling, rather than by phonetic transcription, but it would be easy to reverse. Write a program that maps phonetic sequences like this

$$[\text{'D}, \text{'@}, \text{k}, \text{'\&}, \text{t}, \text{'I}, \text{z}, \text{'O}, \text{n}, \text{'D}, \text{'@}, \text{m}, \text{'\&}, \text{t}]$$

to word sequences like this:

$$[\text{the}, \text{cat}, \text{is}, \text{on}, \text{the}, \text{mat}].$$

b. As in the previous problem (3), connect this translator to the recognizer, so that we can recognize certain phonetic sequences as sentences.

Problem (4), Solution 1:

```
%File: badprog.pl
% not so bad, really!

% first, we load the first 2 columns of the Mitton lexicon
:- [col12].

test :- translate(['D','@',k,'\&',t,'I',z,'O',n,'D','@',m,'\&',t],Words), write(Words).

%translate(Phones,Words)
translate([],[]).
translate(Phones,[Word|Words]) :-
        append(FirstPhones,RestPhones,Phones),
        lex(Word,FirstPhones),
        translate(RestPhones,Words).
```

We can test this program like this:

```
1 ?- [badprog].
%  col12 compiled 2.13 sec, 52 bytes
% badprog compiled 2.13 sec, 196 bytes

Yes
2 ?- translate(['D','@',k,'&',t,'I',z,'O', n,'D','@',m,'&',t],Words).

Words = [the, cat, is, on, the, 'Matt'] ;

Words = [the, cat, is, on, the, mat] ;

Words = [the, cat, is, on, the, matt] ;

No
3 ?-
```

Part b of the problem asks us to integrate this kind of translation into the syntactic recognizer. Since we only want to do a dictionary lookup when we have a syntactic lexical item of the syntax, let's represent the lexical items in the syntax with lists, like this:

```
ip :~ [dp, i1].     i1 :~ [i0, vp].     i0 :~ [].
dp :~ [d1].         d1 :~ [d0, np].     d0 :~ [[t,h,e]].
np :~ [n1].         n1 :~ [n0].         n0 :~ [[c,a,t]].    n0 :~ [[m,a,t]].
vp :~ [v1].         v1 :~ [v0,pp].      v0 :~ [[i,s]].
pp :~ [p1].         p1 :~ [p0,dp].      p0 :~ [[o,n]].
```

Now the syntactic atoms have a phonetic structure, as a list of characters. We test this grammar in the following session – notice each word is spelled out as a sequence of characters.

```
2 ?- [badprog].
% badprog compiled 0.00 sec, 2,760 bytes

Yes
3 ?- ([[t,h,e],[c,a,t],[i,s],[o,n],[t,h,e],[m,a,t]] ?~ [ip]).
```

```
Yes
```

Now to integrate the syntax and the phonetic grammar, let's modify the inference rules of our recognizer simply by adding a new "scan" rule that will notice that when we are trying to find a syntactic atom – now represented by a list of characters – then we should try to parse it as a sequence of phones using our transducer. Before we do the actual dictionary lookup, we put the characters back together with the built-in command `atom_chars`, since this is what or lexicon uses (we will change this in our next solution to the problem).

```
1 ?- atom_chars(cat,Chars).   % just to see how this built-in predicate works

Chars = [c, a, t] ;

No
2 ?- atom_chars(Word,[c, a, t]).

Word = cat ;

No
```

OK, so we extend our inference system with the one extra scan rule that parses the syntactic atoms phonetically, like this:

```
[] ?~ [].
(S0 ?~ Goals0) :- infer(S0,Goals0,S,Goals), (S ?~ Goals).

infer(S,[A|C],S,DC) :- (A :~ D), append(D,C,DC).   % ll
infer([W|S],[W|C],S,C).                            % scan
infer(Phones,[[Char|Chars]|C],Rest,C) :-           % parse words
        atom_chars(Word,[Char|Chars]),
        append([Phon|Phons],Rest,Phones),
        lex(Word,[Phon|Phons]).
```

Now we can test the result.

```
1 ?- [badprog].
%  col12 compiled 2.95 sec, 13,338,448 bytes
% badprog compiled 2.96 sec, 13,342,396 bytes

Yes
2 ?- (['D','@',k,'&',t,'I',z,'0', n,'D','@',m,'&',t] ?~ [ip]).

Yes
3 ?- (['D','@',k,'&',t,'I',z,'0', n,'D','@'] ?~ [ip]).

No
4 ?- (['D','@',k,'&',t] ?~ [dp]).

Yes
5 ?-
```

It works!

Problem (4), Solution 2: We can generate a more efficient representation of the dictionary this way:

```
1 ?- [col12].
%  col12 compiled 2.13 sec, 52 bytes

Yes
2 ?- tell('col12r.pl'),lex(Word,[B|C]),atom_chars(Word,Chars),
     portray_clause(lex(B,C,Chars)),fail;told.

No
```

You do not need to know about the special prolog facilities that are used here, but in case you are interested, here is a quick explanation.

The built-in command `tell` causes all output to be written to the specified file; then each clause of lex is written in the new format using the built-in command `portray_clause` – which is just like write except that its argument is followed by a period, etc. so that it can be read as a clause; then the fail causes the prolog to backtrack an find the next clause, and the next and so on, until all of them are found. When all ways of proving the first conjuncts fail, then the built-in command `told` closes the file that was opened by `tell` and succeeds.

After executing this command, we have a new representation of the lexicon that has clauses like this in it:

```
...
lex(&, [d, m, @, r, @, l, t, 'I'], [a, d, m, i, r, a, l, t, y]).
lex(&, [d, m, @, r, eI, 'S', n], [a, d, m, i, r, a, t, i, o, n]).
lex(@, [d, m, aI, @, 'R'], [a, d, m, i, r, e]).
lex(@, [d, m, aI, @, d], [a, d, m, i, r, e, d]).
lex(@, [d, m, aI, r, @, 'R'], [a, d, m, i, r, e, r]).
lex(@, [d, m, aI, r, @, z], [a, d, m, i, r, e, r, s]).
lex(@, [d, m, aI, @, z], [a, d, m, i, r, e, s]).
lex(@, [d, m, aI, @, r, 'I', 'N'], [a, d, m, i, r, i, n, g]).
lex(@, [d, m, aI, @, r, 'I', 'N', l, 'I'], [a, d, m, i, r, i, n, g, l, y]).
lex(@, [d, m, 'I', s, @, b, 'I', l, 'I', t, 'I'], [a, d, m, i, s, s, i, b, i, l, i, t, y]).
lex(@, [d, m, 'I', s, @, b, l], [a, d, m, i, s, s, i, b, l, e]).
lex(@, [d, m, 'I', 'S', n], [a, d, m, i, s, s, i, o, n]).
lex(@, [d, m, 'I', 'S', n, z], [a, d, m, i, s, s, i, o, n, s]).
lex(@, [d, m, 'I', t], [a, d, m, i, t]).
lex(@, [d, m, 'I', t, s], [a, d, m, i, t, s]).
lex(@, [d, m, 'I', t, n, s], [a, d, m, i, t, t, a, n, c, e]).
...
```

This lexicon is more efficiently accessed because the first symbol of the phonetic transcription is exposed as the first argument.

We just need to modify slightly the translate program to use this new representation of the dictionary:

```
%File: medprog.pl

% first, we load the first 2 columns of the Mitton lexicon in the new format
:- [col12r].

%translate(Phones,Words)
translate([],[]).
translate(Phones,[Word|Words]) :-
        append([First|MorePhones],RestPhones,Phones),  %% minor change here
        lex(First,MorePhones,Chars),                   %% minor change here
        atom_chars(Word,Chars),
        translate(RestPhones,Words).
```

We get a session that looks like this:

```
1 ?- [medprog].
%  col12r compiled 3.68 sec, 17,544,304 bytes
% medprog compiled 3.68 sec, 17,545,464 bytes

Yes
2 ?- translate(['D','@',k,'&',t,'I',z,'O', n,'D','@',m,'&',t],Words).

Words = [the, cat, is, on, the, 'Matt'] ;

Words = [the, cat, is, on, the, mat] ;

Words = [the, cat, is, on, the, matt] ;

No
3 ?-
```

Part b of the problem asks us to integrate this kind of translation into the syntax. Using the same syntax from the previous solution, we just need a slightly different scan rule:

```
[] ?~ [].
(S0 ?~ Goals0) :- infer(S0,Goals0,S,Goals), (S ?~ Goals).

infer(S,[A|C],S,DC) :- (A :~ D), append(D,C,DC).   % ll
infer([W|S],[W|C],S,C).                            % scan
infer(Phones,[[Char|Chars]|C],Rest,C) :-
        append([Phon|Phons],Rest,Phones),
        lex(Phon,Phons,[Char|Chars]).              % minor changes here
```

Now we can test the result.

```
1 ?- [medprog].
%  col12r compiled 3.73 sec, 17,544,304 bytes
% medprog compiled 3.73 sec, 17,548,376 bytes

Yes
2 ?- (['D','@',k,'&',t,'I',z,'O', n,'D','@',m,'&',t] ?~ [ip]).

Yes
3 ?- (['D','@',k,'&',t,'I',z,'O', n,'D','@'] ?~ [ip]).

No
4 ?- (['D','@',k,'&',t] ?~ [dp]).

Yes
5 ?-
```

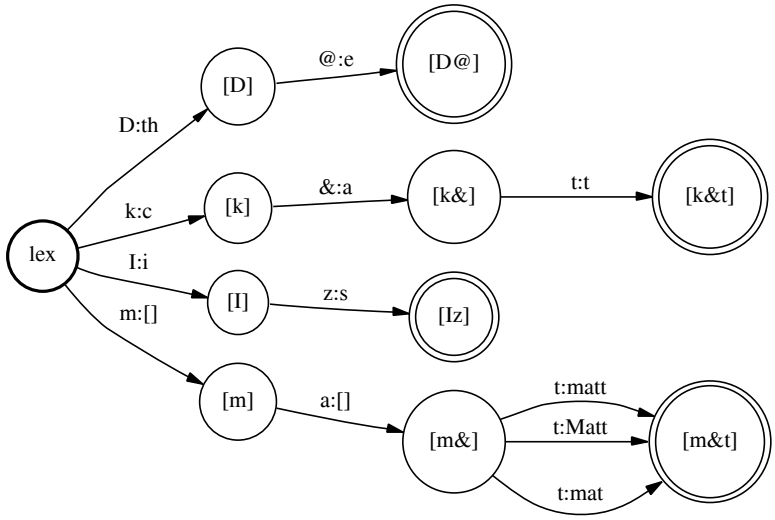
It works!

Problem (4), Solution 3: To get more efficient lookup, we can represent our dictionary as a tree. Prolog is not designed to take advantage of this kind of structure, but it is still valuable to get the idea of how it could be done in principle. We will only do it for a tiny fragment of the dictionary for illustration.

Consider the following entries from Mitton:

```
lex('the',['D','@']).
lex('cat',['k','&','t']).
lex('cat-nap',['k','&','t','y','n','&','p']).
lex('is',['I','z']).
lex('island',['aI','l','@','n','d']).
lex('on',['O','n']).
lex('mat',['m','&','t']).
lex('matt',['m','&','t']).
lex('Matt',['m','&','t']).
```

We can represent this dictionary with the following prefix transducer that maps phones to spelling as follows:



As discussed in class, in order to represent a finite state transducer, which is, in effect, a grammar with "output," we will label all the categories of the morphological component with terms of the form:

```
category(output)
```

So then the machine drawn above corresponds to the following grammar:

```
lex([t,h|Rest]) :~ ['D','[D]'(Rest)].
lex([c|Rest]) :~ [k,'[k]'(Rest)].
lex([i|Rest]) :~ ['I','[I]'(Rest)].
lex([o|Rest]) :~ ['O','[O]'(Rest)].  % in Mitton notation, that's a zero
lex(Rest) :~ [m,'[m]'(Rest)].
'[D]'(['e'|Rest]) :~ ['@','[D@]'(Rest)].
'[D@]'([]) :~ [].
'[k]'([a|Rest]) :~ ['&','[k&]'(Rest)].
'[k&]'([t|Rest]) :~ [t,'[k&t]'(Rest)].
'[k&t]'([]) :~ [].
'[I]'([s|Rest]) :~ [z,'[Iz]'(Rest)].
'[Iz]'([]) :~ [].
```

```
'[O]'([n|Rest]) :~ [n,'[On]'(Rest)].
'[On]'([]) :~ [].
'[m]'(Rest) :~ ['&','[m&]'(Rest)].
'[m&]'(Rest) :~ [t,'[m&t]'(Rest)].
'[m&t]'([m,a,t]) :~ [].
'[m&t]'([m,a,t,t]) :~ [].
'[m&t]'(['M',a,t,t]) :~ [].
```

With can test this grammar this way:

```
2 ?- [goodprog].
% goodprog compiled 0.00 sec, 2,760 bytes

Yes
3 ?- (['D','@'] ?~ [lex(W)]).

W = [t, h, e] ;

No
4 ?- ([m,'&',t] ?~ [lex(W)]).

W = [m, a, t] ;

W = [m, a, t, t] ;

W = ['M', a, t, t] ;

No
5 ?- ([m,'&',t,'D'] ?~ [lex(W)]).

No
6 ?-
```

(It's always a good idea to test your axioms with both positive and negative cases like this!)

Now let's extend this to part b of the problem. We can use the same syntax again, and simply modify the "scan" to notice when we are trying to find a syntactic atom – now represented by a list of characters – then we should try to parse it as a sequence of phones using our transducer.

```
[] ?~ [].
(S0 ?~ Goals0) :- infer(S0,Goals0,S,Goals), (S ?~ Goals).

infer(S,[A|C],S,DC) :- (A :~ D), append(D,C,DC).   % ll
infer([W|S],[W|C],S,C).                            % scan
infer(Phones,[[Char|Chars]|C],Rest,C) :-
        append([Phon|Phons],Rest,Phones),
        ([Phon|Phons] ?~ [lex([Char|Chars])]).    % minor change here
```

Now we can test the result.

```
1 ?- [goodprog].
% goodprog compiled 0.00 sec, 6,328 bytes

Yes
```

```
2 ?- (['D','@',k,'&',t,'I',z,'O', n,'D','@',m,'&',t] ?~ [ip]).

Yes
3 ?- (['D','@',k,'&',t,'I',z,'O', n,'D','@'] ?~ [ip]).

No
4 ?- (['D','@',k,'&',t] ?~ [dp]).

Yes
5 ?-
```

It works!

3 Extensions of the top-down recognizer

3.1 Unification grammars

(1) How should agreement relations be captured in a grammar? We actually already have a powerful mechanism available for this: instead of "propositional grammars" we can use "predicate grammars" where the arguments to the predicates can define subcategorizing features of each category.

We explore this idea here, since it is quite widely used, before considering the idea from transformational grammar that agreement markers are heads of their own categories (Pollock 1994, Sportiche 1998, many others).

(2) Consider the following grammar:

```
% g2.pl
:- op(1200,xfx,:~).

ip :~ [dp(Per,Num), vp(Per,Num)].
dp(1,s) :~ ['I'].                 dp(2,s) :~ [you].                 dp(3,s) :~ [it].
                                                                    dp(3,s) :~ [she].
                                                                    dp(3,s) :~ [he].
dp(3,Num) :~ [d1(Num)].           d1(Num) :~ [d0(Num), np(Num)].    d0(_Num) :~ [the].
d0(p) :~ [most].                  d0(s) :~ [every].                 d0(p) :~ [few].
np(Num) :~ [n1(Num)].             n1(Num) :~ [n0(Num)].             n0(s) :~ [penguin].
                                                                    n0(p) :~ [penguins].
vp(Per,Num) :~ [v1(Per,Num)].     v1(Per,Num) :~ [v0(Per,Num)].     v0(1,s) :~ [sing].
                                                                    v0(2,s) :~ [sing].
                                                                    v0(3,s) :~ [sings].
                                                                    v0(3,p) :~ [sing].
```

With this grammar `g2.pl` I produced the following session:

```
1 ?- [td],[g2].
td compiled, 0.00 sec, 1,116 bytes.
g2 compiled, 0.01 sec, 2,860 bytes.

Yes
2 ?- [every,penguin,sings] ?~ [ip].

Yes
3 ?- [every,penguins,sing] ?~ [ip].

No
4 ?- [it,sing] ?~ [ip].

No
5 ?- [it,sings] ?~ [ip].

Yes
6 ?- [the,penguin,sings] ?~ [ip].

Yes
7 ?- [the,penguin,sing] ?~ [ip].

No
8 ?- [the,penguins,sings] ?~ [ip].

No
```

(3) Dalrymple and Kaplan (2000), Bayer and Johnson (1995), Ingria (1990), and others have pointed out that agreement seems not always to have the "two way" character of unification. That is, while in English, an ambiguous word can be resolved only in one way, this is not always true:

a. The English *fish* is ambiguous between singular and plural, and cannot be both:
   The fish who eats the food gets/*get fat
   The fish who eat the food *gets/get fat
   (This is what we expect if *fish* has a number feature that gets unified with one particular value.)

b. The Polish wh-pronoun *kogo* is ambiguous between accusative and genitive case, and can be both:
   Kogo Janek lubi a Jerzy nienawidzi?
   who Janek likes and Jerzy hates
   (*lubi* requires ACC object and *nienawidzi* requires GEN object.)

c. The German *was* is ambiguous between accusative and nominative case, and can be both:
   Ich habe gegessen was übrig war.
   I have eaten what left was
   (The German *gegessen* requires ACC object and *übrig war* needs a NOM subject.)

Dalrymple and Kaplan (2000) propose that what examples like the last two show is that feature values should not be atoms like *sg, pl* or *nom, acc, gen* but (at least in some cases) sets of atoms.

3.2 More unification grammars: case features

(4) We can easily extend the grammar `g2.pl` to require subjects to have nominative case and objects, accusative case, just by adding a case argument to `dp`:

```
% g3.pl
:- op(1200,xfx,:~).

ip :~ [dp(P,N,nom), i1(P,N)].  i1(P,N) :~ [i0, vp(P,N)].       i0 :~ [].

dp(1,s,nom) :~ ['I'].          dp(2,_,_) :~ [you].             dp(3,s,nom) :~ [she].
                                                               dp(3,s,nom) :~ [he].
                                                               dp(3,s,_) :~ [it].
                                                               dp(3,p,nom) :~ [they].

dp(1,s,acc) :~ ['me'].                                         dp(3,s,acc) :~ [her].
                                                               dp(3,s,acc) :~ [him].
                                                               dp(3,p,acc) :~ [them].

dp(3,s,_) :~ [titus].          dp(3,s,_) :~ [tamora].          dp(3,s,_) :~ [lavinia].

dp(3,N,_) :~ [d1(N)].          d1(N) :~ [d0(N), np(N)].        d0(_) :~ [the].
                                                               d0(p) :~ [most].
d0(s) :~ [every].              d0(s) :~ [some].                d0(p) :~ [few].
np(N) :~ [n1(N)].              n1(N) :~ [n0(N)].               n0(s) :~ [penguin].
                                                               n0(p) :~ [penguins].
                               n0(s) :~ [song].                n0(p) :~ [songs].
vp(P,N) :~ [v1(P,N)].          v1(P,N) :~ [v0(intrans,P,N)].
                               v1(P,N) :~ [v0(trans,P,N),dp(_,_,acc)].

v0(_,1,_) :~ [sing].           v0(_,2,_) :~ [sing].            v0(_,3,s) :~ [sings].
                                                               v0(_,3,p) :~ [sing].

v0(trans,1,_) :~ [praise].     v0(trans,2,_) :~ [praise].      v0(trans,3,s) :~ [praises].
                                                               v0(trans,3,p) :~ [praise].

v0(intrans,1,_) :~ [laugh].    v0(intrans,2,_) :~ [laugh].     v0(intrans,3,s) :~ [laughs].
                                                               v0(intrans,3,p) :~ [laugh].
```

(5) The coordinate structure *Tamora and Lavinia* is plural. We cannot get this kind of construction with rules like the following because they are left recursive, and so problematic for TD:

```
dp(_,p,K) :~ [dp(_,_,K), coord(dp(_,_,K))].  % nb: left recursion
vp(P,N) :~ [vp(P,N), coord(vp(P,N))].        % nb: left recursion
coord(Cat) :~ [and,Cat].
```

We will want to move to a recognizer that allows these, but notice that TD does allow the following restricted case of coordination:[15]

```
dp(_,p,K) :~ [dp(_,s,K), coord(dp(_,_,K))].
coord(Cat) :~ [and,Cat].
```

(6) With the simple grammar above (including the non-left-recursive coord rules), we have the following session:

```
| ?- [td,g3].
td compiled, 0.00 sec, 1,116 bytes.
g3 compiled, 0.01 sec, 5,636 bytes.

| ?- [they,sing] ?~ [ip].

yes
| ?- [them,sing] ?~ [ip].

no
| ?- [they,praise,titus] ?~ [ip].

yes
| ?- [they,sing,titus] ?~ [ip].

yes
| ?- [he,sing,titus] ?~ [ip].

no
| ?- [he,sings,titus] ?~ [ip].

yes
| ?- [he,praises,titus] ?~ [ip].

yes
| ?- [he,praises] ?~ [ip].

no
| ?- [he,laughs] ?~ [ip].

yes
| ?- [he,laughs,titus] ?~ [ip].

no
| ?- [few,penguins,sing] ?~ [ip].

yes
| ?- [few,penguins,sings] ?~ [ip].

no
| ?- [some,penguin,sings] ?~ [ip].

yes
| ?- [you,and,'I',sing] ?~ [ip].

yes
| ?- [titus,and,tamora,and,lavinia,sing] ?~ [ip].

yes
```

[15] We are here ignoring the fact that, for most speakers, the coordinate structure *Tamora or Lavinia* is singular. We are also ignoring the complex interactions between determiner and noun agreement that we see in examples like this:

a. Every cat and dog is/*are fat

b. All cat and dog *is/*are fat

c. The cat and dog *is/are fat

3.3 Recognizers: time and space

Given a recognizer, a (propositional) grammar Γ, and a string $s \in \Sigma^*$,

(7) a proof that s has category $c \in N$ has **space complexity** k iff the goals on the right side of the deduction ("the workspace") never have more than k conjuncts.

(8) For any string $s \in \Sigma^*$, we will say that s has **space complexity** k iff for every category A, every proof that s has category A has space complexity k.

(9) Where S is a set of strings, we will say S has **space complexity** k iff every $s \in S$ has space complexity k.

(10) Set S **has (finitely) bounded memory requirements** iff there is some finite k such that S has space complexity k.

(11) The proof that s has category c has **time complexity** k iff the number of proof steps that can be taken from c is no more than k.

(12) For any string $s \in \Sigma^*$, we will say that s has **time complexity** k iff for every category A, every proof that s has category A has complexity k.

3.3.1 Basic properties of the top-down recognizer

(13) The recognition method introduced last time has these derivation rules:

$$G, \Gamma, S \vdash G \text{ [axiom]} \qquad \text{for definite clauses } \Gamma, \text{ goal } G, S \subseteq \Sigma^*$$

$$\frac{G, \Gamma, S \vdash (?\text{-}p, C)}{G, \Gamma, S \vdash (?\text{-}q_1, \ldots, q_n, C)} \qquad \text{if } (p\text{:-}q_1, \ldots, q_n) \in \Gamma$$

$$\frac{G, \Gamma, pS \vdash (?\text{-}p, C)}{G, \Gamma, S \vdash (?\text{-}C)}\ [scan]$$

To prove that a string $s \in \Sigma^*$ has category a given grammar Γ, we attempt to find a deduction of the following form, where [] is the empty string:

goal		theory		resources		workspace
?-a	,	Γ	,	s	$\vdash$	?-a
...						
?-a	,	Γ	,	[]	$\vdash$	$\square$

Since this defines a top-down recognizer, let's call this logic TD.

(14) There is exactly one TD deduction for each derivation tree. That is: $s \in yield(G, A)$ has n leftmost derivations from A iff there n TD proofs that s has category A.

(15) Every right branching $RS \subseteq yield(G, A)$ has bounded memory requirements in TD.

(16) No infinite left branching $LS \subseteq yield(G, A)$ has bounded memory requirements in TD.

(17) If there is any left recursive derivation of s from A, then the problem of showing that s has category A has infinite space requirements in TD, and prolog may not terminate.

(18) Throughout our study, we will keep an eye on these basic properties of syntactic analysis algorithms which are mentioned in these facts:

1. First, we would like our deductive system to be **sound** (if s can be derived from A, then $\Box$ can be deduced from axioms s and goal ?-A) and **complete** (if $\Box$ can be deduced from axioms s and goal ?-A, then s can be derived from A), and we also prefer to avoid **spurious ambiguity**. That is, we would like there to be n proofs just in case there are n corresponding derivations from the grammar.
2. Furthermore, we would prefer for there to be a substantial subset of the language that can be recognized with finite **memory**.
3. Finally, we would like the **search space** for any particular input to be finite.

(19) Let's call the grammar considered earlier, $G1$, implemented in `g1.pl` as follows:

```
:- op(1200,xfx,:~).
ip :~ [dp, i1].        i1 :~ [i0, vp].        i0 :~ [will].
dp :~ [d1].            d1 :~ [d0, np].        d0 :~ [the].
np :~ [n1].            n1 :~ [n0].            n0 :~ [idea].
                       n1 :~ [n0, cp].
vp :~ [v1].            v1 :~ [v0].            v0 :~ [suffice].
cp :~ [c1].            c1 :~ [c0, ip].        c0 :~ [that].
```

(20) Let's call this top-down, backtracking recognizer, considered last time, td.pl:

```
/*
* file: td.pl = ll.pl
*
*/
:- op(1200,xfx,:~ ). % this is our object language "if"
:- op(1100,xfx,?~ ). % metalanguage provability predicate

[] ?~ [].
(S0 ?~ Goals0) :- infer(S0,Goals0,S,Goals), (S ?~ Goals).

infer(S,[A|C], S,DC) :- (A :~ D), append(D,C,DC). % ll
infer([A|S],[A|C], S,C). % scan

append([],L,L).
append([E|L],M,[E|N]) :- append(L,M,N).
```

(21) We can store the grammar in separate files, `g1.pl` and `td.pl`, and load them both:

```
1 ?- [td,g1].
td compiled, 0.00 sec, 1,116 bytes.
g1 compiled, 0.00 sec, 1,804 bytes.
Yes
2 ?- [the,idea,will,suffice] ?~[ip].
Yes
3 ?- [the,idea,that,the,idea,will,suffice,will,suffice] ?~[ip].
Yes
4 ?- [will,the,idea,suffice] ?~[ip].
No
5 ?- halt.
```

(22) Suppose that we want to extend our grammar to get sentences like:

a. The elusive idea will suffice

b. The idea about the idea will suffice

c. The idea will suffice on Tuesday

We could add the rules:

```
n1 :~ [adjp, n1].      n1 :~ [n1, pp].      i1 :~ [i1,pp].
adjp :~ [adj1].        adj1 :~ [adj0].      adj0 :~ [elusive].
pp :~ [p1].            p1 :~ [p0,dp].       p0 :~ [about].
```

The top left production here is right recursive, the top middle and top right productions are left recursive. If we add these left recursive rules to our grammar, the search space for every input axiom is infinite, and consequently our prolog implementation may fail to terminate.

3.4 Trees, and parsing: first idea

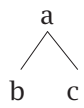
The goal of syntactic analysis is not just to compute whether a string of words is an expression of some category, but rather to compute a structural description for every grammatical string. Linguists typically represent their structural descriptions with trees or bracketings. Since we are already doing recognition by computing derivations, it will be a simple matter to compute the corresponding derivation trees. First, though, we need a notation for trees.

(23) To represent trees, we will use the '/' to represent a kind of immediate domination, but we will let this domination relation hold between a node and a sequence of subtrees.

Prolog allows the binary function symbol '/' to be written in infix notation (since prolog already uses it in some other contexts to represent division).

So for example, the term `a/[]` represents a tree with a single node, labelled `a`, not dominating anything.

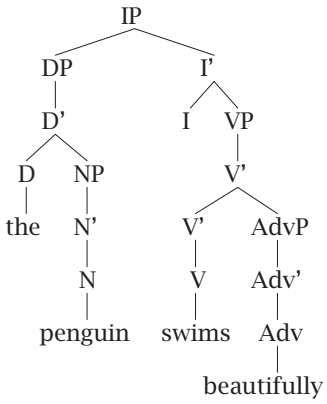
The term `a/[b/[],c/[]]` represents the tree that we would draw this way:



And the term (using quotes so that categories can be capitalized without being variables),

```
'IP'/[  'DP'/[ 'D'''/[ 'D'/[the/[]],
        'NP'/[ 'N'''/[ 'N'/[penguin/[]]]]],
        'I'''/[ 'I'/[],
        'VP'/[ 'V'''/[ 'V'''/[ 'V'/[swims/[]]],
        'AdvP'/[ 'Adv'''/[ 'Adv'/[beautifully/[]]]]]]]]
```

represents the tree:



3.5 The top-down parser

(24) Any TD proof can be represented as a tree, so let's modify the TD provable ~ predicate so that it not only finds proofs, but also builds tree representations of the proofs that it finds.

(25) Recall that the TD ?~ predicate is defined this way:

```
/*
 * file: td.pl = ll.pl
 *
 */
:- op(1200,xfx,:~).   % this is our object language "if"
:- op(1100,xfx,?~).   % provability predicate

[] ?~ [].
(S0 ?~ Goals0) :- infer(S0,Goals0,S,Goals), (S ?~ Goals).

infer(S,[A|C], S,DC) :- (A :~ D), append(D,C,DC).   % ll
infer([A|S],[A|C], S,C).                            % scan

append([],L,L).
append([E|L],M,[E|N]) :- append(L,M,N).
```

The predicate ?~ takes 2 arguments: the list of lexical axioms (the "input string") and the list of goals to be proven, respectively.

In the second rule, when subgoal A expands to subgoals D, we want to build a tree that shows a node labeled A dominating these subgoals.

(26) The parser is trickier; going through it carefully will be left as an optional exercise. We add a third argument in which to hold the proof trees.

```
/*
 * file: tdp.pl = llp.pl
 */
:- op(1200,xfx,:~).   % this is our object language "if"
:- op(1100,xfx,?~).   % provability predicate
:- op(500,yfx,@).     % metalanguage functor to separate goals from trees

[] ?~ []@[].
(S0 ?~ Goals0@T0) :- infer(S0,Goals0@T0,S,Goals@T), (S ?~ Goals@T).

infer(S,[A|C]@[A/DTs|CTs],S,DC@DCTs) :- (A :~ D), new_goals(D,C,CTs,DC,DCTs,DTs). \% ll
infer([A|S],[A|C]@[A/[]|CTs],S,C@CTs).                                           \% scan

%new_goals(NewGoals,OldGoals,OldTrees,AllGoals,AllTrees,NewTrees)
new_goals([],Gs,Ts,Gs,Ts,[]).
new_goals([G|Gs0],Gs1,Ts1,[G|Gs2],[T|Ts2],[T|Ts]) :- new_goals(Gs0,Gs1,Ts1,Gs2,Ts2,Ts).
```

In this code new_goals really does three related things at once. In the second clause of ?~, for example, the call to new_goals

i. appends goals D and C to obtain the new goal sequence DC;

ii. for each element of D, it adds a tree T to the list CTs of trees, yielding DCTs; and

iii. each added tree T is also put into the list of trees DTs corresponding to D.

(27) With this definition, if we also load the following theory,

```
p :~ [q,r].
q :~ [].
r :~ [].
```

then we get the following session:

```
| ?- [] ?~ [p]@[T].

T = p/[q/[], r/[]] ;

No
| ?- [] ?~ [p,q]@[T0,T].

T0 = p/[q/[], r/[]]
T = q/[] ;

No
```

What we are more interested in is proofs from grammars, so here is a session showing the use of our simple grammar `g1.pl` from page 45:

```
| ?- [tdp,g1].

Yes
| ?- [the,idea,will,suffice] ?~ [ip]@[T].

T = ip/[dp/[d1/[d0/[the/[]], np/[n1/[n0/[idea/[]]]]]], i1/[i0/[will/[]], vp/[v1/[v0/[suffice/[]]]]]
```

3.6 Some basic relations on trees

3.6.1 "Pretty printing" trees

(28) Those big trees are not so easy to read! It is common to use a "pretty printer" to produce a more readable text display. Here is the simple pretty printer:

```
/*
 * file: pp_tree.pl
 */

pp_tree(T) :- pp_tree(T, 0).

pp_tree(Cat/Ts, Column) :- !, tab(Column), write(Cat), write(' /['), pp_trees(Ts, Column).
pp_tree(X, Column) :- tab(Column), write(X).

pp_trees([], _) :- write(']').
pp_trees([T|Ts], Column) :-  NextColumn is Column+4, nl, pp_tree(T, NextColumn), pp_rest_trees(Ts, NextColumn).

pp_rest_trees([], _) :- write(']').
pp_rest_trees([T|Ts], Column) :- write(','), nl, pp_tree(T, Column), pp_rest_trees(Ts, Column).
```

The only reason to study the implementation of this pretty printer is as an optional exercise prolog. What is important is that we be able to use it for the work we do that is more directly linguistic.

(29) Here is how to use the pretty printer:

```
| ?- [tdp,g1,pp_tree].

Yes
| ?- ([the,idea,will,suffice] ?~ [ip]@[T]),pp_tree(T).
ip /[
    dp /[
        d1 /[
            d0 /[
                the /[]],
            np /[
                n1 /[
                    n0 /[
                        idea /[]]]]]],
    i1 /[
        i0 /[
            will /[]],
        vp /[
            v1 /[
                v0 /[
                    suffice /[]]]]]]
T = ip/[dp/[d1/[d0/[the/[]],np/[n1/[...]]]],i1/[i0/[will/[]],vp/[v1/[v0/[...]]]]] ?

Yes
```

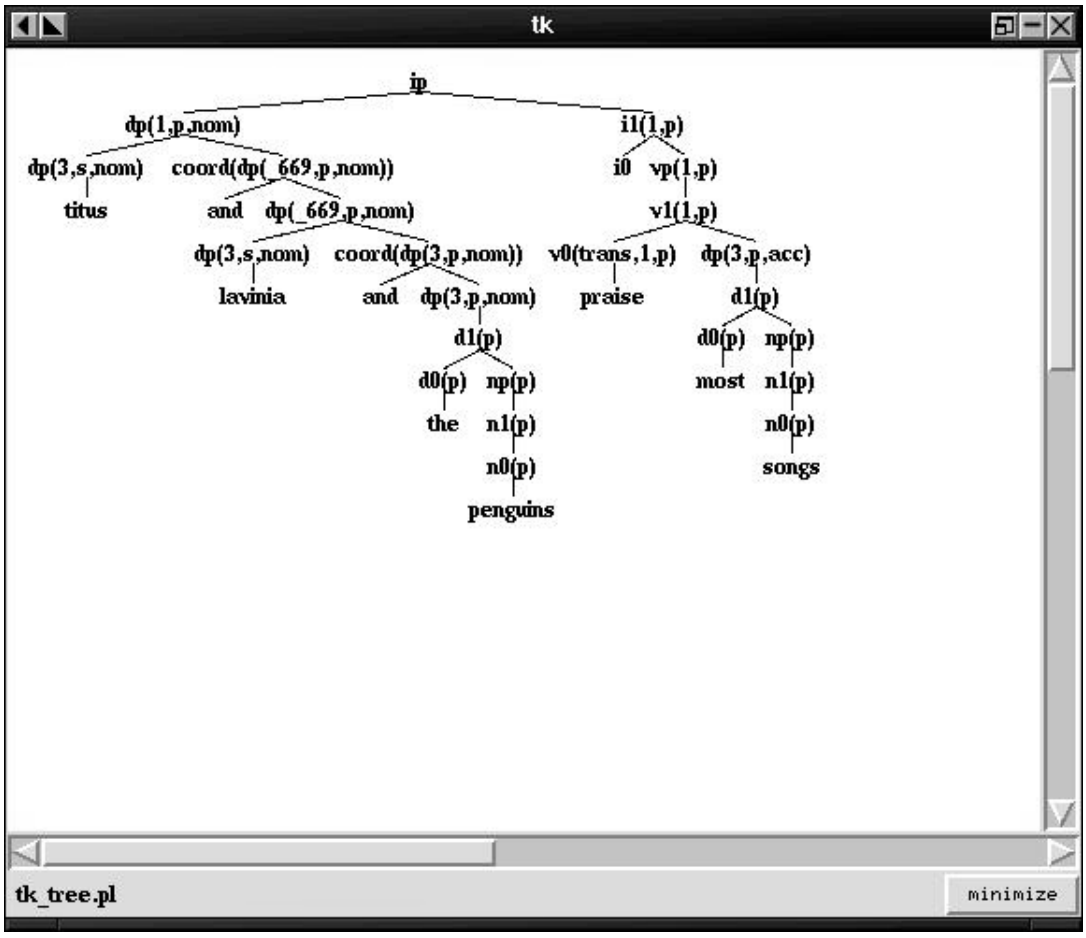
This pretty printer is better than nothing, but really, we can do better!

This is the kind of stuff people had to look at when computers wrote their output to electric typewriters. We can do much better now.

(30) There are various graphical tools that can present your tree in a much more readable format. I will describe using the Tcl/Tk interface which sicstus provides, but I also have tools for drawing trees from prolog through xfig, dot, TeX, and some others.

```
| ?- [tdp,g3,wish_tree].

Yes
| ?- ([titus,and,lavinia,and,the,penguins,praise,most,songs] ?^ [ip]@[T]),wish_tree(T).

T = ip/[dp(1,p,nom)/[dp(3,s,nom)/[titus/[]],coord(dp(_A,p,nom))/[and/[],dp(_A,p,nom)/[dp(...)/[...],coord(...)/[...|...]]]],i1(1,p)/[i0/[],vp(1,p)/[v1(1,p)/[v0(...)/[..

Yes
```

But what appears on your screen will be something like this:



Not the Mona Lisa, but this is only week 4. Notice that with this tool, carefully inspecting the trees becomes much less tedious!

(31) **tcl/tk** tree display on win32 systems (Windows 95, 98, NT, 2000)

a. I went to `http://dev.scriptics.com/software/tcltk/download82.html` and downloaded the install file `tcl823.exe`.

Clicking on this, I let it unpack in the default directory, which was `c:\Program Files\Tcl`

In `c:\Program Files\Tcl\bin` there is a program called: `wish82.exe`. I added `c:\Program Files\Tcl\bin` to my PATH.

This is the program that I use to display trees from SWI Prolog.

NB: If you install one of the more recent versions of tcl/tk, they should still work. But to use them with our wish_tree predicate, you will have to (i) find out the name of your wish executable (the equivalent of our `wish82.exe`, and then (ii) replace occurrences of `wish82.exe` in `wish_tree` with that name.

b. I installed swiprolog, and put the icon for `c:\Program Files\pl\bin\plwin.exe` on my desktop.

c. Clicking on this icon, I set the properties of this program so that it would run in my prolog directory, which is `c:\pl`

d. Then I downloaded all the win32 SWI-Prolog files from the webpage into my prolog directory, `c:\pl`

e. Then, starting pl from the icon on the desktop, I can execute

```
?- [wish_tree.pl].

?- wish_tree(a/[b/[],c/[]]).
```

This draws a nice tree in a wish window.

f. TODO: Really, we should provide a proper tk interface for SWI Prolog, or else an implementation of the tree display in XPCE. If anyone wants to do this, and succeeds, please share the fruits of your labor!

3.6.2 Structural relations

(32) Many of the structural properties that linguists look for are expressed as relations among the nodes in a tree. Here, we make a first pass at defining some of these relations. The following definitions all identify a node just by its label.
So for example, with the definition just below, we will be able to prove that `ip` is the root of a tree even if that tree also contains `ip` constituents other than the root. We postpone the problem of identifying nodes uniquely, even when their labels are not unique.

(33) The relation between a tree and its root has a trivial definition:

```
root(A,A/L).
```

(34) Now consider the parent relation in trees. Using our notation, it can also be defined very simply, as follows:

```
parent(A,B,A/L) :- member(B/_,L).
parent(A,B,_/L) :- member(Tree,L), parent(A,B,Tree).
```

(35) Domination is the transitive closure of the `parent` relation. Notice how the following definition avoids left recursion. And notice that, since no node is a parent of itself, no node dominates itself. Consequently, we also define `dominates_or_eq`, which is the reflexive, transitive closure of the `parent` relation. Every node, in every tree stands in the `dominates_or_eq` relation to itself:

```
dominates(A,B,Tree) :- parent(A,B,Tree).
dominates(A,B,Tree) :- parent(A,C,Tree), dominates(C,B,Tree).

dominates_or_eq(A,A,_).
dominates_or_eq(A,B,Tree) :- dominates(A,B,Tree).
```

(36) We now define the relation between subtrees and the tree that contains them:

```
subtree(T/Subtrees,T/Subtrees).
subtree(Subtree,_/Subtrees) :- member(Tree,Subtrees),subtree(Subtree,Tree).
```

(37) A is a **sister** of B iff A and B are not the same node, and A and B have the same parent.
Notice that, with this definition, no node is a sister of itself. To implement this idea, we use the important relation `select`, which is sort of like `member`, except that it removes a member of a list and returns the remainder in its third argument. For example, with the following definition, we could prove `select(b,[a,b,c],[a,c])`.

```
sisters(A,B,Tree) :-
        subtree(_/Subtrees,Tree),
        select(A/_,Subtrees,Remainder),
        member(B/_,Remainder).

select(A,[A|Remainder],Remainder).
select(A,[B|L],[B|Remainder]) :- select(A,L,Remainder).
```

(38) Various "command" relations play a very important role in recent syntax. Let's say that A **c-commands** B iff A is not equal to B, neither dominates the other, and every node that dominates A dominates B.[16] This is equivalent to the following more useful definition:

[16] This definition is similar to the one in Koopman and Sportiche (1991), and to the IDC-command in Barker and Pullum (1990). But notice that our definition is irreflexive – for us, no node c-commands itself.

A c-commands B iff B is a sister of A, or B is dominated by a sister of A.
This one is easily implemented:

```
c_commands(A,B,Tree) :- sisters(A,AncB,Tree), dominates_or_eq(AncB,B,Tree).
```

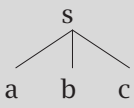
(39) The relation between a tree and the string of its leaves, sometimes called the **yield** relation, is a little bit more tricky to define. I will present a definition here, but not discuss it any detail. (Maybe in discussion sections…)

```
yield(Tree,L) :- yield(Tree,[],L).

yield(W/[], L, [W|L]).
yield(_/[T|Ts], L0, L) :- yields([T|Ts],L0,L).

yields([], L, L).
yields([T|Ts], L0, L) :- yields(Ts,L0,L1), yield(T,L1,L).
```

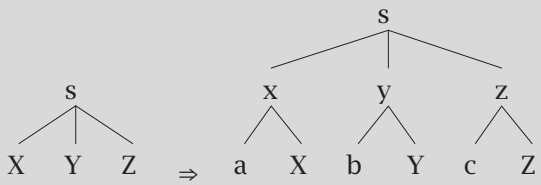
NB: Notice that this does not distinguish empty categories with no yield from terminal vocabulary with no yield.

3.7 Tree grammars

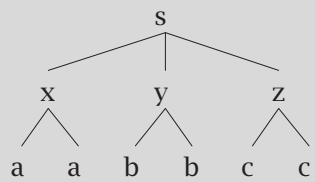
(40) The rules we have been considering so far rewrite strings. But it is not hard to formulate rules that rewrite trees. Suppose for example that we have a tree:



(41) Suppose that we want to expand the category *s* in this tree with a rule that could be schematically expressed as follows, where X, Y, Z are variables standing in for any subtrees:



(42) If we apply this rule to the particular tree we began with, we get the result:



We could apply the rule again to this tree, and so on.

(43) We can define the rule that expands any node `s` with its 3 subtrees like this.
In our notation, our initial tree is: `s/[a/[],b/[],c/[]]` Let this be the only "start tree" (there could be more than one), and consider the set of trees that includes this tree and all the other trees that can be obtained from this tree by applications of the rule.
In Prolog, we can define this set easily. Here is one way to do it:

```
% oktree.pl

ok_tree(s/[a/[],b/[],c/[]]).
ok_tree(s/[x/[a/[],X],y/[b/[],Y],z/[c/[],Z]]) :- ok_tree(s/[X,Y,Z]).
```

(44) The first axiom says that the start tree is allowed as a tree in the tree language, an `ok_tree` (there is only one start tree in this case). The second axiom says that the result of applying our rule to any *ok_tree* is also allowed as an `ok_tree`. Loading just this 2 line definition we can prove:

```
| ?- [ok_tree].
consulted /home/es/tex/185-00/ok_tree.pl in module user, 10 msec 896 bytes

yes
| ?- ok_tree(A).

A = s/[a/[],b/[],c/[]] ? ;

A = s/[x/[a/[],a/[]],y/[b/[],b/[]],z/[c/[],c/[]]] ? ;

A = s/[x/[a/[],x/[a/[],a/[]]],y/[b/[],y/[b/[],b/[]]],z/[c/[],z/[c/[],c/[]]]] ?

yes
```

Combining this definition of `ok_tree` with our previous definition of the `yield` relation, we can prove:

```
| ?- ok_tree(A), yield(A,L).

A = s/[a/[],b/[],c/[]],
L = [a,b,c] ? ;

A = s/[x/[a/[],a/[]],y/[b/[],b/[]],z/[c/[],c/[]]],
L = [a,a,b,b,c,c] ? ;

A = s/[x/[a/[],x/[a/[],a/[]]],y/[b/[],y/[b/[],b/[]]],z/[c/[],z/[c/[],c/[]]]],
L = [a,a,a,b,b,b,c,c,c] ? ;

...
```

(45) So we see that we have defined a set of trees whose yields are the language $a^n b^n c^n$, a language (of strings) that cannot be generated by a simple ("context free") phrase structure grammar of the familiar kind.

(46) Does any construction similar to $a^n b^n c^n$ occur in any natural language? There are arguments that English is not context free, but the best known arguments consider parts of English which are similar to $a^i b^j a^i b^j$ or the language

$$\{xx \mid x \textit{ any nonempty string of terminal symbols}\}$$

These languages are not context-free. Purported examples of this kind of thing occur in phonological/morphological reduplication, in simplistic treatments of the "respectively" construction in English; in some constructions in a Swiss-German dialect. Some classic discussions of these issues are reprinted in Savitch et al. (1987).

(47) Tree grammars and automata that accept trees have been studied extensively (Gecseg and Steinby, 1984), particularly because they allow elegant logical (in fact, model-theoretic) characterizations (Cornell and Rogers, 1999). Tree automata have also been used in the analysis of non-CFLs (Mönnich, 1997; Michaelis, Mönnich, and Morawietz, 2000; Rogers, 2000).

Problem Set:

1. Grammar G1, implemented in `g1.pl` on page 45 and used by `td.pl`, is neither right-branching nor left-branching, so our propositions (15) and (16) do not apply. Does L_{G1} have finite space complexity? If so, what is the finite complexity bound? If not, why is there no bound?
2. Download `g1.pl` and `td.pl` to your own machine. Then extend the grammar in `g1.pl` in a natural way, with an empty I and an inflected verb, to accept the sentences:
   a. The idea suffices
   b. The idea that the idea suffices suffices

   Turn in a listing of the grammar and a session log showing this grammar being used with `td.pl`, with a brief commentary on what you have done and how it works in the implementation.
3. Stowell (1981) points out that the treatment of the embedded clauses in Grammar 1 (which is implemented in `g1.pl`) is probably a mistake. Observe in the first place, that when a derived nominal takes a DP object, *of* is required,
   c. John's claim of athletic superiority is warranted.
   d. * John's claim athletic superiority is warranted.

   But when a *that*-clause appears, we have the reverse pattern:
   e. * John's claim of that he is a superior athlete is warranted.
   f. John's claim that he is a superior athlete is warranted.

   Stowell suggests that the *that*-clauses in these nominals are not complements but appositives, denoting propositions. This fits with the fact that identity claims with *that*-clauses are perfect, and with the fact that whereas events can be witnessed, propositions cannot be:
   g. John's claim was that he is a superior athlete.
   h. I witnessed John's claiming that he is a superior athlete.
   i. * I witnessed that he is a superior athlete.

   Many other linguists have come to similar conclusions. Modify the grammar in `g1.pl` so that *The idea that the idea will suffice will suffice* does not have a `cp` in complement position. Turn in a listing of the grammar and a session log showing this grammar being used with `td.pl`, with a brief commentary on what you have done and how it works in the implementation.
4. Linguists have pointed out that the common treatment of `pp` adjunct modifiers proposed in the phrase structure rules in (22) is probably a mistake. Those rules allow any number of `pp` modifiers in an NP, which seems appropriate, but the rules also have the effect of placing later `pp`'s higher in the NP. On some views, this conflicts with the binding relations we see in sentences like
   j. The picture of Bill_1 near his_1 house will suffice.
   k. The story about $[\text{my mother}]_1$ with her_1 anecdotes will amuse you.

   One might think that the pronouns in these sentences should be c-commanded by their antecedents. Modify the proposed phrase structure rules to address this problem. Turn in a listing of the resulting grammar and a session log showing this grammar being used with `td.pl`, with a brief commentary on what you have done and how it works in the implementation.
5. Linguists have also pointed out that the common treatment of `adjp` adjunct modifiers proposed in (22) is probably a mistake. The proposed rule allows any number of adjective phrases to occur in an `np`, which seems appropriate, but this rule does not explain why prenominal adjective phrases cannot have complements or adjuncts:
   l. * The elusive to my friends idea will suffice.

m. The idea which is elusive to my friends will suffice.

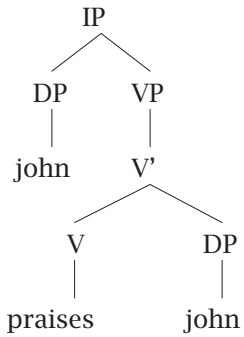
n. * The frightened of my mother friends will not visit.

o. The friends who are frightened of my mother will not visit.

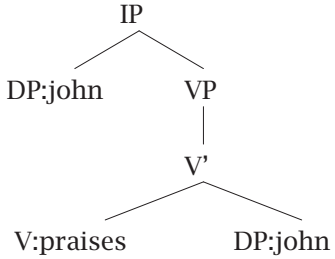
The given rules also do not fit well with the semantic idea that modifiers take the modifiees as argument (Keenan, 1979; Keenan and Faltz, 1985) – which is usually indicated in the syntax by a head with its arguments in complement position. Abney (1987) proposes just this alternative idea about prenominal adjective modifiers: they are functional categories inside DP that obligatorily select nominal NP or AP complements. Since NP or another AP is the complement in these constructions, they cannot take another complement as well, as we see in examples l-o. Modify the proposed phrase structure rules along these lines. Turn in a listing of the resulting grammar and a session log showing this grammar being used with `td.pl`, with a brief commentary on what you have done and how it works in the implementation.

6. There are various reasons that our measure of space complexity is not really a very good measure of the computational resources required for top-down recognition. Describe some of them.

7. a. Write a propositional context free grammar for tdp.pl that generates the following tree from the input axioms `[john,praises,john]`, and test it with either `pp_tree` or `wish_tree` (or both) before submitting the grammar. (Get exactly this tree.)



b. Notice that in the previous tree, while the arcs descending from IP, VP and V' are connecting these constituents to their parts, the arcs descending from the "pre-lexical" categories DP and V are connecting these constituents to their phonetic/orthographic forms. It is perhaps confusing to use arcs for these two very different things, and so it is sometimes proposed that something like the following would be better:



Is there a simple modification of your grammar of the previous question that would produce this tree from the input axioms `[john,praises,john]`? If so, provide it. If not, briefly explain.

8. Define a binary predicate `number_of_nodes(T,N)` which will count the number `N` of nodes in any tree `T`. So for example,

```
| ?- number_of_nodes(a/[b/[],c/[]],N).

N=3

Yes
```

   Test your definition to make sure it works, and then submit it.

9. Write a grammar that generates infinitely many sentences with pronouns and reflexive pronouns in subject and object positions, like
   `he praises titus and himself`
   `himself praises titus`

   a. Define a predicate `cc_testa` that is true of all and only parse trees in which every reflexive pronoun is c-commanded by another DP that is not a reflexive pronoun, so that
      `titus praises himself`
      is OK but
      `himself praises titus`
      is not OK. Test the definition with your grammar and `tdp`.

   b. Define a predicate `cc_testb` that is true of all and only parse trees in which no pronoun is c-commanded by another DP, so that
      `he praises titus`
      is OK but
      `titus praises him`
      is not OK.[17] Test the definition with your grammar and `tdp`.

10. As already mentioned on page 24, human languages frequently have various kinds of "reduplication." The duplication or copying of an earlier part of a string requires access to memory of a kind that CFGs cannot provide, but tree grammars *can*.
    Write a tree grammar for the language

$$\{xx \mid x \textit{ any nonempty string of terminal symbols}\}$$

    where the terminal symbols are only a and b. This is the language:

$$\{aa, bb, abab, baba, aaaa, bbbb, \ldots\}.$$

    Implement your tree grammar in Prolog, and test it by computing some examples and their yields, as we did in the previous section for $a^n b^n c^n$, before submitting the grammar.

[17] Of course, what we should really do is to just block binding in the latter cases, but for the exercise, we just take the first step of identifying the configurations where binding is impossible.

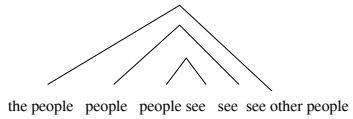
4 Brief digression: simple patterns of dependency

4.1 Human-like linguistic patterns

Human languages apparently show lots of nested dependencies. We get this when we use put a relative clause after the subject of a sentence, where the relative clause itself has a subject and predication which can be similarly modified:

the people see other people
the people [people see] see other people
the people [people [people see] see] see other people
…

Placing an arc between each subject and the verb it corresponds to, we get this "nested pattern:"



This kind of pattern is defined by context-free grammars for the language $\{a^n b^n \mid n \geq 0\}$, like the following one:

```
% anbncn.pl

'S' :~ [a,'S',b].
'S' :~ [].
```

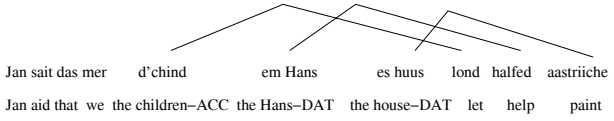
We also find crossing dependencies in human languages, for example, when a string is "reduplicated" – something which happens at the word level in many languages – or where the objects of verbs appear in the order

$$O_1\ O_2\ O_3\ V_1\ V_2\ V_3.$$

Dutch has crossing dependencies of this sort which are semantically clear, though not syntactically marked (Huybregts, 1976; Bresnan et al., 1982). Perhaps the most uncontroversial case of syntactically marked crossing dependencies is found in the Swiss German collected by Shieber (1985):

Jan säit das mer d'chind em Hans es huus lönd hälfed aastriiche
John said that we the children Hans the house let help paint
'John said that we let the children help Hans paint the house'

The dependencies in this construction are crossing, as we can see in the following figure with an arc from each verb to its object:



In Swiss German, the dependency is overtly marked by the case requirements of the verbs: *hälfe* requires dative case, and *lönd* and *aastriiche* require accusative.

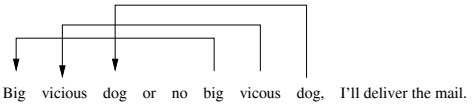
CFLs are closed under intersection with regular languages. But if we assume that there is no bound on the depth of embedding in Swiss German constructions like those shown here, then the intersection of Swiss German with the regular language,

Jan säit das mer (d'chind)* (em Hans)* hænd wele (laa)* (hälfe)* aastriiche
Jan says that we the children Hans have wanted let help paint

is the following language:

Jan säit das mer (d'chind)^i (em Hans)^j hænd wele (laa)^i (hälfe)^j aastriiche.

Some dialects of English have constructions that strongly favor perfect copying (Manaster-Ramer, 1986), which also involves crossing dependencies.



The formal language $\{xx \mid x \in \{a,b\}^*\}$ is a particularly simple formal example of crossing dependencies like this. It is easily defined with a unification grammar like this one:

```
% xx.pl

'S' :~ ['A'(X),'A'(X)].

'A'([a|X]) :~ [a,'A'(X)].
'A'([b|X]) :~ [b,'A'(X)].
'A'([]) :~ [].
```

We can use this grammar in sessions like this:

```
~/tex/185 1%pl
Welcome to SWI-Prolog (Version 5.0.8)
Copyright (c) 1990-2002 University of Amsterdam.
1 ?- [tdp,xx].
\% tdp compiled 0.00 sec, 1,968 bytes
\% xx compiled 0.00 sec, 820 bytes

Yes
2 ?- ([a,b,a,b]?~['S']@[T]).

T = 'S'/['A'([a, b])/[a/[], 'A'([b])/[b/[], 'A'(...)/[]]], 'A'([a, b])/[a/[], 'A'([b])/[b/[], ...

No
3 ?- ([a,b,a]?~['S']@[T]).

No
5 ?-
```

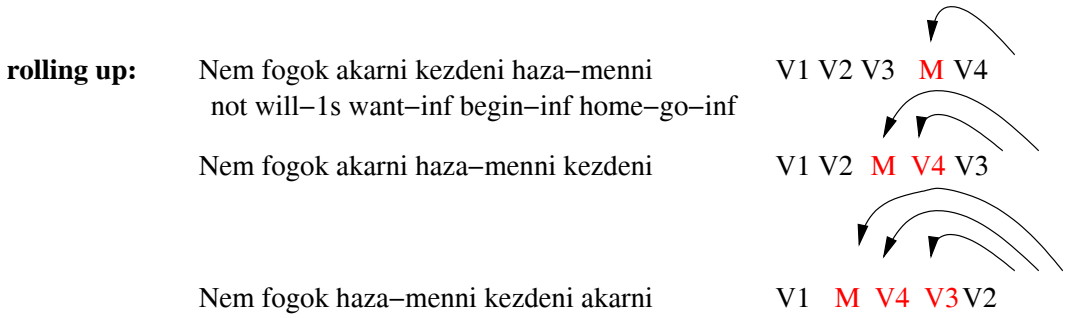
Developing an observation of Kenesei, Koopman and Szabolcsi observe the following pattern in negated or focused sentences of Hungarian, schematized on the right where "M" is used to represent the special category of "verbal modifiers" like *haza-*:

(48) Nem fogok akarni kezdeni haza-menni V1 V2 V3 M V4
not will-1s want-inf begin-inf home-go-inf

(49) Nem fogok akarni haza-menni kezdeni V1 V2 M V4 V3
not will-1s want-inf home-go-inf begin-inf

(50) Nem fogok haza-menni kezdeni akarni V1 M V4 V3 V2
not will-1s begin-inf want-inf home-go-inf

One analysis of verbal clusters in Hungarian (Koopman and Szabolcsi, 2000a) suggests that they "roll up" from the end of the string as shown below:



[M] moves around V4, then [M V4] rolls up around V3, then [M V4 V3] rolls up around V2,...It turns out that this kind of derivation can derive complex patterns of dependencies which can yield formal languages like $\{a^n b^n c^n | \; n \geq 0\}$, or even $\{a^n b^n c^n d^n e^n | \; n \geq 0\}$ – any number of counting dependencies. We can define these languages without any kind of "rolling up" constituents if we help ourselves to (unboundedly many) feature values and unification:

```
% anbncn.pl

'S' :~ ['A'(X),'B'(X),'C'(X)].

'A'(s(X)) :~ [a,'A'(X)].    'B'(s(X)) :~ [b,'B'(X)].    'C'(s(X)) :~ [c,'C'(X)].
'A'(0) :~ [].               'B'(0) :~ [].               'C'(0) :~ [].
```

4.2 Semilinearity and some inhuman linguistic patterns

In the previous section we saw grammars for $\{a^n b^n | \; n \geq 0\}$, $\{xx | \; x \in \{a,b\}^*\}$, and $\{a^n b^n c^n | \; n \geq 0\}$. These languages all have a basic property in common, which can be seen by counting the number of symbols in each string of each language.

For example,

$$\{a^n b^n | \; n \geq 0\} = \{\epsilon, ab, aabb, aaabbb, \ldots\}$$

, we can use (x, y) to represent x a's and y'bs, so we see that the strings in this language have the following counts:

$$\{(0,0),(1,1),(2,2),\ldots\} = \{(x,y) | \; x = y\}.$$

For $\{xx | \; x \in \{a,b\}^*\}$, we have all pairs $\mathbb{N} \times \mathbb{N}$. For $\{a^n b^n c^n | \; n \geq 0\}$ we have the set of triples $\{(x,y,z) | \; x = y = z\}$. If we look at just the number of a's in each language, considering the set of values of first coordinates of the tuples, then we can list those sets by value, obtaining in all three cases the sequence:

$$0, 1, 2, 3, \ldots$$

The patterns of dependencies we looked at above will not always give us the sequence $0, 1, 2, 3, \ldots$, though. For example, the language

$$\{(ab)^n (ba)^n | \; n \geq 0\} = \{\epsilon, abba, ababbaba, \ldots\}$$

also has nested dependencies just like $\{a^n b^n | \ n \geq 0\}$, but this time the number of a's in words of the language is

$$0, 2, 4, 6, \ldots$$

Plotting position in the sequence against value, these sets are both linear.

Let's write scalar product of an integer k and a pair (x, y) this way:

$$k(x, y) = (kx, ky)$$

, and we add pairs in the usual way $(x, y) + (z, w) = (x + y, z + w)$. Then a set S of pairs (or tuples of higher arity) is said to be **linear** iff there are finitely many pairs (tuples) $v_0, v_1, \ldots, v_k$ such that

$$S = \{v_0 + \sum_{i=1}^{k} n v_i | \ n \in \mathbb{N}, 1 \leq i \leq k\}.$$

A set is **semilinear** iff it is the union of finitely many linear sets.

Theorem: Finite state and context free languages are semilinear

Semilinearity Hypothesis: Human languages are semilinear (Joshi, 1985)

Theorem: Many unification grammar languages are not semilinear!

Here is a unification grammar that accepts $\{a^{2^n} | \ n > 0\}$.

```
% apowtwon.pl

'S'(0) :~ [a,a].
'S'(s(X)) :~ ['S'(X),'S'(X)].
```

Michaelis and Kracht (1997) argue against Joshi's semilinearity hypothesis on the basis of the case markings in Old Georgian,[18] which we see in examples like these (cf also Boeder 1995, Bhatt&Joshi 2003):

(51) saidumlo-j igi sasupevel-isa m-is ɣmrt-isa-jsa-j
mystery-NOM the-NOM kingdom-GEN the-GEN God-GEN-GEN-NOM
'the mystery of the kingdom of God'

(52) govel-i igi sisxl-i saxl-isa-j m-is Saul-is-isa-j
all-NOM the-NOM blood-NOM house-GEN-NOM the-NOM Saul-GEN-GEN-NOM
'all the blood of the house of Saul'

Michaelis and Kracht infer from examples like these that in this kind of possessive, Old Georgian requires the embedded nouns to repeat the case markers on all the heads that dominate them, yielding the following pattern (writing K for each case marker):

$$[\mathrm{N}_1 - \mathrm{K}_1[\mathrm{N}_2 - \mathrm{K}_2 - \mathrm{K}_1[\mathrm{N}_3 - \mathrm{K}_3 - \mathrm{K}_2 - \mathrm{K}_1 \ldots [\mathrm{N}_\mathrm{n} - \mathrm{K}_\mathrm{n} - \ldots - \mathrm{K}_1]]]]$$

It is easy to calculate that in this pattern, when there are n nouns, there are $\frac{n(n+1)}{2}$ case markers. Such a language is not semilinear.

[18] A Kartevelian language with translations of the Gospel from the 5th century. Modern Georgian does not show the phenomenon noted here.

5 Trees, and tree manipulation: second idea

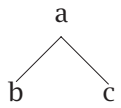
5.1 Nodes and leaves in tree structures

(1) The previous section introduced a standard way of representing non-empty ordered trees uses a two argument term `Label/Subtrees`.[19]

The argument `Label` is the label of the tree's root node, and `Subtrees` is the sequence of that node's subtrees.

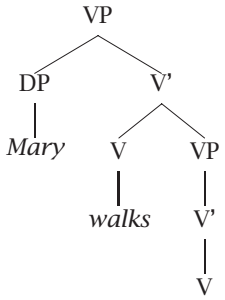
A tree consisting of a single node (necessarily a leaf node) has an empty sequence of subtrees.

For example, the 3 node tree with root labeled `a` and leaves labeled `b` and `c` is represented by the term `a/[b/[],c/[]]`:



While this representation is sufficient to represent arbitrary trees, it is useful extend it by treating phonological forms not as separate terminal nodes, but as a kind of annotation or feature of their parent nodes. This distinguishes "empty nodes" from leaf nodes with phonological content; only the latter possess (non-null) phonological forms. Thus in the tree fragment depicted above, the phonological forms `Mary` and `walks` are to be interpreted not as a separate nodes, but rather as components of their parent `DP` and `V` nodes.

(2) While this representation is sufficient to represent arbitrary trees, it is useful extend it by treating phonological forms not as separate terminal nodes, but as a kind of annotation or feature of their parent nodes. This distinguishes "empty nodes" from leaf nodes with phonological content; only the latter possess (non-null) phonological forms. Thus in the tree fragment depicted just below, the phonological forms *Mary* and *walks* are to be interpreted not as a separate nodes, but rather as components of their parent `DP` and `V` nodes.



There are a number of ways this treatment of phonological forms could be worked out. For example, the phonological annotations could be regarded as features and handled with the feature machinery, perhaps along the lines described in Pollard and Sag (1989, 1993). While this is arguably the formalization most faithful to linguists' conceptions, we have chosen to represent trees consisting of a single

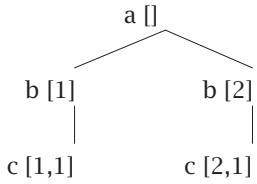
[19] This notation is discussed more carefully in Stabler (1992, p65).

node with a phonological form with terms of the form `Label/-Phon`, where `Label` is the node label and `Phon` is that node's phonological form. The tree fragment depicted above is represented by the term VP/[DP/ `-Mary`, V′/[`V/-walks`, VP/[V′/[V/[]]]]].

From a computational perspective, the primary advantage of this representation is that it provides a simple, structural distinction between (trees consisting of) a node with no phonological content and a node with no phonological content.

(3) As noted by Gorn (1969), a node can be identified by a sequence of integers representing the choices made on the path from the root to the node.

We can see how this method works in the following tree, while identifying nodes by their labels does not:



(4) Gorn's path representations of notes are difficult to reason about if the tree is constructed in a non-top-down fashion, so we consider a another representation is proposed by Hirschman and Dowding (1990) and modified slightly by Johnson (1991), Johnson and Stabler (1993).

A node is represented by a term

`node(Pedigree,Tree,Parent)`

where `Pedigree` is the integer position of the node with respect to its sisters, or else `root` if the node is root; `Tree` is the subtree rooted at this node, and `Parent` is the representation of the parent node, or else `none` if the node is root.

(5) Even though the printed representation of a node can be quadratically larger than the printed representation of the tree that contains it, it turns out that in most implementations structure-sharing will occur between subtrees so that the size of a node is linear in the size of the tree.

(6) With this representation scheme, the leftmost leaf of the tree above is represented by the term

`n(1,c/[],n(1,b/[c/[]],n(root,a/[b/[c/[]],b/[c/[]]],none)))`

(7) With this notation, we can define standard relations on nodes, where the nodes are unambiguously denoted by our terms:

```
% child(I, Parent, Child) is true if Child is the Ith child of Parent.

child(I, Parent, n(I,Tree,Parent)) :- Parent = n(_, _/ Trees, _), nth(I, Trees, Tree).

% ancestor(Ancestor, Descendant) is true iff Ancestor dominates Descendant.
% There are two versions of ancestor, one that works from the
% Descendant up to the Ancestor, and the other that works from the
% Ancestor down to the descendant.

ancestor_up(Ancestor, Descendant) :- child(_I, Ancestor, Descendant).
ancestor_up(Ancestor, Descendant) :- child(_I, Parent, Descendant), ancestor_up(Ancestor, Parent).

ancestor_down(Ancestor, Descendant) :- child(_I, Ancestor, Descendant).
ancestor_down(Ancestor, Descendant) :- child(_I, Ancestor, Child), ancestor_down(Child, Descendant).

% root(Node) is true iff Node has no parent
root(n(root,_,none)).

% subtree(Node, Tree) iff Tree is the subtree rooted at Node.
subtree(n(_,Tree,_), Tree).

% label(Node, Label) is true iff Label is the label on Node.
label(n(_,Label/_,_), Label).

% contents(Node, Contents) is true iff Contents is either
%   the phonetic content of node or the subtrees of node
contents(n(_,_/Contents,_),Contents).

% children(Parent, Children) is true if the list of Parent's
% children is Children.
```

```
children(Parent, Children) :- subtree(Parent, _/Trees), children(Trees, 1, Parent, Children).

children([], _, _, []).
children([Tree|Trees], I, Parent, [n(I,Tree,Parent)|Children]) :- I =< 3, I1 is I+1, children(Trees, I1, Parent, Children).

% siblings(Node, Siblings) is true iff Siblings is a list of siblings
% of Node.  The version presented here only works with unary and
% binary branching nodes.

siblings(Node, []) :- root(Node).          % Node has no siblings if Node is root
siblings(Node, []) :- children(_, [Node]). % Node has no siblings if it's an only child
siblings(Node, [Sibling]) :- children(_, [Node, Sibling]).
siblings(Node, [Sibling]) :- children(_, [Sibling, Node]).
```

5.2 Categories and features

(8) With these notions, we can set the stage for computing standard manipulations of trees, labeled with X-bar style categories:

`x(Category,Barlevel,Features,Segment)`

where `Category` is one of `{n,v,a,p,…}`, `Barlevel` is one of `{0,1,2}`, `Features` is a list of feature values, each of which has the form `Attribute:Value`, and `Segment` is – if the constituent is not a proper segment of an adjunction structure, and is + otherwise.

So with these conventions, we will write `x(d,2,[],-)/ -hamlet` instead of `dp/[hamlet/[]]`.

(9) The following definitions are trivial:

```
category(Node, Category) :- label(Node, x(Category,_,_,_)).
barlevel(Node, BarLevel) :- label(Node, x(_,BarLevel,_,_)).
features(Node, Features) :- label(Node, x(_,_,Features,_)).
extended(Node, EP) :-       label(Node, x(_,_,_,EP)).
no_features(Node) :-        features(Node, []).
```

(10) There are at least two ways of conceptualizing features and feature assignment processes.

First, we can treat features as marks on nodes, and distinguish a node without a certain feature from nodes with this feature (even if the feature's value is unspecified). This is the approach we take in this chapter. As we pointed out above, under this approach it is not always clear what it means for two nodes to "share" a feature, especially in circumstances where the feature is not yet specified on either of the nodes. For example, a node moved by Move-α and its trace may share the case feature (so that the trace "inherits" the case assigned to its antecedent), but if the node does not have a specified case feature before movement it is not clear what should be shared.

Another approach, more similar to standard treatments of features in computational linguistics, is to associate a single set of features with all corresponding nodes at all levels of representations. For example, a DP will have a case feature at D-structure, even if it is only "assigned" case at S-structure or LF. Under this approach it is straightforward to formalize feature-sharing, but because feature values are not assigned but only tested, it can be difficult to formalize requirements that a feature be "assigned" exactly once. We can require that a feature value is assigned at most once by associating each assigner with a unique identifier and recording the assigner with each feature value.[20] Surprisingly, it is an open problem in this approach how best to formalize the linguist's intuitions that a certain feature value value must be set somewhere in the derivation. If feature values are freely assigned and can be checked more than once, it is not even clear in a unification grammar what it means to require that a feature is "assigned" at least once.[21] The intuitive idea of feature-checking is more naturally treated in a resource-logical or formal language framework, as discussed in §9.

(11) To test a feature value, we define a 3-place predicate:

[20] The problem of requiring uniqueness of feature assignment has been discussed in various different places in the literature. Kaplan and Bresnan (1982) discuss the use of unique identifiers to ensure that no grammatical function is filled more than once. Stowell (1981) achieves a similar effect by requiring unique indices in co-indexation. Not all linguists assume that case is uniquely assigned; for example Chomsky and Lasnik (1993) and many more recent studies assume that a chain can receive case more than once.

[21] In work on feature structures, this problem is called the ANY value problem, and as far as I know it has no completely satisfactory solution. See, e.g. Johnson (1988) for discussion.

```
feature(Attribute, Node, Value) :- features(Node, Features), member(Attribute:Value, Features).
```

(12) Let's say that a `terminal` node is a leaf with phonetic content, and an `empty node` is a leaf with no phonetic content:

```
terminal(n(_,_/ -Word,_), Word).

empty(n(_,_/[],_)).

nonempty(Node) :- terminal(Node).
nonempty(Node) :- children(Node,[_|_]).
```

(13) To extend a set of features at a node, it is convenient to have:

```
add_feature(Node0, Attribute, Value, Node) :-
        category(Node0, Category), category(Node, Category),
        barlevel(Node0, Barlevel), barlevel(Node, Barlevel),
        extended(Node0, EP), extended(Node, EP),
        features(Node0, Features0), features(Node, Features),
        (       member(Attribute:Value0, Features0)
        ->      Value = Value0, Features = Features0
        ;       Features = [Attribute:Value|Features0]
        ).
```

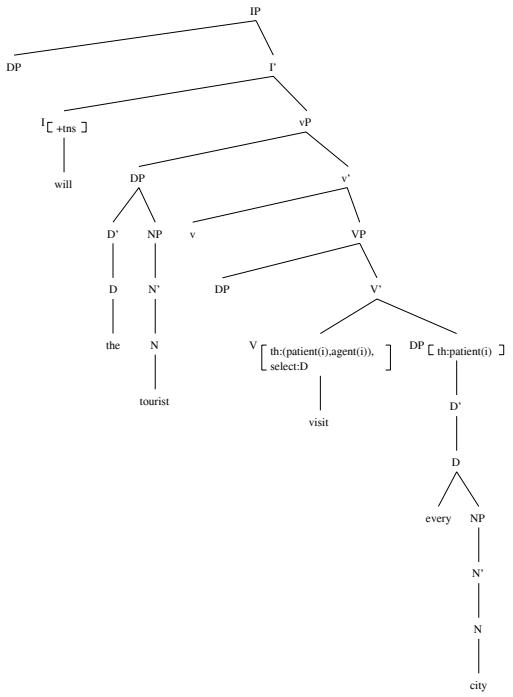
And to copy the values of a list of attributes:

```
copy_features([], OldFeatures, []).
copy_features([Att|Atts], OldFeatures, Features0) :-
        (         member(Att:Val, OldFeatures)
        ->        Features0 = [Att:Val|Features]
        ;         Features0 = Features
        ),
        copy_features(Atts, OldFeatures, Features).
```

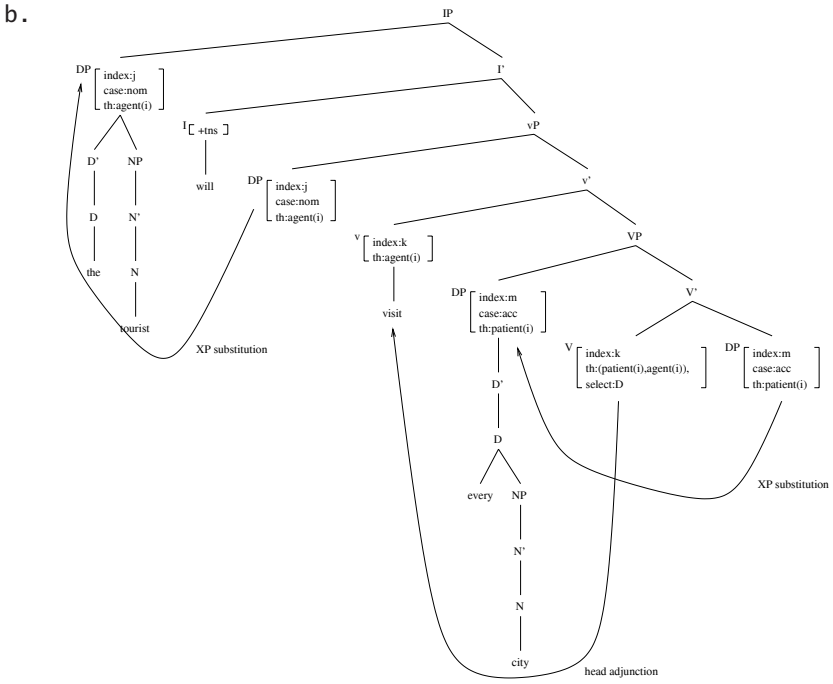
5.3 Movement relations

We can now define node replacement, and movement relations. For example, beginning with a structure like 14a, we might derive a pronounced ("spelled out") structure like 14b and a lf structure like 14c:

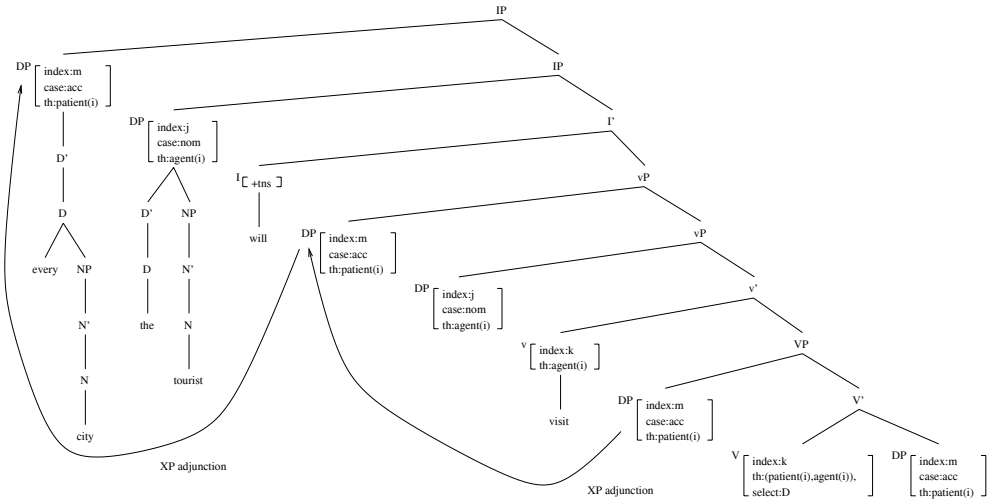
(14) a.



b.



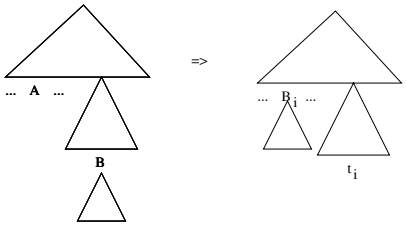
c.



Let's assume that there are two types of movement: substitutions and adjunctions. These involve only phrases (XPs) or heads (X0s); that is, only these levels of structure are "visible" to movement operations. And we will assume that both types of movements must be "structure preserving" in a sense to be defined.

5.3.1 Substitution

(15) A substitution moves a constituent to an empty constituent, a "landing site," elsewhere in the tree, leaving a co-indexed empty category behind:



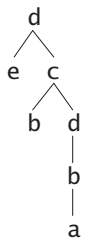
A substitution is often said to be "structure preserving" iff the moved constituent and the landing site have the same category (though this requirement is sometimes relaxed slightly, e.g. to allow V to substitute into an empty I position).

(16) First, we define a relation that holds between two sequences of nodes with the same pedigrees and the same subtrees:

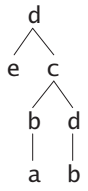
```
iso_subtrees([], []).
iso_subtrees([A|As], [B|Bs]) :- iso_subtree(A, B), iso_subtrees(As, Bs).

iso_subtree(NodeA, NodeB) :-
        subtree(NodeA, Tree),
        subtree(NodeB, Tree),
        same_pedigrees(NodeA,NodeB).

same_pedigrees(A,B) :- child(I,_,A),   child(I,_,B).
same_pedigrees(A,B) :- root(A), root(B).
```

(17) Since every node representation specifies the whole tree of which it is a part, we can define move-α directly on nodes, with the advantage the node representation offers of making the whole tree environment accessible at every point. In effect, the movement relations are defined by traversing an "input" tree from root to frontier, checking its correspondence with the "output" tree as fully as possible at every point.

a.



b.



Consider, for example, how we could substitute the non-empty b of in 17a into the position of the empty b, obtaining the tree in 17b. This involves two basic steps. First, we must replace the nonempty subtree b/[a/[]] by an empty subtree b/[], and then we must replace the other empty subtree b/[] by b/[a/[]]. Both steps involve modifying a tree just by replacing one of its subtrees by something else. We formalize this basic step first, as an operation on our special notation for nodes, with the predicate replace_node.

(18) We define replace_node(A,DescA,B,DescB) to hold just in case nodes A and B are subtree isomorphic except that where the subtree of the former has descendant DescA, the latter has descendant DescB:

```
replace_node(A, A, B, B).                       % A replaced by B
replace_node(A, DescendantA, B, DescendantB) :-  % DescA repl by DescB
        label(A, Label),
        label(B, Label),
        children(A, ChildrenA),
        children(B, ChildrenB),
        replace_nodes(ChildrenA, DescendantA, ChildrenB, DescendantB).
```

The first clause, in effect, just replaces the current node A by B, while the second clause uses the relation replace_nodes to do the replacement in exactly one of the children of the current node.

(19) We extend the previous relation to node sequences:

```
replace_nodes([A|As], DA, [B|Bs], DB) :-
        replace_node(A, DA, B, DB),
        iso_subtrees(As, Bs).
replace_nodes([A|As], DA, [B|Bs], DB) :-
        iso_subtree(A, B),
        replace_nodes(As, DA, Bs, DB).
```

With these axioms, we can establish some basic relations among trees. For example, with two basic replacement steps, we can transform the tree in Figure 17a into the tree in Figure 17b. The first step replaces the subtree b/[a/[]] by b/[], and the second step replaces the original b/[] by b/[a/[]].

Consider the first step, and since we are working with our special node representations, let's focus our attention just on the subtrees dominated by c, where the action is. Taking just this subtree, we establish a relation between the root nodes:

```
A=n(root,c/[b/[],d/[b/[a/[]]]],none),
B=n(root,c/[b/[],d/[b/[]]],none).
```

What we do to obtain B is to replace DescA in A by DescB, where

```
DescA=n(1,b/[a/[]],n(2,d/[b/[a/[]]],
                        n(root,c/[b/[],d/[b/[a/[]]]],none))),
DescB=n(1,b/[],n(2,d/[b/[]],
                        n(root,c/[b/[],d/[b/[]]],none))).
```

We can deduce that these elements stand in the relation

```
replace\_node(A,DescA,B,DescB).
```

(20) We now define substitution. As observed earlier, this kind of movement involves two basic node replacement steps. For this reason, it is convenient to define a relation which holds between root nodes after two such steps. We define

`twice_replace_node(A, DA1, DA2, B, DB1, DB2)`

to hold iff node `B` is formed by changing two distinct descendants in distinct subtrees of `A` as follows:

(i) replacing `DA1` in one subtree of `A` by the empty category `DB1`, and

(ii) replacing `DA2` by `DB2` in another subtree of `A`.

This is easily done.

```
twice_replace_node(A, DA1, DA2, B, DB1, DB2) :-
        label(A, Label),
        label(B, Label),
        children(A, ChildrenA),
        children(B, ChildrenB),
        twice_replace_nodes(ChildrenA, DA1, DA2, ChildrenB, DB1, DB2).

twice_replace_nodes([A|As], DA1, DA2, [B|Bs], DB1, DB2) :-
        replace_node(A, DA1, B, DB1),
        replace_nodes(As, DA2, Bs, DB2).
twice_replace_nodes([A|As], DA1, DA2, [B|Bs], DB1, DB2) :-
        replace_node(A, DA2, B, DB2),
        replace_nodes(As, DA1, Bs, DB1).
twice_replace_nodes([A|As], DA1, DA2, [B|Bs], DB1, DB2) :-
        twice_replace_node(A, DA1, DA2, B, DB1, DB2),
        iso_subtrees(As, Bs).
twice_replace_nodes([A|As], DA1, DA2, [B|Bs], DB1, DB2) :-
        iso_subtree(A, B),
        twice_replace_nodes(As, DA1, DA2, Bs, DB1, DB2).
```

Now we define the special linguistic requirements of the substitution operation, using a basic relation `substitution` and several auxiliary definitions which define the relationships among the nodes that are involved: the moved node, the landing site, and the trace.[22]

```
substitution(OldRoot, NewRoot, MovedNode, Trace) :-
        root(OldRoot),
        root(NewRoot),
        subst_landing(OldNode, EmptyNode),
        subst_moving(OldNode, MovedNode),
        trace(OldNode, Trace),
        twice_replace_node(OldRoot, OldNode, EmptyNode, NewRoot, Trace, MovedNode),
        copy_phi_features(OldNode, Trace0),
        add_feature(Trace0, index, I, Trace),
        copy_psi_features(OldNode, MovedNode0),
        add_feature(MovedNode0, index, I, MovedNode).

% subst_moving(OldNode, MovedNode)  iff OldNode and MovedNode have same
%        Cat,Bar,Level,EP features

subst_moving(OldNode, MovedNode) :-
        category(OldNode, Cat),         category(MovedNode, Cat),
        barlevel(OldNode, Bar),         barlevel(MovedNode, Bar),
        extended(OldNode, EP),          extended(MovedNode, EP),
        contents(OldNode, Contents),    contents(MovedNode, Contents).

% subst_landing(OldNode, EmptyNode)  iff OldNode and EmptyNode have same
%        Cat,Bar features, and EmptyNode is a visible nonterminal with
%        no children and no features

subst_landing(OldNode, EmptyNode) :-
        category(OldNode, Cat),       category(EmptyNode, Cat),
        barlevel(OldNode, Bar),       barlevel(EmptyNode, Bar),
        children(EmptyNode, []),
        features(EmptyNode, []),
        visible(EmptyNode).

% trace(OldNode, Trace)  iff OldNode and Trace have same Cat,Bar,EP features,
%        and Trace is a nonterminal with no children.

trace(OldNode, Trace) :-
        category(OldNode, Category),  category(Trace, Category),
        barlevel(OldNode, Barlevel),  barlevel(Trace, Barlevel),
        extended(OldNode, EP),        extended(Trace, EP),
```

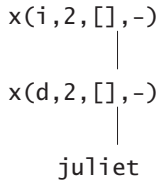
[22] The requirement that the empty node which is that landing site of the substitution have no features may be overly stringent. (This requirement is here imposed by the predicate `subst_landing`.) We could just require that the landing site have no index feature – prohibiting a sort of "trace erasure" (Freidin, 1978). If we remove the restriction on the landing site features altogether, the character of the system changes rather dramatically though, since it becomes possible to have "cycling" derivations of arbitrary length as discussed in Stabler (1992, §14.3). In the system described here, neither a trace nor a moved node can be a landing site.

```
        children(Trace, []).

% visible(Node) iff Node is maximal or minimal, and not a proper segment

visible(Node) :- extended(Node, -), barlevel(Node, 2).
visible(Node) :- extended(Node, -), barlevel(Node, 0).
```

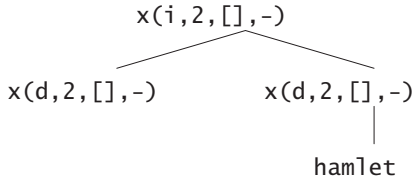
The predicate copy_phi_features, and the similar copy_psi_features are easily defined using our earlier predicate copy_features:

```
phi_features([person, number, case, wh, index, th, finite]).
psi_features([person, number, case, wh, index, th, finite,
        pronominal, anaphoric]).

copy_phi_features(Node0, Node) :-
        features(Node0, Features0), features(Node, Features),
        phi_features(Phi),
        copy_features(Phi, Features0, Features).

copy_psi_features(Node0, Node) :-
        features(Node0, Features0), features(Node, Features),
        psi_features(Psi),
        copy_features(Psi, Features0, Features).
```

(21) With these definitions, substitution cannot apply to the tree:



(22) The following tree, on the other hand, allows exactly one substitution:



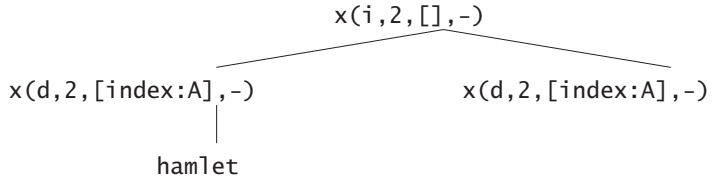
To avoid typing in the term that denotes this tree all the time, let's add the axiom:

```
tree(1,
   x(i,2,[],-)/[
      x(d,2,[],-)/[],
      x(d,2,[],-)/ -hamlet ]).
```

Then we can compute the substitution with a session like this:

```
| ?- tree(1,T),subtree(N,T),substitution(N,NewN,Moved,Trace),subtree(NewN,NewT),tk_tree(NewT).

N = n(root,x(i,2,[],-)/[x(d,2,[],-)/[],x(d,2,[],-)/ -(hamlet)],none),
T = x(i,2,[],-)/[x(d,2,[],-)/[],x(d,2,[],-)/ -(hamlet)],
NewN = n(root,x(i,2,[],-)/[x(d,2,[index:_A],-)/ -(hamlet),x(d,2,[index:_A],-)/[]],none),
NewT = x(i,2,[],-)/[x(d,2,[index:_A],-)/ -(hamlet),x(d,2,[index:_A],-)/[]],
Moved = n(1,x(d,2,[index:_A],-)/ -(hamlet),n(root,x(i,2,[],-)/[x(d,2,[index:_A],-)/ -(hamlet),x(d,2,[index:_A],-)/[]],none)),
Trace = n(2,x(d,2,[index:_A],-)/[],n(root,x(i,2,[],-)/[x(d,2,[index:_A],-)/ -(hamlet),x(d,2,[index:_A],-)/[]],none)) ?

yes
```

And the tree NewT gets displayed:



5.3.2 Adjunction

(23) Like substitution, adjunction basically involves two replacements, but adjunction is, unfortunately, quite a bit more complex. The main reason is that we can have adjunctions like the ones shown in 14c, where a node is extracted and adjoined to an ancestor. That means that one replacement is done inside one of the constituents affected by the other replacement. A second factor that slightly increases the complexity of the relation is that a new adjunction structure is built. It is no surprise, then, that the specifically linguistic restrictions on this operation are also slightly different from those on substitution.

(24) We define the relation `adjunction` in terms of the replacement relation `adjoin_node`. The latter relation is similar to `twice_replace_node`, but builds appropriate adjunction structures. These adjunction structures are defined slightly differently for the two basic situations: the more complex case in which one of the changes is inside a moved constituent, and the simpler case in which the two affected nodes are distinct. The other relations just define the relevant requirements on the nodes involved.

(25) So, to begin with, we define:

```
adjunction(OldRoot, NewRoot, Adjunct, Trace) :-
        root(OldRoot),                root(NewRoot),
        adjunct(OldNode, Adjunct),
        trace(Adjunct, Trace),
        adjunction_structure(AdjnctSite, Adjunct, _Segment, Adjunction),
        adjoin_node(OldRoot, OldNode, AdjnctSite, NewRoot, Trace, Adjunction),
        nonargument(AdjnctSite),
        copy_phi_features(OldNode, Trace0),
        add_feature(Trace0, index, I, Trace),
        copy_psi_features(OldNode, Adjunct0),
        add_feature(Adjunct0, index, I, Adjunct).
```

(26) The `Adjunct` part of the adjunction structure will be similar to the original node to be moved, `OldNode`, as follows:

```
adjunct(OldNode, Adjunct) :-
        category(OldNode, Category),  category(Adjunct, Category),
        barlevel(OldNode, Bar),       barlevel(Adjunct, Bar),
        extended(OldNode, EP),        extended(Adjunct, EP),
        contents(OldNode, Contents),  contents(Adjunct,Contents).
```

(27) Now we turn to the basic replacement operations. For substitution, these were trivial, but adjunction requires a more careful treatment. In the following definition, the first clause is essentially identical to the definition of `twice_replace_node`, but here we must add the second clause to cover the case where `A` is replaced by `DB2` after replacing `DA1` by `DB1` in a segment of `DB2`:

```
adjoin_node(A, DA1, DA2, B, DB1, DB2) :-
        label(A, Label),
        label(B, Label),
        children(A, ChildrenA),
        children(B, ChildrenB),
        adjoin_nodes(ChildrenA, DA1, DA2, ChildrenB, DB1, DB2).

adjoin_node(A, DA1, A, B, DB1, B) :-
        adjunction_structure(A, _Adjunct, Segment, B),
        lower_segment(A,LowerA),
        replace_node(LowerA, DA1, Segment, DB1).

lower_segment(A,LowerA) :-
        category(A,Cat),       category(LowerA,Cat),
        barlevel(A,Bar),       barlevel(LowerA,Bar),
        features(A,F),         features(LowerA,F),
        contents(A,Contents),  contents(LowerA,Contents),
        same_pedigrees(A,LowerA).
```

Notice that the features and the extended feature of `A` are not copied to `LowerA`: this just allows `LowerA` to match the lower segment of the adjunction structure.

(28) Adjunction of one node to another on a distinct branch of the tree is slightly less awkward to handle. Notice how similar the following definition is to the definition of `twice_replace_nodes`:

```
adjoin_nodes([A|As], DA1, DA2, [B|Bs], DB1, DB2) :-
        replace_node(A, DA1, B, DB1),
        replace_nodes(As, DA2, Bs, DB2),
        adjunction_structure(DA2, _Adjunct, Segment, DB2),
        features(DA2, Features), features(Segment, Features),
        contents(DA2, Contents), contents(Segment, Contents).
```

```
adjoin_nodes([A|As], DA1, DA2, [B|Bs], DB1, DB2) :-
        replace_node(A, DA2, B, DB2),
        replace_nodes(As, DA1, Bs, DB1),
        adjunction_structure(DA2, _Adjunct, Segment, DB2),
        features(DA2, Features), features(Segment, Features),
        contents(DA2, Contents), contents(Segment, Contents).

adjoin_nodes([A|As], DA1, DA2, [B|Bs], DB1, DB2) :-
        adjoin_node(A, DA1, DA2, B, DB1, DB2),
        iso_subtrees(As, Bs).

adjoin_nodes([A|As], DA1, DA2, [B|Bs], DB1, DB2) :-
        iso_subtree(A, B),
        adjoin_nodes(As, DA1, DA2, Bs, DB1, DB2).
```

(29) Finally, the whole adjunction structure is defined by its relations to the `AdjunctionSite` and the `Adjunct`, as follows:

```
adjunction_structure(AdjunctionSite, Adjunct, Segment, Adjunction) :-
         category(Adjunction, Cat),  category(AdjunctionSite, Cat),
                                     category(Segment, Cat),
         barlevel(Adjunction, Bar),  barlevel(AdjunctionSite, Bar),
                                     barlevel(Segment, Bar),
         extended(Adjunction, EP),   extended(AdjunctionSite, EP),
                                     extended(Segment, +),
         features(Adjunction, Fea),   features(Segment, Fea),
         right_or_left(Adjunct,Segment,Adjunction),
         visible(AdjunctionSite).

right_or_left(Adjunct,LowerSegment,AdjunctionStructure) :-       % left
        children(AdjunctionStructure, [Adjunct,LowerSegment]).
right_or_left(Adjunct,LowerSegment,AdjunctionStructure) :-       % right
        children(AdjunctionStructure, [LowerSegment,Adjunct]).
```

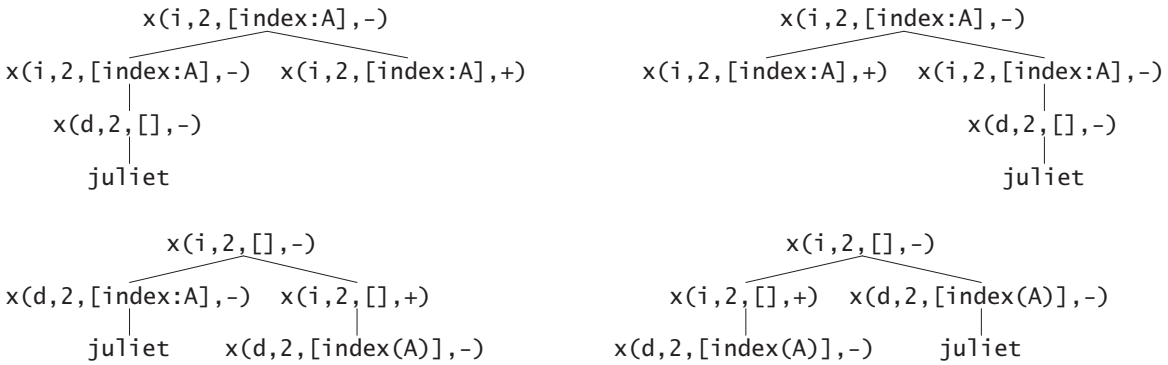
Notice that the contents and features of the lower `Segment` are not specified by `adjunction_structure`. They may not correspond exactly to the contents and features of the original `AdjunctionSite` because they may be changed by the replacement of `OldNode` by `Trace`.

(30) In some theories, like the one in Sportiche (1998a), adjunction is only possible to "non-argument" or A' categories, namely V, I, and A, so we could define:

```
nonargument(Node) :- category(Node,v).
nonargument(Node) :- category(Node,i).
nonargument(Node) :- category(Node,a).
```

(31) We observed above that no substitution is possible in the tree of (21). However, adjunction can apply to that structure. In fact, exactly four adjunctions are allowed by the definitions provided: we can left adjoin the IP to itself; we can right adjoin the IP to itself; we can left adjoin the DP to the IP; or we can right adjoin the DP to the IP.

These four results are shown here, in the order mentioned:



No adjunction of the DP to itself is possible, because DP is an argument.

But clearly, adjunction as formulated here can apply in very many ways, so any theory using it will have to restrict its application carefully.

5.3.3 Move-α

(32) Since a movement can be either a substitution or adjunction, let's say:

```
moveA(OldRoot, NewRoot) :- substitution(OldRoot, NewRoot, MovedNode, Trace), ccl(MovedNode,Trace).
moveA(OldRoot, NewRoot) :- adjunction(OldRoot, NewRoot, MovedNode, Trace), ccl(MovedNode,Trace).
```

The `ccl` predicate, axiomatized below, will hold if the movement satisfies CCL, the condition on chain links. Finally, the reflexive, transitive closure of this relation can be defined in the familiar way:

```
moveAn(Root, Root).
moveAn(DeepRoot, Root) :- moveA(DeepRoot, MidRoot), moveAn(MidRoot, Root).
```

The predicate `moveAn` corresponds closely to the usual notion of move-α.

5.3.4 Tree relations for adjunction structures

(33) The definition of `siblings` given above in 7 is simple, but it is purely geometric and does not pay attention to adjunction structures with segments.

(34) We need to extend the geometric notions of `parent, ancestor, siblings` to the related specialized notions: `imm_dominates, dominates, sister`. To do this, we need to be able to find the minimal and maximal segment of a node. Assuming binary branching, this can be done as follows:

```
maximal_segment(Node,Node) :-
        extended(Node,-).
maximal_segment(Node,MaxSegment) :-
        extended(Node,+),
        child(_,Parent,Node),
        maximal_segment(Parent,MaxSegment).

minimal_segment(Node,Node) :-
        children(Node,[]).
minimal_segment(Node,Node) :-
        children(Node,[Child]),
        extended(Child,-).
minimal_segment(Node,Node) :-
        children(Node,[ChildA,ChildB]),
        extended(ChildA,-),
        extended(ChildB,-).
minimal_segment(Node,MinSegment) :-
        child(_I,Node,Segment),
        extended(Segment,+),
        minimal_segment(Segment,MinSegment).
```

Notice that a node is a minimal segment iff it is not a parent of any proper segment (i.e. any node with a + extended feature).

(35) With these notions, the intended `dominates` and `excludes` relations are easily defined:

```
dominates(Node,Child) :-
        minimal_segment(Node,MinSegment),
        ancestor_down(MinSegment,Child).
excludes(NodeA,NodeB) :-
        maximal_segment(NodeA,MaximalSegment),
        $\backslash$+ ancestor_down(MaximalSegment,NodeB).
```

The predicate `ancestor` was defined earlier and can use these new definitions of domination.

(36) The `sister` and `imm_dominates` relations can be defined as follows, (assuming binary branching):

```
sister(Node,Sister) :-
        maximal_segment(Node,MaxSegment),
        siblings(MaxSegment,[Sister]),
        extended(Sister,-).
sister(Node,Sister) :-
        maximal_segment(Node,MaxSegment),
        siblings(MaxSegment,[Segment]),
        extended(Segment,+),
        imm_dominates(Segment,Sister).

imm_dominates(Node,Child) :-
        child(_I,Node,Child),
        extended(Child,-).
imm_dominates(Node,Child) :-
        child(_I,Node,Segment),
        extended(Segment,+),
        imm_dominates(Segment,Child).
```

(37) With these foundations, it is easy to formalize `i_command` – sometimes called c-command:

α i-commands β iff α is immediately dominated by an ancestor of β, and $\alpha \neq \beta$.

This is equivalent to the earlier formulation, since if the immediately dominating parent of Commander dominates Node, then every node dominating Commander dominates Node. In our formal notation:

```
i_commands(Commander,Node) :- dominates(Ancestor,Node), imm_dominates(Ancestor,Commander), \+ Commander=Node.
```

(38) Consider the top left tree in 31. In this tree, the root IP has adjoined to itself, consequently, the moved constituent has no sister. In fact, the node labeled `x(i,2,[index:A],-)` has no sister. The first node that dominates it is the adjunction structure, and that adjunction structure does not immediately dominate any other node. The trace is itself part of an extended adjunction structure, and has no sister, and no i-commander.

(39) We now have enough to define notions like L-marking, L-dependence, barriers, intervention and government.

5.3.5 Conclusion and prospects

(40) The tree manipulations and relations defined in this section are not trivial, but they are fully explicit and implemented for computation.[23]

(41) In the minimalist program, there are simpler approaches to movement that will be discussed in §9.1-§**??**, below.

[23] The formalization of movement relations in Rogers (1999) and in Kracht (1998) are mathematically more elegant, and it would be interesting to consider whether an implementation of these formalizations could be nicer than the ones given here.

6 Context free parsing: stack-based strategies

6.1 LL parsing

(1) Recall the definition of TD, which uses an "expansion" rule that we will now call "LL," because this method consumes the input string from Left to right, and it constructs a Leftmost parse:

$$G, \Gamma, S \vdash G \text{ [axiom]} \qquad \text{for definite clauses } \Gamma, \text{ goal } G, S \subseteq \Sigma^*$$

$$\frac{G, \Gamma, S \vdash (?\text{-}p, C)}{G, \Gamma, S \vdash (?\text{-}q_1, \ldots, q_n, C)} \text{[ll]} \qquad \text{if } (p\text{:-}q_1, \ldots, q_n) \in \Gamma$$

$$\frac{G, \Gamma, wS \vdash (?\text{-}w, C)}{G, \Gamma, S \vdash (?\text{-}C)} \text{[scan]}$$

(2) As discussed in §1 on page 6, the rule ll is sound. To review that basic idea from a different perspective, notice, for example, that [ll] licenses inference steps like the following:

$$\frac{G, \Gamma, S \vdash (?\text{-}p, q)}{G, \Gamma, S \vdash (?\text{-}r, s, q)} \text{[ll]} \qquad \text{if } (p\text{:-}r, s) \in \Gamma$$

In standard logic, this reasoning might be represented this way:

$$\frac{\neg(p \wedge q) \wedge ((r \wedge s) \rightarrow p)}{\neg(r \wedge s \wedge q)}$$

Is this inference sound in the propositional calculus? Yes. This could be shown with truth tables, or we could, for example, use simple propositional reasoning to deduce the conclusion from the premise using tautologies and modus ponens.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\neg(p \wedge q) \wedge ((r \wedge s) \rightarrow p)}{(\neg p \vee \neg q) \wedge ((r \wedge s) \rightarrow p)}\,{\scriptstyle \neg(A\wedge B)\leftrightarrow(\neg A\vee\neg B)}}{(p \rightarrow \neg q) \wedge ((r \wedge s) \rightarrow p)}\,{\scriptstyle (\neg A\vee B)\leftrightarrow(A\rightarrow B)}}{((r \wedge s) \rightarrow p) \wedge (p \rightarrow \neg q)}\,{\scriptstyle (A\wedge B)\leftrightarrow(B\wedge A)}}{(r \wedge s) \rightarrow \neg q}\,{\scriptstyle ((A\rightarrow B)\wedge(B\rightarrow C))\rightarrow(A\rightarrow C)}}{\neg(r \wedge s) \vee \neg q}\,{\scriptstyle (A\rightarrow B)\leftrightarrow(\neg A\vee B)}}{(\neg r \vee \neg s \vee \neg q)}\,{\scriptstyle \neg(A\wedge B)\leftrightarrow(\neg A\vee\neg B)}}{\neg(r \wedge s \wedge q)}\,{\scriptstyle (\neg A\vee\neg B\vee\neg C)\leftrightarrow\neg(A\wedge B\wedge C)}$$

Mates' 100 important tautologies: A formula is a tautology iff it is true under all interpretations. The following examples from (Mates, 1972) are tautologies, for all formulas A, B, C, D:

1. $(A \rightarrow B) \rightarrow ((B \rightarrow C) \rightarrow (A \rightarrow C))$ (Principle of the Syllogism)
2. $(B \rightarrow C) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
3. $A \rightarrow ((A \rightarrow B) \rightarrow B)$
4. $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
5. $(A \rightarrow (B \rightarrow C)) \rightarrow (B \rightarrow (A \rightarrow C))$
6. $A \rightarrow A$ (Law of Identity)
7. $B \rightarrow (A \rightarrow B)$
8. $\neg A \rightarrow (A \rightarrow B)$ (Law of Duns Scotus)
9. $A \rightarrow (\neg A \rightarrow B)$
10. $\neg\neg A \rightarrow A$
11. $A \rightarrow \neg\neg A$
12. $(\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$
13. $(A \rightarrow \neg B) \rightarrow (B \rightarrow \neg A)$
14. $(\neg A \rightarrow B) \rightarrow (\neg B \rightarrow A)$
15. $(A \rightarrow B) \rightarrow (\neg B \rightarrow \neg A)$ (Principle of Transposition, or contraposition)
16. $(\neg A \rightarrow A) \rightarrow A$ (Law of Clavius)
17. $(A \rightarrow \neg A) \rightarrow \neg A$
18. $\neg(A \rightarrow B) \rightarrow A$
19. $\neg(A \rightarrow B) \rightarrow \neg B$
20. $A \rightarrow (B \rightarrow (A \wedge B))$
21. $(A \rightarrow B) \rightarrow ((B \rightarrow A) \rightarrow (A \leftrightarrow B))$
22. $(A \leftrightarrow B) \rightarrow (A \rightarrow B)$
23. $(A \leftrightarrow B) \rightarrow (B \rightarrow A)$
24. $(A \vee B) \leftrightarrow (B \vee A)$ (Commutative Law for Disjunction)
25. $A \rightarrow (A \vee B)$
26. $B \rightarrow (A \vee B)$
27. $(A \vee A) \leftrightarrow A$ (Principle of Tautology for Disjunction)
28. $A \leftrightarrow A$
29. $\neg\neg A \leftrightarrow A$ (Principle of Double Negation)
30. $(A \leftrightarrow B) \leftrightarrow (B \leftrightarrow A)$
31. $(A \leftrightarrow B) \leftrightarrow (\neg A \leftrightarrow \neg B)$
32. $(A \leftrightarrow B) \rightarrow ((A \wedge C) \leftrightarrow (B \wedge C))$
33. $(A \leftrightarrow B) \rightarrow ((C \wedge A) \leftrightarrow (C \wedge B))$
34. $(A \leftrightarrow B) \rightarrow ((A \vee C) \leftrightarrow (B \vee C))$
35. $(A \leftrightarrow B) \rightarrow ((C \vee A) \leftrightarrow (C \vee B))$
36. $(A \leftrightarrow B) \rightarrow ((A \rightarrow C) \leftrightarrow (B \rightarrow C))$
37. $(A \leftrightarrow B) \rightarrow ((C \rightarrow A) \leftrightarrow (C \rightarrow B))$
38. $(A \leftrightarrow B) \rightarrow ((A \leftrightarrow C) \leftrightarrow (B \leftrightarrow C))$

39. $(A \leftrightarrow B) \rightarrow ((C \leftrightarrow A) \leftrightarrow (C \leftrightarrow B))$
40. $(A \vee (B \vee C) \leftrightarrow (B \vee (A \vee C))$
41. $(A \vee (B \vee C)) \rightarrow ((A \vee B) \vee C)$ (Associative Law for Disjunction)
42. $\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$ (De Morgan's Law)
43. $\neg(A \vee B) \leftrightarrow (\neg A \wedge \neg B)$ (De Morgan's Law)
44. $(A \wedge B) \leftrightarrow \neg(\neg A \vee \neg B)$ (De Morgan's Law)
45. $(A \vee B) \leftrightarrow \neg(\neg A \wedge \neg B)$ (De Morgan's Law)
46. $(A \wedge B) \leftrightarrow (B \wedge A)$ (Commutative Law for Conjunction)
47. $(A \wedge B) \rightarrow A$ (Law of Simplification)
48. $(A \wedge B) \rightarrow B$ (Law of Simplification)
49. $(A \wedge A) \rightarrow A$ (Law of Tautology for Conjunction)
50. $(A \wedge (B \wedge C)) \leftrightarrow ((A \wedge B) \wedge C)$ (Associative Law for Conjunction)
51. $(A \rightarrow (B \rightarrow C)) \leftrightarrow ((A \wedge B) \rightarrow C)$ (Export-Import Law)
52. $(A \rightarrow B) \leftrightarrow \neg(A \wedge \neg B)$
53. $(A \rightarrow B) \leftrightarrow (\neg A \vee B)$ **ES says: know this one!**
54. $(A \vee (B \wedge C)) \leftrightarrow ((A \vee B) \wedge (A \vee C))$ (Distributive Law)
55. $(A \wedge (B \vee C)) \leftrightarrow ((A \wedge B) \vee (A \wedge C))$ (Distributive Law)
56. $((A \wedge B) \vee (C \wedge D)) \leftrightarrow (((A \vee C) \wedge (A \vee D)) \wedge (B \vee C) \wedge (B \vee D))$
57. $A \rightarrow ((A \wedge B) \leftrightarrow B)$
58. $A \rightarrow ((B \wedge A) \leftrightarrow B)$
59. $A \rightarrow ((A \rightarrow B) \leftrightarrow B)$
60. $A \rightarrow ((A \leftrightarrow B) \leftrightarrow B)$
61. $A \rightarrow ((B \leftrightarrow A) \leftrightarrow B)$
62. $\neg A \rightarrow ((A \vee B) \leftrightarrow B)$
63. $\neg A \rightarrow ((B \vee A) \leftrightarrow B)$
64. $\neg A \rightarrow (\neg(A \leftrightarrow B) \leftrightarrow B)$
65. $\neg A \rightarrow (\neg(B \leftrightarrow A) \leftrightarrow B)$
66. $A \vee \neg A$ (Law of Excluded Middle)
67. $\neg(A \wedge \neg A)$ (Law of Contradiction)
68. $(A \leftrightarrow B) \leftrightarrow ((A \wedge B) \vee (\neg A \wedge \neg B))$
69. $\neg(A \leftrightarrow B) \leftrightarrow (A \leftrightarrow \neg B)$
70. $((A \leftrightarrow B) \wedge (B \leftrightarrow C)) \rightarrow (A \leftrightarrow C)$
71. $((A \leftrightarrow B) \leftrightarrow A) \leftrightarrow B$
72. $(A \leftrightarrow (B \leftrightarrow C)) \leftrightarrow ((A \leftrightarrow B) \leftrightarrow C)$
73. $(A \rightarrow B) \leftrightarrow (A \rightarrow (A \wedge B))$
74. $(A \rightarrow B) \leftrightarrow (A \leftrightarrow (A \wedge B))$
75. $(A \rightarrow B) \leftrightarrow ((A \vee B) \rightarrow B)$
76. $(A \rightarrow B) \leftrightarrow ((A \vee B) \leftrightarrow B)$
77. $(A \rightarrow B) \leftrightarrow (A \rightarrow (A \rightarrow B))$
78. $(A \rightarrow (B \wedge C)) \leftrightarrow ((A \rightarrow B) \wedge (A \rightarrow C))$
79. $((A \vee B) \rightarrow C) \leftrightarrow ((A \rightarrow C) \wedge (B \rightarrow C))$
80. $(A \rightarrow (B \vee C)) \leftrightarrow ((A \rightarrow B) \vee (A \rightarrow C))$

81. $((A \wedge B) \rightarrow C) \leftrightarrow (A \rightarrow C) \wedge (B \rightarrow C))$
82. $(A \rightarrow (B \leftrightarrow C)) \leftrightarrow ((A \wedge B) \leftrightarrow (A \wedge C))$
83. $((A \wedge \neg B) \rightarrow C) \leftrightarrow (A \rightarrow (B \vee C))$
84. $(A \vee B) \leftrightarrow ((A \rightarrow B) \rightarrow B)$
85. $(A \wedge B) \leftrightarrow ((B \rightarrow A) \wedge B)$
86. $(A \rightarrow B) \vee (B \rightarrow C)$
87. $(A \rightarrow B) \vee (\neg A \rightarrow B)$
88. $(A \rightarrow B) \vee (A \rightarrow \neg B)$
89. $((A \wedge B) \rightarrow C) \leftrightarrow ((A \wedge \neg C) \rightarrow \neg B)$
90. $(A \rightarrow B) \rightarrow ((C \rightarrow (B \rightarrow D)) \rightarrow (C \rightarrow (A \rightarrow D)))$
91. $((A \rightarrow B) \wedge (B \rightarrow C)) \vee (C \rightarrow A))$
92. $((A \rightarrow B) \wedge (C \rightarrow D)) \rightarrow ((A \vee C) \rightarrow (B \vee D))$
93. $((A \rightarrow B) \vee (C \rightarrow D)) \leftrightarrow ((A \rightarrow D) \vee (C \rightarrow B))$
94. $((A \vee B) \rightarrow C) \leftrightarrow ((A \rightarrow C) \wedge ((\neg A \wedge B) \rightarrow C))$
95. $((A \rightarrow B) \rightarrow (B \rightarrow C)) \leftrightarrow (B \rightarrow C)$
96. $((A \rightarrow B) \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
97. $((A \rightarrow B) \rightarrow C) \rightarrow ((C \rightarrow A) \rightarrow A)$
98. $((A \rightarrow B) \rightarrow C) \rightarrow ((A \rightarrow C) \rightarrow C)$
99. $(\neg A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow ((A \rightarrow B) \rightarrow C))$
100. $(((A \rightarrow B) \rightarrow C) \rightarrow D) \rightarrow ((B \rightarrow C) \rightarrow (A \rightarrow D))$

(3) The top-down recognizer was implemented this way:

```
/*
 * file: ll.pl
 */
:- op(1200,xfx,:~).   % this is our object language "if"
:- op(1100,xfx,?~).   % metalanguage provability predicate

[] ?~ [].
(S0 ?~ Goals0) :- infer(S0,Goals0,S,Goals), (S ?~ Goals).

infer(S,[A|C], S,DC) :- (A :~ D), append(D,C,DC).   % ll
infer([A|S],[A|C], S,C).                            % scan

append([],L,L).
append([E|L],M,[E|N]) :- append(L,M,N).
```

(4) This top-down (TD) parsing method is sometimes called LL, because it uses the input string from Left to right, and it constructs a Leftmost parse (i.e. a derivation that expands the leftmost nonterminal at each point).

(5) The parser was implemented this way:

```
/*
 * file: llp.pl = tdp.pl
 */
:- op(1200,xfx,:~).   % this is our object language "if"
:- op(1100,xfx,?~).   % metalanguage provability predicate
:- op(500,yfx,@).     % metalanguage functor to separate goals from trees

[] ?~ []@[].
(S0 ?~ Goals0@T0) :- infer(S0,Goals0@T0,S,Goals@T), (S ?~ Goals@T).

infer(S,[A|C]@[A/DTs|CTs],S,DC@DCTs) :- (A :~ D), new_goals(D,C,CTs,DC,DCTs,DTs). % ll
infer([A|S],[A|C]@[A/[]|CTs],S,C@CTs).                                           % scan

%new_goals(NewGoals,OldGoals,OldTrees,AllGoals,AllTrees,NewTrees)
new_goals([],Gs,Ts,Gs,Ts,[]).
new_goals([G|Gs0],Gs1,Ts1,[G|Gs2],[T|Ts2],[T|Ts]) :- new_goals(Gs0,Gs1,Ts1,Gs2,Ts2,Ts).
```

(6) **Example.** Let's use `g1.pl` again:

```
/*
 * file: g1.pl
 */
:- op(1200,xfx,:~).

ip :~ [dp, i1].     i1 :~ [i0, vp].     i0 :~ [will].
dp :~ [d1].         d1 :~ [d0, np].     d0 :~ [the].
np :~ [n1].         n1 :~ [n0].         n0 :~ [idea].
                    n1 :~ [n0, cp].
vp :~ [v1].         v1 :~ [v0].         v0 :~ [suffice].
cp :~ [c1].         c1 :~ [c0, ip].     c0 :~ [that].
```

With this grammar and `llp.pl` we get the following session:

```
| ?- [llp,g1,pp_tree].
{consulting /home/es/tex/185-00/llp.pl...}
{consulted /home/es/tex/185-00/llp.pl in module user, 20 msec 2000 bytes}
{consulting /home/es/tex/185-00/g1.pl...}
{consulted /home/es/tex/185-00/g1.pl in module user, 20 msec 2384 bytes}
{consulting /home/es/tex/185-00/pp_tree.pl...}
{consulted /home/es/tex/185-00/pp_tree.pl in module user, 10 msec 1344 bytes}

yes
| ?- [the,idea,will,suffice] ?~ [ip]@[T].

T = ip/[dp/[d1/[d0/[the/[]],np/[n1/[...]]]],i1/[i0/[will/[]],vp/[v1/[v0/[...]]]]] ?

yes
| ?- ([the,idea,will,suffice] ?~ [ip]@[T]), pp_tree(T).
ip /[
    dp /[
        d1 /[
            d0 /[
                the /[]],
            np /[
                n1 /[
                    n0 /[
                        idea /[]]]]]],
    i1 /[
        i0 /[
            will /[]],
        vp /[
            v1 /[
                v0 /[
                    suffice /[]]]]]]
T = ip/[dp/[d1/[d0/[the/[]],np/[n1/[...]]]],i1/[i0/[will/[]],vp/[v1/[v0/[...]]]]] ?

yes
| ?-
```

(7) Assessment of the LL strategy:

a. Unbounded memory requirements on simple left branching.

b. Stupid about left branches – recursion in left branches produces infinite search spaces.

6.2 LR parsing

(8) Bottom-up parsing is sometimes called "LR," because it uses the input string from Left to right, and it constructs a Rightmost parse in reverse.
LR parsers adopt a strategy of "listening first" – that is, rules are never used until all of their conditions (their right sides, their antecedents) have been established.

(9) LR recognition is defined this way:

$$G, \Gamma, S \vdash G \text{ [axiom]} \qquad \text{for definite clauses } \Gamma, \text{ goal } G,\ S \subseteq \Sigma^*$$

$$\frac{G, \Gamma, S \vdash (?\text{-}\neg q_n, \ldots, \neg q_1, C)}{G, \Gamma, S \vdash (?\text{-}\neg p, C)}\ \text{[lr]} \qquad \text{if } (p\text{:-}q_1, \ldots, q_n) \in \Gamma$$

$$\frac{G, \Gamma, S \vdash (?\text{-}\neg q_n, \ldots, \neg q_1, p, C)}{G, \Gamma, S \vdash (?\text{-}C)}\ \text{[lr-complete]} \qquad \text{if } (p\text{:-}q_1, \ldots, q_n) \in \Gamma$$

$$\frac{G, \Gamma, wS \vdash (?\text{-}C)}{G, \Gamma, S \vdash (?\text{-}\neg w, C)}\ \text{[shift]}$$

The lr-complete rule (often called "reduce-complete") is needed because the initial goal is, in effect, a prediction of what we should end up with. (More precisely, the goal is the denial of what we want to prove, since these are proofs by contradiction.)

(10) **Exercise:** (optional!) We saw that the top-down reasoning shown in (2) on page 75 can be interpreted in as sound reasoning in the propositional calculus. Is there a way to interpret bottom-up reasoning as sound too? How could we interpret the expressions in the following step so that the reasoning is sound? (tricky!)

$$\frac{G, \Gamma, S \vdash (?\text{-}\neg s, \neg r, q)}{G, \Gamma, S \vdash (?\text{-}\neg p, q)}\ \text{[lr]} \qquad \text{if } (p\text{:-}r, s) \in \Gamma$$

A solution: To understand this as a sound step, we need an appropriate understanding of the negated elements in the goal. One strategy is to treat the negated elements at the front of the goal as disjoined, and so the shift rule actually disjoins an element from S with the negated elements on the right side, if any. So then we interpret the rule just above like this:

$$\frac{\neg((\neg s \vee \neg r) \wedge q) \wedge ((r \wedge s) \rightarrow p)}{\neg(\neg p \wedge q)}$$

The following propositional reasoning shows that this step is sound:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\neg((\neg s \vee \neg r) \wedge q) \wedge ((r \wedge s) \rightarrow p)}{\neg(q \wedge (\neg s \vee \neg r)) \wedge ((r \wedge s) \rightarrow p)}\,{\scriptstyle (A\wedge B)\leftrightarrow(B\wedge A)}}{\neg(q \wedge (\neg r \vee \neg s)) \wedge ((r \wedge s) \rightarrow p)}\,{\scriptstyle (A\vee B)\leftrightarrow(B\vee A)}}{\neg(q \wedge \neg(r \wedge s)) \wedge ((r \wedge s) \rightarrow p)}\,{\scriptstyle (A\vee B)\leftrightarrow\neg(\neg A\wedge\neg B)}}{(\neg q \vee (r \wedge s)) \wedge ((r \wedge s) \rightarrow p)}\,{\scriptstyle \neg(A\wedge B)\leftrightarrow(\neg A\vee\neg B)}}{(q \rightarrow (r \wedge s)) \wedge ((r \wedge s) \rightarrow p)}\,{\scriptstyle (\neg A\vee B)\leftrightarrow(A\rightarrow B)}}{(q \rightarrow p)}\,{\scriptstyle ((A\rightarrow B)\wedge(B\rightarrow C))\rightarrow(A\rightarrow C)}}{(\neg q \vee p)}\,{\scriptstyle (A\rightarrow B)\leftrightarrow(\neg A\vee B)}}{(p \vee \neg q)}\,{\scriptstyle (A\vee B)\leftrightarrow(B\vee A)}}{\neg(\neg p \wedge q)}\,{\scriptstyle (A\vee B)\leftrightarrow\neg(\neg A\wedge\neg B)}$$

(11) Setting the stage: implementation of `reverse` and `negreverse`

```
reverse([],L,L).
reverse([E|L],M,N) :- reverse(L,[E|M],N).

negreverse([],L,L).
negreverse([E|L],M,N) :- negreverse(L,[-E|M],N).
```

(12) The naive implementation of the LR recognizer:

```
/*
 * file: lr0.pl - first version
 */
:- op(1200,xfx,:~).   % this is our object language "if"
:- op(1100,xfx,?~).   % metalanguage provability predicate

[] ?~ [].
(S0 ?~ Goals0) :- infer(S0,Goals0,S,Goals), (S ?~ Goals).

infer(S,RDC,S,C) :- (A :~ D), negreverse(D,[A],RD), append(RD,C,RDC).        % reduce-complete
provable(S,RDC,S,[-A|C]) :- (A := D), negreverse(D,[],RD), append(RD,C,RDC). % reduce
provable([W|S],C,S,[-W|C]).                                                  % shift

negreverse([],L,L).
negreverse([E|L],M,N) :- negreverse(L,[-E|M],N).

append([],L,L).
append([F|L],M,[F|N]) :- append(L,M,N).
```

Here, RDC is the sequence which is the Reverse of D, followed by C

(13) The slightly improved implementation of the LR recognizer:

```
/*
 * file: lr.pl
 */
:- op(1200,xfx,:~).   % this is our object language "if"
:- op(1100,xfx,?~).   % metalanguage provability predicate

[] ?~ [].
(S0 ?~ Goals0) :- infer(S0,Goals0,S,Goals), (S ?~ Goals).

infer(S,RDC,S,C) :- (A :~ D), preverse(D,RDC,[A|C]).      % reduce-complete
infer(S,RDC,S,[-A|C]) :- (A :~ D), preverse(D,RDC,C).     % reduce
infer([W|S],C,S,[-W|C]).                                  % shift

%preverse(ExpansionD,TempStack,ReversedExpansionD,RestConstituents)
preverse(   [], C,C).
preverse([E|L],RD,C) :- preverse(L,RD,[-E|C]).
```

(14) The implementation of the LR parser:

```
/*
 * file: lrp.pl
 */
:- op(1200,xfx,:~).   % this is our object language "if"
:- op(1100,xfx,?~).   % metalanguage provability predicate
:- op(500,yfx,@).     % metalanguage functor to separate goals from trees

[] ?~ []@[].
(S0 ?~ Goals0@T0) :- infer(S0,Goals0@T0,S,Goals@T), (S ?~ Goals@T).

infer(S,RDC@RDCTs,S,C@CTs) :- (A :~ D), preverse(D,RDC,[A|C],DTs,RDCTs,[A/DTs|CTs]).  % reduce-complete
infer(S,RDC@RDCTs,S,[-A|C]@[A/DTs|CTs]) :- (A :~ D), preverse(D,RDC,C,DTs,RDCTs,CTs). % reduce
infer([W|S],C@CTs,S,[-W|C]@[W/[]|CTs]).                                              % shift

%preverse(ExpansionD,ReversedExpansionD,RestCatsC,ExpansionDTs,ReversedExpansionDTs,RestCatsCTs)
preverse([],C,C,[],CTs,CTs).
preverse([E|L],RD,C,[ETs|LTs],RDTs,CTs) :- preverse(L,RD,[-E|C],LTs,RDTs,[ETs|CTs]).
```

This implementation is conceptually more straightforward than `tdp`, because here, all the trees in our stack are complete, so we just do with the trees exactly the same thing that we are doing with the stack. This is accomplished by taking the 4-argument `preverse` from the lr recognizer and making it an 8-argument predicate in the parser, where the tree stacks are manipulated in just the same way that the recognizer stacks are.

(15) Assessment of the LR strategy:

a. Unbounded memory requirements on simple right branching.

b. Stupid about empty categories – they produce infinite search spaces.

6.3 LC parsing

(16) Left corner parsing is intermediate between top-down and bottom-up. Like LR, LC parsers adopt a strategy of "listening first," but after listening to a "left corner," the rest of the expansion is predicted. In a constituent formed by applying a rewrite rule $A \rightarrow B\ C\ D$, the "left corner" is just the first constituent on the right side – B in the production $A \rightarrow B\ C\ D$.

(17) LC recognition is defined this way:[24]

$G, \Gamma, S \vdash G$ [axiom] for definite clauses Γ, goal G, $S \subseteq \Sigma^*$

$$\frac{G, \Gamma, S \vdash (?\text{-}\neg q_1, C)}{G, \Gamma, S \vdash (?\text{-}q_2, \ldots, q_n, \neg p, C)}\ \text{[lc]} \qquad \text{if } (p\text{:-}q_1, \ldots, q_n) \in \Gamma$$

$$\frac{G, \Gamma, S \vdash (?\text{-}\neg q_1, p, C)}{G, \Gamma, S \vdash (?\text{-}q_2, \ldots, q_n, C)}\ \text{[lc-complete]} \qquad \text{if } (p\text{:-}q_1, \ldots, q_n) \in \Gamma$$

$$\frac{G, \Gamma, wS \vdash (?\text{-}C)}{G, \Gamma, S \vdash (?\text{-}\neg w, C)}\ \text{[shift]}$$

$$\frac{G, \Gamma, wS \vdash (?\text{-}w, C)}{G, \Gamma, S \vdash (?\text{-}C)}\ \text{[shift-complete]} \qquad \text{=scan}$$

We want to allow the recognizer to handle empty productions, that is, productions $(p\text{:-}q_1, \ldots, q_n) \in \Gamma$ where $n = 0$. We do this by saying that is such productions, the "left corner" is the empty string. With this policy, the $n = 0$ instances of the *lc* rules can be written this way:

$$\frac{G, \Gamma, S \vdash (?\text{-}C)}{G, \Gamma, S \vdash (?\text{-}\neg p, C)}\ \text{[lc-e]} \qquad \text{if } (p\text{:-}[\,]) \in \Gamma$$

$$\frac{G, \Gamma, S \vdash (?\text{-}p, C)}{G, \Gamma, S \vdash (?\text{-}C)}\ \text{[lc-e-complete]} \qquad \text{if } (p\text{:-}[\,]) \in \Gamma$$

(18) **Exercise:** Use simple propositional reasoning of the sort shown in (2) on page 75 and in (10) on page 80 to show that the following inference step is sound. (tricky!)

$$\frac{G, \Gamma, S \vdash (?\text{-}\neg r, q)}{G, \Gamma, S \vdash (?\text{-}s, \neg p, q)}\ \text{[lc]} \qquad \text{if } (p\text{:-}r, s) \in \Gamma$$

(19) The implementation of the LC recognizer:

```
/*
* file:  lc.pl
*/
:- op(1200,xfx,:-).  % this is our object language "if"
:- op(1100,xfx,?-).  % metalanguage provability predicate


[] ?-[].
(S0 ?-Goals0) :- infer(S0,Goals0,S,Goals), (S ?-Goals).

infer(S,[-D,A|C],S,DC) :- (A :-[D|Ds]), append(Ds,C,DC). % lc-complete
infer(S,[-D|C],S,DAC) :- (A :-[D|Ds]), append(Ds,[-A|C],DAC). % lc


infer([W|S],[W|C],S,C). % shift-complete=scan
infer([W|S],C,S,[-W|C]). % shift


infer(S,[A|C],S,C) :- (A :-[]).  % lc-e-complete
infer(S,C,S,[-A|C]) :- (A :-[]).  % lc-e


append([],L,L).
append([E|L],M,[E|N]) :- append(L,M,N).
```

(20) Like LR, this parser is stupid about empty categories. For example, using `g3.pl` which has an empty `i0`, we cannot prove `[titus,laughs] ?~[ip]`, unless we control the search by hand:

```
| ?- [g3,lc].
yes
| ?- trace,([titus,laughs] ?-[ip]).
The debugger will first creep -- showing everything (trace)
1 1 Call:  [titus,laughs]?-[ip] ?
2 2 Call:  infer([titus,laughs],[ip],_953,_954) ?
?  2 2 Exit:  infer([titus,laughs],[ip],[laughs],[-(titus),ip]) ?
3 2 Call:  [laughs]?-[-(titus),ip] ?
4 3 Call:  infer([laughs],[-(titus),ip],_2051,_2052) ?
5 4 Call:  ip:-[titus|_2424] ?  f
5 4 Fail:  ip:-[titus|_2424] ?
6 4 Call:  _2430:-[titus|_2428] ?
?  6 4 Exit:  dp(3,s,_2815):-[titus] ?
7 4 Call:  append([],[-(dp(3,s,_2815)),ip],_2052) ?  s
7 4 Exit:  append([],[-(dp(3,s,_2815)),ip],[-(dp(3,s,_2815)),ip]) ?
?  4 3 Exit:  infer([laughs],[-(titus),ip],[laughs],[-(dp(3,s,_2815)),ip])
?  8 3 Call:  [laughs]?-[-(dp(3,s,_2815)),ip] ?
9 4 Call:  infer([laughs],[-(dp(3,s,_2815)),ip],_4649,_4650) ?
10 5 Call:  ip:-[dp(3,s,_2815)|_5028] ?
10 5 Exit:  ip:-[dp(3,s,nom),i1(3,s)] ?
11 5 Call:  append([i1(3,s)],[],_4650) ?  s
11 5 Exit:  append([i1(3,s)],[],[i1(3,s)]) ?
?  9 4 Exit:  infer([laughs],[-(dp(3,s,nom)),ip],[laughs],[i1(3,s)]) ?
12 4 Call:  [laughs]?-[i1(3,s)] ?
13 5 Call:  infer([laughs],[i1(3,s)],_7267,_7268) ?
?  13 5 Exit:  infer([laughs],[i1(3,s)],[],[-(laughs),i1(3,s)]) ?
14 5 Call:  []?-[-(laughs),i1(3,s)] ?  f
14 5 Fail:  []?-[-(laughs),i1(3,s)] ?
13 5 Redo:  infer([laughs],[i1(3,s)],[],[-(laughs),i1(3,s)]) ?
15 6 Call:  i1(3,s):-[] ?
15 6 Fail:  i1(3,s):-[] ?
16 6 Call:  _7639:-[] ?
?  16 6 Exit:  i0:-[] ?
?  13 5 Exit:  infer([laughs],[i1(3,s)],[laughs],[-(i0),i1(3,s)]) ?
17 5 Call:  [laughs]?-[-(i0),i1(3,s)] ?
18 6 Call:  infer([laughs],[-(i0),i1(3,s)],_9100,_9101) ?
19 7 Call:  i1(3,s):-[i0|_9478] ?
19 7 Exit:  i1(3,s):-[i0,vp(3,s)] ?
20 7 Call:  append([vp(3,s)],[],_9101) ?  s
20 7 Exit:  append([vp(3,s)],[],[vp(3,s)]) ?
?  18 6 Exit:  infer([laughs],[-(i0),i1(3,s)],[laughs],[vp(3,s)]) ?
21 6 Call:  [laughs]?-[vp(3,s)] ?
22 7 Call:  infer([laughs],[vp(3,s)],_11713,_11714) ?
?  22 7 Exit:  infer([laughs],[vp(3,s)],[],[-(laughs),vp(3,s)]) ?
23 7 Call:  []?-[-(laughs),vp(3,s)] ?
24 8 Call:  infer([],[-(laughs),vp(3,s)],_12827,_12828) ?
25 9 Call:  vp(3,s):-[laughs|_13204] ?
25 9 Fail:  vp(3,s):-[laughs|_13204] ?
26 9 Call:  _13210:-[laughs|_13208] ?
?  26 9 Exit:  v0(intrans,3,s):-[laughs] ?
27 9 Call:  append([],[-(v0(intrans,3,s)),vp(3,s)],_12828) ?  s
27 9 Exit:  append([],[-(v0(intrans,3,s)),vp(3,s)],[-(v0(intrans,3,s)),vp(3,s)]) ?
?  24 8 Exit:  infer([],[-(laughs),vp(3,s)],[],[-(v0(intrans,3,s)),vp(3,s)]) ?
28 8 Call:  []?-[-(v0(intrans,3,s)),vp(3,s)] ?
29 9 Call:  infer([],[-(v0(intrans,3,s)),vp(3,s)],_15456,_15457) ?
30 10 Call:  vp(3,s):-[v0(intrans,3,s)|_15839] ?
30 10 Fail:  vp(3,s):-[v0(intrans,3,s)|_15839] ?
31 10 Call:  _15845:-[v0(intrans,3,s)|_15843] ?
?  31 10 Exit:  v1(3,s):-[v0(intrans,3,s)] ?
32 10 Call:  append([],[-(v1(3,s)),vp(3,s)],_15457) ?  s
32 10 Exit:  append([],[-(v1(3,s)),vp(3,s)],[-(v1(3,s)),vp(3,s)]) ?
?  29 9 Exit:  infer([],[-(v0(intrans,3,s)),vp(3,s)],[],[-(v1(3,s)),vp(3,s)]) ?
33 9 Call:  []?-[-(v1(3,s)),vp(3,s)] ?
34 10 Call:  infer([],[-(v1(3,s)),vp(3,s)],_18106,_18107) ?
35 11 Call:  vp(3,s):-[v1(3,s)|_18488] ?
35 11 Exit:  vp(3,s):-[v1(3,s)] ?
36 11 Call:  append([],[],_18107) ?  s
36 11 Exit:  append([],[],[]) ?
?  34 10 Exit:  infer([],[-(v1(3,s)),vp(3,s)],[],[]) ?
37 10 Call:  []?-[] ?
?  37 10 Exit:  []?-[] ?
?  33 9 Exit:  []?-[-(v1(3,s)),vp(3,s)] ?
?  28 8 Exit:  []?-[-(v0(intrans,3,s)),vp(3,s)] ?
?  23 7 Exit:  []?-[-(laughs),vp(3,s)] ?
?  21 6 Exit:  [laughs]?-[vp(3,s)] ?
?  17 5 Exit:  [laughs]?-[-(i0),i1(3,s)] ?
?  12 4 Exit:  [laughs]?-[i1(3,s)] ?
?  8 3 Exit:  [laughs]?-[-(dp(3,s,nom)),ip] ?
?  3 2 Exit:  [laughs]?-[-(titus),ip] ?
?  1 1 Exit:  [titus,laughs]?-[ip] ?


yes
```

(21) The implementation of the LC parser:

```
/*
* file:  lcp.pl
*/
:- op(1200,xfx,:-).  % this is our object language "if"
:- op(1100,xfx,?-).  % metalanguage provability predicate
:- op(500,yfx,@).  % metalanguage functor to separate goals from trees

[] ?-[]@[].
(S0 ?-Goals0@T0) :- infer(S0,Goals0@T0,S,Goals@T), (S ?-Goals@T).

infer([A|S],[A|C]@[A/[]|CTs],S,C@CTs).  % scan
infer([W|S], C@CTs,S,[-W|C]@[W/[]|CTs]).  % shift

infer(S,[-D,A|C]@[DT,A/[DT|DTs]|CTs],S,DC@DCTs) :- (A :-[D|Ds]), new_goals(Ds,C,CTs,DC,DCTs,DTs).  % lc-complete
infer(S,[-D|C]@[DT|CTs],S,DC@DCTs) :- (A :-[D|Ds]), new_goals(Ds,[-A|C],[A/[DT|DTs]|CTs],DC,DCTs,DTs).  % lc

infer(S,[A|C]@[A/[]|CTs],S,DC@DCTs) :- (A :-[]), new_goals([],C,CTs,DC,DCTs,[]).  % lc-e-complete
infer(S,C@CTs,S,DC@DCTs) :- (A :-[]), new_goals([],[-A|C],[A/[]|CTs],DC,DCTs,[]).  % lc-e

%new_goals(NewGoals,OldGoals,OldTrees,AllGoals,AllTrees,NewTrees)
new_goals([],Gs,Ts,Gs,Ts,[]).
new_goals([G|Gs0],Gs1,Ts1,[G|Gs2],[T|Ts2],[T|Ts]) :- new_goals(Gs0,Gs1,Ts1,Gs2,Ts2,Ts).
```

(22) Assessment of the LC parser:

a. Bounded memory requirements on simple right and left branching!

b. Unbounded on recursive center embedding (of course)

c. Stupid about empty categories – they can still produce infinite search spaces.

6.4 All the GLC parsing methods (the "stack based" methods)

(23) LC parsing uses a rule after establishing its leftmost element. We can represent how much of the right side is established before the rule is used in the following way:

$$s\text{:-}np \rrbracket vp$$

LL parsing uses a rule predictively, without establishing any of the right side:

$$s\text{:-} \rrbracket np, vp$$

LR parsing uses a rule conservatively, after establishing all of the right side:

$$s\text{:-}np, vp \llbracket$$

Let's call the sequence on the right that triggers the use of the rule, the *trigger*.

In a rule like this with 2 constituents on the right side, these 3 options are the only ones.

This observation is made in Brosgol (1974), and in Demers (1977).

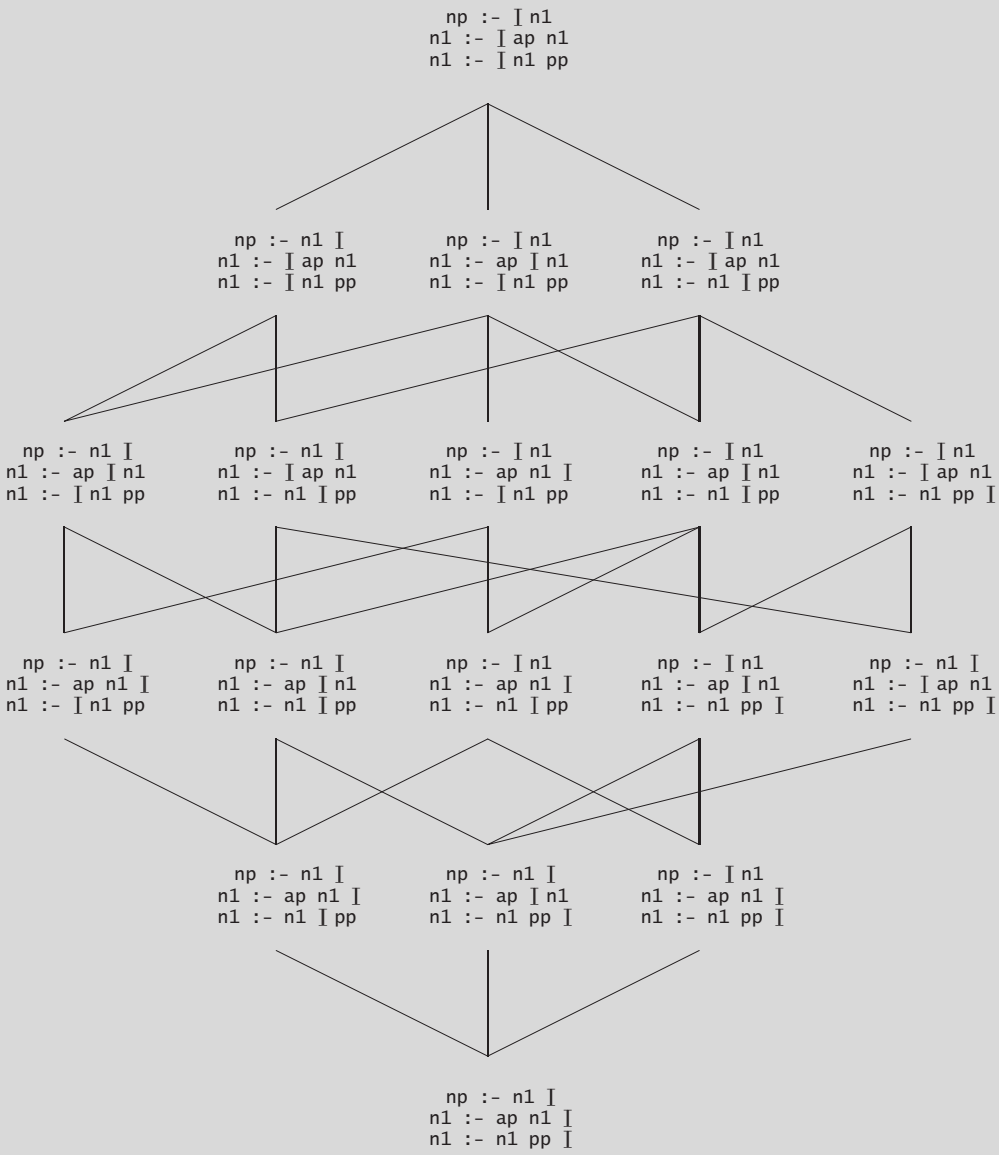
(24) In general, it is clear that with a rule that has n elements on its right side, there are $n + 1$ options for the parser.

Furthermore, the parser need not treat all rules the same way, so in a grammar like the following, the number of parsing options is the product of the number of ways to parse each rule.

(25) As Demers (1977) points out, the collection of trigger functions $\mathcal{F}$ for any grammar can be naturally partially ordered by top-downness:

$F_1 \leq F_2$ if and only if for every production p, the trigger $F_1(p)$ is at least as long as $F_2(p)$.

In other words, a setting of triggers F_1 is as bottom-up as F_2 if and only if for every production p, the triggering point defined by F_1 is at least as far to the right as the triggering point defined by F_2. It is easy to see that $\langle \mathcal{F}, \leq \rangle$ is a lattice, as Demers claims, since for any collection $\mathcal{F}$ of trigger functions for any grammar, the least upper bound of $\mathcal{F}$ is just the function which maps each rule to the trigger which is the shortest of the triggers assigned by any function in $\mathcal{F}$, and the greatest lower bound of $\mathcal{F}$ is the function which maps each rule to the trigger which is the longest assigned by any function in $\mathcal{F}$. Furthermore, the lattice is finite.[25] We call this lattice of recognition strategies the GLC lattice. The simple lattice structure for a 3 rule grammar can be depicted like this:


np :- I n1
n1 :- I ap n1
n1 :- I n1 pp
np :- n1 I
n1 :- I ap n1
n1 :- I n1 pp
np :- I n1
n1 :- ap I n1
n1 :- I n1 pp
np :- I n1
n1 :- ap I n1
n1 :- n1 I pp
np :- n1 I
n1 :- ap I n1
n1 :- I n1 pp
np :- n1 I
n1 :- I ap n1
n1 :- n1 I pp
np :- I n1
n1 :- ap n1 I
n1 :- I n1 pp
np :- I n1
n1 :- ap n1 I
n1 :- n1 I pp
np :- I n1
n1 :- ap I n1
n1 :- n1 I pp
np :- I n1
n1 :- ap I n1
n1 :- n1 pp I
np :- n1 I
n1 :- I ap n1
n1 :- n1 pp I
np :- n1 I
n1 :- ap n1 I
n1 :- I n1 pp
np :- n1 I
n1 :- ap I n1
n1 :- n1 I pp
np :- I n1
n1 :- ap n1 I
n1 :- n1 I pp
np :- I n1
n1 :- ap I n1
n1 :- n1 pp I
np :- n1 I
n1 :- I ap n1
n1 :- n1 pp I
np :- n1 I
n1 :- ap n1 I
n1 :- n1 I pp
np :- n1 I
n1 :- ap I n1
n1 :- n1 pp I
np :- I n1
n1 :- ap n1 I
n1 :- n1 pp I
np :- n1 I
n1 :- ap n1 I
n1 :- n1 pp I

(26) GLC recognition is defined this way:

$G, \Gamma, S \vdash G$ [axiom] for definite clauses Γ, goal G, $S \subseteq \Sigma^*$

$$\frac{G, \Gamma, S \vdash (?\text{-}\neg q_i, \ldots, \neg q_1, C)}{G, \Gamma, S \vdash (?\text{-}q_{i+1}, \ldots, q_n, \neg p, C)}\ \text{[glc]} \qquad \text{if } (p\text{:-}q_1, \ldots, q_i \rrbracket q_{i+1}, \ldots q_n) \in \Gamma$$

$$\frac{G, \Gamma, S \vdash (?\text{-}\neg q_i, \ldots, \neg q_1, p, C)}{G, \Gamma, S \vdash (?\text{-}q_{i+1}, \ldots, q_n, C)}\ \text{[glc-complete]} \qquad \text{if } (p\text{:-}q_1, \ldots, q_i \rrbracket q_{i+1}, \ldots q_n) \in \Gamma$$

$$\frac{G, \Gamma, wS \vdash (?\text{-}C)}{G, \Gamma, S \vdash (?\text{-}\neg w, C)}\ \text{[shift]}$$

$$\frac{G, \Gamma, wS \vdash (?\text{-}w, C)}{G, \Gamma, S \vdash (?\text{-}C)}\ \text{[shift-complete]} \qquad \text{=scan}$$

Like in the LC parser, we allow the possibilities $i = 0$ and $n = 0$.

(27) The implementation of the GLC recognizer:

```
/*
* file:  glc.pl
*/
:- op(1200,xfx,:-).  % this is our object language "if"
:- op(1100,xfx,?-).  % metalanguage provability predicate

[] ?-[].
(S0 ?-Goals0) :- infer(S0,Goals0,S,Goals), (S ?-Goals).

infer([W|S],[W|C],S,C). % shift-complete=scan
infer([W|S],C,S,[-W|C]). % shift

infer(S,RPC,S,DC) :- (A :-P+D), preverse(P,RPC,[A|C]), append(D,C,DC). % reduce-complete
infer(S,RPC,S,DAC) :- (A :-P+D), preverse(P,RPC,C), append(D,[-A|C],DAC). % reduce

%preverse(ExpansionD,ReversedExpansionD,RestConstituents)
preverse([E|L],RD,C) :- preverse(L,RD,[-E|C]).
preverse( [], C,C).

append([],L,L).
append([E|L],M,[E|N]) :- append(L,M,N).
```

(28) We postpone an assessment of the GLC parsers until we have introduced some methods for controlling them, to make infinite searches less of a problem.

6.5 Oracles

The way to see the future is to understand what is possible, and what must follow from the current situation. Knowing the future, one can act accordingly.

6.5.1 Top-down oracles: stack consistency

(29) Some stacks cannot possibly be reduced to empty, no matter what input string is provided. In particular: There is no point in shifting a word if it cannot be part of the trigger of the most recently predicted category.

And there is no point in building a constituent (i.e. using a rule) if the parent category cannot be part of the trigger of the most recently predicted category.

(30) These conditions can be enforced by calculating, for each category C that could possibly be predicted, all of the stack sequences which could possibly be part of a trigger for C.

In top-down parsing, the triggers are always empty.

In left-corner parsing, the possible trigger sequences are always exactly one completed category.

In bottom-up and some other parsing methods, the sequences can sometimes be arbitrarily long, but in some cases they are finitely bounded and can easily be calculated in advance. A test that can check these precalculated possibilities is called an **oracle**.

(31) Given a context free grammar $G = \langle \Sigma, N, \rightarrow, \mathsf{S} \rangle$, we can generate all instances of the **is a beginning of** relation **R** with the following logic:

$$q_1, \ldots, q_i \; R \; p \qquad \text{[axiom]} \qquad \text{if } p\text{:-}q_1, \ldots, q_i \; \mathbb{I} \, q_{i+1}, \ldots, q_n$$

$$\frac{q_1, \ldots, q_i \; R \; p}{q_1, \ldots, q_{i-1} \; R \; p} \text{[unscan]} \qquad \text{if } q_i \in \Sigma$$

$$\frac{q_1, \ldots, q_i \; R \; p}{q_1, \ldots, q_{i-1}, r_1, \ldots, r_j \; R \; p} \text{[unreduce]} \qquad \text{if } q_i\text{:-}r_1, \ldots, r_j \; \mathbb{I} \, r_{j+1}, \ldots, r_n$$

(32) **Example:** Consider the following grammar, which shows one way to separate the trigger from the rest of the right side of a rule

```
/*
 * file:  g5-mix.pl
 */
:- op(1200,xfx,:-).

ip :-[dp,i1]+[].       i1 :-[]+[i0,vp].    i0 :-[]+[will].
dp :-[d1]+[].          d1 :-[d0]+[np].     d0 :-[the]+[].
np :-[n1]+[].          n1 :-[n0]+[].       n0 :-[idea]+[].
                       n1 :-[n1]+[pp].
vp :-[v1]+[].          v1 :-[v0]+[].       v0 :-[suffice]+[].
pp :-[p1]+[].          p1 :-[p0]+[dp].     p0 :-[about]+[].
```

For this the following proof shows that the beginnings of `ip` include [dp, i1], [dp], [d1], [d0], [the], []:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{}{\texttt{[dp i1]} \; R \; \texttt{ip}}}{\texttt{[dp]} \; R \; \texttt{ip}}}{\texttt{[d1]} \; R \; \texttt{ip}}}{\texttt{[d0]} \; R \; \texttt{ip}}}{\texttt{[the]} \; R \; \texttt{ip}}}{\texttt{[]} \; R \; \texttt{ip}}$$

[unreduce] [unreduce] [unreduce] [unreduce] [unshift]

Notice that the beginnings of `ip` do <u>not</u> include `[the,idea]`, `[the,i1]`, `[d0, i0]`, `[i0]`, `[i1]`.

(33) GLC parsing with an oracle is defined so that whenever a completed category is placed on the stack, the resulting sequence of completed categories on the stack must be a beginning of the most recently predicted category.

Let's say that a sequence C is **reducible** iff the sequence C is the concatenation of two sequences $C = C_1C_2$ where

a. C_1 is a sequence $\neg p_i, \ldots, \neg p_1$ of $0 \leq i$ completed (i.e. negated) elements

b. C_2 begins with a predicted (i.e. non-negated) element p, and

c. $p_1, \ldots, p_i$ is a beginning of p

(34) GLC parsing with an oracle:

$$\frac{G, \Gamma, S \vdash (?\text{-}\neg q_i, \ldots, \neg q_1, C)}{G, \Gamma, S \vdash (?\text{-}q_{i+1}, \ldots, q_n, \neg p, C)}\ [\text{glc}] \qquad \text{if } (p\text{:-}q_1, \ldots, q_i \rrbracket q_{i+1}, \ldots q_n) \in \Gamma \text{ and } \neg p, C \text{ is reducible}$$

$$\frac{G, \Gamma, S \vdash (?\text{-}\neg q_i, \ldots, \neg q_1, p, C)}{G, \Gamma, S \vdash (?\text{-}q_{i+1}, \ldots, q_n, C)}\ [\text{glc-complete}] \qquad \text{if } (p\text{:-}q_1, \ldots, q_i \rrbracket q_{i+1}, \ldots q_n) \in \Gamma$$

$$\frac{G, \Gamma, wS \vdash (?\text{-}C)}{G, \Gamma, S \vdash (?\text{-}\neg w, C)}\ [\text{shift}] \qquad \text{if } \neg w, C \text{ is reducible}$$

$$\frac{G, \Gamma, wS \vdash (?\text{-}w, C)}{G, \Gamma, S \vdash (?\text{-}C)}\ [\text{shift-complete}] \qquad =\text{scan}$$

(35) This scheme subsumes almost everything covered up to this point: Prolog is an instance of this scheme in which every trigger is empty and the sequence of available "resources" is empty; LL, LR and LC are obtained by setting the triggers at the left edge, right edge, and one symbol in, on the right side of each rule.

(36) To implement GLC parsing with this oracle, we precalculate the beginnings of every category. In effect, we want to find every theorem of the logic given above.

Notice that this kind of logic can allow infinitely many derivations.

(37) **Example.** Consider again `g5-mix.pl` given in (32) above. There are infinitely many derivations of this trivial result:

$$\dfrac{\dfrac{\dfrac{}{\texttt{[n1]}\ R\ \texttt{n1}}}{\texttt{[n1]}\ R\ \texttt{n1}}\text{[unreduce]}}{\texttt{[n1]}\ R\ \texttt{n1}}\text{[unreduce]}}{\cdots}\text{[unreduce]}$$

Nevertheless, the set of theorems, the set of pairs in the "is a beginning of" relation for the grammar `g5-mix.pl` with the trigger relations indicated there, is finite.

We can compute the whole set by taking the closure of the axioms under the inference relation.

(38) Another wrinkle for the implementation: we store our beginnings in reversed, negated form, to make it maximally easy to apply them in GLC reasoning.

$$\neg q_i, \ldots, \neg q_1 \ \ nrR \ \ p \qquad \text{[axiom-r]} \qquad \text{if } p\text{:-}q_1, \ldots, q_i \ \mathbb{I} q_{i+1}, \ldots, q_n$$

$$\dfrac{\neg q_i, \ldots, \neg q_1 \ \ nrR \ \ p}{\neg q_{i-1}, \ldots, \neg q_1 \ \ nrR \ \ p}\text{[unscan-r]} \qquad \text{if } q_i \in \Sigma$$

$$\dfrac{\neg q_i, \ldots, \neg q_1 \ \ nrR \ \ p}{\neg r_j, \ldots, \neg r_1, \neg q_{i-1}, \ldots, \neg q_1 \ \ nrR \ \ p}\text{[unreduce-r]} \qquad \text{if } q_i\text{:-}r_1, \ldots, r_j \ \mathbb{I} r_{j+1}, \ldots, r_n$$

(39) We will use code from Shieber, Schabes, and Pereira (1993) to compute the closure of these axioms under the inference rules.[26]

The following two files do what we want. (We have one version specialized for sicstus prolog, and one for swiprolog.)

```
/* oracle-sics.pl
* E Stabler, Feb 2000
*/
:- op(1200,xfx,:-).  % this is our object language "if"
:- ['closure-sics'].  % defines closure/2, uses inference/4
:- use_module(library(lists),[append/3,member/2]).

%verbose.  % comment to reduce verbosity of chart construction
computeOracle :- abolish(nrR), setof(Axiom,axiomr(Axiom),Axioms), closure(Axioms, Chart), asserts(Chart).

axiomr(nrR(NRBA)) :- (A :-B+_), negreverse(B,[A|_],NRBA).

negreverse([],M,M).
negreverse([E|L],M,N) :- negreverse(L,[-E|M],N).

inference(unreduce-r/2,
     [ nrR([-Qi|Qs]) ],
     nrR(NewSeq),
     [(Qi :-Prefix+_),
     negreverse(Prefix,[],NRPrefix),
     append(NRPrefix,Qs,NewSeq) ]).

inference(unscan-r/2,
     [ nrR([-Qi|Qs]) ],
     nrR(Qs),
     [ \+ (Qi :-_) ]).

asserts([]).
asserts([nrR([-C|Cs])|L]) :- !, assert(nrR([-C|Cs])), asserts(L).
asserts([_|L]) :- asserts(L).
```

The second file provides the interface to code from Shieber, Schabes, and Pereira (1993):

```
/* closure-sics.pl
* E. Stabler, 16 Oct 99
* interface to the chart mechanism of Shieber, Schabes & Periera (1993)
*/

:- ['shieberetal93-sics/chart.pl'].
:- ['shieberetal93-sics/agenda.pl'].
:- ['shieberetal93-sics/items.pl'].
:- ['shieberetal93-sics/monitor.pl'].
:- ['shieberetal93-sics/driver'].
:- ['shieberetal93-sics/utilities'].

closure(InitialSet,Closure) :-
     init_chart,
     empty_agenda(Empty),
     add_items_to_agenda(InitialSet, Empty, Agenda),
     exhaust(Agenda),
     setof(Member,Index^stored(Index,Member),Closure).


% item_to_key/2 should be specialized for the relations being computed

item_to_key(F, Hash) :- hash_term(F, Hash).
```

(40) The GLC parser that checks the oracle is then:

```
/* file:  glco.pl
* E Stabler, Feb 2000
*/
:- op(1200,xfx,:-).  % this is our object language "if"
:- op(1100,xfx,?-).  % metalanguage provability predicate

[] ?-[].
(S0 ?-Goals0) :- infer(S0,Goals0,S,Goals), (S ?-Goals).

infer([W|S],[W|C],S,C). % shift-complete=scan
infer([W|S],C,S,[-W|C]) :- nrR([-W|C]). % shift

infer(S,RPC,S,DC) :- (A :-P+D), preverse(P,RPC,[A|C]), append(D,C,DC). % reduce-complete
infer(S,RPC,S,DAC) :- (A :-P+D), preverse(P,RPC,C), nrR([-A|C]), append(D,[-A|C],DAC). % reduce

%preverse(ExpansionD,ReversedExpansionD,RestConstituents)
preverse([E|L],RD,C) :- preverse(L,RD,[-E|C]).
preverse( [], C,C).

append([],L,L).
append([E|L],M,[E|N]) :- append(L,M,N).
```

(41) With these files, we get the following session: `| ?- ['g5-mix','oracle-sics',glco].`

```
yes

| ?- computeOracle.
Warning:  abolish(user:nrR) - no matching predicate
...............:::.:.:.:.:.:.:..:..:.:.:.:.:.:.::.:.:.::.:.:.:.:::.:.::.:.::.::.::.::.::::.:
yes
| ?- listing(nrR).
nrR([-(about),p0|_]).
nrR([-(about),p1|_]).
nrR([-(about),pp|_]).
nrR([-(d0),d1|_]).
nrR([-(d0),dp|_]).
nrR([-(d0),ip|_]).
nrR([-(d1),dp|_]).
nrR([-(d1),ip|_]).
nrR([-(dp),ip|_]).
nrR([-(i1),-(dp),ip|_]).
nrR([-(idea),n0|_]).
nrR([-(idea),n1|_]).
nrR([-(idea),np|_]).
nrR([-(n0),n1|_]).
nrR([-(n0),np|_]).
nrR([-(n1),n1|_]).
nrR([-(n1),np|_]).
nrR([-(p0),p1|_]).
nrR([-(p0),pp|_]).
nrR([-(p1),pp|_]).
nrR([-(suffice),v0|_]).
nrR([-(suffice),v1|_]).
nrR([-(suffice),vp|_]).
nrR([-(the),d0|_]).
nrR([-(the),d1|_]).
nrR([-(the),dp|_]).
nrR([-(the),ip|_]).
nrR([-(v0),v1|_]).
nrR([-(v0),vp|_]).
nrR([-(v1),vp|_]).
yes

| ?- [the,idea,will,suffice] ?-[ip].

yes
| ?- [the,idea,about,the,idea,will,suffice] ?-[ip].

yes
| ?- [will,the,idea,suffice] ?-[ip].

no
| ?- [the,idea] ?-[C].

C = d1 ?  ;

C = dp ?  ;


no
```

(42) With this oracle, we can repair `g3.pl` to allow the left recursive rule for coordination, and an empty `i0`. The resulting file `g3-lc.pl` lets us recognize, for example, the ip: `some penguins and most songs praise titus`. As observed earlier, we were never able to do this before.[27]

(43) The calculation of the oracle can still fail to terminate. To ensure termination, we need to restrict both the length of the trigger sequences, and also the complexity of the arguments (if any) associated with the categories in the sequence.

6.5.2 Bottom-up oracles: lookahead

(44) In top-down parsing, there is no point in using an expansion p:-q, r if the next symbol to be parsed could not possibly be the beginning of a q.

To guide top-down steps, it would be useful to know what symbol (or what k symbols) are next, waiting to be parsed. This "bottom-up" information can be provided with a "lookahead oracle."

Obviously, the "lookahead" oracle does not look into the future to hear what has not been spoken yet. Rather, structure building waits for a word (or in general, k words) to be heard.

Again, we will precompute, for each category p, what the first k symbols of the string could be when we are recognizing that category in a successful derivation of any sentence.

(45) In calculating lookahead, we ignore the triggers.

One kind of situation that we must allow for is this. If p:-$q_1, \ldots, q_n$ and $q_1, \ldots, q_i \Rightarrow^* \epsilon$, then every next symbol for q_{i+1} is a next symbol for p.

(46) For any $S \in \Sigma^*$, let $\text{first}_k(S)$ be the first k symbols of S if $|S| \geq k$, and otherwise $\text{first}_k(S) = S$.

We can use the following reasoning to calculate all of the next k words that can be waiting to be parsed as each category symbol is expanded. For some $k > 0$:

$$wLAw \quad \text{[axiom]} \qquad \text{if } w \in \Sigma$$

$$xLAp \quad \text{[axiom]} \qquad \text{if } p\text{:-}q_1, \ldots, q_n \text{ and either } x = q_1, \ldots, q_k \in \Sigma^k \text{ for } k \leq n, \text{ or } x = q_1, \ldots, q_n \in \Sigma^n \text{ for } n < k$$

$$\frac{x_1 LA q_1 \quad \ldots \quad x_n LA q_n}{xLAp} \text{ [la]} \qquad \text{if } p\text{:-}q_1, \ldots, q_n, \text{ and } x = \text{first}_k(x_1 \ldots x_n)$$

And we let $first_k(x_1 \ldots x_n) LA (q_1, \ldots, q_n)$ if $x_i LA q_i$ for $1 \leq i \leq n$.

(47) GLC parsing with two oracles:

$$\frac{G, \Gamma, S \vdash (?\text{-}\neg q_i, \ldots, \neg q_1, C)}{G, \Gamma, S \vdash (?\text{-}q_{i+1}, \ldots, q_n, \neg p, C)} \text{ [glc]} \qquad \begin{array}{c}\text{if } (p\text{:-}q_1, \ldots, q_i \rrbracket q_{i+1}, \ldots q_n) \in \Gamma \\ \neg p, C \text{ is reducible, and } \text{first}_k(S) LA (q_{i+1}, \ldots, q_n)\end{array}$$

$$\frac{G, \Gamma, S \vdash (?\text{-}\neg q_i, \ldots, \neg q_1, p, C)}{G, \Gamma, S \vdash (?\text{-}q_{i+1}, \ldots, q_n, C)} \text{ [glc-complete]} \qquad \begin{array}{c}\text{if } (p\text{:-}q_1, \ldots, q_i \rrbracket q_{i+1}, \ldots q_n) \in \Gamma \\ \text{and } \text{first}_k(S) LA (q_{i+1}, \ldots, q_n)\end{array}$$

$$\frac{G, \Gamma, wS \vdash (?\text{-}C)}{G, \Gamma, S \vdash (?\text{-}\neg w, C)} \text{ [shift]} \qquad \text{if } \neg w, C \text{ is reducible}$$

$$\frac{G, \Gamma, wS \vdash (?\text{-}w, C)}{G, \Gamma, S \vdash (?\text{-}C)} \text{ [shift-complete]} \qquad \text{=scan}$$

(48) We can compute the bottom-up $k = 1$ words of lookahead offline:

```
| ?- ['la-sics'].

yes
| ?- ['g1-lc'].

yes
| ?- computela.
Warning:  abolish(user:la) - no matching predicate
..........:.:.:.:.::.::.::.:.:.:.:.::.:.:.:.
yes
| ?- listing(la).
la([idea], idea).
la([idea], n0).
la([idea], n1).
la([idea], np).
la([suffice], suffice).
la([suffice], v0).
la([suffice], v1).
la([suffice], vp).
la([that], c0).
la([that], c1).
la([that], cp).
la([that], that).
la([the], d0).
la([the], d1).
la([the], dp).
la([the], ip).
la([the], the).
la([will], i0).
la([will], i1).
la([will], will).

yes
| ?-
```

(49) Adding bottom-up k-symbol lookahead to the `glco` recognizers with a bottom-up oracle, we have `glcola(`k`)` recognizers.

For any of the GLC parsers, a language is said to be `glcola(`k`)` iff there is at most one step that can be taken at every point in every proof.

Obviously, when a language has genuine structural ambiguity – more than one successful parse for some strings in the language – the language cannot be `glcola(`k`)` for any k (e.g. LL(k), LC(k), LR(k),…) In the case of an ambiguous language, though, we can consider whether the recognition of unambiguous strings is deterministic, or whether the indeterminism that is encountered is all due to global ambiguities. We return to these questions below.

6.6 Assessment of the GLC ("stack based") parsers

6.6.1 Termination

(50) We have not found any recognition method that is guaranteed to terminate (i.e. has a finite search space) on any input, even when the grammar has left recursion and empty categories. In fact, it is obvious that we do not want to do this, since a context free grammar can have infinitely ambiguous strings.

6.6.2 Coverage

(51) The GLC recognition methods are designed for CFGs. Human languages have structures that are only very inelegantly handled by CFGs, and structures that seem beyond the power of CFGs, as we mentioned earlier (Savitch et al., 1987).

6.6.3 Ambiguity (local and global) vs. glcola(k) parsing

(52) Ambiguity is good.
If you know which Clinton I am talking about, then I do not need to say "William Jefferson Clinton." Doing so violates normal conventions about being brief and to-the-point in conversation (Grice, 1975), and consequently calls for some special explanation (e.g. pomposity, or a desire to signal formality). A full name is needed when one Clinton needs to be distinguished from another. For most of us non-Clintons, in most contexts, using just "Clinton" is enough, even though the name is semantically ambiguous.

(53) For the same reason, it is no surprise that standard Prolog uses the list constructor "." both as a function to build lists and as a predicate whose "proof" triggers loading a file. Some dialects of Prolog also use "/" in some contexts to separate a predicate name from its arity, and in other contexts for division. This kind of multiple use of an expression is harmless in context, and allows us to use shorter expressions.

(54) There is a price to pay in parsing, since structural ambiguities must be resolved. Some of these ambiguities are resolved definitively by the structure of the sentence; other ambiguities persist throughout a whole sentence and are resolved by discourse context. It is natural to assume that these various types of ambiguity are resolved by similar mechanisms in human language understanding, but of course this is an empirical question.

(55) **Global ambiguity (unresolved by local structure)** How much structural ambiguity do sentences of human languages really have?[28] We can get a first impression of how serious the structural ambiguity problem is by looking at simple artificial grammars for these constructions.

a. PP attachment in $[_{VP}$ V D N PP1 PP2 ...]
Consider the grammar:

VP → V NP PP*
NP → D N PP*

A grammar like this cannot be directly expressed in standard context free form. It defines a context free language, but it is equivalent to the following infinite grammar:

np → d n	vp → v np
np → d n pp	vp → v np pp
np → d n pp pp	vp → v np pp pp
np → d n pp pp pp	vp → v np pp pp pp
np → d n pp pp pp pp	vp → v np pp pp pp pp
np → d n pp pp pp pp pp	vp → v np pp pp pp pp pp
…	…

[28] Classic discussions of this point appear in, Church and Patil (1982) and Langendoen, McDaniel, and Langsam (1989).

The number of structures defined by this grammar is an exponential function of the number of words.

N =	1	2	3	4	5	6	7	8	9	10
#trees =	2	5	14	132	469	1430	4862	16796	1053686	…

b. N compounds $[_N$ N N]

n → n n

This series same as for PPs, except PP^1 is like a N^3 compound:

n	1	2	3	4	5	6	7	8	9	10
#trees	1	1	2	5	14	42	132	429	1420	4862

c. Coordination $[_X$ X and X]

NP → NP (and NP)*

This is equivalent to the grammar:

np → np and np
np → np and np and np
np → np and np and np and np
np → np and np and np and np and np
np → np and np and np and np and np and np
np → np and np and np and np and np and np and np
np → np and np and np and np and np and np and np and np
np → np and np and np and np and np and np and np and np and np
…

n	1	2	3	4	5	6	7	8	9	10
#trees	1	1	3	11	45	197	903	4279	20793	103049

(56) **Structurally resolved (local) ambiguity**

a. Agreement: In simple English clauses, the subject and verb agree, even though the subject and verb can be arbitrarily far apart:
   a. The deer {are, is} in the field
   b. The deer, the guide claims, {are, is} in the field

b. Binding: The number of the embedded subject is unspecified in the following sentence:
   a. I expect the deer to admire the pond.

   But in similar sentences it can be specified by binding relations:
   b. I expect the deer to admire {itself,themselves} in the reflecions of the pond.

c. Head movement:
   a. i. Have the students take the exam!
      ii. Have the students taken the exam?
   b. i. Is the block sitting in the box?
      ii. Is the block sitting in the box red?

d. A movement:
   a. i. The chair$_i$ is t_i too wobbly for the old lady to sit on it
      ii. The chair$_i$ is t_i too wobbly for the old lady to sit on t_i

e. A' movement:
   a. i. Who$_i$ did you help t_i to prove that John was unaware that he had been leaking secrets
      ii. Who$_i$ did you help t_i to prove that John was unaware t_i that he had been leaking secrets to t_i

(57) English is not $glcola(k)$ for any triggers and any k.

Marcus (1980) pointed out that the ambiguity in, for example, (56c-a), is beyond the ability of an $LR(k)$ parser, as is easy to see. Since LR is the bottom-up extreme of the GLC lattice, delaying structural decisions the longest, the fact that English and other human languages are not $LR(k)$ means that they are not deterministic with k symbols of lookahead for any of the $glcola(k)$ parsers.

(58) Marcus (1980) made two very interesting proposals about the structural ambiguity of sentence prefixes.[29]

a. **First: (at least some) garden paths indicate failed local ambiguity resolution.** Marcus proposes that humans have difficulty with certain local ambiguities (or fail completely), resulting in the familiar "garden path" effects:[30] The following sentences exhibit extreme difficulty, but other less extreme variations in difficulty may also evidence the greater or less backtracking involved:

   a. The horse raced past the barn fell
   b. Horses raced past barns fall
   c. The man who hunts ducks out on weekends
   d. Fat people eat accumulates
   e. The boat floated down the river sank
   f. The dealer sold the forgery complained
   g. Without her contributions would be impossible

   This initially very plausible idea has not been easy to defend. One kind of problem is that some constructions which should involve backtracking are relatively easy: see for example the discussions in Pritchett (1992) and Frazier and Clifton (1996).

b. **Second: to reduce backtracking to human level, delay decisions until next constituent is built.** Suppose we agree that some garden paths should be taken as evidence of backtracking: we would like to explain why sentences like the ones we were considering earlier (repeated here) are not as difficult as the garden paths just mentioned:

   a. i. Have the students take the exam!
      ii. Have the students taken the exam?
   b. i. Is the block sitting in the box?
      ii. Is the block sitting in the box red?

   The reason that k symbols of lookahead will not resolve these ambiguities is that the disambiguating words are on the other side of a noun phrase, and noun phrases can be arbitrarily long. So Marcus proposes that when confronted with such situations, the human parser delays the decision until after the next phrase is constructed. In effect, this allows the parser to look some finite number of constituents ahead, instead of just a finite number of words ahead.[31] This is an appealing idea which may deserve further consideration in the context of more recent proposals about human languages.

[29] These proposals were developed in various ways by Berwick and Weinberg (1984), Nozohoor-Farshi (1986), and van de Koot (1990). These basic proposals are critiqued quite carefully by Fodor (1985) and by Pritchett (1992).

[30] There are many studies of garden path effects in human language understanding. Some of the prominent early studies are the following: Bever (1970), Frazier (1978), Frazier and Rayner (1982), Ford, Bresnan, and Kaplan (1982), Crain and Steedman (1985), Pritchett (1992).

[31] Parsing strategies of this kind are sometimes called "non-canonical." They were noticed by Knuth (1965), and developed further by Szymanski and Williams (1976). They are briefly discussed in Aho and Ullman (1972, §6.2). A formal study of Marcus's linguistic proposals is carefully done by Nozohoor-Farshi (1986).

6.6.4 A dilemma

(59) A dilemma for models of human language understanding:

a. The fact that ordinary comprehension of spoken language is incremental suggests that the parsing method should be quite top-down (Steedman, 1989; Stabler, 1991; Shieber and Johnson, 1994).

b. The fact that people can readily allow the relative clause in strings like the following to modify either noun, suggests that structure building cannot be top-down:

The daughter of the colonel who was standing on the balcony

At least, the choice about whether to build the high attachment point for the relative clause cannot be made before *of* is scanned.

This problem is discussed by Frazier and Clifton (1996) and many others.

(60) These difficulties merit giving some attention to alternatives that are outside of the range of glc parsers. We consider some quite different parsing strategies in the next section.

6.6.5 Exercise

Consider this modified lr parser which counts the number of steps in the search for the first parse:

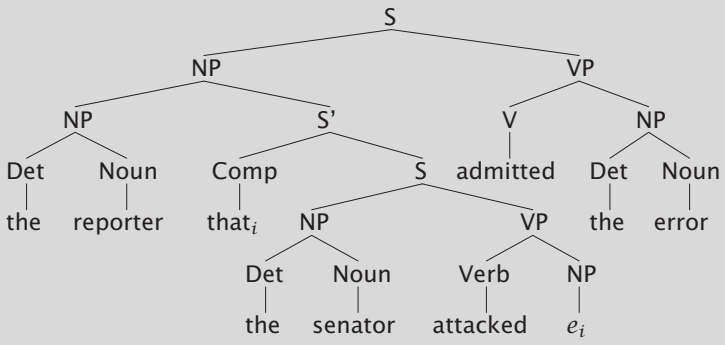
```
/*
 * file: lrp-cnt.pl              lr parser, with count of steps in search
 */
:- op(1200,xfx,:~).   % this is our object language "if"
:- op(1100,xfx,?~).   % metalanguage provability predicate
:- op(500,yfx,@).     % metalanguage functor to separate goals from trees

% PARSER with countStep
[] ?~ []@[].
(S0 ?~ Goals0@T0) :- infer(S0,Goals0@T0,S,Goals@T), countStep, (S ?~ Goals@T).

infer(S,RDC@RDCTs,S,C@CTs) :- (A :~ D), preverse(D,RDC,[A|C],DTs,RDCTs,[A/DTs|CTs]).   % reduce-complete
infer(S,RDC@RDCTs,S,[-A|C]@[A/DTs|CTs]) :- (A :~ D), preverse(D,RDC,C,DTs,RDCTs,CTs). % reduce
infer([W|S],C@CTs,S,[-W|C]@[W/[]|CTs]).                                              % shift

preverse([],C,C,[],CTs,CTs).
preverse([E|L],RD,C,[ETs|LTs],RDTs,CTs) :- preverse(L,RD,[-E|C],LTs,RDTs,[ETs|CTs]).

% COUNTER: count steps to find the first parse, with start category 'S'

parseOnce(S0,T) :-
        retractall(cnt),        % remove any previous counter
        (S0 ?~ ['S']@[T]),!,    % parse the string just once
        retract(cnt(X)), print('Number of steps in search':X), nl.

:- dynamic cnt/1.  % declare a predicate whose definition we change

countStep :- retract(cnt(X)), !, X1 is X+1, assert(cnt(X1)).
countStep :- assert(cnt(1)).

%% ES GRAMMAR, with simple object topicalization

'S' :~ ['DP','VP'].     'S/DP' :~ ['DP','VP/DP'].
'S' :~ ['DP','S/DP'].   'VP/DP' :~ ['Vt'].

'VP' :~ ['V'].          'DP' :~ ['D','NP'].    'NP' :~ ['N'].
'VP' :~ ['Vt','DP'].    'D' :~ [the].          'NP' :~ ['AP','NP'].
'V' :~ [laugh].         'D' :~ [some].         'NP' :~ ['NP','PP'].
'V' :~ [cry].           'D' :~ [no].           'N' :~ [students].
'Vt' :~ [love].                                'N' :~ [homeworks].
'Vt' :~ [hate].                                'N' :~ [trees].
                                               'N' :~ [lists].

'AP' :~ ['A'].          'DegP' :~ ['Deg'].     'PP' :~ ['P','DP'].
'AP' :~ ['DegP','A'].   'Deg' :~ [really].     'P' :~ [about].
'A' :~ [good].          'Deg' :~ [kind,of].    'P' :~ [with].
'A' :~ [bad].

% EXAMPLES for ES GRAMMAR:
% parseOnce([the,students,laugh],T),wish_tree(T).
% parseOnce([the,students,hate,the,homeworks],T).
% parseOnce([the,lists,the,students,hate],T).
% parseOnce([the,bad,homeworks,the,students,love],T).
% parseOnce([the,kind,of,good,students,hate,the,homeworks],T).
% WORST 6-WORD SENTENCE IS: ???

% EXAMPLES for YOUR GRAMMAR:
% WORST 6-WORD SENTENCE IS: ???
```

1. Download the file `lrp-cnt.pl` and modify the definition of `parseOnce` so that in addition to printing out the number of steps in the search, it also prints out the number of steps in the derivation (= the number of nodes in the tree).
2. Which 6 word sentence requires the longest search with this ES grammar? Put your example in at the end of the file.
3. Add 1 or 2 rules to the grammar (don't change the parser) in order to produce an even longer search on a 6 word sentence – as long as you can make it (but not infinite = no empty categories). Put your example at the end of the file, and turn in the results of all these steps.

6.6.6 Additional Exercise (for those who read the shaded blocks)

(61) Gibson (1998) proposes

> For initial purposes, a syntactic theory with a minimal number of syntactic categories, such as Head-driven Phrase Structure Grammar (Pollard and Sag, 1994) or Lexical Functional Grammar (Bresnan, 1982), will be assumed. [Note: The SPLT is also compatible with grammars assuming a range of functional categories such as Infl, Agr, Tense, etc. (e.g. Chomsky 1995) under the assumption that memory cost indexes predicted *chains* rather than predicted categories, where a chain is a set of categories coindexed through movement (Chomsky 1981).] Under these theories, the minimal number of syntactic head categories in a sentence is two: a head noun for the subject and a head verb for the predicate. If words are encountered that necessitate other syntactic heads to form a grammatical sentence, then these categories are also predicted, and an additional memory load is incurred. For example, at the point of processing the second occurrence of the word "the" in the object-extracted RC example,
>
> 1a. The reporter who the senator attacked admitted his error,



> there are four obligatory syntactic predictions: 1) a verb for the matrix clause, 2) a verb for the embedded clause, 3) a subject noun for the embedded clause, and an empty category NP for the wh-pronoun "who."

Is the proposed model a glc parser? If not, is the proposal a cogent one, one that conforms to the behavior of a parsing model that could possibly work?

(62) **References.** Basic parsing methods are often introduced in texts about building compilers for programming languages, like Aho, Sethi, and Ullman (1985). More comprehensive treatments can be found in Aho and Ullman (1972), and in Sikkel (1997).

7 Context free parsing: dynamic programming methods

In this section, we consider recognition methods that work for all context free grammars. These are "dynamic programming" methods in the general sense that they involve computing and recording the solutions to basic problems, from which the full solution can be computed. It is crucial that the number of basic problems be kept finite. As we will see, sometimes it is difficult or impossible to assure this, and then these dynamic methods will not work. Because records of subproblem solutions are kept and looked up as necessary, these methods are commonly called "tabular" or "chart-based."

We saw in §6.6.3 that the degree of ambiguity in English sentences can be very large. (We showed this with a theoretical argument, but we will see that, in practice, the situation is truly problematic). The parsing methods introduced here can avoid this problem, since instead of producing explicit representations of each parse, we will produce a chart representation of a grammar that generates exactly the input.

If you have studied formal language theory, you might remember that the result of intersecting any context free language with a finite state language is still context free. The standard proof for this is constructive: it shows how, given any context free grammar and finite state grammar (or automaton), you can construct a context free grammar that generates exactly the language that is the intersection you want. This is the idea we use here, and it can be used not only for recognition but also for parsing, since the grammar represented by a chart will indicate the derivations from the original grammar (even when there are a large number, or infinitely many of them).

7.1 CKY recognition for CFGs

(1) For simplicity, we first consider context free grammars $G = \langle \Sigma, N, \rightarrow, \mathsf{S} \rangle$ in Chomsky normal form: Chomsky normal form grammars have rules of only the following forms, for some $A, B, C \in N$, $w \in \Sigma$,

$$\mathrm{A} \rightarrow \mathrm{BC} \qquad \mathrm{A} \rightarrow \mathrm{w}$$

If ϵ is in the language, then the following form is also allowed, subject to the requirement that S does not occur on the right side of any rule:

$$\mathrm{S} \rightarrow \epsilon$$

(2) A Chomsky normal form grammar has no unit or empty productions, and hence no "cycles" $A \Rightarrow^+ A$, and no infinitely ambiguous strings.

(3) These grammars allow an especially simple CKY-like tabular parsing method. To parse a string $w_1, \ldots, w_n$, for $n > 0$ we use the following logic:

$$(i-1, i) : w_i \qquad \text{[axioms]}$$

$$\frac{(i, j) : w}{(i, j) : A} \qquad \text{[reduce1]} \qquad \text{if } A \rightarrow w$$

$$\frac{(i, j) : B \quad (j, k) : C}{(i, k) : A} \qquad \text{[reduce2]} \qquad \text{if } A \rightarrow BC$$

Recognition is successful iff $(0, n) : S$ is in the closure of the axioms under these inference rules.

This recognition method is based on proposals of Cocke, Kasami and Younger (Aho and Ullman, 1972; Shieber, Schabes, and Pereira, 1993; Sikkel and Nijholt, 1997).

As will become clear when we collect trees from the closure, the closure, in effect, represents all derivations, but the representation is reasonably sized even when the number of derivations is infinite, because the number of possible items is finite.

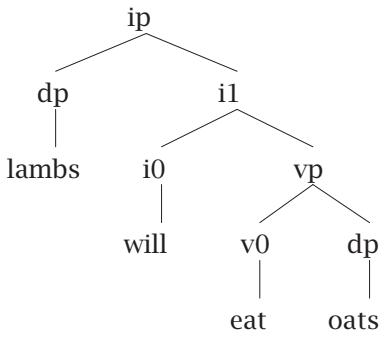
(4) The soundness and completeness of this method are shown in Shieber, Schabes, and Pereira (1993). Aho and Ullman (1972, §4.2.1) show in addition that for a sentence of length n, the maximum number of steps needed to compute the closure is proportional to n^3.

Aho and Ullman (1972) say of this recognition method:

It is essentially a "dynamic programming" method and it is included here because of its simplicity. It is doubtful, however, that it will find practical use, for three reasons:

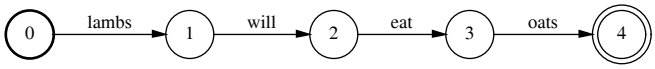
1. n^3 time is too much to allow for parsing.
2. The method uses an amount of space proportional to the square of the input length.
3. The method of the next section (Earley's algorithm) does at least as well in all respects as this one, and for many grammars does better.

7.1.1 CKY example 1




```
ip → dp i1
dp → lambs
i1 → i0 vp
i0 → will
vp → v0 dp
v0 → eat
dp → oats
```

The axioms can be regarded as specifying a finite state machine representation of the input:



Given an n state finite state machine representation of the input, computing the CKY closure can be regarded as filling in the "upper triangle" of an $n \times n$ matrix, from the (empty) diagonal up:[32]

	0	1	2	3	4
0		(0,1):dp (0,1):lambs			(0,4):ip
1			(1,2):i0 (1,2):will		(1,4):i1
2				(2,3):v0 (2,3):eat	(2,4):vp
3					(3,4):dp (3,4):oats
4					

[32] CKY tables and other similar structures of intermediate results are frequently constructed by matrix operations. This idea has been important in complexity analysis and in attempts to find the fastest possible parsing methods (Valiant, 1975; Lee, 1997). Extensions of matrix methods to more expressive grammars are considered by Satta (1994) and others.

7.1.2 CKY extended

(5) We can relax the requirement that the grammar be in Chomsky normal form. For example, to allow arbitrary empty productions, and rules with right sides of length 3,4,5,6, we could add the following rules:

$$\frac{}{(i,i):A} \quad \text{[reduce0]} \quad \text{if } A \to \epsilon$$

$$\frac{(i,j):B \quad (j,k):C \quad (k,l):D}{(i,l):A} \quad \text{[reduce3]} \quad \text{if } A \to BCD$$

$$\frac{(i,j):B \quad (j,k):C \quad (k,l):D \quad (l,m):E}{(i,m):A} \quad \text{[reduce4]} \quad \text{if } A \to BCDE$$

$$\frac{(i,j):B \quad (j,k):C \quad (k,l):D \quad (l,m):E \quad (m,n):F}{(i,n):A} \quad \text{[reduce5]} \quad \text{if } A \to BCDEF$$

$$\frac{(i,j):B \quad (j,k):C \quad (k,l):D \quad (l,m):E \quad (m,n):F \quad (n,o):G}{(i,o):A} \quad \text{[reduce6]} \quad \text{if } A \to BCDEFG$$

(6) While this augmented parsing method is correct, it pays a price in efficiency. The Earley method of the next section can do better.

(7) Using the strategy of computing closures (introduced in (39) on page 91), we can implement the CKY method with these extensions easily:

```
/* ckySWI.pl
 * E Stabler, Feb 2000
 * CKY parser, augmented with rules for 3,4,5,6-tuples
 */
:- op(1200,xfx,:~).    % this is our object language "if"
:- ['closure-swi'].   % Shieber et al's definition of closure/2, uses inference/4

%verbose. % comment to reduce verbosity of chart construction

computeClosure(Input) :-
        computeClosure(Input,Chart),
        nl,portray_clauses(Chart).

/*  ES tricky way to get the axioms from reduce0:                    */
/*  add them all right at the start                                  */
/* (there are more straightforward ways to do it but they are slower) */
computeClosure(Input,Chart) :-
        lexAxioms(0,Input,Axioms0),
        findall((Pos,Pos):A,(A :~ []),Empties),  /* reduce0 is here! */
        append(Axioms0,Empties,Axioms),
        closure(Axioms, Chart).

lexAxioms(_Pos,[],[]).
lexAxioms(Pos,[W|Ws],[(Pos,Pos1):W|Axioms]) :-
        Pos1 is Pos+1,
        lexAxioms(Pos1,Ws,Axioms).

inference(reduce1,
          [ (Pos,Pos1):W ],
            (Pos,Pos1):A,
          [(A :~ [W])] ).

inference(reduce2,
          [ (Pos,Pos1):B, (Pos1,Pos2):C],
            (Pos,Pos2):A,
          [(A :~ [B,C])] ).

/* for efficiency, comment out the rules you never use */

inference(reduce3,
          [ (Pos,Pos1):B, (Pos1,Pos2):C, (Pos2,Pos3):D],
            (Pos,Pos3):A,
          [(A :~ [B,C,D])] ).

inference(reduce4,
          [ (Pos,Pos1):B, (Pos1,Pos2):C, (Pos2,Pos3):D, (Pos3,Pos4):E],
            (Pos,A,Pos4),
          [(A :~ [B,C,D,E])] ).

inference(reduce5,
          [ (Pos,Pos1):B, (Pos1,Pos2):C, (Pos2,Pos3):D, (Pos3,Pos4):E, (Pos4,Pos5):F],
            (Pos,Pos5):A,
          [(A :~ [B,C,D,E,F])] ).
```

```
inference(reduce6,
          [ (Pos,Pos1):B, (Pos1,Pos2):C, (Pos2,Pos3):D, (Pos3,Pos4):E, (Pos4,Pos5):F, (Pos5,Pos6):G],
            (Pos,Pos6):A,
          [(A :¯ [B,C,D,E,F,G])] ).

portray_clauses([]).
portray_clauses([C|Cs]) :- portray_clause(C), portray_clauses(Cs).
```

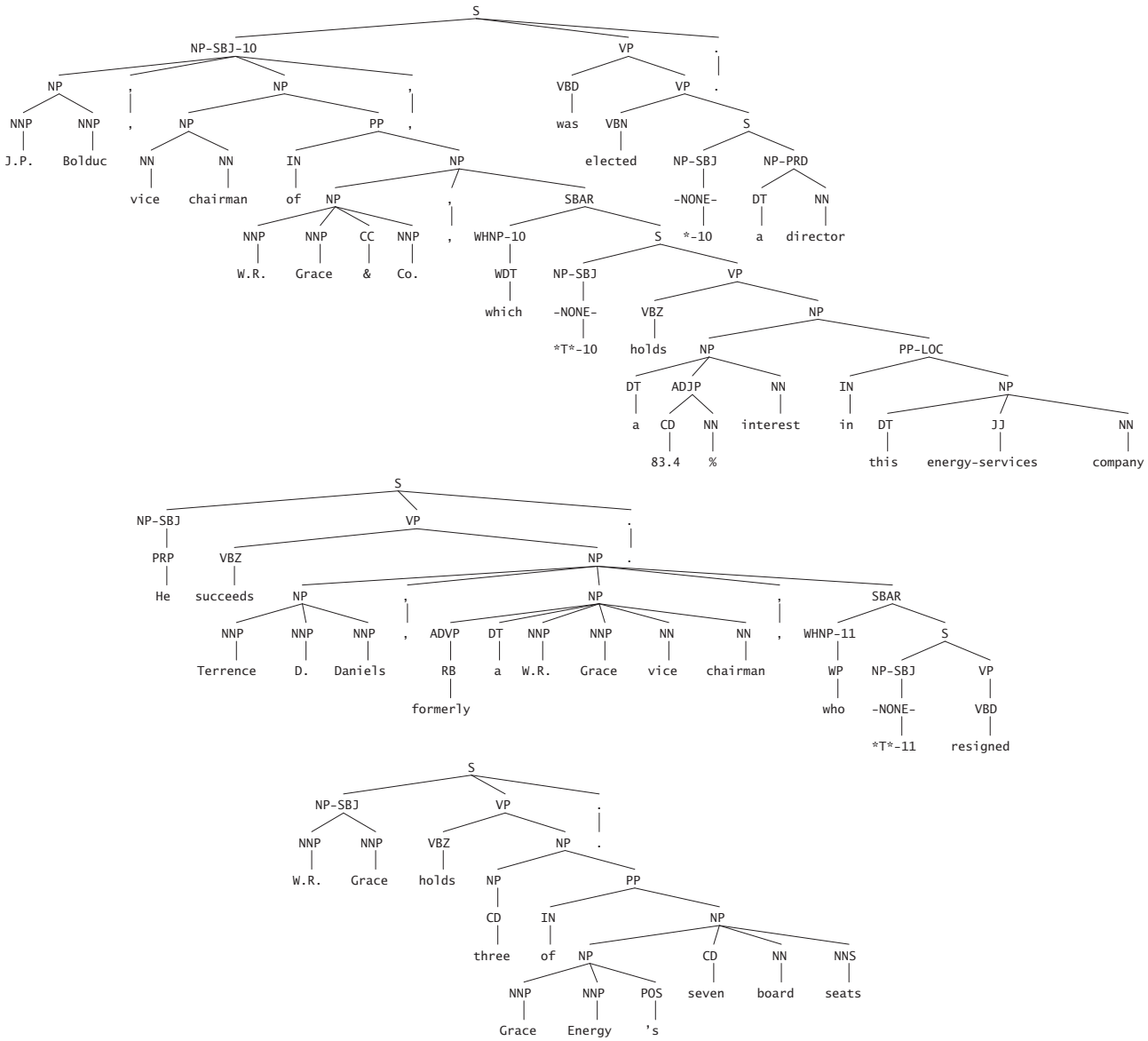
(8) With this code, we get sessions like the following. Notice that the first but not the second example produces the goal item `(0,4):ip`, which is correct.

```
1 ?- [ckySWI].
%   chart compiled 0.00 sec, 1,672 bytes
%   agenda compiled 0.00 sec, 3,056 bytes
%   items compiled 0.00 sec, 904 bytes
%   monitor compiled 0.00 sec, 2,280 bytes
%   driver compiled 0.00 sec, 3,408 bytes
%   utilities compiled 0.00 sec, 1,052 bytes
%  closure-swi compiled 0.00 sec, 13,936 bytes
% ckySWI compiled 0.00 sec, 18,476 bytes

Yes
2 ?- [g1].
% g1 compiled 0.00 sec, 2,388 bytes

Yes
3 ?- computeClosure([the,idea,will,suffice]).
'.''.''.''.':'.':'.':'.':'.':'.'::'.'::'.':'.':'.':'.':'.':'.':'.'::'.':
(0, 1):d0.
(0, 1):the.
(0, 2):d1.
(0, 2):dp.
(0, 4):ip.
(1, 2):idea.
(1, 2):n0.
(1, 2):n1.
(1, 2):np.
(2, 3):i0.
(2, 3):will.
(2, 4):i1.
(3, 4):suffice.
(3, 4):v0.
(3, 4):v1.
(3, 4):vp.

Yes
4 ?- computeClosure([will,the,idea,suffice]).
'.''.''.''.':'.':'.':'.':'.':'.':'.':'.':'.':'.':'.':'.':'.':'.'::'.':
(0, 1):i0.
(0, 1):will.
(1, 2):d0.
(1, 2):the.
(1, 3):d1.
(1, 3):dp.
(2, 3):idea.
(2, 3):n0.
(2, 3):n1.
(2, 3):np.
(3, 4):suffice.
(3, 4):v0.
(3, 4):v1.
(3, 4):vp.

Yes
5 ?-
```

Notice that unlike the first example the latter example is not accepted by the grammar, as we can see from the fact that there is no ip (or any other category) that spans from the beginning of the string to the end, from 0 to 4.

7.1.3 CKY example 2

Since we can now recognize the language of any context free grammar, we can take grammars written by anyone else and try them out. For example, we can take the grammar defined by the Penn Treebank and try to parse with it. For example, in the file `wsj_0005.mrg` we find the following 3 trees:



Notice that these trees indicate movement relations, with co-indexed traces. If we ignore the movement relations and just treat the traces as empty, though, we have a CFG – one that will accept all the strings that are parsed in the treebank plus some others as well.

We will study how to parse movements later, but for the moment, let's collect the (overgenerating) context free rules from these trees. Dividing the lexical rules from the others, and showing how many times each rule is used, we have first:

```
1 ('SBAR':¯['WHNP-11','S']).
```

```
1 ('SBAR':~['WHNP-10','S']).
2 ('S':~['NP-SBJ','VP']).
2 ('S':~['NP-SBJ','VP','.']).
1 ('S':~['NP-SBJ-10','VP','.']).
1 ('S':~['NP-SBJ','NP-PRD']).
3 ('VP':~['VBZ','NP']).
1 ('VP':~['VBN','S']).
1 ('VP':~['VBD']).
1 ('VP':~['VBD','VP']).
3 ('NP-SBJ':~['-NONE-']).
2 ('PP':~['IN','NP']).
2 ('NP':~['NP','PP']).
1 ('PP-LOC':~['IN','NP']).
1 ('NP-SBJ-10':~['NP',',','NP',',']).
1 ('NP-SBJ':~['PRP']).
1 ('NP-SBJ':~['NNP','NNP']).
1 ('NP-PRD':~['DT','NN']).
1 ('NP':~['NP','PP-LOC']).
1 ('NP':~['NP','CD','NN','NNS']).
1 ('NP':~['NP',',','SBAR']).
1 ('NP':~['NP',',','NP',',','SBAR']).
1 ('NP':~['NNP','NNP']).
1 ('NP':~['NNP','NNP','POS']).
1 ('NP':~['NNP','NNP','NNP']).
1 ('NP':~['NNP','NNP','CC','NNP']).
1 ('NP':~['NN','NN']).
1 ('NP':~['DT','JJ','NN']).
1 ('NP':~['DT','ADJP','NN']).
1 ('NP':~['CD']).
1 ('NP':~['ADVP','DT','NNP','NNP','NN','NN']).
1 ('ADVP':~['RB']).
1 ('ADJP':~['CD','NN']).
1 ('WHNP-11':~['WP']).
1 ('WHNP-10':~['WDT']).
```

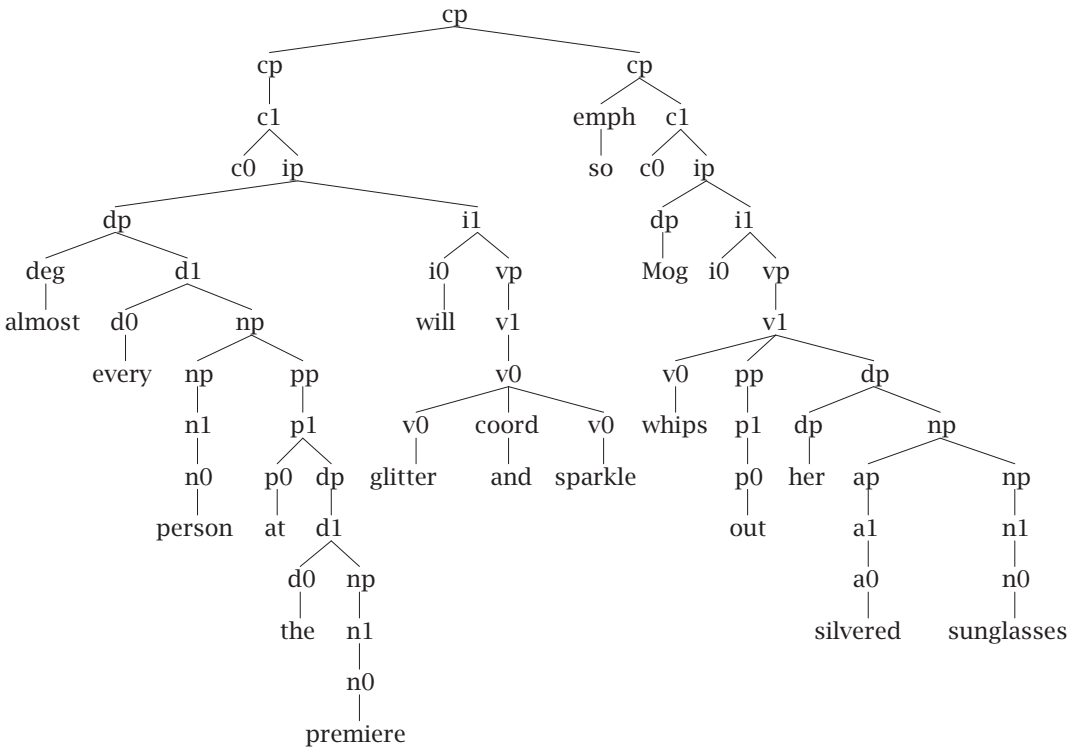
And then the lexical rules:

```
5 (',':~[',']).
4 ('NNP':~['Grace']).
3 ('NNP':~['W.R.']).
3 ('DT':~[a]).
3 ('.':~['.']).
2 ('VBZ':~[holds]).
2 ('NN':~[vice]).
2 ('NN':~[chairman]).
2 ('IN':~[of]).
1 ('WP':~[who]).
1 ('WDT':~[which]).
1 ('VBZ':~[succeeds]).
1 ('VBN':~[elected]).
1 ('VBD':~[was]).
1 ('VBD':~[resigned]).
1 ('RB':~[formerly]).
1 ('PRP':~['He']).
1 ('POS':~['\'s']).
1 ('NNS':~[seats]).
1 ('NNP':~['Terrence']).
1 ('NNP':~['J.P.']).
1 ('NNP':~['Energy']).
1 ('NNP':~['Daniels']).
1 ('NNP':~['D.']).
1 ('NNP':~['Co.']).
1 ('NNP':~['Bolduc']).
1 ('NN':~[interest]).
1 ('NN':~[director]).
1 ('NN':~[company]).
1 ('NN':~[board]).
1 ('NN':~['%']).
1 ('JJ':~['energy-services']).
1 ('IN':~[in]).
1 ('DT':~[this]).
1 ('CD':~[three]).
1 ('CD':~[seven]).
1 ('CD':~[83.4]).
1 ('CC':~[&]).
1 ('-NONE-':~['*T*-11']).
1 ('-NONE-':~['*T*-10']).
1 ('-NONE-':~['*-10']).
```

Exercises:

To begin, download our implementation of the CKY recognizer from the web page. (This implementation has several files, so download them all into the same directory, and run your prolog session in that directory.)

1. a. Modify g1.pl so that it generates exactly the following tree:



(Notice that this grammar has left recursion, right recursion, and empty productions.)

b. Use your modified grammar with the ckySWI recognizer to recognize the string as a cp:

so every silvered premiere will sparkle

Turn in (a) the modified grammar and (b) a session log showing the successful run of the ckySWI parser with that sentence.

Extra Credit: That last exercise was a little bit tedious, but we can automate it!

a. Download the file `wsj_0009.pl`, which has some parse trees for sentences from the Wall Street Journal.

b. Write a program that will go through the derivation trees in this file and write out every rule that would be used in those derivations, in our prolog grammar rule format.

c. Take the grammar generated by the previous step, and use ckySWI to check that you accept the strings in the file, and show at least one other sentence this grammar accepts.

7.2 Tree collection

7.2.1 Collecting trees: first idea

(9) The most obvious way to collect trees is this:

```
/* ckyCollect.pl
 * E Stabler, Feb 2000
 * collect a tree from a CKY parser chart
 */
:- op(1200,xfx,:¯).    % this is our object language "if"
:- use_module(library(lists),[member/2]).

ckyCollect(Chart,N,S,S/STs) :- collectTree(Chart,0,S,N,S/STs).

collectTree(Chart,I,A,J,A/Subtrees) :- member((I,A,J),Chart), (A :¯ L), collectTrees(L,Chart,I,J,Subtrees).
collectTree(Chart,I,A,J,A/[]) :- member((I,A,J),Chart), $\backslash$+((A :¯ _)).

collectTrees([],_,I,I,[]).
collectTrees([A|As],Chart,I,J,[A/ATs|Ts]) :- collectTree(Chart,I,A,K,A/ATs), collectTrees(As,Chart,K,J,Ts).
```

(10) With this file, we can have sessions like this:

```
Process prolog finished
SICStus 3.8.1 (x86-linux-glibc2.1): Sun Feb 20 14:49:19 PST 2000
Licensed to humnet.ucla.edu
| ?- [ckySics].
{consulting /home/es/tex/185-00/ckySics.pl...}
{consulted /home/es/tex/185-00/ckySics.pl in module user, 260 msec 46700 bytes}

yes
| ?- [ckyCollect].
{consulting /home/es/tex/185-00/ckyCollect.pl...}
{consulted /home/es/tex/185-00/ckyCollect.pl in module user, 20 msec 24 bytes}

yes
| ?- [g1].
{consulting /home/es/tex/185-00/g1.pl...}
{consulted /home/es/tex/185-00/g1.pl in module user, 20 msec 2816 bytes}

yes
| ?- [pp_tree].
{consulting /home/es/tex/185-00/pp_tree.pl...}
{consulted /home/es/tex/185-00/pp_tree.pl in module user, 20 msec 1344 bytes}

yes
| ?- computeClosure([the,idea,will,suffice],Chart),nl,ckyCollect(Chart,4,ip,T),pp_tree(T).
....:.:.:.:.::.::.:.:.:.:.::.:
ip /[
    dp /[
        d1 /[
            d0 /[
                the /[]],
            np /[
                n1 /[
                    n0 /[
                        idea /[]]]]]],
    i1 /[
        i0 /[
            will /[]],
        vp /[
            v1 /[
                v0 /[
                    suffice /[]]]]]]
T = ip/[dp/[d1/[d0/[the/[]],np/[n1/[...]]]],i1/[i0/[will/[]],vp/[v1/[v0/[...]]]]],
Chart = [(0,d0,1),(0,d1,2),(0,dp,2),(0,ip,4),(0,the,1),(1,idea,2),(1,n0,2),(1,n1,2),(1,...,...),(...,...)|...] ? ;

no
| ?-
```

(11) This works, but it makes tree collection almost as hard as parsing!
When we are collecting the constituents, this collection strategy will sometimes follow blind alleys.

7.2.2 Collecting trees: a better perspective

(12) We can make tree collection easier by putting additional information into the chart, so that the chart can be regarded as a "packed forest" of subtrees from which any successful derivations can easily be extracted. We call the forest "packed" because a single item in the chart can participate in many (even infinitely many) trees.

One version of this idea was proposed by Tomita (1985), and Billot and Lang (1989) noticed the basic idea mentioned in the introduction: that what we want is a certain way of computing the intersection between a regular language (represented by a finite state machine) and a context free language (represented by a context free grammar).

(13) We can implement this idea as follows: in each item, we indicate which rule was used to create it, and we also indicate the "internal" positions:

```
/* ckypSWI.pl
 * E Stabler, Feb 2000
 * CKY parser, augmented with rules for 0,3,4,5,6-tuples
 */
:- op(1200,xfx,:~).    % this is our object language "if"
:- ['closure-swi'].   % defines closure/2, uses inference/4

%verbose. % comment to reduce verbosity of chart construction
computeClosure(Input) :-
        lexAxioms(0,Input,Axioms),
        closure(Axioms, Chart),
        nl, portray_clauses(Chart).

computeClosure(Input,Chart) :-
        lexAxioms(0,Input,Axioms),
        closure(Axioms, Chart).

lexAxioms(_Pos,[],L) :-
        bagof0(((X,X):(A:~[]),(A :~ []),L).
lexAxioms(Pos,[W|Ws],[((Pos,Pos1):(W:~[]))|Axioms]) :-
        Pos1 is Pos+1,
        lexAxioms(Pos1,Ws,Axioms).

inference(reduce1,
          [ (Pos,Pos1):(W:~_) ],
            ((Pos,Pos1):(A:~[W])),
          [(A :~ [W])] ).

inference(reduce2,
          [ ((Pos,Pos1):(B:~_)), ((Pos1,Pos2):(C:~_))],
            ((Pos,Pos2):(A:~[B,Pos1,C])),
          [(A :~ [B,C])] ).

inference(reduce3,
          [ ((Pos,Pos1):(B:~_)), ((Pos1,Pos2):(C:~_)), ((Pos2,Pos3):(D:~_))],
            (Pos,(A:~[B,Pos1,C,Pos2,D]),Pos3),
          [(A :~ [B,C,D])] ).

inference(reduce4,
          [ ((Pos,Pos1):(B:~_)), ((Pos1,Pos2):(C:~_)), ((Pos2,Pos3):(D:~_)), ((Pos3,Pos4):(E:~_))],
            ((Pos,Pos4):(A:~[B,Pos1,C,Pos2,D,Pos3,E])),
          [(A :~ [B,C,D,E])] ).

inference(reduce5,
          [ ((Pos,Pos1):(B:~_)), ((Pos1,Pos2):(C:~_)), ((Pos2,Pos3):(D:~_)), ((Pos3,Pos4):(E:~_)), ((Pos4,Pos5):(F:~_))],
            ((Pos,Pos5):(A:~[B,Pos1,C,Pos2,D,Pos3,E,Pos4,F])),
          [(A :~ [B,C,D,E,F])] ).

inference(reduce6,
          [ ((Pos,Pos1):(B:~_)), ((Pos1,Pos2):(C:~_)), ((Pos2,Pos3):(D:~_)), ((Pos3,Pos4):(E:~_)), ((Pos4,Pos5):(F:~_)), ((Pos5,Pos6):(F:~_))],
            ((Pos,Pos6):(A:~[B,Pos1,C,Pos2,D,Pos3,E,Pos4,F,Pos5,G])),
          [(A :~ [B,C,D,E,F,G])] ).

portray_clauses([]).
portray_clauses([C|Cs]) :- portray_clause(C), portray_clauses(Cs).

bagof0(A,B,C) :- bagof(A,B,C), !.
bagof0(_,_,[]).
```

(14) With this parsing strategy, we can avoid all blind alleys in collecting a tree.

```
/* ckypCollect.pl
 * E Stabler, Feb 2000
 * collect a tree from a CKY parser chart
 */
:- op(1200,xfx,:~).    % this is our object language "if"

ckypCollect(Chart,N,S,S/STs) :- collectTree(Chart,0,S,N,S/STs).

collectTree(Chart,I,A,J,A/ATs) :- member(((I,J):(A:~L)),Chart), collectTrees(L,Chart,I,J,ATs).
collectTree(Chart,I,W,J,W/[]) :- member(((I,J):(W:~[])),Chart).

collectTrees([],_,I,I,[]).
collectTrees([A],Chart,I,J,[A/ATs]) :- collectTree(Chart,I,A,J,A/ATs).
collectTrees([A,K|As],Chart,I,J,[A/ATs|Ts]) :- collectTree(Chart,I,A,K,A/ATs), collectTrees(As,Chart,K,J,Ts).
```

(15) We have sessions like this:

```
7 ?- [g1,ckypSWI,pp_tree].
% g1 compiled 0.00 sec, 672 bytes
%   chart compiled 0.00 sec, 0 bytes
%   agenda compiled 0.00 sec, 0 bytes
%   items compiled 0.00 sec, 0 bytes
%   monitor compiled 0.01 sec, 0 bytes
%   driver compiled 0.00 sec, 0 bytes
%   utilities compiled 0.00 sec, 0 bytes
%  closure-swi compiled 0.01 sec, 0 bytes
% ckypSWI compiled 0.01 sec, 0 bytes
% pp_tree compiled 0.00 sec, 1,692 bytes

Yes
8 ?- computeClosure([the,idea,will,suffice]).
'.'.'.'.'.'.':'.':'.':'.':'.':'.'::'.'::'.':'.':'.':'.':'.':'.':'.'::'.':
(0, 1): (d0:~[the]).
(0, 1): (the:~[]).
(0, 2): (d1:~[d0, 1, np]).
(0, 2): (dp:~[d1]).
(0, 4): (ip:~[dp, 2, i1]).
(1, 2): (idea:~[]).
(1, 2): (n0:~[idea]).
(1, 2): (n1:~[n0]).
(1, 2): (np:~[n1]).
(2, 3): (i0:~[will]).
(2, 3): (will:~[]).
(2, 4): (i1:~[i0, 3, vp]).
(3, 4): (suffice:~[]).
(3, 4): (v0:~[suffice]).
(3, 4): (v1:~[v0]).
(3, 4): (vp:~[v1]).

Yes
9 ?- [ckypCollect].
% ckypCollect compiled 0.00 sec, 1,764 bytes

Yes
10 ?- computeClosure([the,idea,will,suffice],Chart),nl,ckypCollect(Chart,4,ip,T),pp_tree(T).
'.'.'.'.'.'.':'.':'.':'.':'.':'.'::'.'::'.':'.':'.':'.':'.':'.':'.'::'.':
ip /[
    dp /[
        d1 /[
            d0 /[
                the /[]],
            np /[
                n1 /[
                    n0 /[
                        idea /[]]]]]],
    i1 /[
        i0 /[
            will /[]],
        vp /[
            v1 /[
                v0 /[
                    suffice /[]]]]]]

Chart = [ (0, 1): (d0:~[the]), (0, 1): (the:~[]), (0, 2): (d1:~[d0, 1, np]), (0, 2): (dp:~[d1]), (0, 4): (ip:~[dp, 2|...]), (1, 2): (idea:~[]), (1, 2): (n0:~[...]), (...
T = ip/[dp/[d1/[d0/[the/[]], np/[... /...]]], i1/[i0/[will/[]], vp/[v1/[...]]]]

Yes
11 ?-
```

Exercise: What grammar is the intersection of `g1.pl` and the finite state language with just one string, `{the idea will suffice}`.

7.3 Earley recognition for CFGs

(16) Earley (1968) showed, in effect, how to build an oracle into a chart construction algorithm for any grammar $G = \langle \Sigma, N, \rightarrow, \mathsf{s} \rangle$. With this strategy, the algorithm has the "prefix property," which means that, processing a string from left to right, an ungrammatical prefix (i.e. a sequence of words that is not a prefix of any grammatical string) will be recognized at the the earliest possible point.

For $A, B, C \in N$ and some designated $s' \notin N$, for $S, T, U, V \in (N \cup \Sigma)^*$, and for input $w_1 \ldots w_n \in \Sigma^n$,

$$(0,0) : s' \rightarrow [\,] \bullet s \qquad \text{[axiom]}$$

$$\frac{(i,j) : A \rightarrow S \bullet w_{j+1} T}{(i,j+1) : A \rightarrow S w_{j+1} \bullet T} \qquad \text{[scan]}$$

$$\frac{(i,j) : A \rightarrow S \bullet B T}{(j,j) : B \rightarrow \bullet U} \qquad \text{[predict]} \qquad \text{if } B\text{:-}U \text{ and } (U = \epsilon \vee U = CV \vee (U = w_{j+1} V)$$

$$\frac{(i,k) : A \rightarrow S \bullet B T \quad (k,j) : B \rightarrow U \bullet}{(i,j) : A \rightarrow S B \bullet T} \qquad \text{[complete]}$$

The input is recognized iff $(0,n) : S' \rightarrow S\bullet$ is in the closure of the axioms (in this case, the set of axioms has just one element) under these inference rules.

Also note that in order to apply the scan rule, we need to be able to tell which word is in the $j+1$'th position.

```
    /* earley.pl
* E Stabler, Feb 2000
* Earley parser, adapted from Shieber et al.
* NB: the grammar must specify:  startCategory(XXX).
*/
:- op(1200,xfx,:-).  % this is our object language "if"
:- ['closure-sics'].  % Shieber et al's definition of closure/2, uses inference/4

%verbose.  % comment to reduce verbosity of chart construction
computeClosure(Input) :-
     retractall(word(_,_)), % get rid of words from previous parse
     lexAxioms(0,Input,Axioms),
     closure(Axioms, Chart),
     nl, portray_clauses(Chart).

computeClosure(Input,Chart) :-
     retractall(word(_,_)), % get rid of words from previous parse
     lexAxioms(0,Input,Axioms),
     closure(Axioms, Chart).

% for Earley, lexAxioms *asserts* word(i,WORDi) for each input word,
% and then returns the single input axiom:  item(start,[],[s],0,0)

lexAxioms(_Pos,[],[item(start,[],[S],0,0)]) :-
     startCategory(S).

lexAxioms(Pos,[W|Ws],Axioms) :-
     Pos1 is Pos+1,
     assert(word(Pos1,W)),
     lexAxioms(Pos1,Ws,Axioms).

inference( scan,
     [ item(A, Alpha, [W|Beta], I, J) ],
% -------------------------------------
     item(A, [W|Alpha], Beta, I, J1),
% where
     [J1 is J + 1,
     word(J1, W)] ).

inference( predict,
     [ item(_A, _Alpha, [B|_Beta], _I,J) ],
% ---------------------------------------
     item(B, [], Gamma, J,J),
% where
     [(B :-Gamma),
     eligible(Gamma,J)] ).

inference( complete,
     [ item(A, Alpha, [B|Beta], I,J),
     item(B, _Gamma, [], J,K) ],
% ---------------------------------
     item(A, [B|Alpha], Beta, I,K),
% where [] ).

eligible([],_).
eligible([A|_],_) :- \+ (\+ (A :-_)), !.  % the double negation leaves A unbound
eligible([A|_],J) :- J1 is J+1, word(J1,A).

portray_clauses([]).
portray_clauses([C|Cs]) :- portray_clause(C), portray_clauses(Cs).
```

(17) With this code we get sessions like this:

```
SICStus 3.8.1 (x86-linux-glibc2.1):  Sun Feb 20 14:49:19 PST 2000
Licensed to humnet.ucla.edu
| ?- [earley,grdin].
yes
| ?- computeClosure([the,idea,will,suffice,'.']).
.:.:.:.:.:.:.:.:.:..:.:.:.:.:..:.:.:.:.:.::.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:.:
item(c1, [], [c0,ip], 2, 2).
item(cp, [], [c1], 2, 2).
item(d0, [], [the], 0, 0).
item(d0, [the], [], 0, 1).
item(d1, [], [d0,np], 0, 0).
item(d1, [d0], [np], 0, 1).
item(d1, [np,d0], [], 0, 2).
item(dp, [], [d1], 0, 0).
item(dp, [d1], [], 0, 2).
item(i0, [], [will], 2, 2).
item(i0, [will], [], 2, 3).
item(i1, [], [i0,vp], 2, 2).
item(i1, [i0], [vp], 2, 3).
item(i1, [vp,i0], [], 2, 4).
item(ip, [], [dp,i1], 0, 0).
item(ip, [dp], [i1], 0, 2).
item(ip, [i1,dp], [], 0, 4).
item(n0, [], [idea], 1, 1).
item(n0, [idea], [], 1, 2).
item(n1, [], [n0], 1, 1).
item(n1, [], [n0,cp], 1, 1).
item(n1, [n0], [], 1, 2).
item(n1, [n0], [cp], 1, 2).
item(np, [], [n1], 1, 1).
item(np, [n1], [], 1, 2).
item(s, [], [ip,terminator], 0, 0).
item(s, [ip], [terminator], 0, 4).
item(s, [terminator,ip], [], 0, 5).
item(start, [], [s], 0, 0).
item(start, [s], [], 0, 5).
item(terminator, [], ['.'], 4, 4).
item(terminator, ['.'], [], 4, 5).
item(v0, [], [suffice], 3, 3).
item(v0, [suffice], [], 3, 4).
item(v1, [], [v0], 3, 3).
item(v1, [v0], [], 3, 4).
item(vp, [], [v1], 3, 3).
item(vp, [v1], [], 3, 4).
yes
| ?-
```

(18) Collecting trees from the Earley chart is straightforward.

```
/* earleyCollect.pl
* E Stabler, Feb 2000
* collect a tree from an Earley parser chart,
* adapted from Aho&Ullman's (1972) Algorithm 4.6
*/
:- op(1200,xfx,:-).  % this is our object language "if"
:- use_module(library(lists),[member/2]).

earleyCollect(Chart,N,StartCategory,Tree) :-
     member(item(start,[StartCategory],[],0,N),Chart),
     collectNewSubtrees([StartCategory],[],0,N,[Tree],Chart).

collectNewSubtrees(SubCats,[],I,J,Trees,Chart) :-
     length(SubCats,K),
     collectSubtrees(SubCats,I,J,K,[],Trees,Chart).

collectSubtrees([],_,_,_,Trees,Trees,_).
collectSubtrees([Xk|Xs],I,J,K,Trees0,Trees,Chart) :-
     word(_,Xk),!,
     J1 is J-1,
     K1 is K-1,
     collectSubtrees(Xs,I,J1,K1,[Xk/[]|Trees0],Trees,Chart).  collectSubtrees([Xk|Xs],I,J,K,Trees0,Trees,Chart) :-
     member(item(Xk,Gamma,[],R,J),Chart),
     memberck(item(_A,Xks,[Xk|_R],I,R),Chart),
     length([_|Xks],K),
     collectNewSubtrees(Gamma,[],R,J,Subtrees,Chart),
     K1 is K-1,
     collectSubtrees(Xs,I,R,K1,[Xk/Subtrees|Trees0],Trees,Chart).

memberck(A,L) :- member(A,L), !.  % just check to make sure such an item exists
```

(19) With this tree collector, we can find all the trees in the chart (when there are finitely many).

8 Stochastic influences on simple language models

8.1 Motivations and background

(1) Our example parsers have tiny dictionaries. If you just add in a big dictionary, we get many structural ambiguities.

Just to illustrate how bad the problem is, the following simple examples from Abney (1996a) have ambiguities that most people would not notice, but our parsing methods will:

a. I know the cows are grazing in the meadow

b. I know John saw Mary

The word `are` is a noun in `a hectare is a hundred ares`, and `saw` can be a noun, so the non-obvious readings of those two sentences are the ones analogous to the natural readings of these:

a. I know the sales force (which is) meeting in the office

b. I know Gatling gun Joe

There are many other readings too, ones which would be spelled differently (if we were careful about quotes, which most people are not) but pronounced the same:

a. I know "The Cows are Grazing in the Meadow"

b. I know "The Cows are Grazing" in the meadow

c. I know "The Cows are" grazing in the meadow

…

… I know ""The Cows are Grazing in the Meadow""

…

This kind of thing is a problem for mimicking, let alone modeling, human recognition capabilities. Abney concludes:

> The problem of how to identify the correct structure from among the in-principle possible structures provides one of the central motivations for the use of weighted grammars in computational linguistics.

(2) Martin Gardner gives us the following amusing puzzle. Insert the minimum number of quotation marks to make the best sense of the following sentence:

> Wouldn't the sentence I want to put a hyphen between the words fish and and and and and chips in my fish and chips sign have looked cleaner if quotation marks had been placed before fish and between fish and and and and and and and and and and and and and and and and and and and and and and and and chips as well as after chips?

In effect, we solve a problem like this every time we interpret a spoken sentence.

(3) Another demonstration of the ambiguity problem comes from studies like Charniak, Goldwater, and Johnson (1998). Applying the grammar of the Penn Treebank II to sentences in that Treebank shorter than 40 words from the Wall Street Journal, they found that their charts had, on average, 1.2 million items per sentence – obviously, very few of these are actually used in the desired derivation, and the rest come from local and global ambiguities.

They say:

> Numbers like this suggest that any approach that offers the possibility of reducing the work load is well worth pursuing…

To deal with this problem, Charniak, Goldwater, and Johnson (1998) explore the prospects for using a probabilistic chart parsing method that builds only the n best analyses (of each category for each span of the input) for some n.

(4) Is it reasonable to think that a probabilistic language models can handle these disambiguation problems? It is not clear that this question has any sense, since the term 'probabilistic language model' apparently covers almost everything, including, as a limiting case, the simple, discrete models that we have been studying previously.

However, it is important to recognize that the disambiguation problem is a hard one, and clearly involves background factors that cannot be regarded as linguistic.

It has been generally recognized since the early studies of language in the tradition of analytic philosophy, and since the earliest developments in modern formal semantics, that the problem of determining the intended reading of a sentence, like the problem of determining the intended reference of a name or noun phrase is, at least, well beyond the analytical tools available now. See, e.g., Partee (1975, p80), Kamp (1984, p39), Fodor (1983), Putnam (1986, p222), and many others. Putnam argues, for example, that

> …deciding – at least in hard cases – whether two terms have the same meaning or whether they have the same reference or whether they *could* have the same reference may involve deciding what is and what is not a good scientific explanation.

From this perspective, the extent to which simple statistical models account for human language use is surprising! As we will see, while we say surprising and new things quite a lot, it is easy to discern creatures of habit behind language use as well.

We briefly survey some of the most basic concepts of probability theory and information. Reading quickly over at least §8.1.1-§8.1.3 is recommended, but the main thread of development can be followed by skipping directly to §8.2.1 on page 159.

8.1.1 Corpus studies and first results

We first show how standard (freely distributed) gnu text utilities can be used to edit, sort and count things. (These utilities are standardly provided in linux and unix implementations. In ms-windows systems, you can get them with cygwin. In mac-osX systems, you can get them with fink.)

(5) Jane Austen's *Persuasion*:

```
1%dir persu11.txt
 460 -rw-r--r--   1 es       users      467137 Apr 30 18:00 persu11.txt
2%wc -l persu11.txt
    8466 persu11.txt
3%wc -w persu11.txt
   83309 persu11.txt
4%more persu11.txt
Persuasion by Jane Austen

Chapter 1


Sir Walter Elliot, of Kellynch Hall, in Somersetshire, was a man who,
for his own amusement, never took up any book but the Baronetage;
there he found occupation for an idle hour, and consolation in a
distressed one; there his faculties were roused into admiration and
respect, by contemplating the limited remnant of the earliest patents;
there any unwelcome sensations, arising from domestic affairs
changed naturally into pity and contempt as he turned over
the almost endless creations of the last century; and there,
if every other leaf were powerless, he could read his own history
with an interest which never failed.  This was the page at which
the favorite volume always opened:

           "ELLIOT OF KELLYNCH HALL.

"Walter Elliot, born March 1, 1760, married, July 15, 1784, Elizabeth,
daughter of James Stevenson, Esq. of South Park, in the county of
Gloucester, by which lady (who died 1800) he has issue Elizabeth,
q
```

In 1%, we get the number of bytes in the file.

In 2%, we get the number of lines in the file.

In 3%, we get the number of "words" – character sequences surrounded by spaces or newlines.

(6) Here we use the Gnu version of `tr`. Check your man pages if your `tr` does not work the same way.

```
4%tr ' ' '\012' < persu11.txt > persu11.wds
6%more persu11.wds
Persuasion
by
Jane
Austen

Chapter
1


Sir
Walter
Elliot,
of
Kellynch
Hall,
in
Somersetshire,
was
a
man
who,
for
```

(7)

```
7%tr -sc 'A-Za-z' '\012' < persu11.txt > persu11.wds
8%more persu11.wds
Persuasion
by
Jane
Austen
Chapter
Sir
Walter
Elliot
of
Kellynch
Hall
in
Somersetshire
was
a
man
```

```
who
for
his
own
amusement
never
```

The command 4% in (6) changes the space characters to newlines in `persu11.wds`

The command 7% in (7) changes everything in the complement of A-Za-z to newlines and then squeezes repeated occurrences of newlines down to a single occurrence.

(8)
```
9%sort -d persu11.wds > persu11.srt
10%more persu11.srt
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
A
About
Abydos
Accordingly
```

(9)
```
11%tr -sc 'A-Z' 'a-z' '\012' < persu11.txt > persu11.wds
tr: too many arguments
Try 'tr --help' for more information.
12%tr 'A-Z' 'a-z' < persu11.txt > persu11.low
13%more persu11.wds
persuasion by jane austen

chapter 1


sir walter elliot, of kellynch hall, in somersetshire, was a man who,
for his own amusement, never took up any book but the baronetage;
there he found occupation for an idle hour, and consolation in a
distressed one; there his faculties were roused into admiration and
respect, by contemplating the limited remnant of the earliest patents;
there any unwelcome sensations, arising from domestic affairs
changed naturally into pity and contempt as he turned over
the almost endless creations of the last century; and there,
if every other leaf were powerless, he could read his own history
with an interest which never failed.  this was the page at which
the favorite volume always opened:

           "elliot of kellynch hall.

"walter elliot, born march 1, 1760, married, july 15, 1784, elizabeth,
daughter of james stevenson, esq. of south park, in the county of
gloucester, by which lady (who died 1800) he has issue elizabeth,
```

(10)
```
17%tr -sc 'A-Za-z' '\012' < persu11.low > persu11.wds
18%wc -l persu11.wds
   84125 persu11.wds
19%more persu11.wds
persuasion
by
jane
austen
chapter
sir
walter
elliot
of
kellynch
hall
in
somersetshire
was
a
man
who
for
his
own
amusement
never
```

Why has the number of words increased from what we had in 3% of (5)? (Think about what happened

to punctuation, the dashes, apostrophes, etc.)

(11)
```
23%sort -d persu11.wds > persu11.srt
24%more persu11.srt
a
a
a
a
a
a
a
a
a
a
a
a
a
```

(12)
```
25%uniq -c persu11.srt > persu11.cnt
26%more persu11.cnt
   1595 a
      1 abbreviation
      1 abdication
      1 abide
      3 abilities
     30 able
      1 abode
      1 abominable
      1 abominate
      1 abominates
     97 about
      6 above
      5 abroad
      9 absence
      4 absent
      1 absenting
      3 absolute
      5 absolutely
      1 abstraction
      6 absurd
      1 absurdity
      1 abundance
```

The command `uniq -c` inserts a count of consecutive identical lines before 1 copy of that line.

(13) At this point we have a count of word "tokens." That is, we know that "a" occurs 1595 times in this text. Notice that by "tokens" here, we do not mean particular inscriptions, particular patterns of ink on paper or of states in computer memory as is usual. Rather, we mean the **occurrences** in the novel, where the novel is an abstract thing that has many different realizations in the physical world.

When we execute `wc -l persu11.cnt` we get `5742` – the number of word **types**.

(14)
```
27%sort -d persu11.cnt > persu11.sct
28%more persu11.sct
      1 abbreviation
      1 abdication
      1 abide
      1 abode
      1 abominable
      1 abominate
      1 abominates
      1 absenting
      1 abstraction
      1 absurdity
      1 abundance
      1 abuse
      1 abydos
      1 accent
      1 acceptance
      1 accession
      1 accessions
      1 accidental
      1 accommodating
      1 accompanied
      1 accompany
      1 accompanying
```

(15)
```
31%sort -nr persu11.cnt > persu11.sct
32%more persu11.sct
   3329 the
   2808 to
   2801 and
   2570 of
   1595 a
   1389 in
   1337 was
   1204 her
   1186 had
   1146 she
   1124 i
   1038 it
    962 he
    950 be
    934 not
    882 that
```

```
    809 as
    707 for
    664 but
    659 his
    654 with
    628 you
    589 have
    533 at
    530 all
    497 anne
    496 been
    485 s
    467 him
    451 could
    434 very
    433 they
    426 were
    418 by
    416 which
    398 is
    396 on
    359 so
    356 no
....
      1 tranquility
      1 trained
      1 trafalgar
      1 tradespeople
      1 toys
      1 toward
      1 tossed
      1 torn
      1 topic
      1 tolerated
      1 tolerate
```

The command `sort -nr` sorts the file in numeric reverse order, looking for a number at the beginning of each line.

Notice that almost all of these most frequent words are one syllable.

(16) Zipf (1949) observed that longer words tend to be infrequent, and that frequencies are distributed in a regular, non-normal way. This is discussed below.

(17)
```
33%more persu11.wds
persuasion
by
jane
austen
chapter
sir
walter
elliot
of
kellynch
hall
in
somersetshire
was
a
man
who
for
his
own
amusement
never
```

(18)
```
34%tail +2 persu11.wds > persu11.w2
35%more persu11.w2
by
jane
austen
chapter
sir
walter
elliot
of
kellynch
hall
in
somersetshire
was
a
man
who
for
his
own
amusement
never
took
```

(19)
```
36%paste persu11.wds persu11.w2 > bigrams
37%more bigrams
persuasion      by
by      jane
```

```
jane    austen
austen  chapter
chapter sir
sir     walter
walter  elliot
elliot  of
of      kellynch
kellynch        hall
hall    in
in      somersetshire
somersetshire   was
was     a
a       man
man     who
who     for
for     his
his     own
own     amusement
amusement       never
never   took
```

(20)
```
38%sort -d bigrams > bigrams.srt
39%more bigrams.srt
a       bad
a       bad
a       bad
a       bad
a       bad
a       ball
a       baronet
a       baronet
a       baronet
a       baronet
a       baronet
a       baronetcy
a       baronetcy
a       baronight
a       barouche
a       beautiful
a       bed
```

(21)
```
40%uniq -c bigrams.srt > bigrams.cnt
41%more bigrams.cnt
      5 a       bad
      1 a       ball
      5 a       baronet
      2 a       baronetcy
      1 a       baronight
      1 a       barouche
      1 a       beautiful
      1 a       bed
      1 a       beloved
      1 a       bend
```

Here, `wc -l bigrams.cnt` gives us `42,728` – the number of **bigram types** – sequences of two word types that occur in the text.

Notice that `42,728 < 5742`^{2}`=32,970,564`, – the number of bigrams is very much less than the number of possible word pairs, which is the square of the number of word-types.

(22)
```
42%sort -nr bigrams.cnt > bigrams.sct
43%more bigrams.sct
    429 of      the
    378 to      be
    323 in      the
    255 had     been
    227 she     had
    220 it      was
    196 captain wentworth
    191 he      had
    176 to      the
    174 mr      elliot
    164 she     was
    148 could   not
    147 lady    russell
    134 he      was
    131 sir     walter
    129 of      her
    127 all     the
    125 i       have
    123 i       am
    117 have    been
    114 she     could
    112 of      his
    112 and     the
    111 for     the
    109 they    were
     96 to      her
     96 that    he
     96 did     not
     95 on      the
     94 to      have
     91 a       very
     90 of      a
     89 would   be
     88 it      is
     84 that    she
```

```
    83 in      a
    82 was     a
    81 not     be
    81 at      the
    80 the     same
```

We could use these as an estimate of word-transition possibilities in a "Markov chain" or "pure Markov source" – these notions are introduced below.[33]

It is very important to note the following point:

(23) Notice how high in the ranking `captain wentworth` is!

Obviously, this reflects not on the structure of the grammar, but on extra-linguistic factors.

This raises the important question: **what do these numbers mean?** They confound linguistic and extralinguistic factors.

Apparently, extralinguistic factors could easily rerank these bigrams substantially without changing the language in any significant way! We will return to this important point.

(24) We can take a peek at the least common bigrams as well. Some of them have unusual words like *baronight*, but others are perfectly ordinary.

```
44%tail -10 bigrams.sct
      1 a       bias
      1 a       bewitching
      1 a       benevolent
      1 a       bend
      1 a       beloved
      1 a       bed
      1 a       beautiful
      1 a       barouche
      1 a       baronight
      1 a       ball
```

(25)
```
45%grep baronight persu11.wds
baronight

46%grep baronight persu11.txt
his master was a very rich gentleman, and would be a baronight some day."

47%
```

(26) Facts like these have led to the view that studying the range of possible linguistic structures is quite a different thing from studying what is common. At the conclusion of a statistical study, Mandelbrot (1961, p213) says, for example,

> …because statistical and grammatical structures seem uncorrelated, in the first approximation, one might expect to encounter laws which are independent of the grammar of the language under consideration. Hence from the viewpoint of significance (and also of the mathematical method) there would be an enormous difference between, *on the one hand*, the collection of data that are unlikely to exhibit any regularity other than the approximate stability of relative frequencies, and *on the other hand,* the study of laws that are valid for natural discourse but not for other organized systems of signs.

8.1.2 Vocabulary growth

(27) Vocabulary growth varies with texts: some authors introduce new words at a much greater rate than other words (and this is a common test used in author-identification).

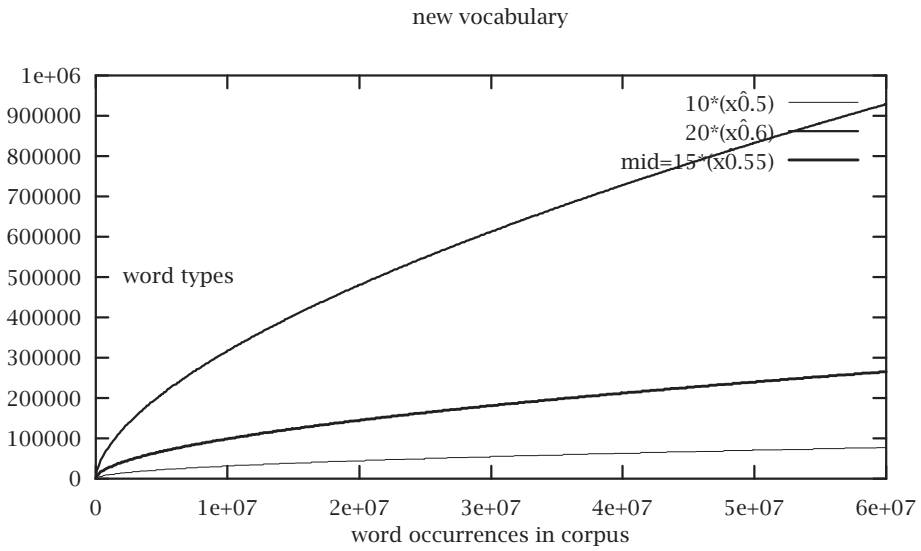
And of course, as we read a corpus of diverse texts, vocabulary growth is "bursty" as you would expect.

In previous studies, it has been found that the number of word types V grows with the number of words in the corpus roughly according to

$$V = kN^{\beta}$$

[33] Using them to set the parameters of a Markov model, where the states do not correspond 1 for 1 to the words, is a much more delicate matter which can be handled in any number of ways. We will mention this perspective, where the states are not visible, again below, since it is the most prominent one in the "Hidden Markov Model" literature.

where usually $10 \leq k \leq 20$ and $0.5 \leq \beta \leq 0.6$.



There is some work on predicting vocabulary size: Good (1953), Salton (1988), Zobel et al. (1995), Samuelsson (1996).

Exercise: a. Get `CharlesDarwin-VoyageOfTheBeagle.txt` from the class machine.

b. What percentage of word-types in this text occur only once?

c. Build trigrams for the text.

d. What percentage of trigrams occur only once?

e. *Extra credit 1:* Generate 100 words of text strictly according to trigam probabilities.
Submit the 100 word text and also the program that generates it.

f. *Extra credit 2:* Plot the rate of vocabulary growth in this text. Does it have roughly the shape of the function $V = kN^{\beta}$?
Extra extra: For what k, β does the function $V = kN^{\beta}$ best fit this curve?

g. Delete all the files you created!

8.1.3 Zipf's law

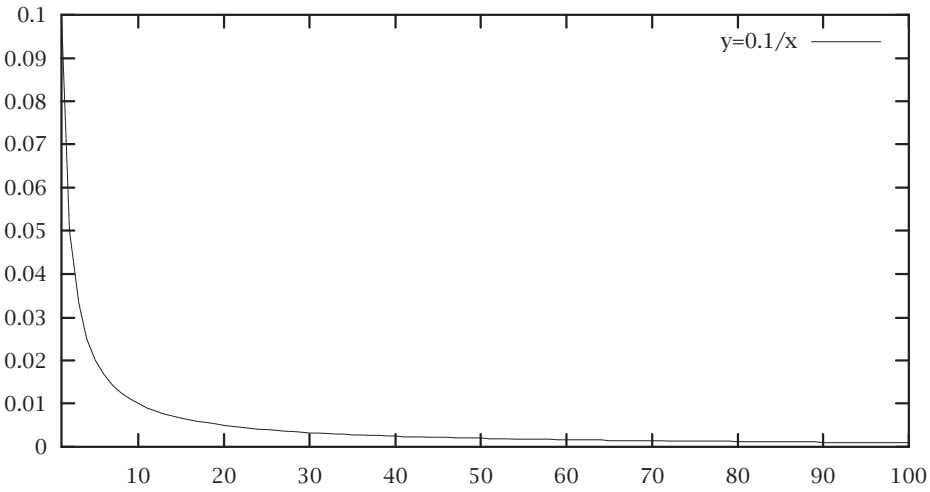
In early studies of texts, Zipf (1949) noticed that the distribution of word frequencies is not normal, and that there is a relation between word frequency and word length. In particular, in most natural texts, when words are ranked by frequency, from most frequent to least frequent, the product of rank r and frequency μ is constant. That is, in natural texts with vocabulary Σ,

$$\exists k \forall x \in \Sigma, r(x)\mu(x) = k,$$

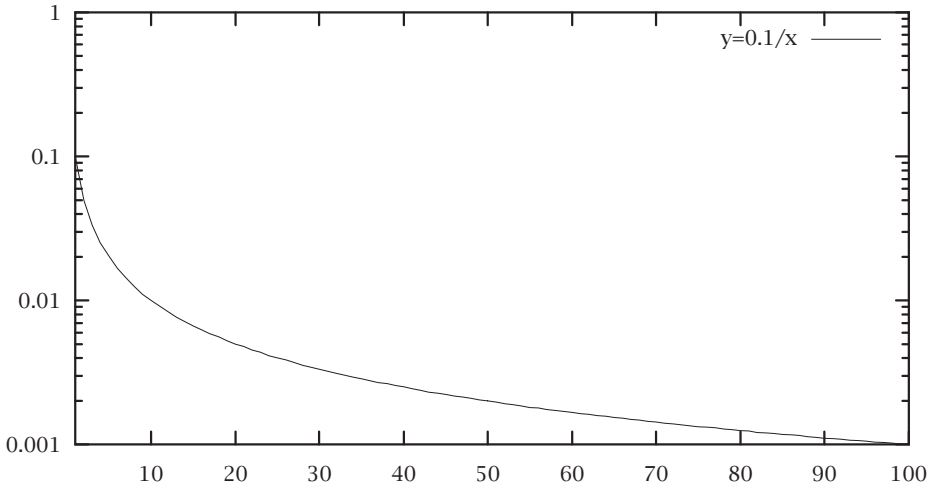
In other words, in natural texts the function f from ranks r to frequency is a function

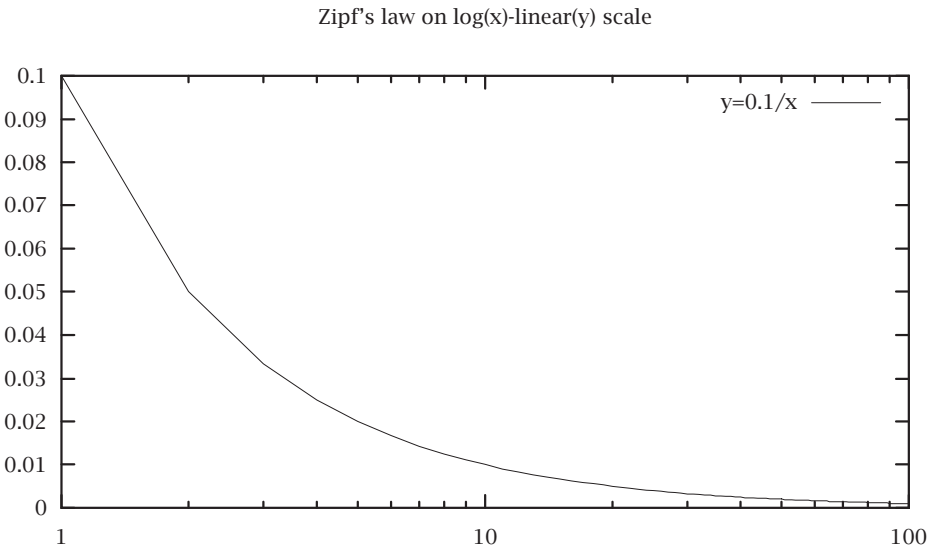
$$f(r) = \frac{k}{r}.$$

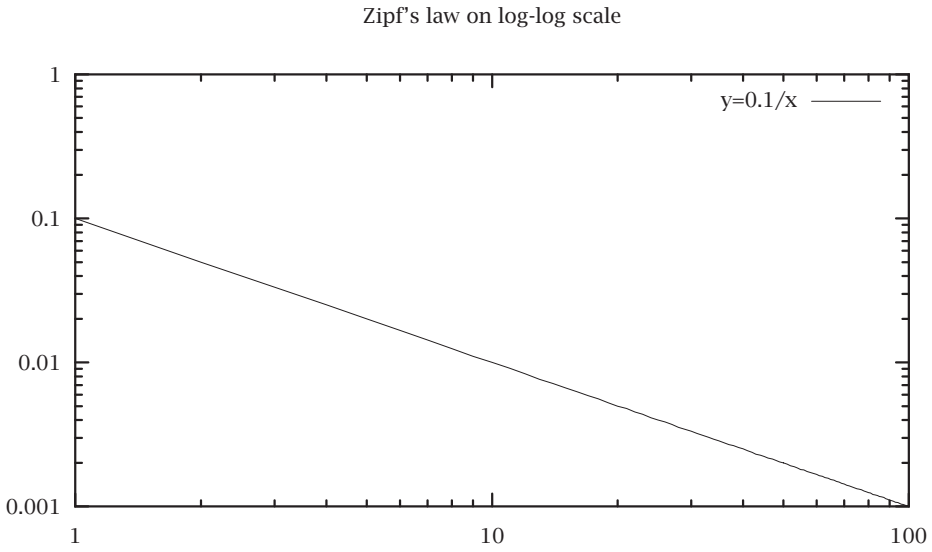
Zipf's law on linear scale



Zipf's law on linear(x)-log(y) scale

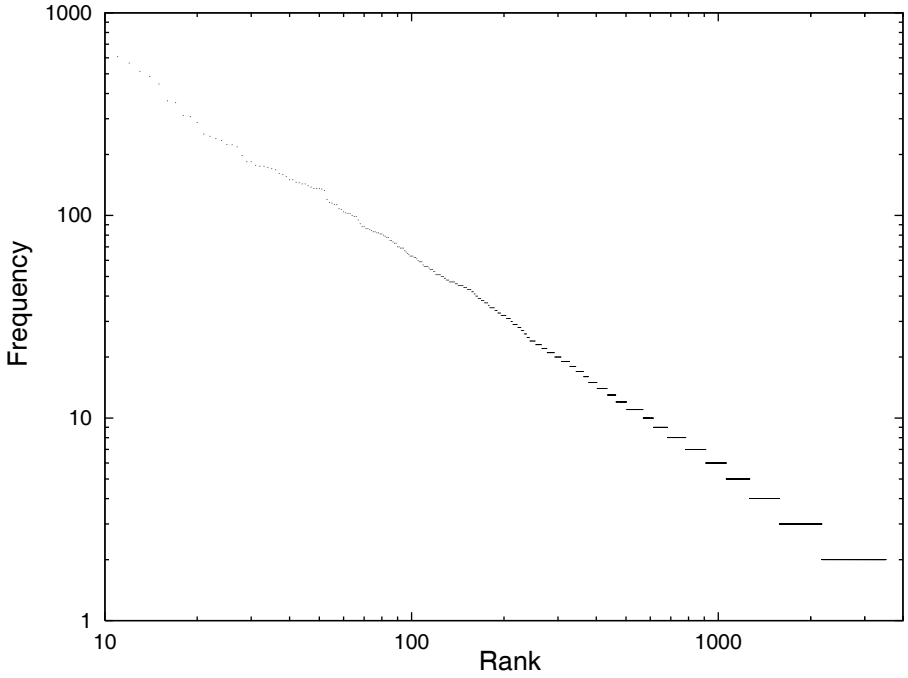






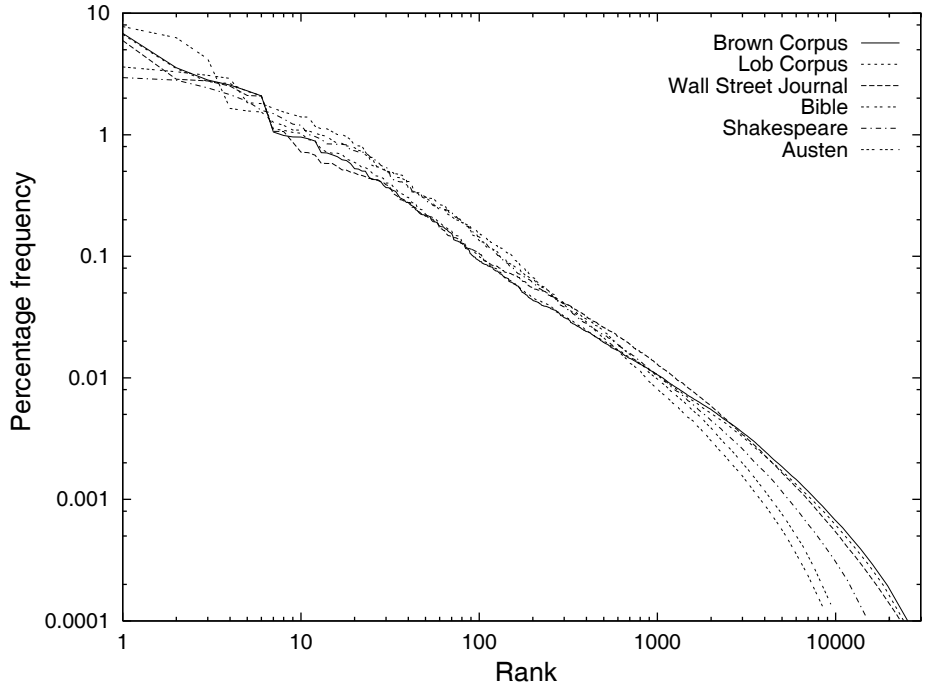
Zipf proposed that this regularity comes from a "principle of least effort:" frequently used vocabulary tends to be shortened. This idea seems intuitively right, but the evidence for it here is very very weak! Miller and Chomsky (1963, pp456-463) discuss Mandelbrot's (1961) point that this happens even in a random distribution, as long as the word termination character is among the randomly distributed elements. Consequently, there is no reason to assume a process of abbreviation unless the distribution of words of various sizes departs significantly from what might be expected anyway. No one has been able to establish that this is so. Cf. Li (1992), Perline (1996), Teahan (1998), Goldman (1998). Teahan (1998) presents a number of useful results that we survey here.

James Joyce's *Ulysses* fits Zipf's hypothesis fairly well, with some falling off at the highest and lowest ranks.

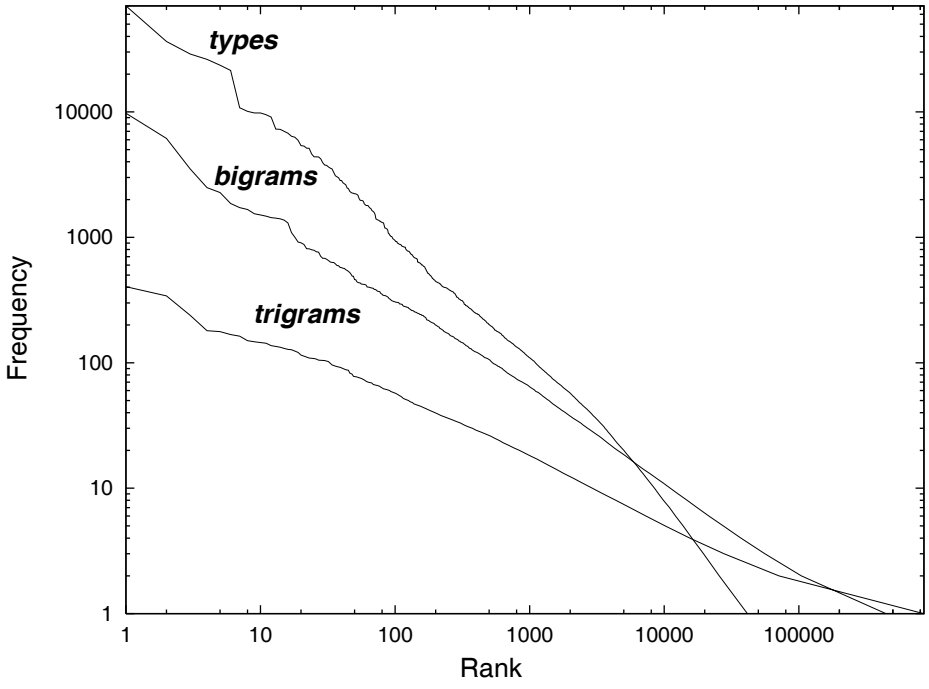


Most word occurrences are occurrences of those few types that occur very frequently.

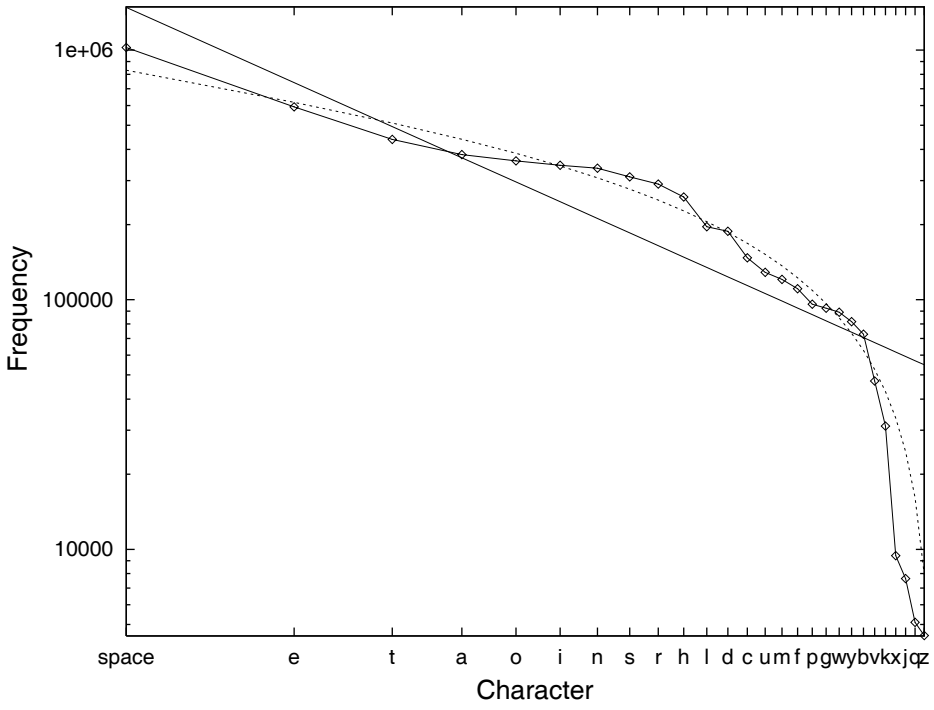
We get almost the same curve for other texts and collections of texts:

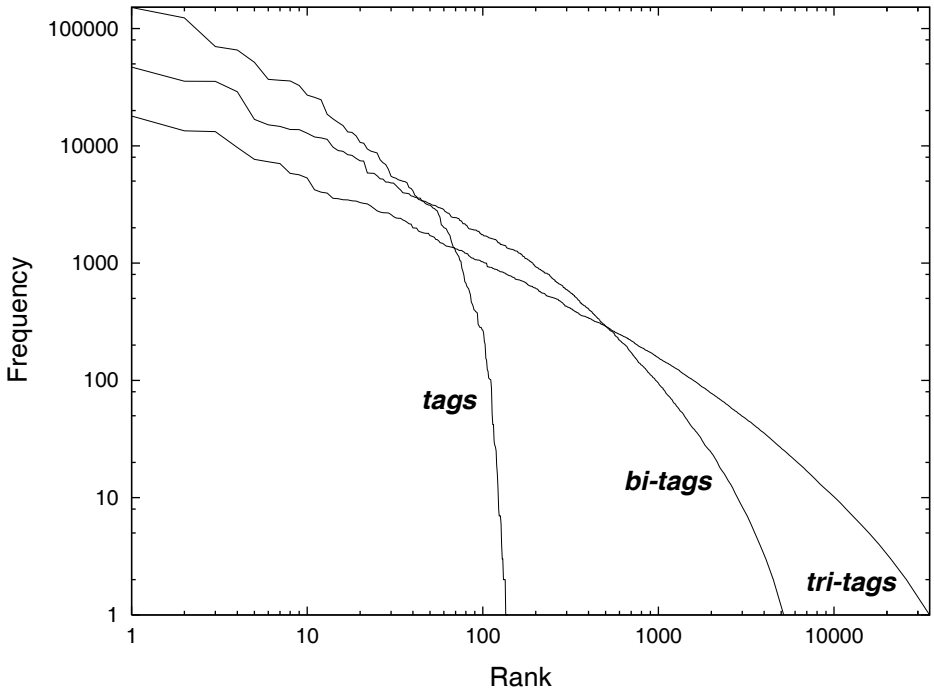


NB: Zipf's relationship holds when texts are combined into a corpus – what explains this? Also, the same kind of curve with steeper fall for n-grams (Brown corpus):

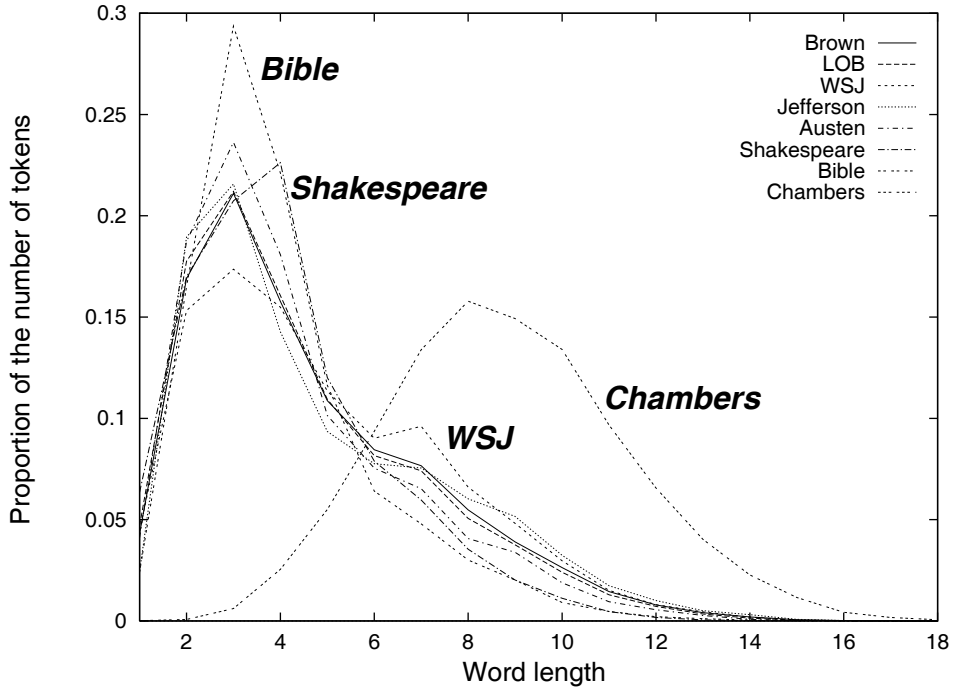


Similar curve for character frequencies, and tags (i.e. part of speech labels) too:

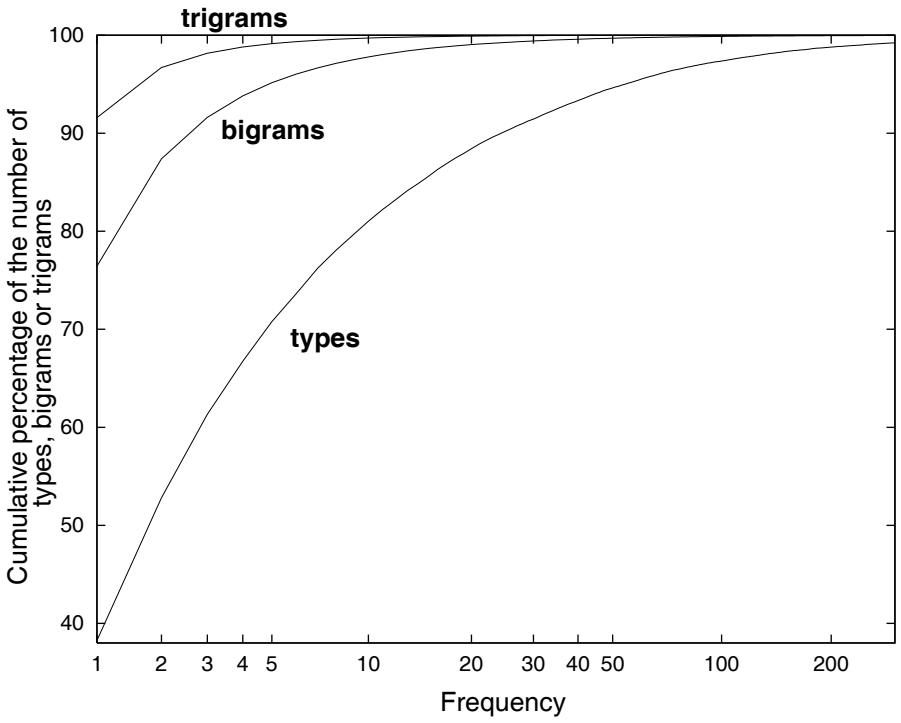




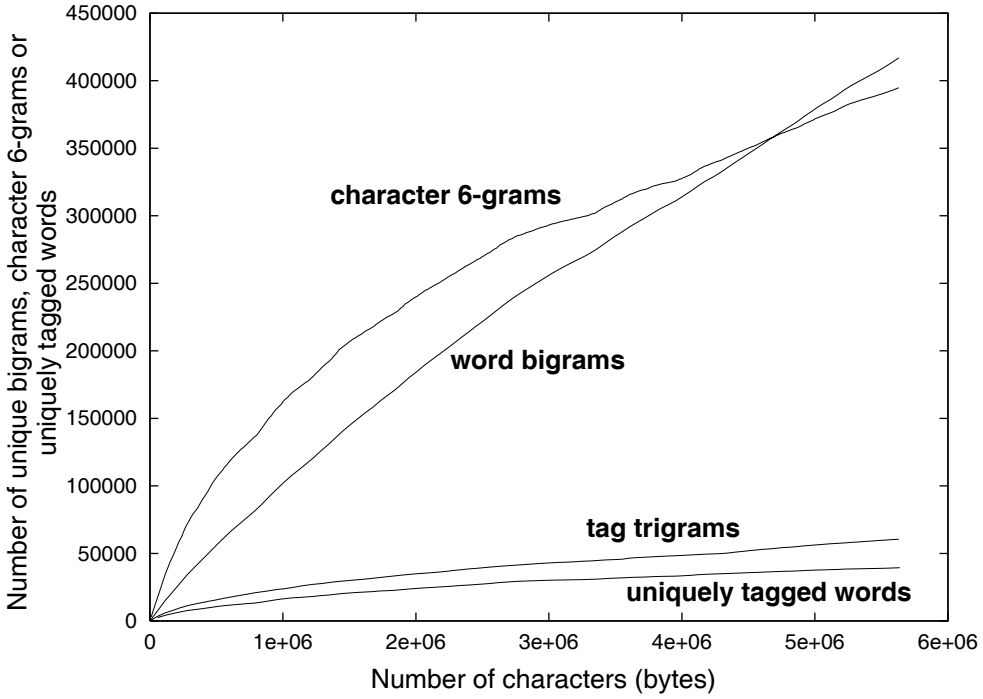
Word length in naturally occurring text has a similar relation to frequency – with dictionaries as unsurprisingly exceptional:

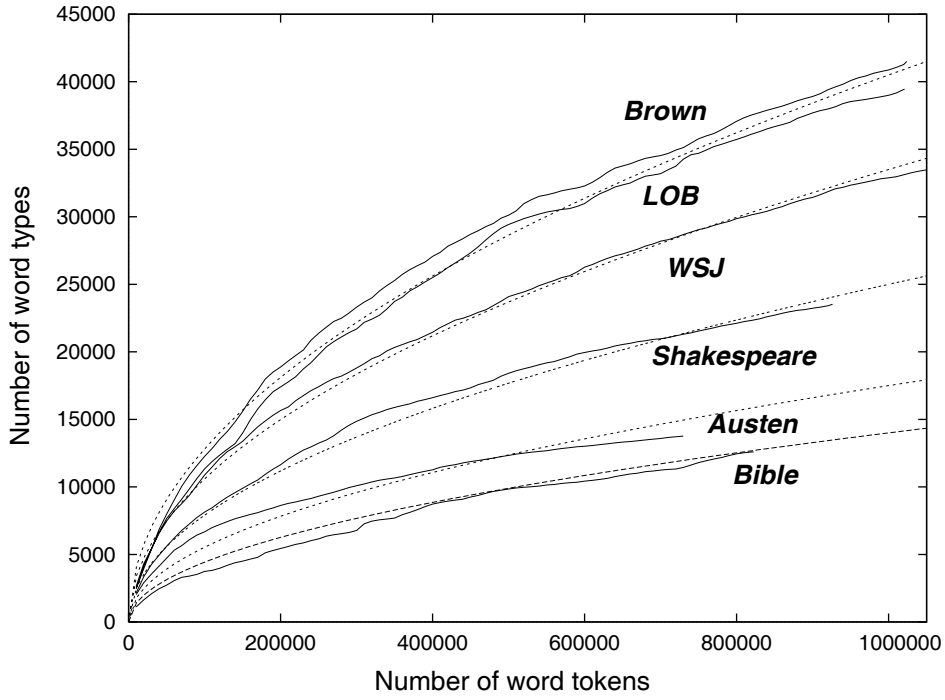


Most words types are rare; bigrams, trigrams more so



And of course, the language grows continually:





8.1.4 Probability measures and Bayes' Theorem

(28) A **sample space** Ω is a set of **outcomes**.
An **event** $A \subseteq \Omega$.
Letting 2^Ω be the set of subsets of Ω, the power set of Ω, then an event $A \in 2^\Omega$.

(29) Given events A, B, the usual notions $A \cap B, A \cup B$ apply.
For the complement of A let's write $\overline{A} = \Omega - A$.
Sometimes set subtraction $A - B = A \cap \overline{B}$ is written $A \backslash B$, but since we are not using the dash for complements, we can use it for subtraction without excessive ambiguity.

(30) $(2^\Omega, \subseteq)$ is a **Boolean (set) algebra**, that is, it is a collection of subsets of Ω such that

a. $\Omega \in 2^\Omega$

b. $A_0, A_1 \in 2^\Omega$ implies $A_0 \cup A_1 \in 2^\Omega$

c. $A \in 2^\Omega$ implies $\overline{A} \in 2^\Omega$

The English mathematician George Boole (1815-1864) is best known for his work in propositional logic and algebra. In 1854 he published *An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities*.

(31) When $A \cap B = \emptyset$, A and B are **(mutually) disjoint.**
$A_0, A_1, \ldots, A_n$ is a sequence of **(mutually) disjoint** events if
for all $0 \leq i, j \leq n$ where $i \neq j$, the pair A_i and A_j is disjoint.

(32) When Ω is countable (finite or countably infinite), it is **discrete**.
Otherwise Ω is **continuous**.

(33) $[0, 1]$ is the closed interval $\{x \in \mathsf{R} \mid 0 \leq x \leq 1\}$
$(0, 1)$ is the open interval $\{x \in \mathsf{R} \mid 0 < x < 1\}$

(34) **Kolmogorov's 3 axioms** define a **probability measure** as a function $P : 2^\Omega \to [0, 1]$ such that

i. $0 \leq P(A) \leq 1$ for all $A \subseteq \Omega$

ii. $P(\Omega) = 1$

iii. **finite additivity:** $P(A \cup B) = P(A) + P(B)$ for any disjoint events $A, B \in 2^\Omega$,
In some settings we will assume **countable additivity**, for any sequence of disjoint events $A_0, A_1, \ldots \in 2^\Omega$,

$$P(\bigcup_{i=0}^{\infty} A_i) = \sum_{i=0}^{\infty} P(A_i)$$

(Notice that axiom i follows from the indicated range of P.)

The Russian mathematician Andrey Nikolayevich Kolmogorov (1903-1987) developed the axiomatic approach to probability theory based on set theory.

(35) When P satisfies i-iii, $(\Omega, 2^\Omega, P)$ is a (finitely, or countably additive) **probability space**

(36) **Theorem:** $P(\overline{A}) = 1 - P(A)$

Proof: Obviously $\overline{A}$ and A are disjoint, so by axiom iii, $P(A \cup \overline{A}) = P(\overline{A}) + P(A)$
Since $\overline{A} \cup A = \Omega$, axiom ii tells us that $P(\overline{A}) + P(A) = 1$ □

(37) **Theorem:** $P(\varnothing) = 0$

(38) **Theorem:** $P(A \cup B) = P(A) + P(B) - P(A \cap B)$

Proof: Since A is the union of disjoint events $(A \cap B)$ and $(\overline{B} \cap A)$, $P(A) = P(A \cap B) + (\overline{B} \cap A)$.
Since B is the union of disjoint events $(A \cap B)$ and $(\overline{A} \cap B)$, $P(B) = P(A \cap B) + (\overline{A} \cap B)$.
And finally, since $(A \cup B)$ is the union of disjoint events $(\overline{B} \cap A), (A \cap B)$ and $(\overline{A} \cap B)$, $P(A \cup B) = P(\overline{B} \cap A) + P(A \cap B) + P(\overline{A} \cap B)$.
Now we can calculate $P(A) + P(B) = P(A \cap B) + (\overline{B} \cap A) + P(A \cap B) + (\overline{A} \cap B)$, and so $P(A \cup B) = P(A) + P(B) - P(A \cap B)$. □

(39) **Theorem (Boole's inequality):** $P(\bigcup_0^\infty A_i) \leq \sum_0^\infty P(A_i)$

(40) **Exercises**

a. Prove that if $A \subseteq B$ then $P(A) \leq P(B)$

b. In (38), we see what $P(A \cup B)$ is. What is $P(A \cup B \cup C)$?

c. Prove Boole's inequality.

(41) The **conditional probability of A given B,** $P(A|B) =_{df} \frac{P(A \cap B)}{P(B)}$

(42) **Bayes' theorem:** $P(A|B) = \frac{P(A)P(B|A)}{P(B)}$

Proof: From the definition of conditional probability just stated in (41), (i) $P(A \cap B) = P(B)P(A|B)$. The definition of conditional probability (41) also tells us $P(B|A) = \frac{P(A \cap B)}{P(A)}$, and so (ii) $P(A \cap B) = P(A)P(B|A)$. Given (i) and (ii), we know $P(A)P(B|A) = P(B)P(A|B)$, from which the theorem follows immediately. □

The English mathematician Thomas Bayes (1702-1761) was a Presbyterian minister. He distributed some papers, and published one anonymously, but his influential work on probability, containing a version of the theorem above, was not published until after his death.

Bayes is also associated with the idea that probabilities may be regarded as degrees of belief, and this has inspired recent work in models of scientific reasoning. See, e.g. Horwich (1982), Earman (1992), Pearl (1988).

In fact, in a Microsoft advertisement we are told that their Lumiere Project uses "a Bayesian perspective on integrating information from user background, user actions, and program state, along with a Bayesian analysis of the words in a user's query…this Bayesian information-retrieval component of Lumiere was shipped in all of the Office '95 products as the Office Answer Wizard…As a user works, a probability distribution is generated over areas that the user may need assistance with. A probability that the user would not mind being bothered with assistance is also computed."

See, e.g. `http://www.research.microsoft.com/research/dtg/horvitz/lum.htm`.

For entertainment, and more evidence of the Bayesian cult that is sweeping certain subcultures, see, e.g. `http://www.afit.af.m`

For some more serious remarks on Microsoft's "Bayesian" spelling correction, and a new proposal inspired by trigram and Bayesian methods, see e.g. Golding and Schabes (1996).

For some serious proposals about Bayesian methods in perception: Knill and Richards (1996); and in language acquisition: Brent and Cartwright (1996), de Marcken (1996).

(43) A and B are **independent** iff $P(A \cap B) = P(A)P(B)$.

8.1.5 Random variables

(44) A **random (or stochastic) variable** on probability space $(\Omega, 2^\Omega, P)$ is a function $X : \Omega \to \mathsf{R}$.

(45) Any set of numbers $A \in 2^\mathsf{R}$ determines (or "generates") an event, a set of outcomes, namely $X^{-1}(A) = \{e|\ X(e) \in A\}$.

(46) So then, for example, $P(X^{-1}(A)) = P(\{e|\ X(e) \in A\})$ is the probability of an event, as usual.

(47) Many texts use the notation $X \in A$ for an event, namely, $\{e|\ X(e) \in A\})$.
So $P(X \in A)$ is just $P(X^{-1}(A))$, which is just $P(\{e|\ X(e) \in A\})$.
Sometimes you also see $P\{X \in A\}$, with the same meaning.

(48) Similarly, for some $a \in \mathsf{R}$, it is common to see $P(X = s)$, where $X = s$ is the event $\{e|\ X(e) = s\})$.

(49) The range of X is sometimes called the **sample space of the stochastic variable** X, Ω_X.
X is **discrete** if Ω_X is finite or countably infinite. Otherwise it is **continuous**.

- Why do things this way? What is the purpose of these functions X?
  The answer is: the functions X just formalize the classification of events, the sets of outcomes that we are interested in, as explained in (45) and (48).
  This is a standard way to name events, and once you are practiced with the notation, it is convenient.
  The events are classified numerically here, that is, they are named by real numbers, but when the set of events Ω_X is finite or countable, obviously we could name these events with any finite or countable set of names.

8.1.6 Stochastic processes and n-gram models of language users

(50) A **stochastic process** is a function X from times (or "indices") to random variables.
If the time is continuous, then $X : \mathsf{R} \to [\Omega \to \mathsf{R}]$, where $[\Omega \to \mathsf{R}]$ is the set of random variables.
If the time is discrete, then $X : \mathsf{N} \to [\Omega \to \mathsf{R}]$

(51) For stochastic processes X, instead of the usual argument notation $X(t)$, we use subscripts X_t, to avoid confusion with the arguments of the random variables.
So X_t is the value of the stochastic process X at time t, a random variable.
When time is discrete, for $t = 0, 1, 2, \ldots$ we have the sequence of random variables $X_0, X_1, X_2, \ldots$

(52) We will consider primarily discrete time stochastic processes, that is, sequences $X_0, X_1, X_2, \ldots$ of random variables.
So X_i is a random variable, namely the one that is the value of the stochastic process X at time i.

(53) $X_i = q$ is interpreted as before as the event (now understood as occurring at time i) which is the set of outcomes $\{e|\ X_i(e) = q\}$.
So, for example, $P(X_i = q)$ is just a notation for the probability, at time i, of an outcome that is named q by X_i, that is, $P(X_i = q)$ is short for $P(\{e|\ X_i(e) = q\})$.

(54) Notice that it would make perfect sense for all the variables in the sequence to be identical, $X_0 = X_1 = X_2 = \ldots$. In that case, we still think of the process as one that occurs in time, with the same classification of outcomes available at each time.
Let's call a stochastic process **time-invariant** (or stationary) iff all of its random variables are the same function. That is, for all $q, q' \in \mathsf{N}, X_q = X_{q'}$.

(55) A **finite stochastic process** X is one where sample space of all the stochastic variables, $\Omega_X = \bigcup_{i=0}^{\infty} \Omega_{X_i}$ is finite.
The elements of Ω_X name <u>events</u>, as explained in (45) and (48), but in this context the elements of Ω_X are often called <u>states</u>.

Markov chains

(56) A stochastic process $X_0, X_1, \ldots$ is **first order** iff for each $0 \leq i$, $\sum_{x \in X_i} P(x) = 1$ and the events in Ω_{X_i} are all independent of one another.
(Some authors number from 0, calling this one 0-order).

(57) A stochastic process $X_0, X_1, \ldots$ has the **Markov property** (that is, it is **second order**) iff the probability of the next event may depend only on the current event, not on any other part of the history.
That is, for all $t \in \mathbb{R}$ and all $q_0, q_1, \ldots \in \Omega_X$,

$$P(X_{t+1} = q_{t+1} | X_0 = q_0, \ldots, X_t = q_t) = P(X_{t+1} = q_{t+1} | X_t = q_t)$$

The Russian mathematician Andrei Andreyevich Markov (1856-1922) was a student of Pafnuty Chebyshev. He used what we now call Markov chains in his study of consonant-vowel sequences in poetry.

(58) A **Markov chain** or Markov process is a stochastic process with the Markov property.

(59) A **finite Markov chain**, as expected, is a Markov chain where the sample space of the stochastic variables, Ω_X is finite.

(60) It is sometimes said that an n-state Markov chain can be specified with an $n \times n$ matrix that specifies, for each pair of events $s_i, s_j \in \Omega_X$ the probability of next event s_j given current event s_i.
Is this true? Do we know that for some X_j where $i \neq j$ that $P(X_{i+1} = q' | X_i = q) = P(X_{j+1} = q' | X_j = q)$? No.[34] For example, we can perfectly well allow that $P(\{e|\ X_{i+1}(e) = q'\}) \neq P(\{e|\ X_{j+1}(e) = q'\})$, simply by letting $\{e|\ X_{i+1}(e) = q'\}) \neq \{e|\ X_{j+1}(e) = q'\}$. This can happen quite naturally when the functions X_{i+1}, X_{j+1} are different, a quite natural assumption when these functions have in their domains outcomes that happen at different times.
The condition that disallows this is: time-invariance, defined in (54) above.

(61) Given a time-invariant, finite Markov process X, the probabilities of events in Ω_X can be specified by

i. an "initial probability vector" which defines a probability distribution over $\Omega_X = \{q_0, q_1, \ldots, q_{n-1}\}$. This can be given as a vector, a $1 \times n$ matrix, $[P_0(q_0) \quad \ldots \quad P_0(q_{n-1})]$, where $\sum_{i=0}^{n-1} P(q_i) = 1$

ii. a $|\Omega_X| \times |\Omega_X|$ matrix of conditional probabilities, here called **transition** or **digram probabilities**, specifying for each $s_i, s_j \in \Omega_X$ the probability of next event/state q_i given current event/state q_j. We introduce the notation $P(q_i|q_j)$ for $P(X_{t+1} = q_i | X_t = q_j)$.

(62) Given an initial probability distribution I on Ω_X and the transition matrix M, the probability of state sequence $q_0 q_1 q_2 \ldots q_n$ is determined. Writing $P_0(q_i)$ for the initial probability of q_i,

$$P(q_0 q_1 q_2 \ldots q_n) = P_0(q_0) P(q_1|q_0) P(q_2|q_1) \ldots P(q_n|q_{n-1})$$

Writing $P(q_i|q_j)$ for the probability of the transition from state q_j to state q_i,

$$\begin{aligned} P(q_1 \ldots q_n) &= P_0(q_1) P(q_2|q_1) \ldots P(q_n|q_{n-1}) \\ &= P_0(q_1) \textstyle\prod_{1 \leq i \leq n-1} P(q_{i+1}|q_i) \end{aligned}$$

(63) Given an initial probability distribution I on Ω_X and the transition matrix M, the probability distribution for the events of a finite time-invariant Markov process at time t is given by the matrix product IM^t. That is, at time 0 $P = I$, at time 1 $P = IM$, at time 2 $P = IMM$, and so on.

(64) To review your matrix arithmetic, there are many good books. The topic is commonly covered in calculus books like Apostol (1969), but many books focus on this topic: Bretscher (1997), Jacob (1995), Golub and Van Loan (1996), Horn and Johnson (1985) and many others. Here is a very brief summary of some basics.

[34] The better written texts are careful about this, as in Papoulis (1991, p637) and Drake (1967, p165), for example.

Matrix arithmetic review: An $m \times n$ matrix A is an array with m rows and n columns.
Let $A(i, j)$ be the element in row i and column j.

1. We can add the $m \times n$ matrices A, B to get the $m \times n$ matrix $A + B = C$ in which
$$C(i, j) = A(i, j) + B(i, j).$$
2. Matrix addition is associative and commutative:
$$A + (B + C) = (A + B) + C$$
$$A + B = B + A$$
3. For any $n \times m$ matrix M there is an $n \times m$ matrix M' such that $M + M' = M' + M = M$, namely the $n \times m$ matrix M' such that every $M(i, j) = 0$.
4. We can multiply an $m \times n$ matrix A and a $n \times p$ matrix to get an $m \times p$ matrix C in which
$$C(i, j) = \sum_{k=1}^{n} A(i, k)B(k, j).$$
This definition is sometimes called the "row by column" rule. To find the value of $C(i, j)$, you add the products of all the elements in row i of A and column j of B. For example,
$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 8 & 9 & 0 \\ 2 & 6 & 0 \end{bmatrix} = \begin{bmatrix} (1 \cdot 8) + (4 \cdot 2) & (1 \cdot 9) + (4 \cdot 6) & (1 \cdot 0) + (4 \cdot 0) \\ (2 \cdot 8) + (5 \cdot 2) & (2 \cdot 9) + (5 \cdot 6) & (2 \cdot 2) + (5 \cdot 0) \end{bmatrix}$$
Here we see that to find $C(1, 1)$ we sum the products of all the elements row 1 of A times the elements in column 1 of B. – The number of elements in the rows of A must match the number of elements in the columns of B or else AB is not defined.
5. Matrix multiplication is associative, but not commutative:
$$A(BC) = (AB)C$$
$$\text{For example, } \begin{bmatrix} 3 & 5 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \end{bmatrix} \neq \begin{bmatrix} 2 \\ 4 \end{bmatrix} \begin{bmatrix} 3 & 5 \end{bmatrix}$$
It is interesting to notice that Lambek's (1958) composition operator is also associative but not commutative:
$(X \bullet Y) \bullet Z) \Rightarrow X \bullet (Y \bullet Z)$
$X/Y \bullet Y \Rightarrow X$
$Y \bullet X/Y \not\Rightarrow X$
The connection between the Lambek calculus and matrix algebra is actually a deep one (Parker, 1995).
6. For any $m \times m$ matrix M there is an $m \times m$ matrix I_m such that $TI_m = I_m T = T$, namely the $m \times m$ matrix I_m such that every $I_m(i, i) = 1$ and for every $i \neq j, I_m(i, j) = 0$.
7. *Exercise:*
   a. Explain why the claims in 2 are obviously true.
   b. Do the calculation to prove that my counterexample to commutativity in 5 is true.
   c. Explain why 6 is true.
   d. Make sure you can use `octave` or some other system to do the calculations once you know how to do them by hand:

```
1% octave
Octave, version 2.0.12 (i686-pc-linux-gnulibc1).
Copyright (C) 1996, 1997, 1998 John W. Eaton.
```

```
octave:1> x=[1,2]
x =

  1  2

octave:2> y=[3;4]
y =

  3
  4

octave:3> z=[5,6;7,8]
z =

  5  6
  7  8

octave:4> x+x
ans =

  2  4

octave:5> x+y
error: operator +: nonconformant arguments (op1 is 1x2, op2 is 2x1)
error: evaluating assignment expression near line 5, column 2
octave:5> 2*x
ans =

  2  4

octave:6> x*y
ans = 11
octave:7> x*z
ans =

  19  22

octave:8> y*x
ans =

  3  6
  4  8

octave:9> z*x
error: operator *: nonconformant arguments (op1 is 2x2, op2 is 1x2)
error: evaluating assignment expression near line 9, column 2
```

(65) To apply the idea in (63), we will always be multiplying a $1 \times n$ matrix times a square $n \times n$ matrix, to get the new $1 \times n$ probability distribution for the events of the n state Markov process.

(66) For example, suppose we have a coffee machine that (upon inserting money and pressing a button) will do one of 3 things:
(q_1) produce a cup of coffee,
(q_2) return the money with no coffee,
(q_3) keep the money and do nothing.
Furthermore, after an occurrence of (q_2), following occurrences of (q_2) or (q_3) are much more likely than they were before. We could capture something like this situation with the following initial distribution for q_1, q_2 and q_3 respectively,

$$I = [0.7 \quad 0.2 \quad 0.1]$$

and if the transition matrix is:

$$\mathrm{T}=\begin{bmatrix} 0.7 & 0.2 & 0.1 \\ 0.1 & 0.7 & 0.2 \\ 0 & 0 & 1 \end{bmatrix}$$

a. What is the probability of state sequence $q_1 q_2 q_1$?

$$P(q_1 q_2 q_1) = P(q_1)P(q_2|q_1)p(q_1|q_2) = 0.7 \cdot 0.2 \cdot 0.1 = 0.014$$

b. What is the probability of the states Ω_X at a particular time t?
At time 0 (maybe, right after servicing) the probabilities of the events in Ω_X are given by I.
At time 1, the probabilities of the events in Ω_X are given by

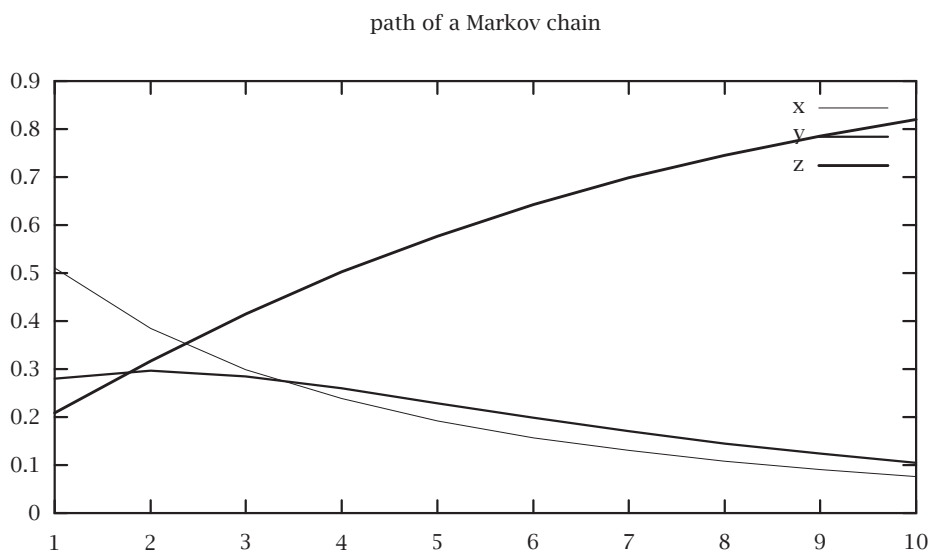
$$\begin{aligned} \mathrm{IT} &= \begin{bmatrix} 0.7 \cdot 0.7 + 0.2 \cdot 0.1 + 0.1 \cdot 0 & 0.7 \cdot 0.2 + 0.2 \cdot 0.7 + 0.1 \cdot 0 & 0.7 \cdot 0.1 + 0.2 \cdot 0.2 + 0.1 \cdot 1 \end{bmatrix} \\ &= \begin{bmatrix} 0.49 + 0.02 & 0.14 + 0.14 & 0.07 + 0.04 + .1 \end{bmatrix} \\ &= \begin{bmatrix} 0.51 & 0.28 & .21 \end{bmatrix} \end{aligned}$$

At time 2, the probabilities of the events in Ω_X are given by IT^2.
At time t, the probabilities of the events in Ω_X are given by IT^t.

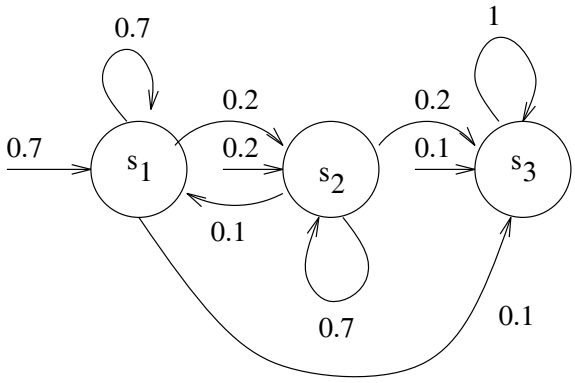
```
octave:10> i=[0.7,0.2,0.1]
i =

  0.70000  0.20000  0.10000

octave:11> t=[0.7,0.2,0.1;0.1,0.7,0.2;0,0,1]
t =

  0.70000  0.20000  0.10000
  0.10000  0.70000  0.20000
  0.00000  0.00000  1.00000

octave:12> i*t
ans =

  0.51000  0.28000  0.21000

octave:13> i*t*t
ans =

  0.38500  0.29800  0.31700

octave:14> i*t*t*t
ans =

  0.29930  0.28560  0.41510

octave:15> i*t*t*t*t
ans =

  0.23807  0.25978  0.50215

octave:16> i*(t**1)
ans =

  0.51000  0.28000  0.21000

octave:17> i*(t**2)
ans =

  0.38500  0.29800  0.31700
```

```
octave:18> result=zeros(10,4)
result =

  0  0  0  0
  0  0  0  0
  0  0  0  0
  0  0  0  0
  0  0  0  0
  0  0  0  0
  0  0  0  0
  0  0  0  0
  0  0  0  0
  0  0  0  0

octave:19>  for x=1:10
>   result(x,:)= [x,(i*(t**x))]
> endfor

octave:20>  result
result =

   1.000000   0.510000   0.280000   0.210000
   2.000000   0.385000   0.298000   0.317000
   3.000000   0.299300   0.285600   0.415100
   4.000000   0.238070   0.259780   0.502150
   5.000000   0.192627   0.229460   0.577913
   6.000000   0.157785   0.199147   0.643068
   7.000000   0.130364   0.170960   0.698676
   8.000000   0.108351   0.145745   0.745904
   9.000000   0.090420   0.123692   0.785888
  10.000000   0.075663   0.104668   0.819669
```

```
octave:21> gplot [1:10] result title "x",\
> result using 1:3 title "y", result using 1:4 title "z"
```



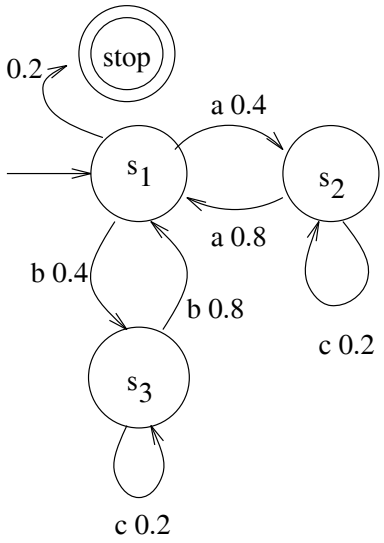
(67) Notice that the initial distribution and transition matrix can be represented by a finite state machine with no vocabulary and no final states:



(68) Notice that no Markov chain can be such that after a sequence of states $acccc$ there is a probability of 0.8 that the next symbol will be an a, that is,
$P(a|acccc) = 0.8$
when it is also the case that
$P(b|bcccc) = 0.8$
This follows from the requirement mentioned in (61) that in each row i, the sum of the transition probabilities from that state $\sum_{q_j \in \Omega_X} P(q_j|q_i) = 1$, and so we cannot have both $P(b|c) = 0.8$ and $P(a|c) = 0.8$.

(69) Chomsky (1963, p337) observes that the Markovian property that we see in state sequences does not always hold in regular languages. For example, the following finite state machine, to which we have

added probabilities, is such that the probability of generating (or accepting) an a next, after generating $acccc$ is $P(a|acccc) = 0.8$, and the probability of generating (or accepting) a b next, after generating $bcccc$ is $P(b|bcccc) = 0.8$. That is, the strings show a kind of history dependence.



However the corresponding state sequences of this same machine are Markovian in some sense: they are not history dependent in the way the strings seem to be. That is, we can have both $P(q_1|q_1q_2q_2q_2q_2) = P(q_1|q_2) = 0.8$ and $P(q_1|q_1q_3q_3q_3q_3) = P(q_1|q_3) = 0.8$ since these involve transitions from different states.

We will make this idea clearer in the next section.

8.1.7 Markov models

(70) A Markov chain can be specified by an initial distribution and state-state transition probabilities can be augmented with stochastic outputs, so that we have in addition an initial output distribution and state-output emission probabilities.

One way to do this is to is to define a **Markov model** as a pair X, O where X is a Markov chain $X : \mathsf{N} \rightarrow [\Omega \rightarrow \mathsf{R}]$ and $O : \mathsf{N} \rightarrow [\Omega \rightarrow \Sigma]$ where the latter function provides a way to classify outcomes by the symbols $a \in \Sigma$ that they are associated with.

In a Markov chain, each number $n \in \mathsf{R}$ names an event under each X_i, namely $\{e|\ X_i(e) = n\}$.

In a Markov model, each output symbol $a \in \Sigma$ names an event in each O_i, namely $\{e|\ 0_i(e) = a\}$.

(71) In problems concerning Markov models where the state sequence is not given, the model is often said to be "hidden," a **hidden Markov model** (HMM).

See, e.g., Rabiner (1989) for an introductory survey on HMMs. Some interesting recent ideas and applications appear in, e.g., Jelinek and Mercer (1980), Jelinek (1985), De Mori, Galler, and Brugnara (1995), Deng and Rathinavelu (1995), Ristad and Thomas (1997b), Ristad and Thomas (1997a).

(72) Let's say that a Markov model (X, O) is a **Markov source** iff the functions X and O are "aligned" in the following sense:[35]

$$\forall e \in \Omega,\ \forall i \in \mathsf{N},\ \forall n \in \mathsf{R},\ \exists a \in \Sigma, \quad X_i(e) = n \text{ implies } O_i(e) = a$$

Then for every X_i, for all $n \in \Omega_{X_i}$, there is a particular output $a \in \Sigma$ such that $P(O_i = a|X_i = n) = 1$.

[35] This is nonstandard. I think "Markov source" is usually just another name for a Markov model.

(73) Intuitively, in a Markov source, the symbol emitted at time i depends only on the state n of the process at that time. Let's formalize this idea as follows.

Observe that, given our definition of Markov source, when O_i extended pointwise to subsets of Ω, the set of outputs associated with outcomes named n has a single element, $O_i(\{e|\ X_i(e) = n\}) = \{a\}$.

So define $Out_i : \Omega_{X_i} \rightarrow \Sigma$ such that for any $n \in \Omega_{X_i}$, $Out_i(n) = a$ where $O_i(\{e|\ X_i(e) = n\}) = \{a\}$.

(74) Let a **pure Markov source** be a Markov model in which Out_i is the identity function on Ω_{X_i}.[36]

Then the outputs of the model are exactly the event sequences.

(75) Clearly, no Markov source can have outputs like those mentioned in (69) above, with $P(a|acccc) = 0.8$ and $P(b|bcccc) = 0.8$.

(76) Following Harris (1955), a Markov source in which the functions Out_i are not 1-1 is a **grouped (or projected) Markov source**.

(77) The output sequences of a grouped Markov source may lack the Markov property. For example, it can easily happen that $P(a|acccc) = 0.8$ and $P(b|bcccc) = 0.8$.

This happens, for example, the 2-state Markov model given by the following initial state matrix I, transition matrix T and output matrix O:

$$I = [0.5 \quad 0.5]$$

$$\mathrm{T}=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\mathrm{O}=\begin{bmatrix} 0.8 & 0 & 0.2 \\ 0 & 0.8 & 0.2 \end{bmatrix}$$

The entry in row i column j of the output matrix represents the probability of emitting the j'th element of $\langle a, b, c\rangle$ given that the system is in state i. Then we can see that we have described the desired situation, since the system can only emit a a if it is in state 1, and the transition table says that once the system is in state 1, it will stay in state 1. Furthermore, the output table shows that in state 1, the probability of emitting another a is 0.8. On the other hand, the system can only emit a b if it is in state 2, and the transition table says that once the system is in state 2, it will stay in state 2, with a probability of 0.8 of emitting another b.

(78) Miller and Chomsky (1963, p427) say that any finite state automaton over which an appropriate probability measure is defined "can serve as" a Markov source, by letting the transitions of the finite state automaton correspond to states of a Markov source.

(Chomsky (1963, p337) observes a related result by Schützenberger (1961) which says that every regular language is the homomorphic image of a 1-limited finite state automaton.)

We return to formalize this claim properly in (106) on page 151, below.

N-gram models

(79) Following Shannon (1948), a Markov model is said to be **n+1'th order** iff the next state is depends only on the previous n symbols emitted.

A pure Markov source is always 2nd order.

A grouped Markov source can have infinite order, as we saw in (77), following Harris (1955).

8.1.8 Computing output sequence probabilities: naive

(80) As noted in (62), given any Markov model and any sequence of states $q_1 \ldots q_n$,

[36] With this definition, pure Markov sources are a special case of the general situation in which the functions Out_i are 1-1.

$$P(q_1 \dots q_n) = P_0(q_1) \prod_{1 \le i \le n-1} P(q_{i+1}|q_i) \quad \text{(i)}$$

Given $q_1 \dots q_n$, the probability of output sequence $a_1 \dots a_n$ is

$$\prod_{t=1}^{n} P(a_t|q_t). \quad \text{(ii)}$$

The probability of $q_1 \dots q_n$ occurring with outputs $a_1 \dots a_n$ is the product of the two probabilities (i) and (ii), that is,

$$P(q_1 \dots q_n, a_1 \dots a_n) = P_0(q_1) \prod_{1 \le i \le n-1} P(q_{i+1}|q_i) \prod_{t=1}^{n} P(a_t|q_t). \quad \text{(iii)}$$

(81) Given any Markov model, the probability of output sequence $a_1 \dots a_n$ is the sum of the probabilities of this output for all the possible sequences of n states.

$$\sum_{q_i \in \Omega_X} P(q_1 \dots q_n, a_1 \dots a_n) \quad \text{(iv)}$$

(82) Directly calculating this is infeasible, since there are $|\Omega_X|^n$ state sequences of length n.

8.1.9 Computing output sequence probabilities: forward

Here is a feasible way to compute the probability of an output sequence $a_1 \dots a_n$.

(83) a. Calculate, for each possible initial state $q_i \in \Omega_X$,

$$P(q_i, a_1) = P_0(q_i)P(a_1|q_i).$$

b. **Recursive step:** Given $P(q_i, a_1 \dots a_t)$ for all $q_i \in \Omega_X$, calculate $P(q_j, a_1 \dots a_{t+1})$ for all $q_j \in \Omega_X$ as follows

$$P(q_j, a_1 \dots a_{t+1}) = \Big(\sum_{i \in \Omega_X} P(q_i, a_1 \dots a_t)P(q_j|q_i) \Big) P(a_{t+1}|q_j)$$

c. Finally, given $P(q_i, a_1 \dots a_n)$ for all $q_i \in \Omega_X$,

$$P(a_1 \dots a_n) = \sum_{q_i \in \Omega_X} P(q_i, a_1 \dots a_n)$$

(84) Let's develop the coffee machine example from (66), adding outputs so that we have a Markov model instead of just a Markov chain. Suppose that there are 3 output messages:
(s_1) `thank you`
(s_2) `no change`
(s_3) `x@b*/!`
Assume that these outputs occur with the probabilities given in the following matrix where row i column j represents the probability of emitting symbol s_j when in state i:

$$\mathrm{O} = \begin{bmatrix} 0.8 & 0.1 & 0.1 \\ 0.1 & 0.8 & 0.1 \\ 0.2 & 0.2 & 0.6 \end{bmatrix}$$

Exercise: what is the probability of the output sequence

$$s_1 s_3 s_3$$

Solution sketch: (do it yourself first! note the trellis-like construction)

a. probability of the first symbol s_1 from one of the initial states

$$\begin{aligned} p(q_i|s_1) = p(q_i)p(s_1|q_i) &= \begin{bmatrix} 0.7 \cdot 0.8 & 0.2 \cdot 0.1 & 0.1 \cdot 0.2 \end{bmatrix} \\ &= \begin{bmatrix} 0.56 & 0.02 & 0.02 \end{bmatrix} \end{aligned}$$

b. probabilities of the following symbols from each state (transposed to column matrix)

$$\begin{aligned} p(q_i|s_1s_3)' &= \begin{bmatrix} ((p(q_1,s_1)\cdot p(q_1|q_1)) + (p(q_2,s_1)\cdot p(q_1|q_2)) + (p(q_3,s_1)\cdot p(q_1|q_3)))\cdot p(s_3|q_1) \\ ((p(q_1,s_1)\cdot p(q_2|q_1)) + (p(q_2,s_1)\cdot p(q_2|q_2)) + (p(q_3,s_1)\cdot p(q_2|q_3)))\cdot p(s_3|q_2) \\ ((p(q_1,s_1)\cdot p(q_3|q_1)) + (p(q_2,s_1)\cdot p(q_3|q_2)) + (p(q_3,s_1)\cdot p(q_3|q_3)))\cdot p(s_3|q_3) \end{bmatrix} \\ &= \begin{bmatrix} ((0.56\cdot 0.7) + (0.02\cdot 0.2) + (0.02\cdot 0))\cdot 0.1 \\ ((0.56\cdot 0.2) + (0.02\cdot 0.7) + (0.02\cdot 0))\cdot 0.1 \\ ((0.56\cdot 0.1) + (0.02\cdot 0.1) + (0.02\cdot 1))\cdot 0.6 \end{bmatrix} \\ &= \begin{bmatrix} (0.392 + 0.04)\cdot 0.1 \\ (0.112 + 0.014)\cdot 0.1 \\ (0.056 + 0.002 + 0.02)\cdot 0.6 \end{bmatrix} \\ &= \begin{bmatrix} 0.0432 \\ 0.0126 \\ 0.0456 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} p(q_i|s_1s_3s_3)' &= \begin{bmatrix} ((p(q_1,s_1s_3)\cdot p(q_1|q_1)) + (p(q_2,s_1s_3)\cdot p(q_1|q_2)) + (p(q_3,s_1s_3)\cdot p(q_1|q_3)))\cdot p(s_3|q_1) \\ ((p(q_1,s_1s_3)\cdot p(q_2|q_1)) + (p(q_2,s_1s_3)\cdot p(q_2|q_2)) + (p(q_3,s_1s_3)\cdot p(q_2|q_3)))\cdot p(s_3|q_2) \\ ((p(q_1,s_1s_2)\cdot p(q_3|q_1)) + (p(q_2,s_1s_3)\cdot p(q_3|q_2)) + (p(q_3,s_1s_3)\cdot p(q_3|q_3)))\cdot p(s_3|q_3) \end{bmatrix} \\ &= \begin{bmatrix} ((0.0432\cdot 0.7) + (0.0126\cdot 0.2) + (0.0456\cdot 0))\cdot 0.1 \\ ((0.0432\cdot 0.2) + (0.0126\cdot 0.7) + (0.0456\cdot 0))\cdot 0.1 \\ ((0.0432\cdot 0.1) + (0.0126\cdot 0.1) + (0.0456\cdot 1))\cdot 0.6 \end{bmatrix} \\ &= \begin{bmatrix} (0.03024 + 0.00252)\cdot 0.1 \\ (0.00864 + 0.00882)\cdot 0.1 \\ (0.00432 + 0.00126 + 0.0456)\cdot 0.6 \end{bmatrix} \\ &= \begin{bmatrix} 0.003276 \\ 0.001746 \\ 0.030708 \end{bmatrix} \end{aligned}$$

c. Finally, we calculate $p(s_1s_3s_3)$ as the sum of the elements of the last matrix:

$$p(s_1s_3s_3) = 0.03285$$

8.1.10 Computing output sequence probabilities: backward

Another feasible way to compute the probability of an output sequence $a_1 \dots a_n$.

(85) a. Let $P(q_i \Rightarrow a_1 \dots a_n)$ be the probability of emitting $a_1 \dots a_n$ beginning from state q_i. And for each possible final state $q_i \in \Omega_X$, let

$$P(q_i \Rightarrow \epsilon) = 1$$

(With this base case, the first use of the recursive step calculates $P(q_j \Rightarrow a_n)$ for each $q_i \in \Omega_X$.)

b. **Recursive step:** Given $P(q_i \Rightarrow a_t \dots a_n)$ for all $q_i \in \Omega_X$, calculate $P(q_j \Rightarrow a_{t-1} \dots a_n)$ for all $q_j \in \Omega_X$ as follows:

$$P(q_j \Rightarrow a_{t-1} \dots a_n) = (\sum_{j \in \Omega_X} P(q_i \Rightarrow a_t \dots a_n)P(q_j|q_i))P(a_{t-1}|q_j)$$

c. Finally, given $P(q_i \Rightarrow a_1 \dots a_n)$ for all $q_i \in \Omega_X$,

$$P(a_1 \dots a_n) = \sum_{q_i \in \Omega_X} P_0(q_i) P(q_i \Rightarrow a_1 \dots a_n)$$

(86) **Exercise:** Use the coffee machine as elaborated in (84) and the backward method to compute the probability of the output sequence

$$s_1 s_3 s_3.$$

8.1.11 Computing most probable parses: Viterbi's algorithm

(87) Given a string $a_1 \dots a_n$ output by a Markov model, what is the most likely sequence of states that could have yielded this string? This is analogous to finding a most probable parse of a string.

Notice that we could solve this problem by calculating the probabilities of the output sequence for each of the $|\Omega_X|^n$ state sequences, but this is not feasible!

(88) The **Viterbi algorithm** allows efficient calculation of the most probable sequence of states producing a given output (Viterbi 1967; Forney 1973), using an idea that is similar to the forward calculation of output sequence probabilities in §8.1.9 above.

Intuitively, once we know the best way to get to any state in Ω_X at a time t, the best path to the next state is an extension of one of those.

(89) a. Calculate, for each possible initial state $q_i \in \Omega_X$,

$$P(q_i, a_1) = P_0(q_i) P(a_1 | q_i).$$

and record: $q_i : P(q_i, a_1)@\epsilon$.

That is, for each state q_i, we record the probability of the state sequence ending in q_i.

b. **Recursive step:** Given $q_i : P(\vec{q} q_i, a_1 \dots a_t)@\vec{q}$ for each $q_i \in \Omega_X$,
for each $q_j \in \Omega_X$ find a q_i that maximizes

$$P(\vec{q} q_i q_j, a_1 \dots a_t a_{t+1}) = P(\vec{q} q_i, a_1 \dots a_t) P(q_j | q_i) P(a_{t+1} | q_j)$$

and record: $q_j : P(\vec{q} q_i q_j, a_1 \dots a_t a_{t+1})@\vec{q} q_i$.[37]

c. After these values have been computed up to the final state t_n, we choose a $q_i : P(\vec{q} q_i, a_1 \dots a_n)@\vec{q}$ with a maximum probability $P(\vec{q} q_i, a_1 \dots a_n)$.

(90) **Exercise:** Use the coffee machine as elaborated in (84) to compute the most likely state sequence underlying the output sequence

$$s_1 s_3 s_3.$$

(91) The Viterbi algorithm is not incremental: at every time step $|\Omega_X|$ different parses are being considered.

As stated, the algorithm stores arbitrarily long state paths at each step, but notice that each step only needs the results of the previous step: $|\Omega_X|$ different probabilities (an unbounded memory requirement, unless precision can be bounded)

[37] In case more than one q_i ties for the maximum $P(\vec{q} q_i q_j, a_1 \dots a_t a_{t+1})$, we can either make a choice, or else carry all the winning options forward.

8.1.12 Markov models in human syntactic analysis?

(92) Shannon (1948, pp42-43) says:

> *We can also approximate to a natural language by means of a series of simple artificial language…To give a visual idea of how this series approaches a language, typical sequences in the approximations to English have been constructed and are given below…*
>
> 5. *First order word approximation…Here words are chosen independently but with their appropriate frequencies.*
>
>   *REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE*
>
> 6. *Second order word approximation. The word transition probabilities are correct but no further structure is included.*
>
>   *THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED*
>
> *The resemblance to ordinary English text increases quite noticeably at each of the above steps…It appears then that a sufficiently complex stochastic source process will give a satisfactory representation of a discrete source.*

(93) Damerau (1971) confirms this trend in an experiment that involved generating 5th order approximations.

All these results are hard to interpret though, since (i) sparse data in generation will tend to yield near copies of portions of the source texts (on the sparse data problem, remember the results from Jelinek mentioned in 95, above), and (ii) human linguistic capabilities are not well reflected in typical texts.

(94) **Miller and Chomsky objection 1:** The number of parameters to set is enormous.

Notice that for a vocabulary of $100,000$ words, where each different word is emitted by a different event, we would need at least 100,000 states. The full transition matrix then has $100,000^2 = 10^{10}$ entries. Notice that the last column of the transition matrix is redundant, and so a 10^9 matrix will do.

Miller and Chomsky (1963, p430) say:

> *We cannot seriously propose that a child learns the value of* 10^9 *parameters in a childhood lasting only* 10^8 *seconds.*

Why not? This is very far from obvious, unless the parameters are independent, and there is no reason to assume they are.

(95) **Miller and Chomsky (1963, p430) objection 2:** The amount of input required to set the parameters of a reasonable model is enormous.

Jelinek (1985) reports that after collecting the trigrams from a 1,500,000 word corpus, he found that, in the next 300,000 words, 25% of the trigrams were new.

No surprise! Some generalization across lexical combinations is required. In this context, the "generalization" is sometimes achieved with various "smoothing" functions, which will be discussed later. With generalization, setting large numbers of parameters becomes quite conceivable.

Without a better understanding of the issues, I find objection 2 completely unpersuasive.

(96) **Miller and Chomsky (1963, p425) objection 3:**

> *Since human messages have dependencies extending over long strings of symbols, we know that any pure Markov source must be too simple…*

This is persuasive! Almost everyone agrees with this.

The "n-gram" elaborations of the Markov models are not the right ones, since dependencies in human languages do not respect any principled bound (in terms of the number of words n that separate the dependent items).

(97) Abney (1996a) says:

> *Shannon himself was careful to call attention to precisely this point: that for any n, there will be some dependencies affecting the well-formedness of a sentence that an n-th order model does not capture.*

Is that true? Reading Shannon (1948), far from finding him careful on this point, I can find no mention at all of this now commonplace fact, that no matter how large n gets, we will miss some of the dependencies in natural languages.

8.1.13 Controversies

(98) Consider the following quote from Charniak (1993, p32):

> *After adding the probabilities, we could call this a "probabilistic finite-state automaton," but such models have different names in the statistical literature. In particular, that in figure 2.4 is called a* Markov chain.

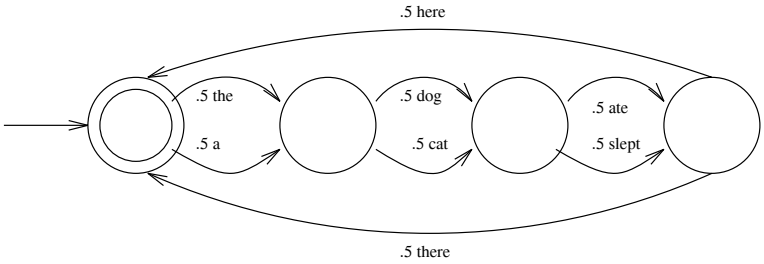


Figure 2.4 A trivial model of a fragment of English

> *Like finite-state automata, Markov chains can be thought of as acceptors or generators. However, associated with each arc is the probability of taking that arc given that one is in the state at the tail of the arc. Thus the numbers associated with all of the arcs leaving a state must sum to one. The probability then of generating a given string in such a model is just the product of the probabilities of the arcs traversed in generating the string. Equivalently, as an acceptor the Markov chain assigns a probability to the string it is given. (This only works if all states are accepting states, something standardly assumed for Markov processes.)*

This paragraph is misleading with respect to one of the fundamental points in this set of notes:
The figure shows a finite automaton, not a Markov chain or Markov model.
While Markov models are similar to probabilistic finite automata in the sense mentioned in (78), Markov chains are not like finite automata, as we saw in (77).

In particular, Markov models can define output sequences with (a finite number of) unbounded dependencies, but Markov chains define only state sequences with the Markovian requirement that blocks non-adjacent dependencies.

(99) Charniak (1993, p39) says:

> *One of the least sophisticated but most durable of the statistical models of English is the n-gram model. This model makes the drastic assumption that only the previous n-1 words have any effect on the probabilities for the next word. While this is clearly false, as a simplifying assumption it often does a serviceable job.*

Serviceable? What is the job??
For the development of the science and of the field, the question is: how can we move towards a model that is not "clearly false."

(100) Abney (1996a, p21) says:

> *In fact, probabilities make Markov models more adequate then their non-probabilistic counterparts, not less adequate. Markov models are surprisingly effective, given their finite-state substrate. For example, they are the workhorse of speech recognition technology. Stochastic grammars can also be easier to learn than their non-stochastic counterparts…*

We might agree about the interest of (non-finite state) stochastic grammars. Certainly, developing stochastic grammars, one of the main questions is: which grammars, which structural relations do we find in human languages? This is the traditional focus of theoretical linguistics. As for the stochastic influences, it is not yet clear what they are, or how revealing they will be.

As for the first sentence in this quoted passage, and the general idea that we can develop good stochastic models without attention to the expressive capabilities of the "substrate," you decide.

(101) It is quite possible that "lexical activation" is sensitive to word co-occurrence frequencies, and this might be modeled with a probabilistic finite automaton (e.g. a state-labeled Markov model or a standard, transition-labeled probabilistic fsa).

The problem of detecting stochastic influences in the grammar itself depends on knowing what parts of the grammar depend on the lexical item. In CFGs, for example, we get only a simple category for each word, but in lexicalized TAGs, and in recent transformational grammars, the lexical item can provide a rich specification of its role in derivations.

8.1.14 Probabilistic finite automata and regular grammars

Finite (or "finite state") automata (FSAs) are usually defined by associating "emitted" vocabulary elements with transitions between non-emitting states. These automata can be made probabilistic by distributing probabilities across the various transitions from each state (counting termination as, e.g., a special transition to a "stop" state).

Finite, time-invariant Markov models (FMMs) are defined by (probabilistic) transitions between states that themselves (probabilistically) emit vocabulary elements.

We can specify a "translation" between FSAs and FMMs.

(102) Recall that a **finite state automaton** can be defined with a 5-tuple $A = \langle Q, \Sigma, \delta, I, F \rangle$ where

Q is a finite set of states ($\neq \emptyset$);

Σ is a finite set of symbols ($\neq \emptyset$);

δ : $Q \times \Sigma \rightarrow 2^Q$,

I $\subseteq Q$, the initial states;

F $\subseteq Q$, the final states.

We allow Σ to contain the empty string ϵ.

(103) Identifying derivations by the sequence of productions used in a leftmost derivation, and assuming that all derivations begin with a particular "start" category, we can distribute probabilities over the set of possible rules that rewrite each category.

This is a **probabilistic finite state automaton**.

(We can generalize this to the case where there is more than one initial state by allowing an initial vector that determines a probability distibution across the initial states.)

(104) As observed in (16) on page 31, a language is accepted by a FSA iff it is generated by some right linear grammar.

(105) **Exercise**

1. Define an ambiguous, probabilistic right linear grammar such that, with the prolog top-down GLC parser, no ordering of the clauses will be such that parses will always be returned in order of most probable first.
2. Implement the probabilistic grammar defined in the previous step, annotating the categories with features that specify the probability of the parse, and run a few examples to illustrate that the more probable parses are not always being returned first.

(106) **SFSA→FMM correspondence**

1. **Insert emitting states**

   Given finite automaton $A = \langle Q, \Sigma, \delta, I, F \rangle$, first define $A' = \langle Q', \Sigma, \delta', I, F \rangle$ as follows:

   We define new states corresponding to each transition of the original automaton:

   $$Q' = Q \cup \{q1.a.q2 |\ q2 \in \delta(a, q1)\}$$

   We then interrupt each a-transition from $q1$ to $q2$ with an empty transition to $q1.a.q2$, so that all transitions become empty:

   The probability P of the transition $q1, a \vdash_P q2$ in A is associated with the new transition $q1 \vdash_P q1.a.q2$, and the new transition $q1.a.q2 \vdash_{P=1} q2$ has probability 1.

2. **Eliminate non-emitting states**

   Change $qi.a.qj \vdash_{P=1} qk \vdash_{P=p} qk.b.ql$ to $qi.a.qj \vdash_{P=p} qk.b.ql$

3. If desired, add a final absorbing state

XXX This should be filled out, and results established

(107) **Sketch of a FMM→SFSA correspondence**

1. **Insert non-emitting states**
   For each a emitted with non-zero probability P_a by q_i, and for each q_j which has a non-zero probability P_{ij} of following q_i, introduce new state $q_i.q_j$ with the new FSA arcs:
   $(q_i, a) \vdash_{P_a} q_i.q_j$
   $(q_i.q_j, \epsilon) \vdash_{P_{ij}} q_j$
2. **Special treatment for inital and final states**

   XXX This should be filled out, and results established

8.1.15 Information and entropy

(108) Suppose $|\Omega_X| = 10$, where these events are equally likely and partition Ω.

If we find out that $X = a$, how much information have we gotten?

9 possibilities are ruled out.

The possibilities are reduced by a factor of 10.

But Shannon (1948, p32) suggests that a more natural measure of the amount of information is the number of "bits." (A name from J.W. Tukey? Is it an acronym for BInary digiT?)

How many binary decisions would it take to pick one element out of the 10? We can pick 1 out of 8 with 3 bits; 1 out of 16 with 4 bits; so 1 out of 10 with 4 (and a little redundancy). More precisely, the number of bits we need is $\log_2(10) \approx 3.32$.

Exponentiation and logarithms review

$$k^m \cdot k^n = k^{m+n}$$

$$k^0 = 1$$

$$k^{-n} = \tfrac{1}{k^n}$$

$$\tfrac{a^m}{a^n} = a^{m-n}$$

$$\log_k x = y \text{ iff } k^y = x$$

$$\log_k(k^x) = x \text{ since: } k^x = k^x$$

and so: $\log_k k = 1$

and: $\log_k 1 = 0$

$$\log_k(\tfrac{M}{N}) = \log_k M - \log_k N$$

$$\log_k(MN) = \log_k M + \log_k N$$

so, in general: $\log_k(M^p) = p \cdot \log_k M$

and we will use: $\log_k \tfrac{1}{x} = \log_k x^{-1} = -1 \cdot \log_k x = -\log_k x$

E.g. $512 = 2^9$ and so $\log_2 512 = 9$. And $\log_{10} 3000 = 3.48 = 10^3 \cdot 10^{0.48}$. And $5^{-2} = \frac{1}{25}$, so $\log_5 \frac{1}{25} = -2$

We'll stick to $\log_2$ and "bits," but another common choice is $\log_e$, where

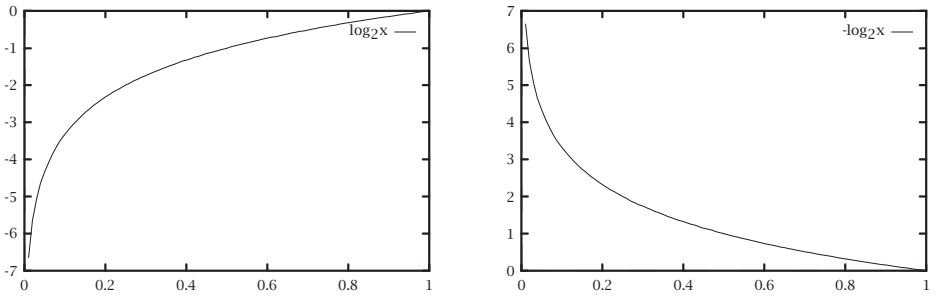
$$e = \lim_{x \to 0}(1+x)^{\frac{1}{x}} = \sum_{n=0}^{\infty} \frac{1}{n!} \approx 2.7182818284590452$$

Or, more commonly, e is defined as the x such that a unit area is found under the curve $\frac{1}{u}$ from $u = 1$ to $u = x$, that is, it is the positive root x of $\int_1^x \frac{1}{u} du = 1$.

This number is irrational, as shown by the Swiss mathematician Leonhard Euler (1707-1783), in whose honor we call it e. In general: $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$. And furthermore, as Euler discovered, $e^{\pi\sqrt{-1}} + 1 = 0$. This is sometimes called the most famous of all formulas (Maor, 1994). It's not, but it's amazing.

Using $\log_2$ gives us "bits," $\log_e$ gives us "nats," and $\log_{10}$ gives us "hartleys."

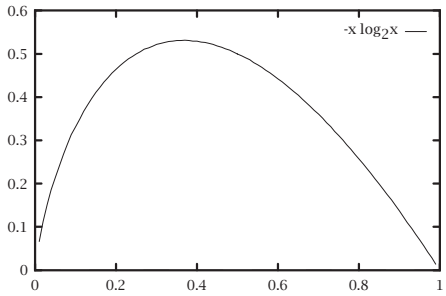
It will be useful to have images of some of the functions that will be defined in the next couple of pages.



surprisal as a function of $p(A)$: $-\log p(A)$

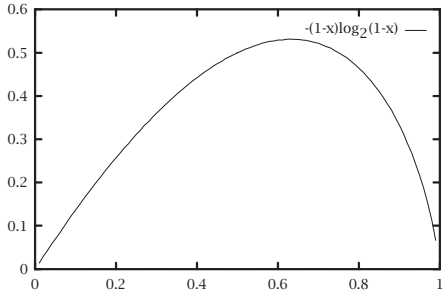
OCTAVE makes drawing these graphs a trivial matter. The graph above is drawn with the command:

```
>x=(0.01:0.01:0.99)';data = [x,log2(x)];gplot [0:1] data
```



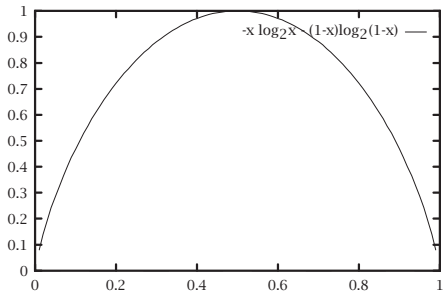
entropy of $p(A)$ as a function of $p(A)$: $-p(A)\log p(A)$

```
>x=(0.01:0.01:0.99)';data = [x,(-x .* log2(x))];gplot [0:1] data
```



entropy of $1 - p(A)$ as a function of $p(A)$: $-(1 - p(A))(\log(1 - p(A))$

```
>x=(0.01:0.01:0.99)';data = [x,(-(1-x) .* log2(1-x))];gplot [0:1] data
```



sum of previous two: $p(A)\log p(A) - (1 - p(A))(\log(1 - p(A))$

```
>x=(0.01:0.01:0.99)';data = [x,(-x .* log2(x))-(1-x).*log2(1-x)];gplot [0:1] data
```

(109) If the outcomes of the binary decisions are not equally likely, though, we want to say something else. The amount of information (or "self-information" or the "surprisal") of an event A,

$$i(A) = \log \frac{1}{P(A)} = -\log P(A)$$

So if we have 10 possible events with equal probabilities of occurrence, so $P(A) = 0.1$, then

$$i(A) = \log \frac{1}{0.1} = -\log 0.1 \approx 3.32$$

(110) The simple cases still work out properly.
In the easiest case where probability is distributed uniformly across 8 possibilities in Ω_X, we would have exactly 3 bits of information given by the occurrence of a particular event A:

$$i(A) = \log \frac{1}{0.125} = -\log 0.125 = 3$$

The information given by the occurrence of $\cup\Omega_X$, where $P(\cup\Omega_X) = 1$, is zero:

$$i(A) = \log \frac{1}{1} = -\log 1 = 0$$

And obviously, if events $A, B \in \Omega_X$ are independent, that is, $P(AB) = P(A)P(B)$, then

$$\begin{aligned} i(AB) &= \log \tfrac{1}{P(AB)} \\ &= \log \tfrac{1}{P(A)P(B)} \\ &= \log \tfrac{1}{P(A)} + \log \tfrac{1}{P(B)} \\ &= i(A) + i(B) \end{aligned}$$

(111) However, in the case where $\Omega_X = \{A, B\}$ where $P(A) = 0.1$ and $P(B) = 0.9$, we will still have

$$i(A) = \log \frac{1}{0.1} = -\log 0.1 \approx 3.32$$

That is, this event conveys more than 3 bits of information even though there is only one other option. The information conveyed by the other event

$$i(B) = \log \frac{1}{0.9} \approx .15$$

Entropy

(112) Often we are interested not in the information conveyed by a particular event, but by the information conveyed by an information source:

> *…from the point of view of engineering, a communication system must face the problem of handling any message that the source can produce. If it is not possible or practicable to design a system which can handle everything perfectly, then the system should handle well the jobs it is most likely to be asked to do, and should resign itself to be less efficient for the rare task. This sort of consideration leads at once to the necessity of characterizing the statistical nature of the whole ensemble of messages which a given kind of source can and will produce. And* information*, as used in communication theory, does just this.* (Weaver, 1949, p14)

(113) For a source X, the average information of an arbitrary outcome in Ω_X is

$$H = \sum_{A \in \Omega_X} P(A) i(A) = - \sum_{A \in \Omega_X} P(A) \log P(A)$$

This is sometimes called "entropy" of the random variable – the average number of bits per event (Charniak, 1993, p29). So called because each $P(A)$ gives us the "proportion" of times that A occurs.

(114) For a source X of an infinite sequence of events, the **entropy** or **average information**, the entropy of the source is usually given as their average probability over an infinite sequence $X_1, X_2, \ldots$, easily calculated from the previous formula to be:

$$H(X) = \lim_{n \to \infty} \frac{G_n}{n}$$

where $G_n =$

$$-n \sum_{A_1 \in \Omega_X} \sum_{A_2 \in \Omega_X} \ldots \sum_{A_n \in \Omega_X} P(X_1 = A_1, X_2 = A_2, \ldots, X_n = A_n) \log P(X_1 = A_1, X_2 = A_2, \ldots, X_n = A_n)$$

(115) When the space Ω_X consists of independent time-invariant events whose union has probability 1, then

$$G_n = -n \sum_{A \in \Omega_X} P(A) \log P(A),$$

and so the **entropy** or **average information** of the source in the following way:

$$H(X) = \sum_{A \in \Omega_X} P(A) i(A) = - \sum_{A \in \Omega_X} P(A) \log P(A)$$

Charniak (1993, p29) calls this the **per word entropy** of the process.

(116) If we use some measure other than bits, a measure that allows r-ary decisions rather than just binary ones, then we can define $H_r(X)$ similarly except that we use $\log_r$ rather than $\log_2$.

(117) Shannon shows that this measure of information has the following intuitive properties (as discussed also in the review of this result in Miller and Chomsky (1963, pp432ff)):

a. Adding any number of impossible events to Ω_X does not change $H(X)$.

b. $H(X)$ is a maximum when all the events in Ω_X are equiprobable.
(see the last graph on page 154)

c. $H(X)$ is additive, in the sense that $H(X_i \cup X_j) = H(X_i) + H(X_j)$ when X_i and X_j are independent.

(118) We can, of course, apply this notion of average information, or entropy to a Markov chain X. In the simplest case, where the events are independent and identically distributed,

$$H(X) = \sum_{q_i \in \Omega_X} P(q_i) H(q_i)$$

Cross-entropy, mutual information, and related things

(119) How can we tell when a model of a language user is a good one? One idea is that the better models are those that maximize the probability of the evidence, that is, minimizing the entropy of the evidence.

Let's consider how this idea could be formalized. One prominent proposal uses the measure "per word cross entropy."

(120) First, let's reflect on the proposal for a moment. Charniak (1993, p27) makes the slightly odd suggestion that one of the two great virtues of probabilistic models is that they have an answer to question (119). (The first claim he makes for statistical models is, I think, that they are "grounded in real text" and "usable" - p.xviii.)

> *The second claim we made for statistical models is that they have a ready-made figure of merit that can be used to compare models, the* per word cross entropy *assigned to the sample text.*

Consider the analogous criterion for non-stochastic models, in which sentences are not more or less probable, but rather they are either in the defined language or not. We could say that the better models are those that define languages that include more of the sentences in a textual corpus. But we do not really want to if the corpus contains strange things that are there for non-linguistic reasons: typos, interrupted utterances, etc.

And on the other hand, we could say that the discrete model should also be counted as better if most of the expressions that are not in the defined language do not occur in the evidence. But we do not want to say this either. First, many sentences that are in the language will not occur for non-linguistic reasons (e.g. they describe events which never occur and which have no entertainment value for us). In fact, there are so many sentences of this sort that it is common here to note a second point: if the set of sentences allowed by the language is infinite, then there will always be infinitely many sentences in the language that never appear in any finite body of evidence.

Now the interesting thing to note is that moving to probabilistic models does not remove the worries about the corresponding probabilistic criterion! Taking the last worry first, since it is the most serious: some sentences will occur never or seldom for purely non-linguistic and highly contingent reasons (i.e. reasons that can in principle vary wildly from one context to another). It does not seem like a good idea to try to incorporate some average probability of occurrence into our language models. And the former worry also still applies: it does not seem like a good idea to assume that infrequent expressions are infrequent because of properties of the language. The point is: we cannot just assume that having these things in the model is a good idea. On the contrary, it does not seem like a good idea, and if it turns out to give a better account of the language user, that will be a significant discovery. In my view, empirical study of this question has not yet settled the matter.

It has been suggested that frequently co-occurring words become associated in the mind of the language user, so that they activate each other in the lexicon, and may as a result tend to co-occur in the language user's speech and writing. This proposal is well supported by the evidence. It is quite another thing to propose that our representation of our language models the relative frequencies of sentences in general. In effect, the representation of the language would then contain a kind of model of the world, a model according to which our knowledge of the language tells us such things as the high likelihood of "President Clinton," "Camel cigarettes," "I like ice cream" and "of the," and the relative unlikelihood of "President Stabler," "Porpoise cigarettes," "I like cancer" and "therefore the." If that is true, beyond the extent predicted by simple lexical associations, that will be interesting.

One indirect argument for stochastic models of this kind could come from the presentation of a theory of human language acquisition based on stochastic grammars.

8.1.16 Codes

(121) Shannon considers the information in a discrete, noiseless message. Here, the space of possible events Ω_X is given by an alphabet (or "vocabulary") Σ.

A fundamental result is Shannon's result that the entropy of the source sets a lower bound on the size of the messages. We present this result in §129 below after setting the stage with the basic ideas we need.

(122) Sayood (1996, p26) illustrates some basic points about codes with some examples. Consider:

message	code 1	code 2	code 3	code 4
a	0	0	0	0
b	0	1	10	01
c	1	00	110	011
d	10	11	111	0111
avg length	1.125	1.125	1.75	1.875

Notice that baa in code 2 is 100. But 100 is also the encoding of bc.

We might like to avoid this. Codes 3 and 4 have the nice property of *unique decodability*. That is, the map from message sequences to code sequences is 1-1.

(123) Consider encoding the sequence
9 11 11 11 14 13 15 17 16 17 20 21

a. To transmit these numbers in binary code, we would need 5 bits per element.
b. To transmit 9 different digits: 9, 11, 13, 14, 15, 16, 17, 20, 21, we could hope for a somewhat better code! 4 bits would be more than enough.
c. An even better idea: notice that the sequence is close to the function $f(n) = n+8$ for $n \in \{1,2,\ldots\}$ The **perturbation** or **residual** $X_n - f(n) = 0,1,0,-1,1,-1,0,1,-1,-1,1,1$, so it suffices to transmit the perturbation, which only requires two bits.

(124) Consider encoding the sequence,
27 28 29 28 26 27 29 28 30 32 34 36 38

This sequence does not look quite so regular as the previous case.

However, each value is near the previous one, so one strategy is to let your receiver know the starting point and then send just the changes:
(27) 1 1 -1 -2 1 2 -1 2 2 2 2 2

(125) Consider the follow sequence of 41 elements, generated by a probabilistic source:
axbarayaranxarrayxranxfarxfaarxfaaarxaway

There are 8 symbols here, so we could use 3 bits per symbol.

On the other hand, we could use the following variable length code:

a	1
x	001
b	01100
f	0100
n	0111
r	000
w	01101
y	0101

With this code we need only about 2.58 bits per symbol

(126) Consider
1 2 1 2 3 3 3 3 1 2 3 3 3 3 1 2 3 3 1 2
Here we have $P(1) = P(2) = \frac{1}{4}$ and $P(3) = \frac{1}{2}$, so the entropy is 1.5/bits per symbol.
The sequence has length 20, so we should be able to encode it with 30 bits.
However, consider blocks of 2. $P(1\ 2) = \frac{1}{2}$, $P(3\ 3) = \frac{1}{2}$, and the entropy is 1 bit/symbol.
For the sequence of 10 blocks of 2, we need only 10 bits.
So it is often worth looking for structure in larger and larger blocks.

8.1.17 Kraft's inequality and Shannon's theorem

(127) MacMillan:

If uniquely decodable code C has K codewords of lengths $l_1, \ldots, l_K$ then

$$\sum_{i=1}^{K} 2^{-l_i} \leq 1.$$

(128) Kraft (1949):

If a sequence $l_1, \ldots, l_K$ satisfies the previous inequality, then there is a uniquely decodable code C that has K codewords of lengths $l_1, \ldots, l_K$

(129) **Shannon's theorem.** Using the definition of H_r in (116), Shannon (1948) proves the following famous theorem which specifies the information-theoretic limits of data compression:

Suppose that X is a first order source with outcomes (or outputs) Ω_X. Encoding the characters of Ω_X in a code with characters Γ where $|\Gamma| = r > 1$ requires an average of $H_r(X)$ characters of Γ per character of Ω_X.
Furthermore, for any real number $\epsilon > 0$, there is a code that uses an average of $H_r(X) + \epsilon$ characters of Γ per character of Ω_X.

8.1.18 String edits and other varieties of sequence comparison

Overviews of string edit distance methods are provided in Hall and Dowling (1980) and Kukich (1992).
Masek and Paterson (1980) present a fast algorithm for computing string edit distances.
Ristad (1997) and Ristad and Yianilos (1996) consider the problem of learning string edit distances.

8.2 Probabilisitic context free grammars and parsing

8.2.1 PCFGs

(130) A probabilistic context free grammar (PCFG)

$$G = \langle \Sigma, N, (\rightarrow), \mathsf{S}, P \rangle,$$

where

1. Σ, N are finite, nonempty sets,
2. S is some symbol in N,
3. the binary relation $(\rightarrow) \subseteq N \times (\Sigma \cup N)^*$ is also finite (i.e. it has finitely many pairs),

4. the function $P : (\rightarrow) \rightarrow [0,1]$ maps productions to real numbers in the closed interval between 0 and 1 in such a way that

$$\sum_{\langle \mathsf{c},\beta\rangle \in (\rightarrow)} P(\mathsf{c} \rightarrow \beta) = 1$$

We will often write the probabilities assigned to each production on the arrow: $c \xrightarrow{p} \beta$

(131) The probability of a parse is the product of the probabilities of the productions in the parse

(132) The probability of a string is the sum of the probabilities of its parses

8.2.2 Stochastic CKY parser

(133) We have extended the CKY parsing strategy can handle any CFG, and augmented the chart entries so that they indicate the rule used to generate each item and the positions of internal boundaries.

We still have a problem getting the parses out of the chart, since there can be too many of them: we do not want to take out one at a time!

One thing we can do is to extract just the most probable parse. An equivalent idea is to make all the relevant comparisons before adding an item to the chart.

(134) For any input string, the CKY parser chart represents a grammar that generates only the input string. We can find the most probable parse using the Trellis-like construction familiar from Viterbi's algorithm.

(135) For any positions $0 \leq i, j \leq |input|$, we can find the rule $(i, j) : A : X$ with maximal probability.

$$(i-1, a_i, i) \qquad [axiom]$$

$$\frac{(i, a, j)}{(i, A, j, p)} \qquad [reduce1] \quad \text{if } A \xrightarrow{p} a \text{ and } \neg\exists A \xrightarrow{p'} a \text{ such that } p' > p$$

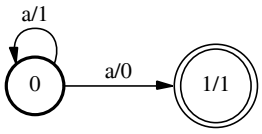
$$\frac{(i, B, j, p_1) \;\; (j, C, k, p_2)}{(i, A, k, p_1 * p_2 * p_3)} \qquad [reduce2] \quad \text{if } A \xrightarrow{p_3} BC \text{ and } \neg\exists (i, B', j, p_1'), (j, C', k, p_2'), A \xrightarrow{p_3'} B'C' \text{ such that } p_1' * p_2' * p_3' > p_1 * p_2 * p_3$$

(136) This algorithm does (approximately) as many comparisons of items as the non-probabilistic version, since the reduce rules require identifying the most probable items of each category over each span of the input.

To reduce the chart size, we need to restrict the rules so that we do not get all the items in there – and then there is a risk of missing some analyses.

8.2.3 Assessment

(137) **Consistent/proper probabilities over languages:**
So far we have been thinking of the derivation steps defined by a grammar or automaton as events which occur with a given probability. But in linguistics, the grammar is posited by the linguist as a model of the **language**. That is, the language is the space of possible outcomes that we are interested in. Keeping this in mind, we see that we have been very sloppy so far! Our probabilistic grammars will often fail to assign probability 1 to the whole language, as for example in this trivial example (labeling arcs with output-symbol/probability):



Intuitively, in this case, the whole probability mass is lost to infinite derivations. Clearly, moving away from this extreme case, there is still a risk of losing some of the probability mass to infinite derivations, meaning that, if the outcome space is really the language (with finite derivations), we are systematically underestimating the probabilities there.
This raises the question: when does a probabilistic automaton or grammar provide a "consistent", or "proper" probability measure over the language generated?

(138) PCFGs cannot reliably decide attachment ambiguities like the following:

a. I bought the lock with the two keys

b. I bought the lock with the two dollars

Obviously, the problem is that the structure of PCFGs is based on the (false!) assumption that (on the intended and readily recognized readings) the probablility of the higher and lower attachment expansions is independent of which noun is chosen at the end of the sentence.
It is interesting to consider how much of the ambiguity of Abney's examples could be resolved by PCFGs.

(139) The previous example also shows that essentially arbitrary background knowledge can be relevant to determining the intended structure for a sentence.

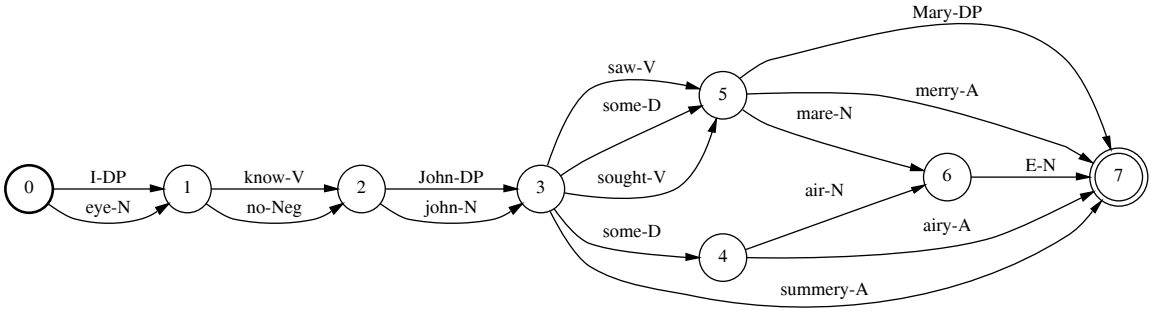
8.3 Multiple knowledge sources

(140) When a language user recognizes what what has been said, it is clear that various sorts of information are used.

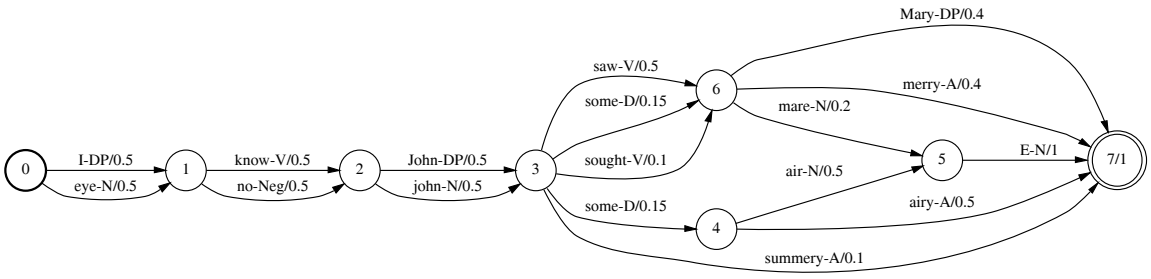
Given an ambiguous acoustic signal (which is certainly very common!), various hypotheses about the words of the utterance will fit the acoustic data more or less well. These hypotheses can also vary with respect to whether they are syntactically well-formed, semantically coherent, and pragmatically plausible.

To study how recognition could work in this kind of setting, let's return to simplified problems like the one that was considered in exercise (3) on page 32.

(141) Suppose, for example, that we have heard "I know John saw Mary" clearly pronounced in a setting with no significant background noise. Leaving aside quote names and neologisms, the acoustic evidence might suggest a set of possibilities like this:



The acoustic evidence will support these various hypotheses differentially, so suppose that our speech processor ranks the hypotheses as follows:



What does this model above suggest as the value of

P(I-DP know-V John-DP saw-V Mary-DP|acoustic signal)?

(142) We could have a stochastic grammar that recognizes some of these possible sentences too.

```
/*
 * file: g6rdin.pl
 */
:- op(1200,xfx,:~).

s  :~ [ip,terminator]/1.                    terminator :~ ['.']/1.

ip :~ [dp, i1]/1.    i1 :~ [i0, vp]/1.0.    i0 :~ [will]/0.4.
                                            i0 :~ []/0.6.

dp :~ ['I']/0.1.     dp :~ ['Mary']/0.1.    dp :~ ['John']/0.1.
dp :~ [dp,ap]/0.1.
dp :~ [d1]/0.6.      d1 :~ [d0, np]/1.0.    d0 :~ [the]/0.55.
                                            d0 :~ [a]/0.44.
                                            d0 :~ [some]/0.01.

np :~ [n1]/1.        n1 :~ [n0]/1.        n0 :~ [saw]/0.2.
                                          n0 :~ [eye]/0.2.
                                          n0 :~ [air]/0.2.
                                          n0 :~ [mare]/0.2.
                                          n0 :~ ['E']/0.2.

vp :~ [v1]/1.        v1 :~ [v0,dp]/0.7.   v0 :~ [know]/0.3
                     v1 :~ [v0,ip]/0.3.   v0 :~ [saw]/0.7.

ap :~ [a1]/1.        a1 :~ [a0]/1.        a0 :~ [summery].
                                          a0 :~ [airy].
                                          a0 :~ [merry].

startCategory(s).
```

What does the PCFG suggest as the value of

P(I-DP know-V John-DP saw-V Mary-DP|Grammar)?

How should these estimates of the two models be combined?

(143) What is $P_{combined}$(I-DP know-V John-DP saw-V Mary-DP)?

a. backoff: prefer the model which is regarded as more reliable.
(E.g. the most specific one, the one based on the largest n-gram,...)
But this just ignores all but one model.

b. interpolate: use a weighted average of the different models.
But the respective relevance of the various models can vary over the domain. E.g. the acoustic evidence may be reliable when it has just one candidate, but not when various candidates are closely ranked.

c. maximum entropy (roughly sketched):

i. rather than requiring P(I-DP know-V John-DP saw-V Mary-DP|acoustic signal)= $k1$, use a corresponding "constraint" which specifies the expected value of $P_{combined}$(I-DP know-V John-DP saw-V Mary-DP).[38]
So we could require, for example,
$E(P_{combined}$(I-DP know-V John-DP saw-V Mary-DP|acoustic signal)) $= k1$,
We can express all requirements in this way, as constraint functions.

ii. if these constraint functions do not contradict each other, they will typically be consistent with infinitely many probability distributions, so which distribution should be used? Jaynes' idea is:
let $p_{combined}$ be the probability distribution with the maximum entropy.[39]
Remarkably: there is exactly one probability distribution with maximum entropy, so our use of the definite article here is justified! It is the probability distribution that is as uniform as possible given the constraints.

Maximum entropy models appear to be more successful in practice than any other known model combination methods. See, e.g. Berger, Della Pietra, and Della Pietra (1996), Rosenfeld (1996), Jelinek (1999, §§13,14), Ratnaparkhi (1998), all based on basic ideas from Jaynes (1957), Kullback (1959).

[38] The "expectation" $E(X)$ of random variable X is $\sum_{x \in rng(X)} x \cdot p(x)$. For example, given a fair 6-sided die with outcomes $\{1,2,3,4,5,6\}$,

$$E(X) = \sum_{i=1}^{6} i \cdot \frac{1}{6} = \frac{1}{6} + (2 \cdot \frac{1}{6}) + \ldots + (6 \cdot \frac{1}{6}) = \frac{7}{2}.$$

Notice that the expectation is not a possible outcome. It is a kind of weighted average. The dependence of expectation on the probability distribution is often indicated with a subscript $E_p(X)$ when the intended distribution is not obvious from context.

[39] Jelinek (1999, p220) points out that, if there is some reason to let the default assumption be some distribution other than the uniform one, this framework extends straightforwardly to minimize the difference from an arbitrary distribution.

8.4 Next steps

a. Important properties of PCFGs and of CFGs with other distributions are established in Chi (1999).

b. Train a stochastic context free grammar with a "treebank," etc, and then "smooth" to handle the sparse data problem: Chen and Goodman (1998).

c. Transform the grammar to carry lexical particulars up into categories: Johnson (1999), Eisner and Satta (1999)

d. Instead of finding the very best parse, use an "n-best" strategy: Charniak, Goldwater, and Johnson (1998) and many others.

e. Probabilistic Earley parsing and other strategies: Stolcke (1995), Magerman and Weir (1992)

f. Stochastic unification grammars: Abney (1996b), Johnson et al. (1999)

g. Use multiple information sources: Ratnaparkhi (1998)

h. Parse mildly context sensitive grammars: slightly more powerful than context free, but much less powerful than unification grammars and unrestricted rewrite grammars. Then, stochastic versions of these grammars can be considered.

We will pursue the last of these topics first, returning to some of the other issues later.

9 Beyond context free: a first small step

(1) Many aspects of language structure seem to be slightly too complex to be captured with context free grammars. Many different, more powerful grammar formalisms have been proposed, but almost all of them lie in the class that Joshi and others have called "mildly context sensitive" (Joshi, Vijay-Shanker, and Weir, 1991; Vijay-Shanker, Weir, and Joshi, 1987).

This class includes certain Tree Adjoining Grammars (TAGs), certain Combinatory Categorial Grammars (CCGs), and also a certain kind of grammar with movements (MGs) that that has been developed since 1996 by Cornell, Michaelis, Stabler, Harkema, and others.

It also includes a number of other approaches that have not gotten so much attention from linguists: Linear Context Free Rewrite Systems (Weir, 1988), Multiple Context Free Grammars (Seki et al., 1991), Simple Range Concatenation Grammars (Boullier, 1998). As pointed out by Michaelis, Mönnich, and Morawietz (2000), these grammars are closely related to literal movement grammars (Groenink, 1997), local scattered context languages (Greibach and Hopcroft, 1969; Abramson and Dahl, 1989), string generating hyperedge replacement grammars (Engelfriet, 1997), deterministic tree-walking tree-to-string transducers (Kolb, Mönnich, and Morawietz, 1999), yields of images of regular tree languages under finite-copying top-down tree transductions, and more! The convergence of so many formal approaches on this neighborhood might make one optimistic about what could be found here.

These methods can all be regarded as descending from Pollard's (1984) insight that the expressive power of context-free-like grammars can be enhanced by marking one or more positions in a string where further category expansions can take place. The tree structures in TAGs and in transformational grammars play this role.

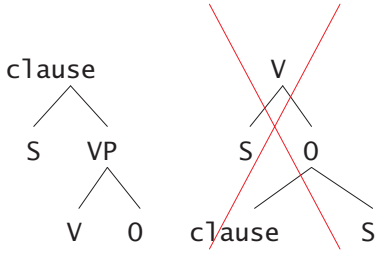
This chapter will define MGs, which were inspired by the early "minimalist" work of Chomsky (1995), Collins (1997), and many others in the transformational tradition. There are various other formal approaches to parsing transformational grammar, but this one is the simplest.[40]

[40] See, for example, Marcus (1980), Berwick and Weinberg (1984), Merlo (1995), Crocker (1997), Fong (1999), Yang (1999). Rogers and Kracht have developed elegant formal representations of parts of transformational grammar, but no processing model for these representations has been presented (Rogers, 1995; Rogers, 1999; Kracht, 1993; Kracht, 1995; Kracht, 1998).

9.1 "Minimalist" grammars

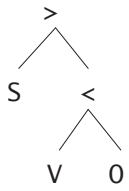
(2) **Phrase structure:** eliminating some redundancy.
Verb phrases alwasy have verbs in them; noun phrases have nouns in them. Nothing in the context free formalism enforces this:

1. cfg



2. X-bar theory says that VPs have V "heads," but then the category of each lexical item gets repeated three times (V,V',VP; D,D',DP; etc):

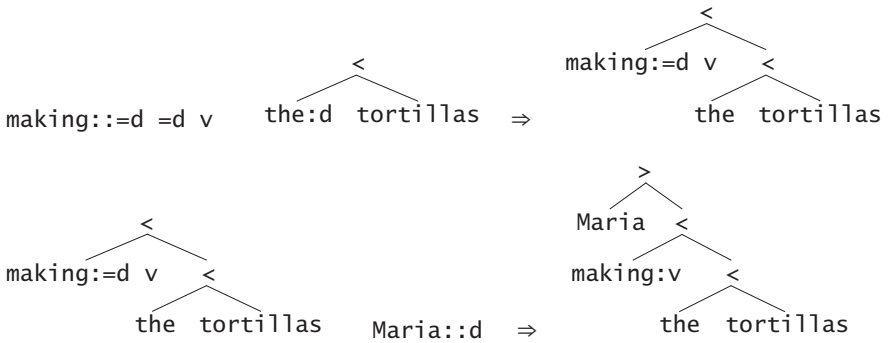
3. bare grammar eliminates this redundancy in the labelling of the tree structures by labelling internal nodes with only > or <, which "point" to the daughter that has the head.



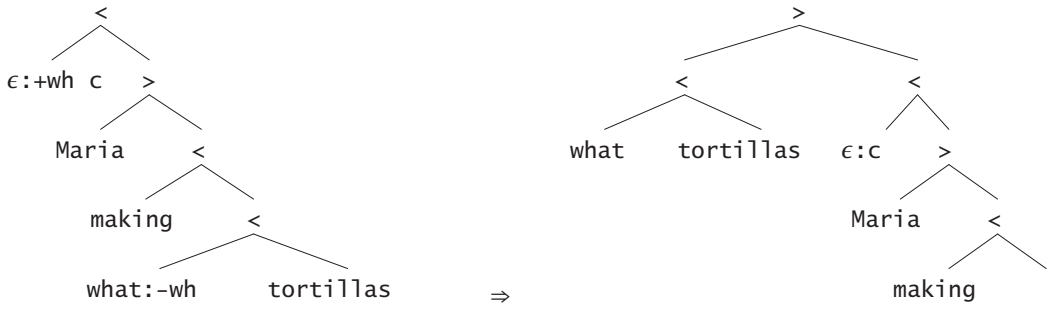
The development of restrictions on hierarchical (dominance) relations in transformational grammar is fairly well known.
Very roughly, we could use cfg derivation trees (as base structures), but these can define many sorts of structures that we will never need, like verb phrases with no verbs in them.
To restrict that class: x-bar theory requires that every phrase (or at least, every complex phrase) of category X is the projection of a head of category X.
Bare grammar makes explicit the fact that lexical insertion must respect the category of the head. In other words, all features are lexical. Shown here is my notation, where the order symbols just "point" to the projecting subtree.

(3) **"minimalist" grammars** $\mathcal{MG}$

- **vocabulary** Σ**:**
  every,some,student,... (non-syntactic, phonetic features)
- **two types** T**:**
  :: (lexical items)
  : (derived expressions)
- **features** F**:**
  c, t, d, n, v, p,... (selected categories)
  =c, =t, =d, =n, =v, =p,... (selector features)
  +wh, +case, +focus,... (licensors)
  -wh, -case, -focus,... (licensees)
- **expressions** E**:** trees with non-root nodes ordered by < or >
- **lexicon:** $\Sigma^* \times \{::\} \times F^*$, a finite set
- **Two structure building rules (partial functions on expressions):**
  • $merge: (E \times E) \to E$



  • $move: E \to E$



(4) More formally, the structure building rules can be formulated like this:

- • structure building (partial) function: $merge: (exp \times exp) \rightarrow exp$
  Letting $t[f]$ be the result of prefixing feature f to the sequence of features at the head of t, for all trees t_1, t_2 all $\mathsf{c} \in Cat$,

$$merge(t_1[\mathsf{=c}], t_2[\mathsf{c}]) = \begin{cases} \overset{<}{\wedge}\ t_1\ t_2 & \text{if } t_1 \in Lex \\ \overset{>}{\wedge}\ t_2\ t_1 & \text{otherwise} \end{cases}$$

  • structure building (partial) function: $move: exp \rightarrow exp$
  Letting $t^>$ be the maximal projection of t,
  for any tree $t_1[\mathsf{+f}]$ which contains exactly one node with first feature $\mathsf{-f}$

$$move(t_1[\mathsf{+f}]) = \overset{>}{\wedge}\ t_2^>\ t_1\{t_2[\mathsf{-f}]^>/\epsilon\}$$

(5) In sum:

Each lexical item is a (finite) sequence of features.

Each structure building operation "checks" and cancels a pair of features.

Features in a sequence are canceled from left to right.

Merge applies to a simple head and the first constituent it selects by attaching the selected constituent on the right, in **complement** position. If a head selects any other constituents, these are attached to the left in **specifier** positions.

All movement is overt, phrasal, leftward. A maximal subtree moves to attach on the left as a specifier of the licensing phrase.

One restriction on movement here comes from the requirement that movement cannot apply when two outstanding $\mathsf{-f}$ requirements would compete for the same position. This is a strong version of the "shortest move" condition (Chomsky, 1995).

We may want to add additional restrictions on movement, such as the idea proposed by Koopman and Szabolcsi (2000a): that the moved tree must be a comp$^+$ or the specifier of a comp$^+$. We will consider these issues later.

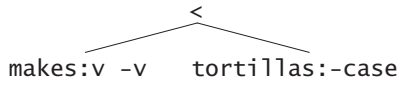
These operations are asymmetric with respect to linear order! So they fit best with projects in "asymmetric syntax:" Kayne (1994), Kayne (1999), Koopman and Szabolcsi (2000a), Sportiche (1999), Mahajan (2000), and many others. It is not difficult to extend these grammars to allow head movement, adjunction and certain other things, but we will stick to these simple grammars for the moment.

9.1.1 First example

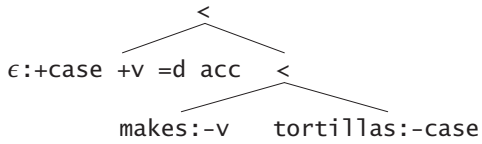
(6) To review, let's quickly rehearse a derivation from the following very simple example grammar:

```
ϵ::=t c                 ϵ::=t +wh c
ϵ::=acc +case t
ϵ::=v +case +v =d acc
makes::=d v -v
maria::d -case          tortillas::d -case      what::d -case -wh
```

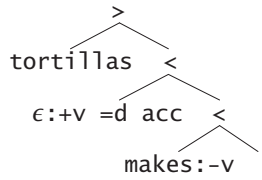
This grammar has eight lexical items. Combining the lexical item `makes` with the lexical item `tortillas`, we obtain:



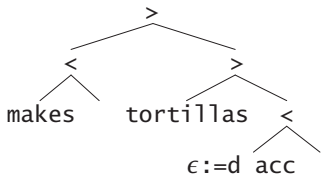
This complex is a VP, which can be selected by the lexical item with category `acc` to yield:



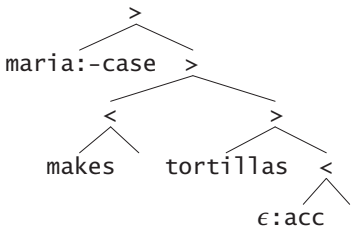
Move can apply to this constituent, yielding



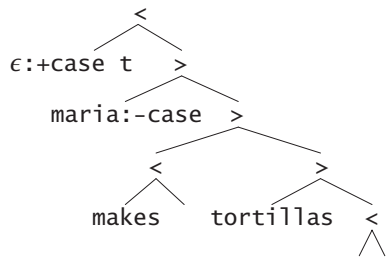
This is an accP, with another movement trigger on its head, so move applies again to yield:



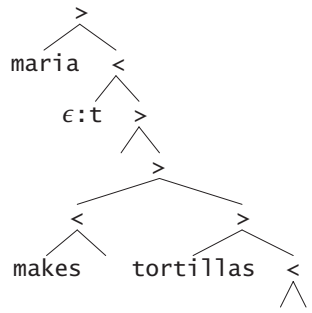
This is still an accP that now want to select another dP, the "external argument", so we can let this phrase merge with the lexical item `maria` to obtain:



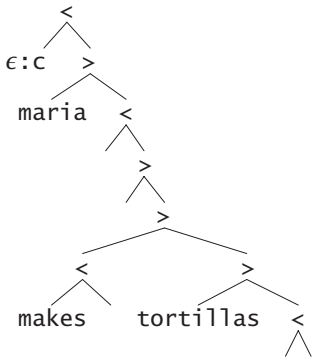
This is a completed accP, which can be selected by the lexical item with category t, to obtain:



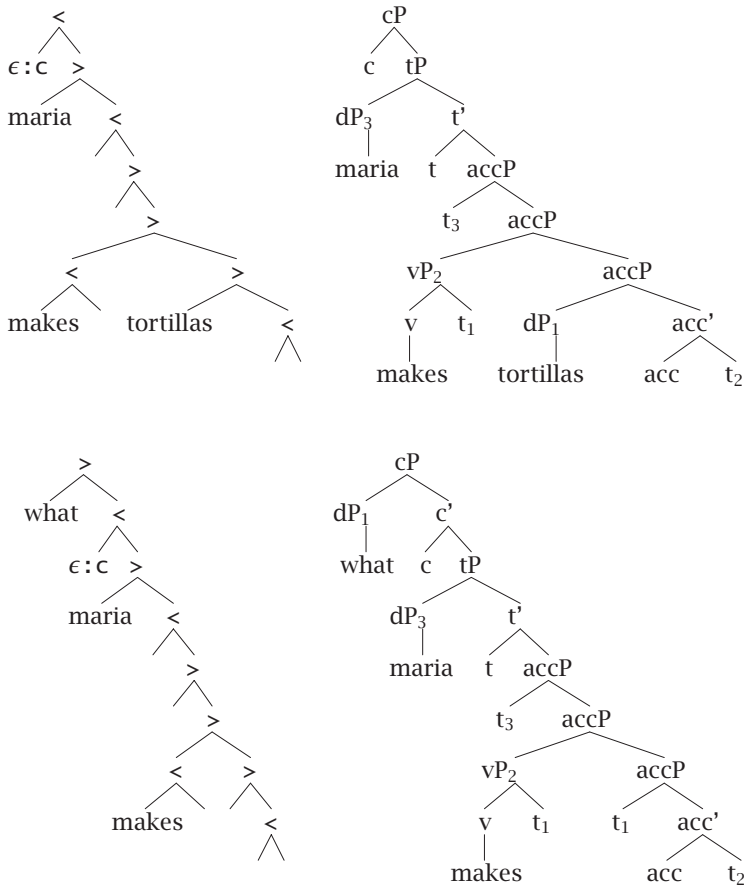
This is a tP, with a movement trigger on its head, so applying movement we obtain:



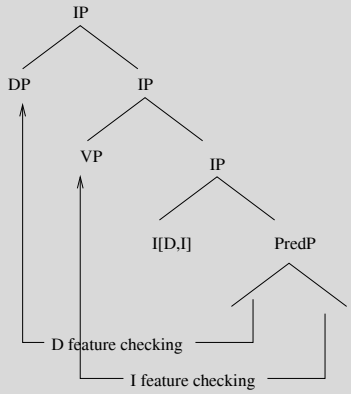
This is a completed tP, which can be selected by the simple lexical item with category c, yielding a tree that has no outstanding syntactic features except the "start" category c:



The final derived tree is shown below, next to the conventional depiction of the same structure. Notice that the conventional depiction makes the history of the derivation much easier to see, but obscures whether all syntactic features were checked. For actually calculating derivations, our "bare trees" are easier to use. We show one other tree that can be derived from the same grammar: a wh-question (without auxiliary inversion, since our rules do not yet provide that):



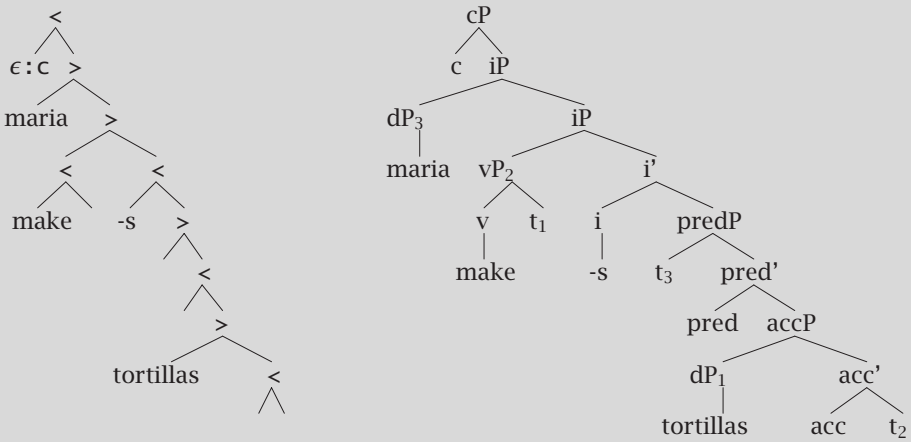
(7) Let's extend this first example slightly by considering how closely the proposed grammar formalism could model the proposals of Mahajan (2000). He proposes that SVIO order could be derived like this, creating an IP with 2 specifier positions:



For this to happen, neither the subject DP nor the object DP can be contained in the VP when it moves. So here is a first idea: the object is selected by the verb but then gets its case checked in an AccP; the AccP is selected by PredP which also provides a position for the subject; the PredP is selected by I, which then checks the V features of the VP and the case features of the subject; and finally the IP is selected by C. The following lexicon implements this analysis:

```
ϵ::=i +wh c
ϵ::=i c
-s::=pred +v +k i
-s::=acc =d pred
-s::=v +k acc
make::=d v -v
maria::d -k        tortillas::d -k      what::d -k -wh
```

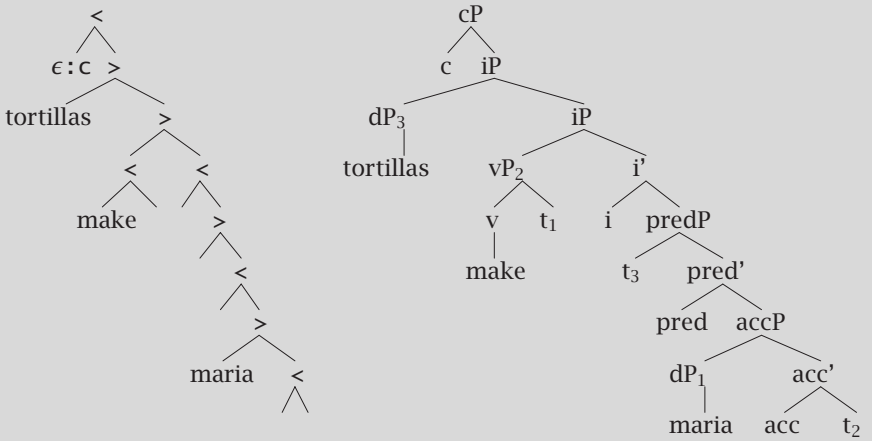
With this grammar, we derive a tree displayed on the left; a structure which would have the more conventional depiction on the right:

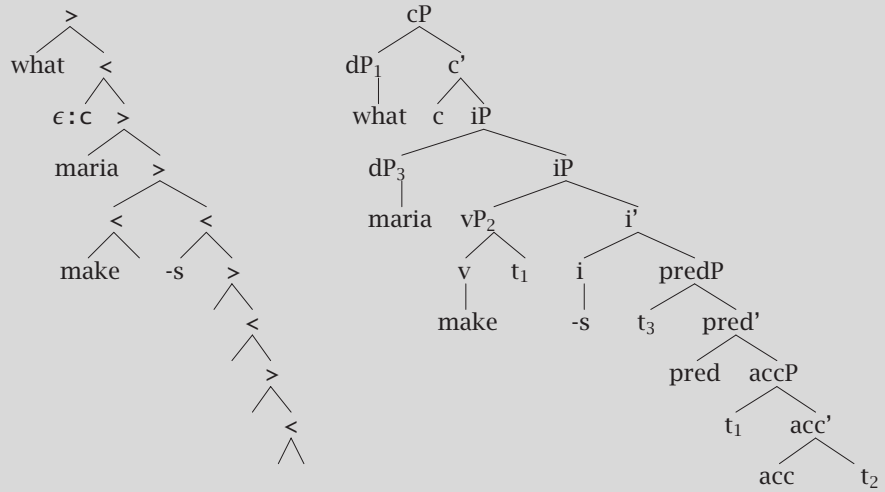


(8) The previous lexicon allows us to derive both `maria make -s tortillas` and `tortillas make -s maria`, as well as `what maria make -s`. The `-s` is only appropriate when the subject is third singular, but in this grammar, `maria` and `tortillas` have identical syntactic features. In particular, the feature `-k` which is checked by the inflection is identical for these two words, even though they differ in number. We can fix this easily. Again, going for just a simple fix first, for illustrative purposes, the following does the trick:

```
ϵ::=i +wh c
ϵ::=i c
-s::=pred +v +k3s i    ϵ::=pred +v +k i
-ed::=pred +v +k3s i   -ed::=pred +v +k i
ϵ::=acc =d pred
ϵ::=v +k acc           ϵ::=v +k3s acc
make::=d v -v
maria::d -k3s          tortillas::d -k tortillas     what::d -k3s -wh
```

With this grammar, we derive the tree shown on the previous page. In addition, we derive the tree displayed below on the left; a structure which would have the more conventional depiction on the right:





And, as desired, we cannot derive:

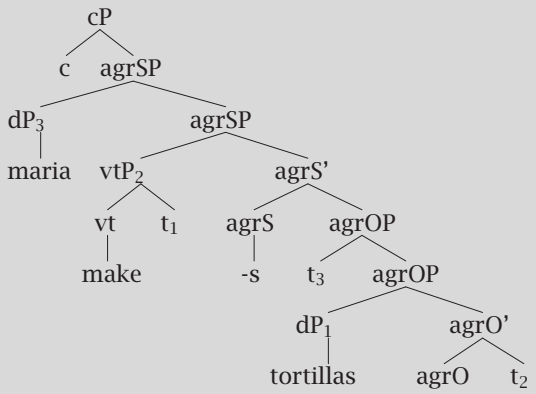
* tortillas make -s maria

* what make tortillas

(9) Let's extend the example a little further. First, Burzio (1986) observed that, at least to a good first approximation, a verb assigns case iff it takes an external argument. In the previous grammar, these two consequences of using a transitive verb are achieved by two separate categories: acc and pred. We can have both together, more in line with Burzio's suggestion, if we allow a projection with two specifiers. We will call this projection agrO, and we can accordingly rename i to agr0. To distinguish transitive and intransitive verbs, we use the categories vt and v, respectively. So then our previous grammar becomes:

```
ϵ::=agrS +wh c
ϵ::=agrS c

-s::=agrO +v +k3s agrS   ϵ::=agrO +v +k agrS    % infl
-ed::=agrO +v +k3s agrS  -ed::=agrO +v +k agrS  % infl

ϵ::=vt +k =d agrO        ϵ::=vt +k3s =d agrO    % Burzio's generalization
ϵ::=v agrO               ϵ::=v agrO             % for intransitives

make::=d vt -v
eat::=d vt -v            eat::=d v -v
```

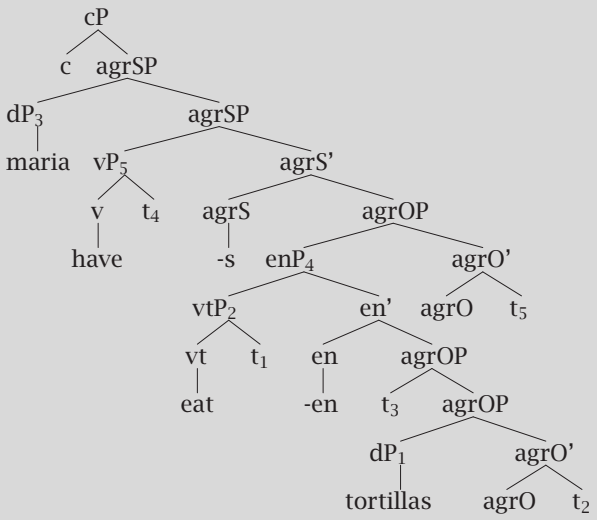
With this grammar, we get trees like this (showing the conventional depiction):



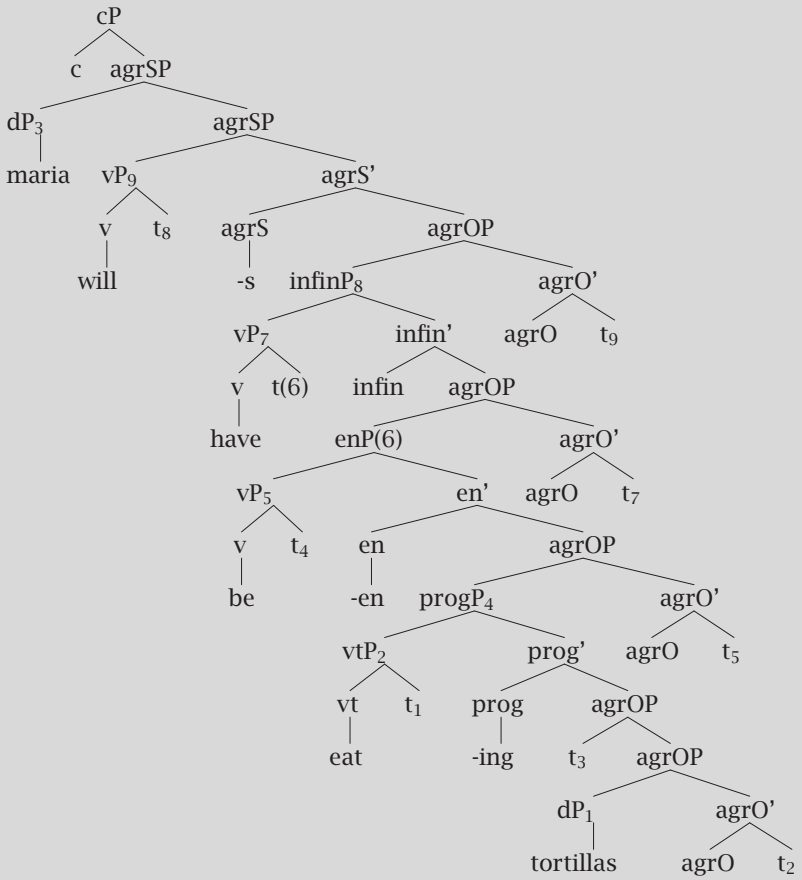
(10) Consider now how we could add auxiliary verbs. One idea is to let the verb `have` select a `-en`-verb phrase (or perhaps a full CP, as Mahajan (2000) proposes). But then we need to get the -en phrase (or CP) out of the VP before the VP raises to agrS. It is doubtful that the -en phrase raises to a case position, but it is a position above the VP. For the moment, let's assume it moves to a specifier of agrO, adding the following lexical items to the previous grammar:

```
=v +aux agrO    =en v -v have       =agrO +v en -aux '-en'
                =prog v -v be       =agrO +v prog -aux '-ing'
                =infin v -v will    =agrO +v infin -aux '-inf'
```

This gives us trees like the following (these lexical items are in `g32.pl` and can be used by the the parser which we present in §9.2 on page 183 below):



(11) The previous tree has the right word order, but notice: the subject is extracted from the embedded enP after that phrase has moved. – Some linguists have proposed that this kind of movement should not be allowed. Also, notice that with this simple approach, we also need to ensure, not only that `be -en` gets spelled out as `been`, but also that `will -s` gets spelled out as `will`.[41] And notice that it will be tricky to get the corresponding yes-no question from this structure, since the auxiliary `will -s` is not a constituent. (We will have to do something like extracting the lower AgrOP and then moving the AgrSP above it.)



Exercise: Derive the two trees shown just above by hand, from the lexical items given.

9.1.2 Second example: modifier orders

(12) Dimitrova-Vulchanova and Giusti (1998) observes some near symmetries in the nominal systems of English and Romance, symmetries that have interesting variants in the Balkan languages. [42]

It is interesting to consider whether the restricted grammars we have introduced – grammars with only overt, phrasal movement – can at least get the basic word orders.

In fact, this turns out to be very easy to do. The adjectives in English order appear in the preferred order,

poss>card>ord>qual>size>shape>color>nationality>n

with (partial) mirror constructions:

a. an expensive English fabric

b. un tissu anglais cher

(13) sometimes, though, other orders, as in Albanian:

a. një fustan fantastik blu
   a dress fantastic blue

b. fustan-i fantastik blu
   dress-the fantastic blue

(14) AP can sometimes appear on either side of the N, but with a (sometimes subtle) meaning change:

a. un bon chef (good at cooking)

b. un chef bon (good, more generally)

(15) scopal properties, e.g. obstinate>young in both

a. an obstinate young man

b. un jeune homme obstiné

To represent these possible selection possibilities elegantly, we use the notation >poss to indicate a feature that is either poss or a feature that follows poss in this hierarchy. And similarly for the other features, so >color indicates one of the features in the set >color={color,nat,n}.

We also put a feature (-f) in parentheses to indicate that it is optional.

```
% English
=>poss d -case a(n)
=>qual qual expensive
=>nat nat English
n fabric
```

```
% French
=>poss d -case un
=>qual +f qual (-f) cher
=>nat +f nat (-f) anglais
n (-f) tissu

=>qual (+f) qual (-f) bon
n (-f) chef
```

```
% Albanian
=>poss +f d -case i
=>poss d -case nje
=>qual (+f) qual fantastik
=>color color blu
n -f fustan
```

This grammar gets the word orders shown in (12-14).

The 4 English lexical items allow us to derive [a fabric], [an expensive fabric] and [an expensive English fabric] as determiner phrases (i.e. as trees with no unchecked syntactic features except the feature `d`), but NOT: [an English expensive fabric].

The first 4 French items are almost the same as the corresponding English ones, except for `+f,-f` features that trigger inversions of exactly the same sort that we saw in the approach to Hungarian verbal complexes in Stabler (1999). To derive [un tissu], we must use the lexical item `n tissu` – that is, we cannot include the optional feature `-f`, because that feature can only be checked by inversion with an adjective. The derivation of [un [tissu anglais] cher] has the following schematic form:

i. [anglais tissu] ⟶ (`nat` selects `n`)

ii. [tissu$_i$ anglais t_i] ⟶ (`nat` triggers inversion of `n`)

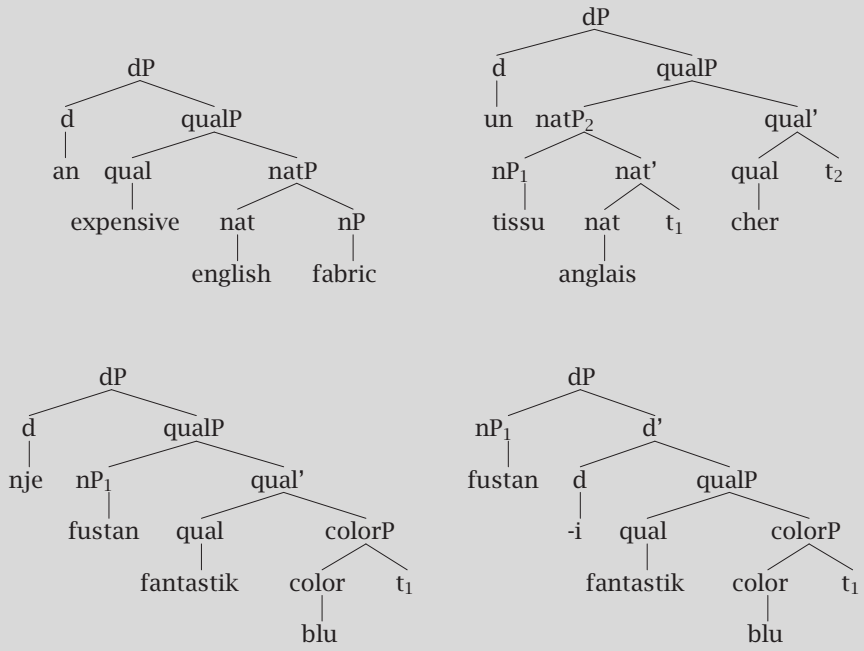
iii. cher [tissu$_i$ anglais t_i] ⟶ (`qual` selects `nat`)

iv. [[tissu$_i$ anglais t_i]$_j$ cher t_j] ⟶ (`qual` triggers inversion of `nat`)

v. un [[tissu$_i$ anglais t_i]$_j$ cher t_j] ⟶ (`d` selects `qual`)

(The entries for `bon`, on the other hand, derive both orders.) So we see that with this grammar, the APs are in different structural configurations when they appear on different sides of the NP, which fits with the (sometimes subtle) semantic differences.

The lexical items for Albanian show how to get English order but with N raised to one side or the other of the article. We can derive [nje fustan fantastik blu] and [fustan-i fantastik blu] but not the other, impossible orders.



Exercise: Check some of the claims made about this last grammar, by trying to derive these trees by hand. (In a moment, we will have a parser that can check them too!)

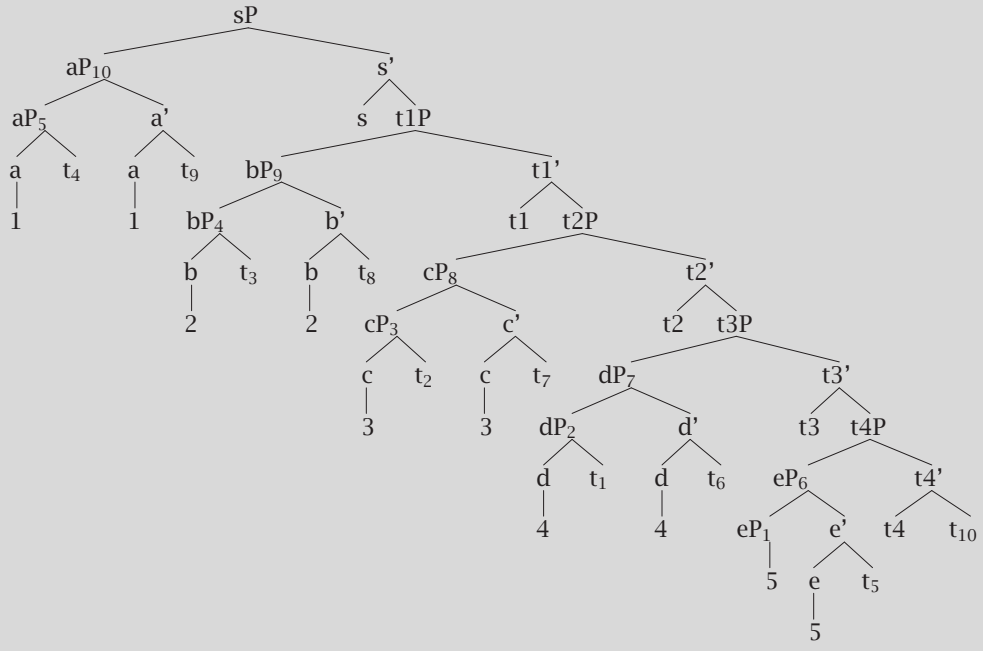
9.1.3 Third example: $1^n2^n3^n4^n5^n$

(16) The language $1^n2^n3^n4^n5^n$ is of interest, because it is known to be beyond the generative power of TAGs, CCGs (as formalized in Vijay-Shanker and Weir 1994), and a number of other similar grammars.

In this grammar, we let the start category be `s`, so intuitively, a successful derivation builds `sP`'s that have no outstanding syntactic requirements.

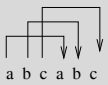
```
s           =t1 +a s
=t2 +b t1
=t3 +c t2
=t4 +d t3
=a +e t4
=b a -a 1   =b +a a -a 1
=c b -b 2   =c +b b -b 2
=d c -c 3   =d +c c -c 3
=e d -d 4   =e +d d -d 4
e -e 5      =a +e e -e 5
```

With this grammar, we build trees like this:



9.1.4 Fourth example: reduplication

(17) While it is not easy to see patterns like $1^n2^n3^n4^n5^n$ in human languages, there are many cases of reduplication. In reduplication, each element of the reduplicated pattern must correspond to an element in the "reduplicant," and these correspondences are "crossing:"



These crossing correspondences are beyond the power of CFGs (unless they are bounded in size as a matter of grammar). They are easily captured in MGs.

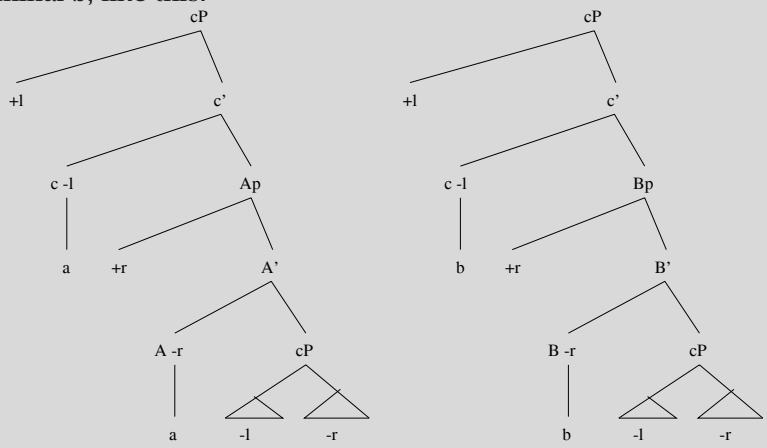
To construct a grammar for a pattern like this, think of how an inductive proof could show that the grammar does what you want.

Considering the recursive step first, each structure could have a substructure with two pieces that can move independently.

Call the features triggering the movement

-l(eft) and -r(ight)

and then the recursive step can be pictured as having two cases, one for AP's with terminal a and one for BP's with terminal b, like this:



Notice that in these pictures, c has a +l and a -l, while A and B each have a +r and -r. That makes the situation nicely symmetric. We can read the lexical items off of these trees:

```
=A +l c -l a          =B +l c -l b
=c +r A -r a          =c +r B -r b
```

With this recursion, we only need to add the base case. Since we already have recursive structures that expect CPs to have a +r and -r, so we just need

=c +r +l c

to finish the derivation, and to begin a derivation, we use:

c -r -l

This grammar, containing 6 lexical items, has 23 occurrences of ten features (3 cats, 3 selectors, 2 licensees, 2 licensors). The grammar makes a big chart even for small trees, partly because it does not know where the middle of the string is. (See if you can find a simpler formulation!)

For each new terminal element, this grammar needs two new lexical items: one for the projection of the terminal in left halves, and one for the right halves of the strings.

9.2 CKY recognition for MGs

(18) **The basic idea:** MGs derive trees rather than strings, and so it is not so easy to imagine how they could be used in an efficient parsing system.

The basic insight we need for handling them is to regard the tree as a structure that stores a number of strings. And in the case of MGs, with "movements" defined as has been done, there is a strict bound on the number of strings that a tree needs to keep distinct, no matter how big the tree is.

Each categorized string in any expression generated by the grammar is called a "chain," because it represents a constituent that may be related to other positions by movement.[43]

(19) Instead of generating categorized strings, MGs can be regarded as generating tuples of categorized strings, where the categorial classification of each string is given by a "type" and a sequence of features. We call each categorized string a "chain," and each expression is then a (nonempty, finite) sequence of chains.

(20) **The formal definition**

A minimalist grammar $G = (\Sigma, F, Types, Lex, \mathcal{F})$, where

Alphabet $\Sigma \neq \emptyset$

Features $F = base$ (basic features, $\neq \emptyset$)

$\cup \{=f \mid f \in base\}$ (selection features)

$\cup \{+f \mid f \in base\}$ (licensor features)

$\cup \{-f \mid f \in base\}$ (licensee features)

Types $= \{::, :\}$ (lexical, derived)

For convenience: Chains $C = \Sigma^* \times Types \times F^*$

Expressions $E = C^+$ (nonempty sequences of chains)

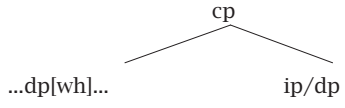
Lexicon $Lex \subseteq C^+$ is a finite subset of $\Sigma^* \times \{::\} \times F^*$.

Generating functions $\mathcal{F} = \{merge, move\}$, partial functions from E* to E, defined below.

Language $L(G) = closure(Lex, \mathcal{F})$.

And for any $\mathsf{f} \in F$, the strings of category f, $S_{\mathsf{f}}(G) = \{s \mid s \cdot \mathsf{f} \in L(G) \text{ for some } \cdot \in \text{Types}\}$.

[43] The "traditional" approach to parsing movements involves passing dependencies (sometimes called "slash dependencies" because of the familiar slash notation for them) down to c-commanded positions, in configurations roughly like this:



Glancing at the trees in the previous sections, we see that this method cannot work: there is no bound on the number of movements through any given part of a path through the tree, landing sites do not c-command their origins, etc.

This intuitive difference also corresponds to an expressive power difference, as pointed out just above: minimalist grammars can define languages like $a^n b^n c^n d^n e^n$ which are beyond the expressive power of TAGs, CCGs (as formalized in Vijay-Shanker and Weir 1994), and standard trace-passing regimes. We need some other strategy.

To describe this strategy, it will help to provide a different view of how our grammars are working.

(21) The generating functions $merge$ and $move$ are partial functions from tuples of expressions to expressions. We present the generating functions in an inference-rule format for convenience, "deducing" the value from the arguments. We write st for the concatenation of s and t, for any strings s, t, and let ϵ be the empty string.

$merge : (E \times E) \rightarrow E$ is the union of the following 3 functions, for $s, t \in \Sigma^*$, for $\cdot \in \{:, ::\}$, for $f \in base$, $\gamma \in F^*$, $\delta \in F^+$, and for chains $\alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l$ $(0 \leq k, l)$

$$\frac{s :: =f\,\gamma \qquad t \cdot f, \alpha_1, \ldots, \alpha_k}{st : \gamma, \alpha_1, \ldots, \alpha_k}$$ merge1: lexical item selects a non-mover

$$\frac{s : =f\,\gamma, \alpha_1, \ldots, \alpha_k \qquad t \cdot f, \iota_1, \ldots, \iota_l}{ts : \gamma, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l}$$ merge2: derived item selects a non-mover

$$\frac{s \cdot =f\,\gamma, \alpha_1, \ldots, \alpha_k \qquad t \cdot f\delta, \iota_1, \ldots, \iota_l}{s : \gamma, \alpha_1, \ldots, \alpha_k, t : \delta, \iota_1, \ldots, \iota_l}$$ merge3: any item selects a mover

Notice that the domains of merge1, merge2, and merge3 are disjoint, so their union is a function.

$move : E \rightarrow E$ is the union of the following 2 functions, for $s, t \in \Sigma^*$, $f \in base$, $\gamma \in F^*$, $\delta \in F^+$, and for chains $\alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l$ $(0 \leq k, l)$ satisfying the following condition,

(SMC) none of $\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k$ has $-f$ as its first feature,

$$\frac{s : +f\,\gamma, \alpha_1, \ldots, \alpha_{i-1}, t : -f, \alpha_{i+1}, \ldots, \alpha_k}{ts : \gamma, \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k}$$ move1: final move of licensee phrase

$$\frac{s : +f\,\gamma, \alpha_1, \ldots, \alpha_{i-1}, t : -f\delta, \alpha_{i+1}, \ldots, \alpha_k}{s : \gamma, \alpha_1, \ldots, \alpha_{i-1}, t : \delta, \alpha_{i+1}, \ldots, \alpha_k}$$ move2: nonfinal move of licensee phrase

Notice that the domains of move1 and move2 are disjoint, so their union is a function.

(22) The (SMC) restriction on the domain of $move$ is a simple version of the "shortest move condition" (Chomsky, 1995,), briefly discussed in §10.6.1 below.

9.2.1 Implementing CKY recognition and parsing

(23) We can use the methods for computing closures that have already been introduced. Instead of writing $string :: y$ or $string : y$, in prolog we write `s:[(Q0,Q):Gamma]` or `c:[(Q0,Q):Gamma]`, respectively, where `s` indicates that the $string$ spanning the input from Q0 to Q is lexical, and `c` indicates that the $string$ spanning the input from Q0 to Q is derived. We need only specify the inference steps, which is easily done:

```
inference(merge1/3,
          [ s:[[X,Y]:[=C|Gamma]],
            _:[[Y,Z]:[C]|Chains] ],
            c:[[X,Z]:Gamma|Chains],
          [smc([[X,Z]:Gamma|Chains])]).

inference(merge2/3,
          [ c:[[X,Y]:[=C|Gamma]|Chains1],
            _:[[V,X]:[C]|Chains2] ],
            c:[[V,Y]:Gamma|Chains],
          [append(Chains1,Chains2,Chains),
           smc([[V,Y]:Gamma|Chains]) ]) .

inference(merge3/3,
          [ _:[[X,Y]:[=C|Gamma]|Chains1],
            _:[[V,W]:[C|[Req|Delta]]|Chains2] ],
            c:[[X,Y]:Gamma,[V,W]:[Req|Delta]|Chains],
          [append(Chains1,Chains2,Chains),
           smc([[X,Y]:Gamma,[V,W]:[Req|Delta]|Chains]) ]).

inference(move1/2,
          [ c:[[X,Y]:[+F|Gamma]|Chains1] ],
            c:[[V,Y]:Gamma|Chains],
          [append(Chains2,[[V,X]:[-F]|Chains3],Chains1),
           append(Chains2,Chains3,Chains),
           smc([[V,Y]:Gamma|Chains]) ]).

inference(move2/2,
          [ c:[([X,Y]:[+F|Gamma])|Chains1] ],
            c:[([X,Y]:Gamma),([V,W]:[Req|Delta])|Chains],
          [append(Chains2,[[V,W]:[-F|[Req|Delta]]|Chains3], Chains1),
           append(Chains2,Chains3,Chains),
           smc([[X,Y]:Gamma,[V,W]:[Req|Delta]|Chains]) ]).

% tentative SMC: no two -f features are exposed at any point in the derivation

smc(Chains) :- smc0(Chains,[]).

smc0([],_).
smc0([_:[-F|_]|Chains],Fs) :- !, \+member(F,Fs), smc0(Chains,[F|Fs]).
smc0([_:_|Chains],Fs) :- smc0(Chains,Fs).
```

(24) It is also an easy matter to modify the chart so that it holds a "packed forest" of trees, from which successful derivations can be extracted easily (if there are any), using the same method discussed for CKY parsing of CFGs (on page 111).

For parsing, we augment the items in the chart exactly as was done for the context free grammars: that is, we add enough information to each item so that we can tell exactly which other items could have been used in its derivation.

Then, we can collect the tree, beginning from any successful item, where a successful item is an expression that spans the whole input and has just the "start category" as its only feature.

(25) I use a file called `setup.pl` to load the needed files:

```
% file: setup.pl
% origin author : E Stabler
% origin date: Feb 2000
% purpose: load files for CKY-like mg parser, swi version,
%      building "standard" trees, real derivation trees, and bare trees
% updates:
% todo:

:- op(500, fx, =). % for selection features
:- op(500, xfy, ::). % lexical items

:- [mgp].  % the parser (builds a "packed forest")
:- [lp].   % builds the tree in various formats

% uncomment one grammar (see associated test examples just below)
:- ['g-ne'].

%SVIO - tests for g-ne
ne_eg(a) :- parse([titus,praise,'-s',lavinia]).
ne_eg(b) :- parse([titus,laugh,'-s']).

% for tree display
```

```
:- ['pp_tree'].
:- ['wish_tree'].  % for windows
%:- ['wish_treeSWI'].  % for unix
:- ['latex_tree'].
%:- ['latex_treeSWI']. % for unix
```

With this code, we get sessions like this:

```
Welcome to SWI-Prolog (Version 4.1.0)
Copyright (c) 1990-2000 University of Amsterdam.

1 ?- [setup].
%    chart compiled 0.00 sec, 1,672 bytes
%    agenda compiled 0.00 sec, 3,056 bytes
%    items compiled 0.00 sec, 904 bytes
%    monitor compiled 0.00 sec, 2,280 bytes
%    driver compiled 0.00 sec, 3,408 bytes
%    utilities compiled 0.00 sec, 1,052 bytes
%   closure-swi compiled 0.00 sec, 13,892 bytes
%   step compiled 0.00 sec, 13,056 bytes
Warning: (/home/es/tex/185/mgp.pl:22):
        Redefined static procedure parse/1
%  mgp compiled 0.00 sec, 46,180 bytes
%  lp compiled 0.01 sec, 20,452 bytes
%  g-ne compiled 0.00 sec, 1,908 bytes
%  pp_tree compiled 0.00 sec, 1,560 bytes
%   draw_tree compiled into draw_tree 0.01 sec, 10,388 bytes
%   fonttbr12 compiled into fonttbr12 0.00 sec, 16,932 bytes
%  wish_tree compiled into wish_tree 0.01 sec, 40,980 bytes
%   fontcmtt10 compiled into fontcmtt10 0.00 sec, 2,324 bytes
%  latex_tree compiled into latex_tree 0.00 sec, 11,640 bytes
% setup compiled 0.02 sec, 123,716 bytes

Yes
2 ?- parse([titus,praise,'-s',lavinia]).
building chart...'.''.''.''.''.''.''.':::::::'.'::'.':'.':'.':'.':'.':'.':'.''.':'.''.':::'.':'.':'.':'.':'.'::'.''.'::
s:  (A, A, empty):=v pred
s:  (A, A, empty):=vt +k =d pred
s:  (A, A, empty):=i +wh c
s:  (A, A, empty):=i c
s:  (0, 1, lex([titus])):d -k
s:  (1, 2, lex([praise])):=d vt -v
s:  (2, 3, lex([-s])):=pred +v +k i
s:  (3, 4, lex([lavinia])):d -k
c:  (1, 2, r3(d, 1, 0)):vt -v   (0, 1, A):-k
c:  (1, 2, r3(d, 4, 0)):vt -v   (3, 4, A):-k
c:  (A, A, r3(vt, 2, 0)):+k =d pred   (1, 2, A):-v   (0, 1, A):-k
c:  (A, A, r3(vt, 2, 0)):+k =d pred   (1, 2, A):-v   (3, 4, A):-k
c:  (0, 1, v1(k, 1, s(0))):=d pred   (1, 2, A):-v
c:  (3, 4, v1(k, 4, s(0))):=d pred   (1, 2, A):-v
c:  (0, 1, r3(d, 1, s(0))):pred   (0, 1, A):-k   (1, 2, A):-v
c:  (0, 1, r3(d, 4, s(0))):pred   (3, 4, A):-k   (1, 2, A):-v
c:  (3, 4, r3(d, 1, s(0))):pred   (0, 1, A):-k   (1, 2, A):-v
c:  (3, 4, r3(d, 4, s(0))):pred   (3, 4, A):-k   (1, 2, A):-v
c:  (2, 4, r1(pred, 3)):+v +k i   (0, 1, A):-k   (1, 2, A):-v
c:  (2, 4, r1(pred, 3)):+v +k i   (3, 4, A):-k   (1, 2, A):-v
c:  (1, 4, v1(v, 2, s(0))):+k i   (0, 1, A):-k
c:  (1, 4, v1(v, 2, s(0))):+k i   (3, 4, A):-k
c:  (0, 4, v1(k, 1, 0)):i
c:  (0, 4, r1(i, 0)):+wh c
c:  (0, 4, r1(i, 0)):c

accepted as category c: titus praise '-s' lavinia
derivation:[1, 3, 4, 6, 8, 9].

Yes
3 ?- showParse([titus,praise,'-s',lavinia]).
building chart...'.''.''.''.''.''.''.':::::::'.'::'.':'.':'.':'.':'.':'.':'.''.':'.''.':::'.':'.':'.':'.':'.'::'.''.'::
s:  (A, A, empty):=v pred
s:  (A, A, empty):=vt +k =d pred
s:  (A, A, empty):=i +wh c
s:  (A, A, empty):=i c
s:  (0, 1, lex([titus])):d -k
s:  (1, 2, lex([praise])):=d vt -v
s:  (2, 3, lex([-s])):=pred +v +k i
s:  (3, 4, lex([lavinia])):d -k
c:  (1, 2, r3(d, 1, 0)):vt -v   (0, 1, A):-k
c:  (1, 2, r3(d, 4, 0)):vt -v   (3, 4, A):-k
c:  (A, A, r3(vt, 2, 0)):+k =d pred   (1, 2, A):-v   (0, 1, A):-k
c:  (A, A, r3(vt, 2, 0)):+k =d pred   (1, 2, A):-v   (3, 4, A):-k
c:  (0, 1, v1(k, 1, s(0))):=d pred   (1, 2, A):-v
c:  (3, 4, v1(k, 4, s(0))):=d pred   (1, 2, A):-v
c:  (0, 1, r3(d, 1, s(0))):pred   (0, 1, A):-k   (1, 2, A):-v
c:  (0, 1, r3(d, 4, s(0))):pred   (3, 4, A):-k   (1, 2, A):-v
c:  (3, 4, r3(d, 1, s(0))):pred   (0, 1, A):-k   (1, 2, A):-v
c:  (3, 4, r3(d, 4, s(0))):pred   (3, 4, A):-k   (1, 2, A):-v
c:  (2, 4, r1(pred, 3)):+v +k i   (0, 1, A):-k   (1, 2, A):-v
c:  (2, 4, r1(pred, 3)):+v +k i   (3, 4, A):-k   (1, 2, A):-v
c:  (1, 4, v1(v, 2, s(0))):+k i   (0, 1, A):-k
c:  (1, 4, v1(v, 2, s(0))):+k i   (3, 4, A):-k
c:  (0, 4, v1(k, 1, 0)):i
c:  (0, 4, r1(i, 0)):+wh c
c:  (0, 4, r1(i, 0)):c

accepted as category c: titus praise '-s' lavinia
derivation:[1, 3, 4, 6, 8, 9].
more? ?
```

```
At the prompt:  more?
                  <cr>  to finish
                   ;    for more results
                   t    display derivation, x-bar,and bare trees with tk
                   d    print derivation tree to tk and ltree.tex
                   b    print bare tree to tk and ltree.tex
                   x    print x-bar tree to tk and ltree.tex
                   p    pprint derivation tree to terminal and ltree.tex
                   q    pprint bare tree to terminal and ltree.tex
                   r    pprint x-bar tree to terminal and ltree.tex
                 or anything else       for this help
more? r
cP /[
    c' /[
        c /[
            [] /[]],
        iP /[
            dP(3) /[
                d' /[
                    d /[
                        [titus] /[]]]],
            i' /[
                vtP(2) /[
                    vt' /[
                        vt /[
                            [praise] /[]],
                        dP /[
                            t(1) /[]]]],
                i' /[
                    i /[
                        [-s] /[]],
                    predP /[
                        dP /[
                            t(3) /[]],
                        pred' /[
                            dP(1) /[
                                d' /[
                                    d /[
                                        [lavinia] /[]]]],
                            pred' /[
                                pred /[
                                    [] /[]],
                                vtP /[
                                    t(2) /[]]]]]]]]]]
more? x
more? q
< /[
    []:[c] /[],
    > /[
        [titus]:[] /[],
        > /[
            < /[
                [praise]:[] /[],
                 /[]],
            < /[
                [-s]:[] /[],
                > /[
                    /[],
                   > /[
                       [lavinia]:[] /[],
                       < /[
                           []:[] /[],
                            /[]]]]]]]]
more? b
more?

Yes
4 ?-
```

The MG derivation trees displayed in these notes were all generated by `mgp`.

Complexity: (Harkema, 2000) shows that the time complexity of this parsing method is bounded by $\mathcal{O}(n^{4m+4})$ where m is the number of different movement triggers.

Morawietz has also shown how this kind of parsing strategy can be implemented as a kind of constraint propagation (Morawietz, 2001).

9.2.2 Some CKY-like derivations

(26) The grammar we formulated for the CKY parsing strategy is more succinct than the tree-based formulations, and most explicitly reveals the insight of Pollard (1984) and Michaelis (1998) that trees play the role of dividing the input string into a (finite!) number of related parts.

Furthermore, since the latter formulation of the grammar derives tuples of categorized strings, it becomes feasible to present derivations trees again, since the structures at the derived nodes are tuples of strings rather than trees.

In fact, the latter representation is succinct enough that it is feasible to display many complete derivations, every step in fully explicit form.

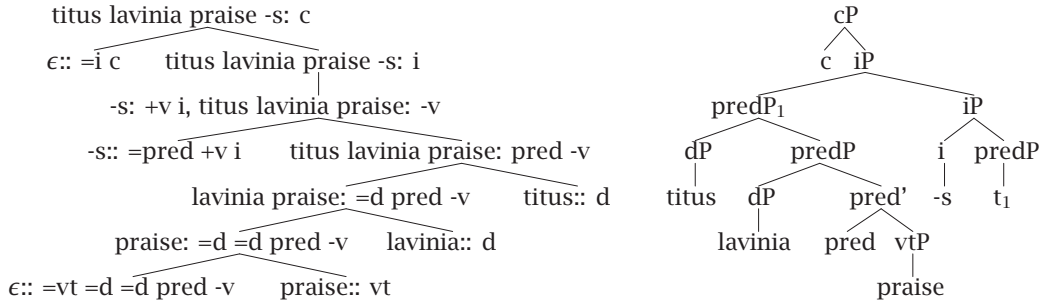
With a little practice, it is not difficult to translate back and forth betweeen these explicit derivation trees and the corresponding X-bar derivations that are more familiar from mainstream linguistic theory. We briefly survey some examples.

(27) **SOVI: Naive Tamil**. We first consider some more very simple examples inspired by Mahajan (2000). The order Subject-Object-Verb-Inflection is defined by the following grammar:

```
lavinia::d                  titus::d
praise::vt                  criticize::vt
laugh::v                    cry::v
ϵ::=i c                     -s::=pred +v i
ϵ::=vt =d =d pred -v        ϵ::=v =d pred -v
```

Notice that the `-s` in the string component of an expression signals that this is an affix, while the `-v` in the feature sequence of an expression signals that this item must move to a `+v` licensing position.

With this lexicon, we have the following derivation of the string *titus lavinia praise -s* $\in S_c(NT)$:



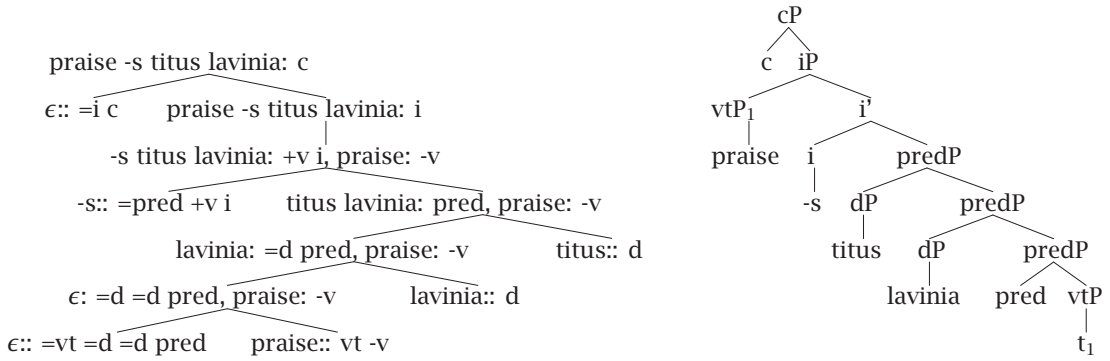
These conventional structures show some aspects of the history of the derivations, something which can be useful for linguists even though it is not necessary for the calculation of derived expressions.

(28) **VISO: Naive Zapotec**

An VSO language like Zapotec can be obtained by letting the verb select its object and then its subject, and then moving the just the lowest part of the SOV complex move to the "specifier" of I(nflection). The following 10 lexical items provide a naive grammar of this kind:

```
lavinia::d                  titus::d
praise::vt -v               laugh::v -v
ϵ::=i c                     -s::=pred +v i
ϵ::=vt =d =d pred           ϵ::=v =d pred
```

With this lexicon, we have the following derivation of the string *praise -s titus lavinia* $\in S_c(NT)$:



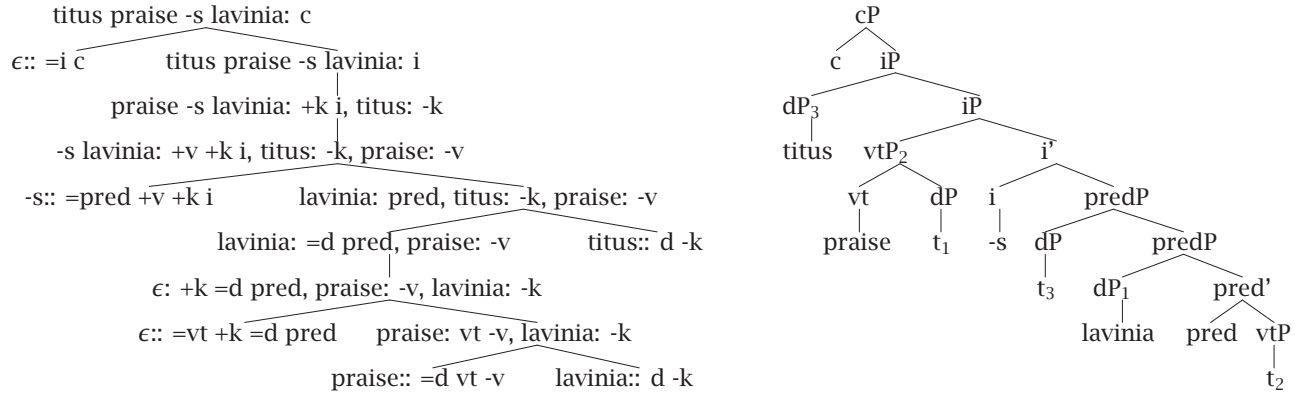
(29) **SVIO: naive English**

The following 16 lexical items provide a slightly more elaborate fragment of an English-like SVIO language:

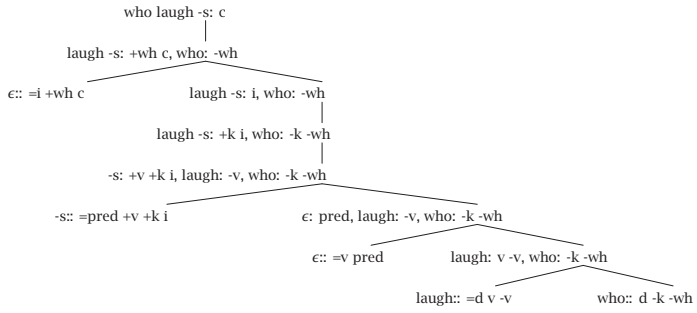
lavinia:: d -k	titus:: d -k	who:: d -k -wh	
some:: =n d -k	every:: =n d -k	noble:: n	kinsman:: n
laugh:: =d v -v	cry:: =d v -v		
praise:: =d vt -v	criticize:: =d vt -v		
-s:: =pred +v +k i	ϵ:: =vt +k =d pred	ϵ:: =v pred	
ϵ:: =i c	ϵ:: =i +wh c		

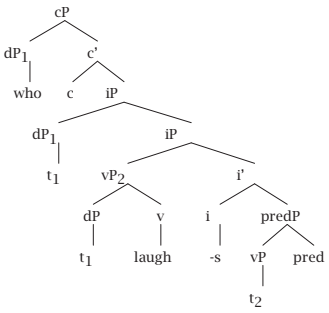
Notice that an SVIO language must break up the underlying SVO complex, so that the head of inflection can appear postverbally. This may make the SVIO order more complex to derive than the SOVI and VISO orders, as in our previous examples.

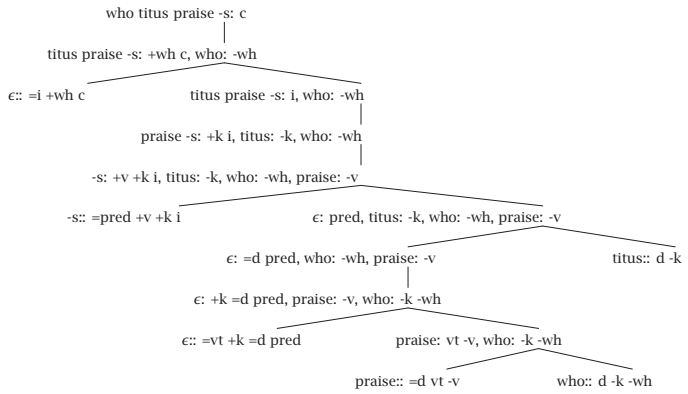
With this lexicon, we have the following derivation of the string *titus praise -s lavinia* $\in S_c(NE)$:

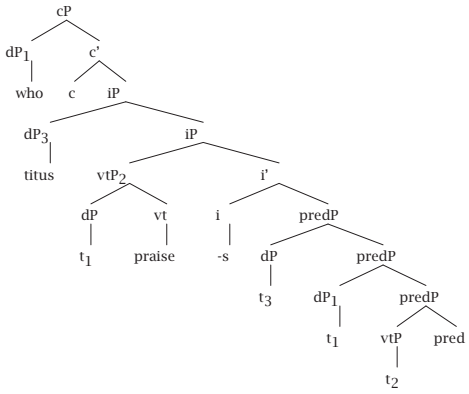


These lexical items allow wh-phrases to be fronted from their "underlying" positions, so we can derive *who laugh -s* and (since "do-support" is left out of the grammar for simplicity) *who titus praise -s*:


who laugh -s: c
laugh -s: +wh c, who: -wh
ϵ:: =i +wh c
laugh -s: i, who: -wh
laugh -s: +k i, who: -k -wh
-s: +v +k i, laugh: -v, who: -k -wh
-s:: =pred +v +k i
ϵ: pred, laugh: -v, who: -k -wh
ϵ:: =v pred
laugh: v -v, who: -k -wh
laugh:: =d v -v
who:: d -k -wh


cP
dP1
c'
who
c
iP
dP1
iP
t1
vP2
i'
dP
v
i
predP
t1
laugh
-s
vP
pred
t2


who titus praise -s: c
titus praise -s: +wh c, who: -wh
ϵ:: =i +wh c
titus praise -s: i, who: -wh
praise -s: +k i, titus: -k, who: -wh
-s: +v +k i, titus: -k, who: -wh, praise: -v
-s:: =pred +v +k i
ϵ: pred, titus: -k, who: -wh, praise: -v
ϵ: =d pred, who: -wh, praise: -v
titus:: d -k
ϵ: +k =d pred, praise: -v, who: -k -wh
ϵ:: =vt +k =d pred
praise: vt -v, who: -k -wh
praise:: =d vt -v
who:: d -k -wh


cP
dP1
c'
who
c
iP
dP3
iP
titus
vtP2
i'
dP
vt
i
predP
t1
praise
-s
dP
predP
t3
dP1
predP
t1
vtP
pred
t2

(30) **Relative clauses according to Kayne.**

As noted in §9.1.2, we can capture order preferences among adjectives by assuming that they are heads selecting nominal phrases rather than left adjuncts of nominal phrases. The idea that some such adjustment is needed proceeds from a long line of interesting work including Barss and Lasnik (1986), Valois (1991), Sportiche (1994), Kayne (1994), Pesetsky (1995), Cinque (1999), Dimitrova-Vulchanova and Giusti (1998).

Since right adjuncts are not generated by our grammar, Kayne (1994, §8) proposes that the raising analyses of relative clauses look most promising in this framework, in which the "head" of the relative is raised out of the clause. This kind of analysis was independently proposed much earlier because of an apparent similarity between relative clauses and certain kinds of focus constructions (Schacter, 1985; Vergnaud, 1982; Åfarli, 1994; Bhatt, 1999):

a. i. This is the cat that chased the rat

ii. It's the cat that chased the rat

b. i. * That's the rat that this is the cat that chased

ii. * It's that rat that this is the cat that chased

c. i. Sun gaya wa yaron (Hausa)
PERF.3PL tell IOBJ child
'they told the child'

ii. yaron da suka gaya wa
child REL 3PL tell IOBJ
'the child that they told'

iii. yaron ne suka gaya wa
child FOCUS 3PL tell IOBJ
'it's the child that they told'

d. i. nag-dala ang babayi sang bata (Ilonggo)
AGT-bring TOPIC woman OBJ child
'the woman brought a child'

ii. babanyi nga nag-dala sang bata
woman REL AGT-bring OBJ child
'a woman that brought a child'

iii. ang babanyi nga nag-dala sang bata
TOPIC woman REL AGT-bring OBJ child
'it's the woman that brought a child'

The suggestion is that in all of these constructions, the focused noun raises to a prominent position in the clause. In the relative clauses, the clause with the raised noun is the sister of the determiner; in the clefts, the clause is the sister of the copula.

We could assume that these focused elements land in separate focus projections, but for the moment let's assume that they get pulled up to the CP.

Kayne assumes that the relative pronoun also originates in the same projection as the promoted head, so we get analyses with the structure:

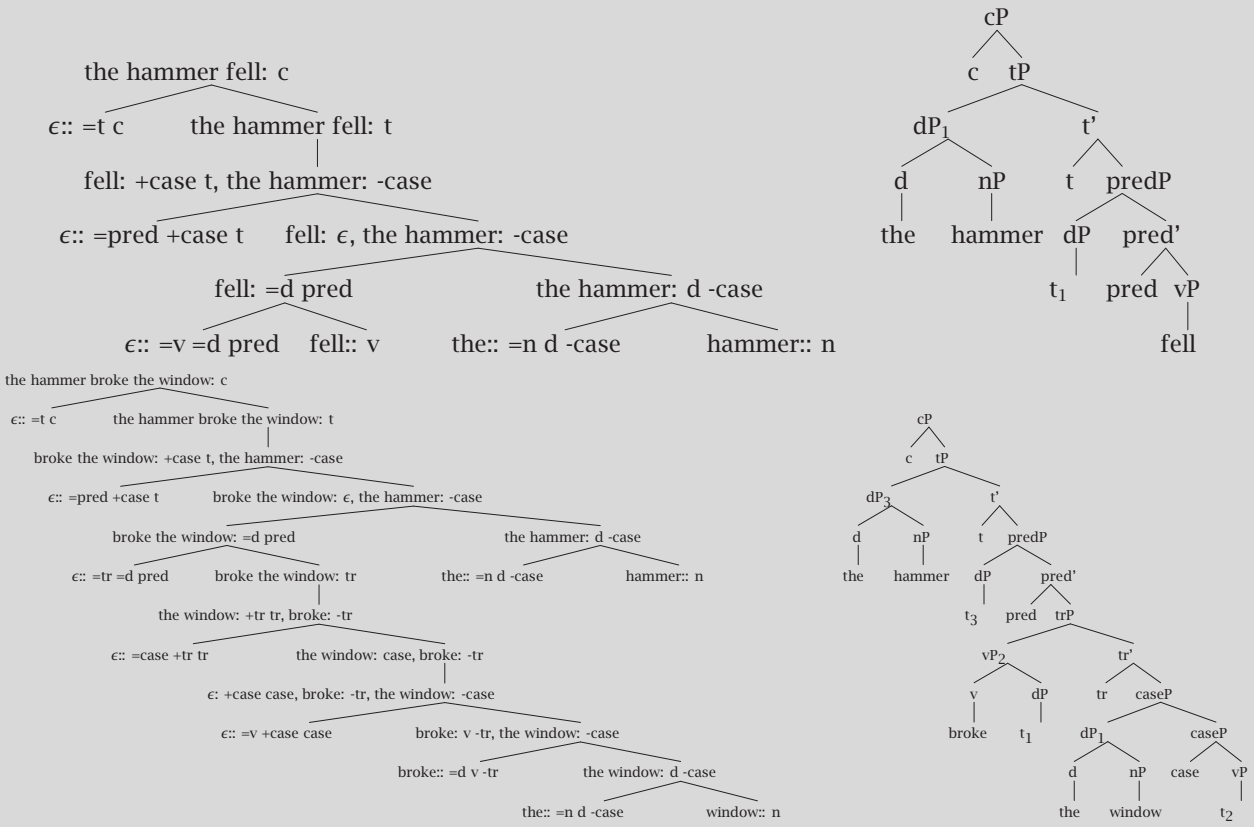
a. The hammer$_i$ [which t_i]$_j$ [$t_j$broke t_h]$_k$ [the window]$_h$ t_k

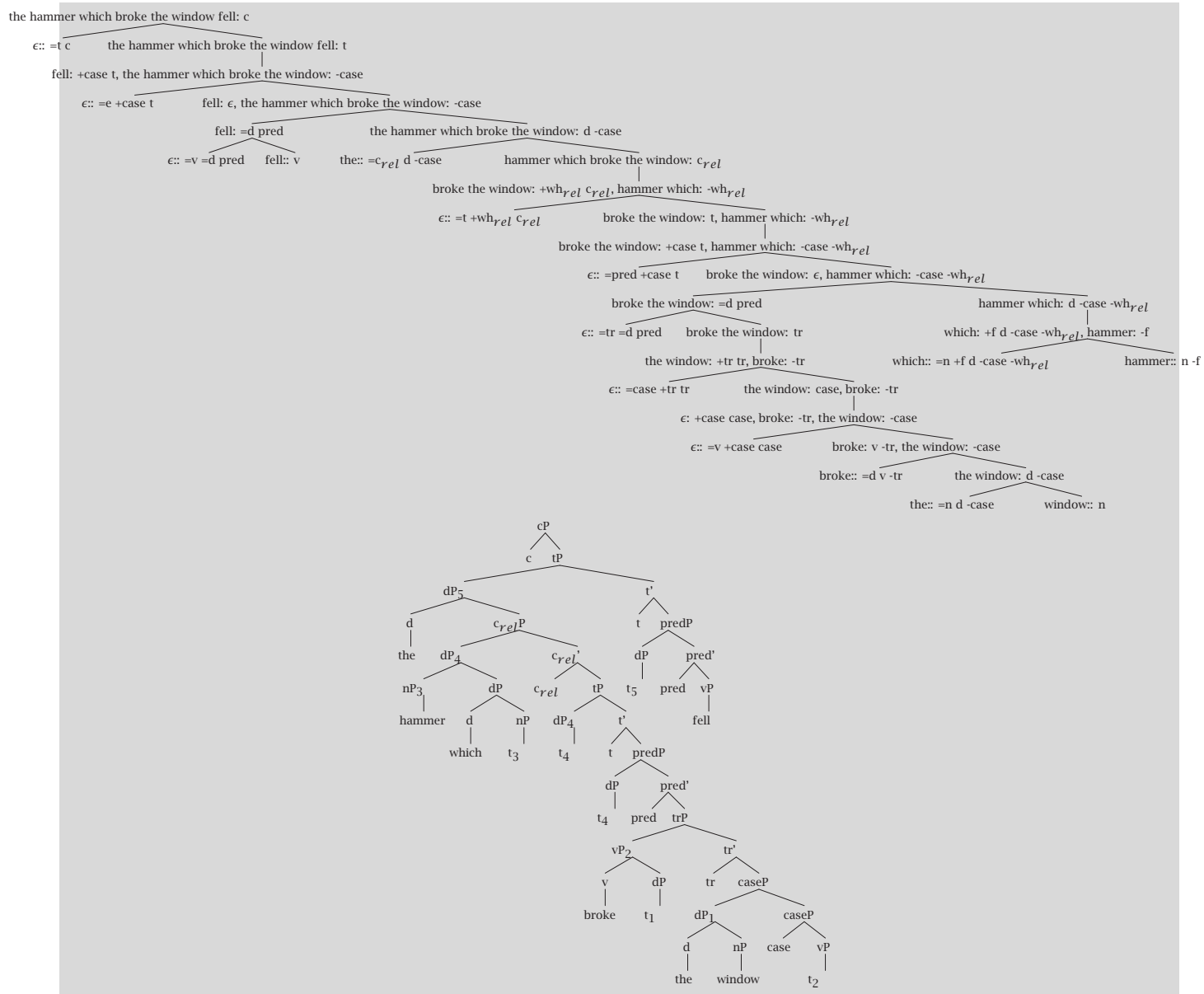
b. The window$_i$ [which t_i]$_j$ [the hammer]$_h$ [$t_h$broke t_j]$_k$ t_k

We can obtain this kind of analysis by allowing noun heads of relative clauses to be focused, entering the derivation with some kind of focus feature -f.

=t c	=t +wh$_{rel}$ c$_{rel}$
=pred +case t	
=n d -case the	=n +f d -case -wh$_{rel}$ which
	=c$_{rel}$ d -case the
n hammer	n -f hammer
n window	n -f window
=v +case case	
=tr =d pred	
=case +tr tr	
=d v -tr broke	

NB: focused lexical items in the second column.





(31) Buell (2000) shows that Kayne's analysis of relative clauses does not extend easily to Swahili. In Swahili, it is common to separate the NP head of the relative clause from the relative pronoun –cho:

a. Hiki ni kitabu ni- li- cho- ki- soma
7.this cop 7.book 1s.subj- past- 7.o.relpro 7.obj- read
'This is the book that I read'

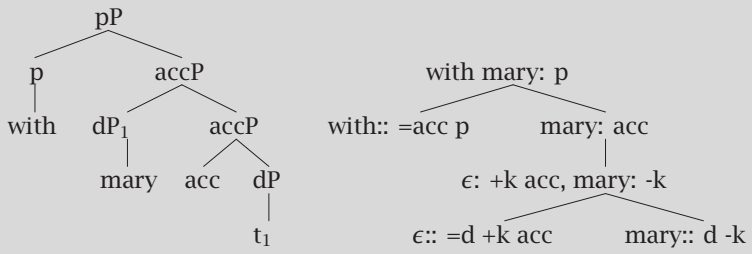
b. Hiki ni kitabu ni- ki- soma -cho
7.this cop 7.book 1s.subj- 7.obj- read -7.o.relpro read
'This is the book that I read'

(32) **Various ideas about prepositional phrases.**
Lots of ideas have about PPs have been proposed. Here we consider just a few, sticking to proposals that do not require head movement.

a. **PPs and case assignment**
The objects of prepositions are case marked. This can be done either by

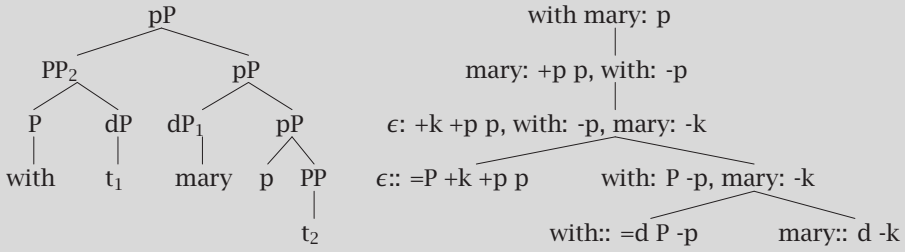
i. having the prepositions select phrases in which the objects already have case,

```
=acc p with      =d +k acc      d -k mary
```



ii. or by selecting the bare object, and then "rolling" up the object and then the preposition.

```
=d P -p with      =P +k +p p      d mary
```



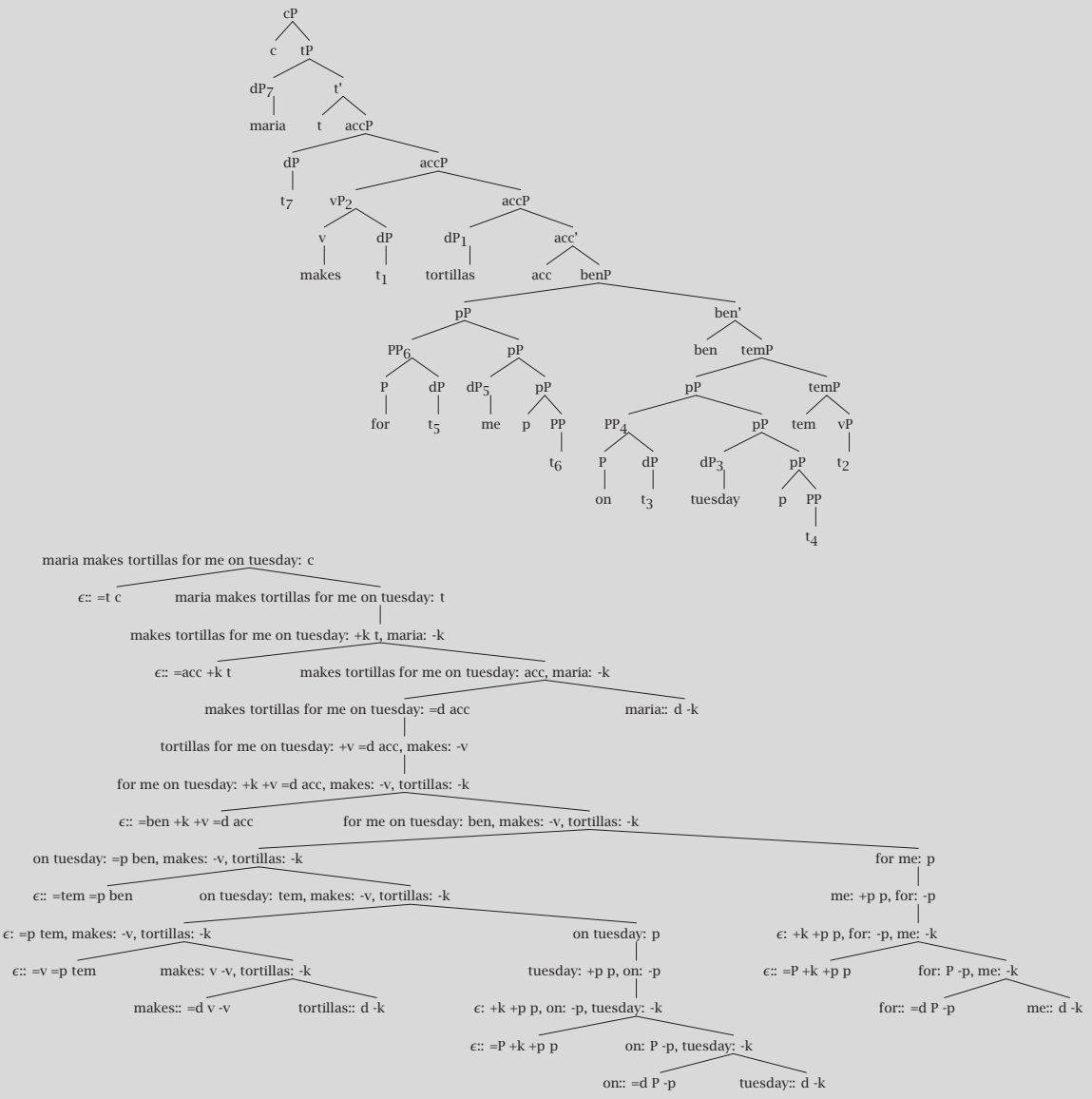
The former analysis is closer to "traditional" ideas; the latter sets the stage for developing an analogy between prepositional and verbal structures (Koopman, 1994), and allows the selection relation between a head and object to be immediate, with all structural marking done afterwards.

b. **PPs in a sequence of specifiers**
Recall our simple grammar for *maria makes tortillas* from page 171:

```
=t c                =t +wh c
=acc +k t
=v +k +v =d acc
=d v -v makes
d -k maria          d -k tortillas   d -k -wh what
```

One simple (too simple!) way to extend this for the PPs in a sentence like
maria makes tortillas for me on Tuesday
is to just provide a sequence of positions in the VP, so that adjuncts are sort of like "deep complements" (Larson, 1988) of the VP. Since the VP here is "rolled up", let's use the second analysis of PPs from (32a), and add the PPs above vP but below accP like this

```
=ben +k +v =d acc          =tem =p ben          =v =p tem
```

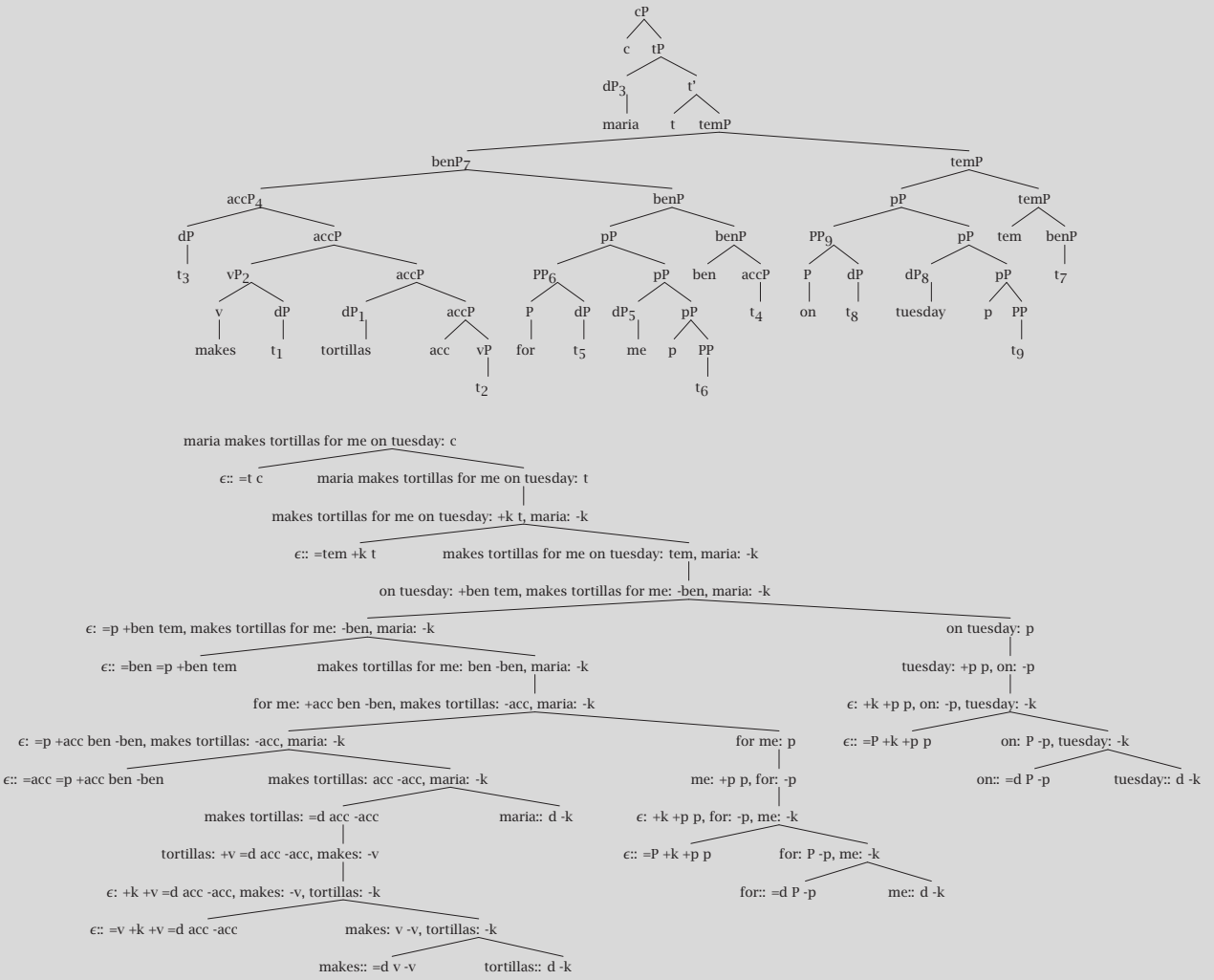


This is not very appealing, since the temporal adverb is so low in the structure. It is more natural to think that the temporal adverb should be attached high, outside of the rest of the verbal structure, maybe even at the tense projection. Something like this can be done as follows.

c. **PPs with "rolling up" of lower structure**

We get more natural selection relations following the spirit of (Koopman and Szabolcsi, 2000b) by recursively rolling up our structures:

```
=tem +k t     =ben =p +ben tem     =acc =p +acc ben -ben     =v +k +v =d acc -acc
```



d. **PPs in a "cascade"**

Yet another idea is to reject the idea that the complement of a preposition is generally its object, in favor of the view that the complement is another PP with the object in its specifier (Pesetsky, 1995). We could also capture this idea, but we leave it as an exercise.

Exercises:

You can download the MG CKY-like parser `mgp.pl` and tree collector `lp.pl` from the class web page.

1. Extend the "naive English" grammar on page 189 to fit the following data (i.e. your grammar should provide reasonable derivations for the first five strings, and it should not accept the last five, starred strings):
   i. Titus severely criticize s Lavinia
   ii. Titus never criticize s Lavinia
   iii. Titus probably criticize s Lavinia
   iv. Titus probably never criticize s Lavinia
   v. Titus probably never severely criticize s Lavinia
   vi. * Titus severely probably criticize s Lavinia
   vii. * Titus never probably criticize s Lavinia
   viii. * Titus severely never probably criticize s Lavinia
   ix. * Titus severely probably never criticize s Lavinia
   x. * Titus probably severely never criticize s Lavinia

   a. Type your grammar in our prolog format, and run it with `mgp.pl`, and turn in a session log showing tests of the previous 10 strings.
   b. Append to your session log a brief assessment of your grammar from a linguist's perspective (just a few sentences).

2. (Obenauer, 1983) observed that some Adverbs in French like *beaucoup* seem to block extraction while others like *attentivement* do not:
   a. i. [Combien de problèmes] sais-tu résoudre t?
      ii. Combien sais-tu résoudre t de problèmes?
   b. i. [Combien de livres] a-t-il beaucoup consultés t?
      ii. * [Combien] a-t-il beaucoup consultés t de livres?
   c. i. [Combien de livres] a-t-il attentivement consultés t?
      ii. * [Combien] a-t-il attentivement consultés t de livres?

   Design an MG which gets the following similar pattern:
   i. he consults many books
   ii. how many books he well consults?
   iii. how many books he attentively consults?
   iv. * how many he well consults books?
   v. how many he attentively consults books?

   Present the grammar, a derivation of e, and an explanation of why d cannot be derived.

3. Imagine that you thought grammar in (10) on page 176 was on the right track. Provide the most natural extension you can that gets passive forms like *the pie be -s eat -en*. Present the grammar and a hand worked derivation.

   If you are ambitious, try this harder problem: modify the grammar in (10) to get yes-no questions.

10 Towards standard transformational grammar

In the previous section the grammars had only:

- selection
- phrasal movement

It is surprisingly easy to modify the grammar to add a couple of the other common structure building options:

- head movement to the left or the right
- affix hopping to the left or the right
- adjunction on the left or the right

Many linguists doubt that all these mechanisms are needed, but the various proposals for unifying them are controversial. Fortunately for us, it turns out that all of them can be handled as small variations on the devices of the "minimalist" framework. Consequently, we will be able to get quite close to the processing problems posed by grammars of the sort given by introductory texts on transformational grammar!

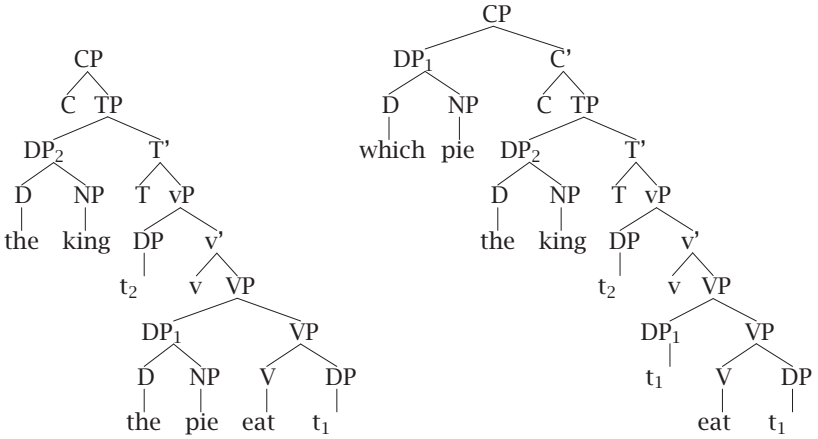
10.1 Review: phrasal movement

A simple approach to wh-movement allows us to derive simple sentences and wh-questions like the following, in an artificial Subject-Object-Verb language with no verbal inflections:

(1) the king the pie eat

(2) which pie the king eat

Linguists have proposed that not only is the question formed by moving the wh determiner phrase (DP) [which pie] from object position to the front, but in all clauses the pronounced DPs move to case positions, where transitive verbs assign case to their objects ("Burzio's generalization"). So then the clauses above get depictions rather like this, indicating movements by leaving coindexed "traces" (t) behind:



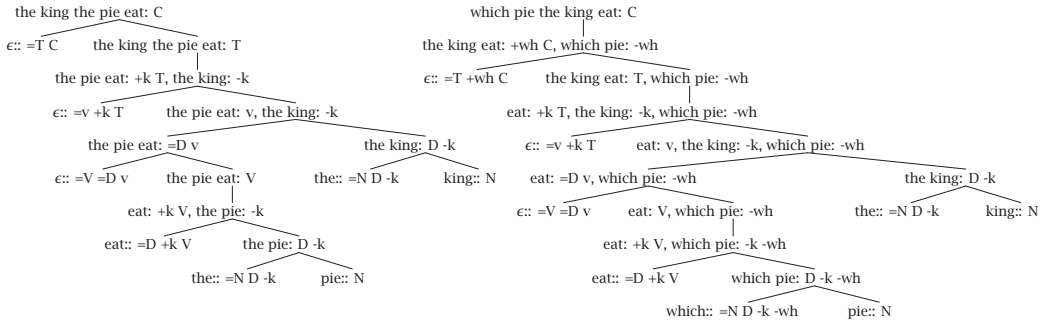
As indicated by coindexing, in the tree on the left, there are two movements, while the tree on the right has three movements because [which pie] moves twice: once to a case position, and then to the front, wh-question position. The sequences of coindexed constituents are sometimes called "chains."

Notice that if we could move *eat* from its V position in these trees to the v position, we would have the English word order. In fact, we will do this, but first let's recall how this non-English word order can be derived with the mechanisms we already have.

These expressions above can be defined by an MG with the following 10 lexical items (writing ϵ for the empty string, and using k for the abstract "case" feature):

ϵ:: =T C	ϵ:: =T +wh C
ϵ:: =v +k T	ϵ:: =V =D v
eat:: =D +k V	laugh:: V
the:: =N D -k	which:: =N D -k -wh
king:: N	pie:: N

With this grammar, we can derive strings of category C as follows, where in these trees the leaves are lexical items, a node with two daughters represents the result of *merge*, and a node with one daughter represents the result of a *move*.



Since *merge* is binary and *move* is unary, it is easy to see that the tree on the left has two movements, while the one on the right has three.

Let's elaborate this example just slightly, to introduce auxiliary verbs. We can capture many of the facts about English auxiliary verb cooccurrence relations with the mechanism of selection we have defined here. Consider for example the following sentences:

He might have been eating	He has been eating	He is eating
He eats	He has been eating	He has eaten
He might eat	He might be eating	He might have eaten

If we put the modal verbs in any other orders, the results are no good:

* He have might been eating * He might been eating * He is have ate * He has will eat

The regularities can be stated informally as follows:[44]

(3) English auxiliaries occur in the order MODAL HAVE BE. So there can be as many as 3, or as few as 0.

(4) A MODAL (when used as an auxiliary) is followed by a tenseless verb, [-TNS]

(5) HAVE (when used as an auxiliary) is followed by a past participle, [PASTPART]

(6) Be (when used as an auxiliary) is followed by a present participle, [PRESPART]

(7) The first verb after the subject is always the one showing agreement with the subject and a tense marking (if any), [+TNS]

[44] Many of these auxiliary verbs have other uses too, which will require other entries in the lexicon.

(1) He willed me his fortune. His mother contested the will. (WILL as main verb, or noun)

(2) They can this beer in Canada. The can ends up in California. (CAN as main verb, or noun)

(3) The might of a grizzly bear is nothing to sneeze at. (MIGHT as noun)

(4) I have hiking boots. (HAVE as main verb)

(5) I am a hiker. (BE as main verb)

We can enforce these requirements with selection features. For example, we can augment the previous grammar as follows:

ϵ:: =T C	ϵ:: =T +wh C		
-s:: =Modal +k T	-s:: =Have +k T	-s:: =Be +k T	-s:: =v +k T
will:: =Have Modal	will:: =Be Modal	will:: =v Modal	
have:: =Been Have	have:: =v_{en} Have		
be:: =v_{ing} Be	been:: =v_{ing} Been		
ϵ:: =V =D v	-en:: =V =D v_{en}	-ing:: =V =D v_{ing}	
eat:: =D +k V	laugh:: V		
the:: =N D -k	which:: =N D -k -wh		
king:: N	pie:: N		

Notice that if we could move any of these auxiliary verbs from their position in T to C, we would form yes-no questions:

He has been eating	He is eating	He has eaten	He will be eating
Has he been eating?	Is he eating?	Has he eaten?	Will he be eating?

And notice that when there is more than one auxiliary verb, only the <u>first</u> one can move:

He will have been eating
Will he have been eating?
* Have he will been eating?
* Been he will have eating?

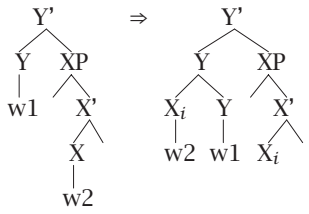
This observation and many other similar cases in other languages support the idea that head movement, if it exists, is very tightly constrained. One simple version of the idea is this one:

Head movement constraint (HMC): A head can only move to the head that selects it

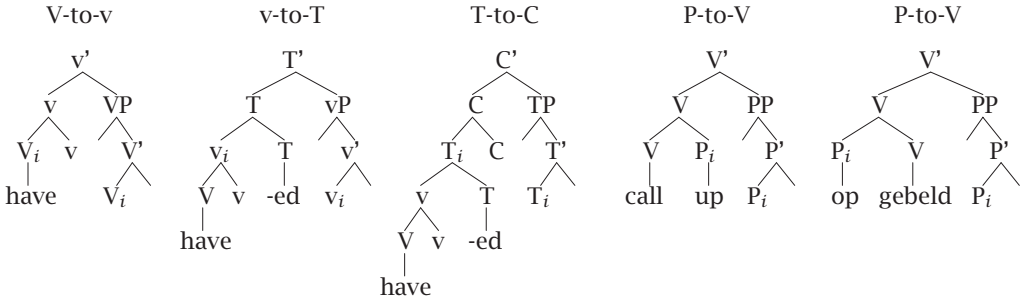
This motivates the following simple extension of the minimalist grammar framework.

10.2 Head movement

Many linguists believe that in addition to phrasal movement, there is "head movement", which moves not the whole phrase but just the "head". In the simplest, "canonical" examples, a head X of a phrase XP moves to adjoin to the left or right of the head Y that selects XP. Left-adjoining X to Y is often depicted this way:



For example, questions with inversion of subject and inflected verb may be formed by moving the T head to C (sometimes called T-to-C or I-to-C movement); verbs may get their inflections by V-to-T movement; particles may get associated with verbs by P-to-V movement; objects may incorporate into the verb with N-to-V movement, and there may also be V-to-v movement.



As indicated by these examples of v-to-T and T-to-C movement, heads can be complex. And notice that the P-to-V movement is right-adjoining in the English [call up] but left-adjoining in the Dutch [opgebeld] (Koopman 1993, 1994). Similarly (though not shown here) when a verb incorporates a noun, it is usually attached on the left, but sometimes on the right (Baker, 1996, 32).

The MGs defined above can be extended to allow these sorts of movements. Since they involve the configuration of selection, we regarded them as part of the *merge* operation (Stabler, 1997). Remembering the essential insight from Pollard and Michaelis, mentioned on the first page, the key thing is to keep the phonetic contents of any movable head in a separate component. A head X is not movable after its phrase XP has been merged, so we only need to distinguish the head components of phrases until they have been merged. So rather than expressions of the form:

$$\mathtt{s}_1 \cdot \mathtt{Features}_1, \mathtt{s}_2 \cdot \mathtt{Features}_2, \ldots, \mathtt{s}_k \cdot \mathtt{Features}_k,$$

we will use expressions in which the string part $\mathtt{s}_1$ of the first chain is split into three (possibly empty) pieces `s(pecifier), h(head), c(omplement)`:

$$\mathtt{s,h,c} \cdot \mathtt{Features}_1, \mathtt{s}_2 \cdot \mathtt{Features}_2, \ldots, \mathtt{s}_k \cdot \mathtt{Features}_k.$$

So **lexical chains** now have a triple of strings, but only the head can be non-empty: $LC = \epsilon, \Sigma^*, \epsilon :: F^*$. As before, a **lexicon** is a finite set of lexical chains.

Head movement will be triggered by a specialization of the selecting feature. The feature =>V will indicate that the head of the selected VP is to be adjoined on the left; and V<= will indicate that the head of the selected VP is to be adjoined on the right. The former set of features is thus extended by adding these two new functions on the base categories B: **right-incorporators** $R = \{f\texttt{<=} \mid f \in B\}$, and **left-incorporators** $L = \{\texttt{=>}f \mid f \in B\}$. So now the set of syntactic features $F = B \cup S \cup M \cup N \cup R \cup L$. The new work of placing heads properly is done by the $merge$ function, so the earlier functions r1 and r3 each break into 3 cases. Define $merge$ as the union of the following 7 functions:

$$\frac{\epsilon, s, \epsilon :: =f\,\gamma \qquad t_s, t_h, t_c \cdot f, \alpha_1, \ldots, \alpha_k}{\epsilon, s, t_s t_h t_c : \gamma, \alpha_1, \ldots, \alpha_k}\ \text{r1'}$$

$$\frac{\epsilon, s, \epsilon :: f\texttt{<=}\gamma \qquad t_s, t_h, t_c \cdot f, \alpha_1, \ldots, \alpha_k}{\epsilon, s t_h, t_s t_c : \gamma, \alpha_1, \ldots, \alpha_k}\ \text{r1right}$$

$$\frac{\epsilon, s, \epsilon :: \texttt{=>}f\,\gamma \qquad t_s, t_h, t_c \cdot f, \alpha_1, \ldots, \alpha_k}{\epsilon, t_h s, t_s t_c : \gamma, \alpha_1, \ldots, \alpha_k}\ \text{r1left}$$

$$\frac{s_s, s_h, s_c : =f\,\gamma, \alpha_1, \ldots, \alpha_k \qquad t_s, t_h, t_c \cdot f, \iota_1, \ldots, \iota_l}{t_s t_h t_c s_s, s_h, s_c : \gamma, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l}\ \text{r2'}$$

$$\frac{s_s, s_h, s_c \cdot =f\,\gamma, \alpha_1, \ldots, \alpha_k \qquad t_s, t_h, t_c \cdot f\,\delta, \iota_1, \ldots, \iota_l}{s_s, s_h, s_c : \gamma, t_s t_h t_c : \delta, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l}\ \text{r3'}$$

$$\frac{s_s, s_h, s_c :: f\texttt{<=}\gamma, \alpha_1, \ldots, \alpha_k \qquad t_s, t_h, t_c \cdot f\,\delta, \iota_1, \ldots, \iota_l}{s_s, s_h t_h, s_c : \gamma, t_s t_c : \delta, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l}\ \text{r3right}$$

$$\frac{s_s, s_h, s_c :: \texttt{=>}f\,\gamma, \alpha_1, \ldots, \alpha_k \qquad t_s, t_h, t_c \cdot f\,\delta, \iota_1, \ldots, \iota_l}{s_s, t_h s_h, s_c : \gamma, t_s t_c : \delta, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l}\ \text{r3left}$$

And $move$ changes only trivially. It is the union of the following functions:

$$\frac{s_s, s_h, s_c : +f\,\gamma, \alpha_1, \ldots, \alpha_{i-1}, t : -f, \alpha_{i+1}, \ldots, \alpha_k}{t s_s, s_h, s_c : \gamma, \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k}\ \text{m1'}$$

$$\frac{s_s, s_h, s_c : +f\,\gamma, \alpha_1, \ldots, \alpha_{i-1}, t : -f\,\delta, \alpha_{i+1}, \ldots, \alpha_k}{s_s, s_h, s_c : \gamma, \alpha_1, \ldots, \alpha_{i-1}, t : \delta, \alpha_{i+1}, \ldots, \alpha_k}\ \text{m2'}$$

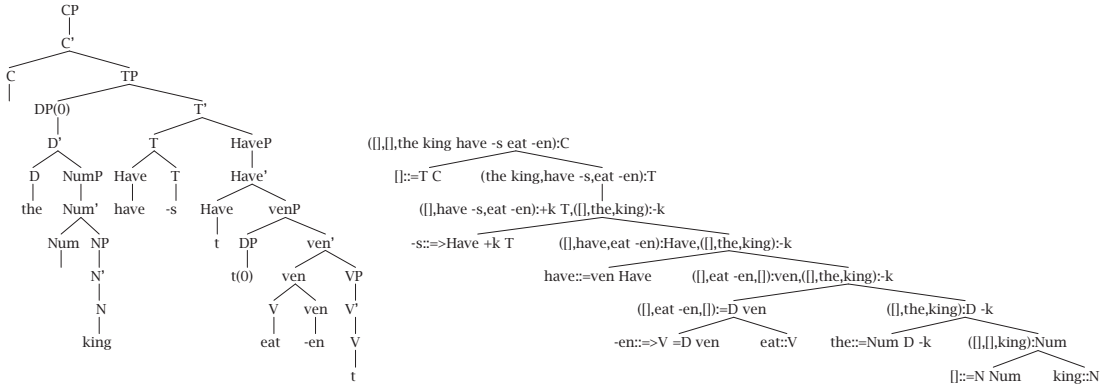
As before, for any grammar $G = Lex$, the language $L(G)$ is the closure of Lex under the fixed set of structure building functions $\mathcal{F} = \{merge, move\}$. And for any $\mathtt{f} \in B$, the strings of category $\mathtt{f}$, $S_{\mathtt{f}}(G) = \{s_s s_h s_c \mid s_s, s_h, s_c \cdot \mathtt{f} \in L(G) \text{ for some } \cdot \in T\}$.

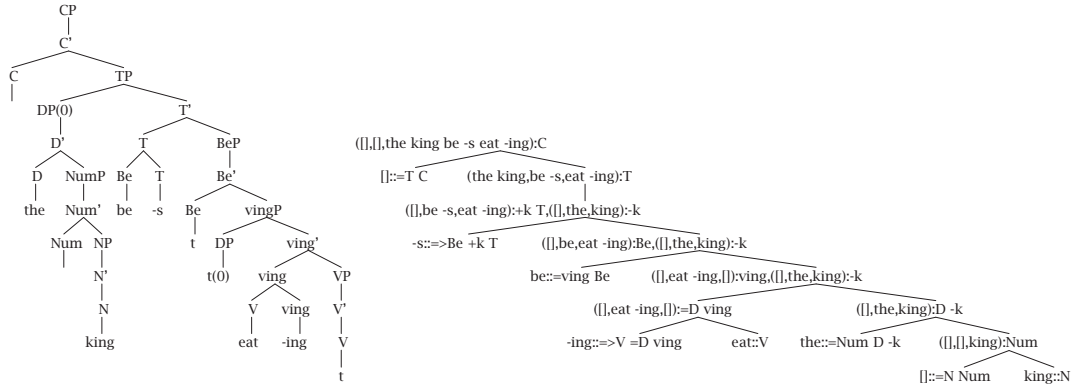
10.2.1 Subject-auxiliary inversion in English

Introductions to transformational syntax like Koopman, Sportiche, and Stabler (2002) and (Fromkin, 2000, §5) often present a simplified account of English auxiliaries and question formation that can now be be presented with a lexicon like the following (writing $\mathtt{s}_h\mathtt{::Fs}$ for each $(\mathtt{s}_s,\mathtt{s}_h,\mathtt{s}_c\mathtt{::Fs}) \in Lex$, since $\mathtt{s}_s$ and $\mathtt{s}_c$ are always empty in the lexicon):[45]

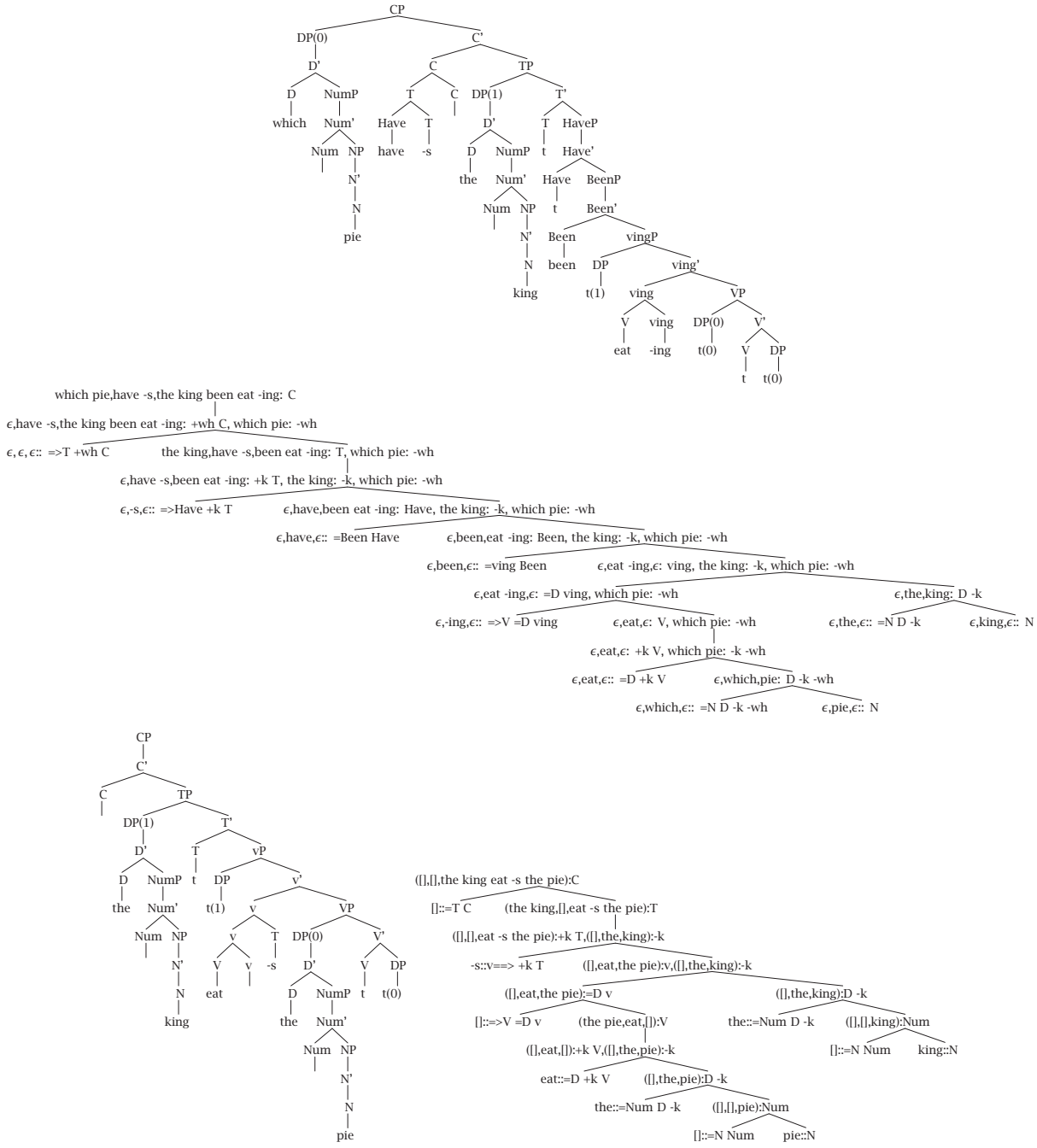
ϵ:: =T C	ϵ:: =>T C	ϵ:: =>T +wh C	
-s:: =>Modal +k T	-s:: =>Have +k T	-s:: =>Be +k T	-s:: =v +k T
will:: =Have Modal	will:: =Be Modal	will:: =v Modal	
have:: =Been Have	have:: =v_{en} Have		
be:: =v_{ing} Be	been:: =v_{ing} Been		
ϵ:: =>V =D v	-en:: =>V =D v_{en}	-ing:: =>V =D v_{ing}	
eat:: =D +k V	laugh:: V		
the:: =N D -k	which:: =N D -k -wh		
king:: N	pie:: N		

With this grammar we have derivations like the following





[45] I follow the linguistic convention of punctuating a string like -s to indicate that it is an affix. This dash that occurs next to a string should not be confused with the dash that occurs next to syntactic features like -wh.



The behavior of this grammar is English-like on a range of constructions:

(8) will -s the king laugh

(9) the king be -s laugh -ing

(10) which king have -s eat -en the pie

(11) the king will -s have been eat -ing the pie

We also derive

(12) -s the king laugh

which will be discussed in §17. This string "triggers do-support."

10.2.2 Affix hopping

The grammar of §10.2.1 does not derive the simple tensed clause: *the king eat -s the pie*. The problem is that if we simply allow the verb *eat* to pick up this inflection by head movement to T, as the auxiliary verbs do, then we will mistakenly also derive **eat -s the king the pie*. Also, assuming that *will* fills *T*, there are VP modifiers that can follow *T*

He will completely solve the problem.

So if the verb moves to the T affix *-s*, we would expect to find it before such a modifier, which is not what we find:

He completely solve -s the problem.
* He solve -s completely the problem.

Since Chomsky (1957), one common proposal about this is that when there is no auxiliary verb, the inflection can lower to the main verb. This lowering is sometimes called "affix hopping." In the present context, it is interesting to notice that once the head of unmerged phrases is distinguished for head movement, no further components are required for affix hopping.

We can formalize this idea in our grammars as follows. We introduce two new kinds of features $<\!\!=\!\!f$ and $f\!\!=\!\!>$ (for any $f \in B$), and we add the following additional cases to definition of $merge$:

$$\frac{\epsilon, s, \epsilon :: f\!\!=\!\!>\!\!\gamma \qquad t_s, t_h, t_c \cdot f, \alpha_1, \ldots, \alpha_k}{\epsilon, \epsilon, t_s t_h s t_c : \gamma, \alpha_1, \ldots, \alpha_k}\ \text{r1hopright}$$

$$\frac{\epsilon, s, \epsilon :: <\!\!=\!\!f\gamma \qquad t_s, t_h, t_c \cdot f, \alpha_1, \ldots, \alpha_k}{\epsilon, \epsilon, t_s s t_h t_c : \gamma, \alpha_1, \ldots, \alpha_k}\ \text{r1hopleft}$$

$$\frac{\epsilon, s, \epsilon :: f\!\!=\!\!>\!\!\gamma, \alpha_1, \ldots, \alpha_k \qquad t_s, t_h, t_c \cdot f\delta, \iota_1, \ldots, \iota_l}{\epsilon, \epsilon, \epsilon : \gamma, t_s t_h s t_c : \delta, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l}\ \text{r3hopright}$$

$$\frac{\epsilon, s, \epsilon :: <\!\!=\!\!f\gamma, \alpha_1, \ldots, \alpha_k \qquad t_s, t_h, t_c \cdot f\delta, \iota_1, \ldots, \iota_l}{\epsilon, \epsilon, \epsilon : \gamma, t_s s t_h t_c : \delta, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l}\ \text{r3hopleft}$$

This formulation of affix-hopping as a sort of string-inverse of head movement has the consequence that an affix can only "hop" to the head of a selected phrase, not to the head of the head selected by a selected phrase. That is, affix hopping can only take place in the configuration of selection.[46] It is now a simple matter to obtain a grammar G2 that gets simple inflected clauses.

We can elaborate grammar G1 by adding a single lexical item:

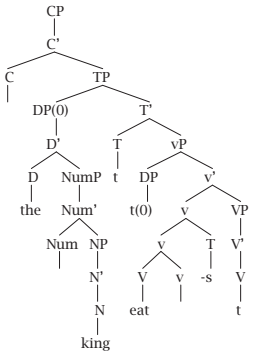
```
-s::  v=> +k T
```

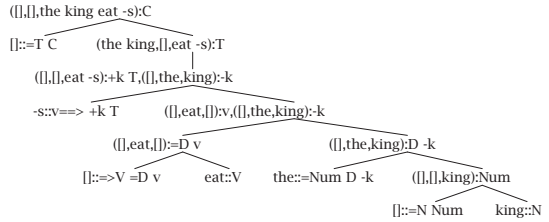
It is left as an exercise for the reader to verify that the set of strings of category C now allows main verbs to be inflected but not fronted, as desired:

(13) the king eat -s the pie

(14) *eat -s the king the pie

[46](Sportiche, 1998b, 382) points out that the proposal in (Chomsky, 1993) for avoiding affix hopping also has the consequence that affixes on main verbs in English can only occur in the configuration where head movement would also have been possible.





This kind of account of English clause structure commonly adds one more ingredient: do-support. Introductory texts sometimes propose that *do* can be attached to any stranded affix, perhaps by a process that is not part of the syntax proper. We accordingly take it up in the next section.

A note on the implementation.

Although the basic idea behind this treatment of head movement is very simple, it is now a bigger job to take care of all the details in parsing. There are many more special cases of *merge* and *move*. Prolog does not like all the operators we are using to indicate the different kinds of selection we have, so unfortunately we need a slightly different notation there. The whole collection so far is this:

feature	meaning	prolog notation
x	category x	x
=x	select and merge with a phrase with category x (first selection attached as comp, on right, second as spec, on left)	=x
+x	move a phrase with feature -x to spec	+x
–x	move to the spec of a category with a +x feature	–x
=>x	select phrase with cat x and adjoin its head on the left of yours	=>x
x<=	select phrase with cat x and adjoin its head on the right of yours	x<=
<=x	select phrase with cat x and hop your head to the left of the selected head	<==x
x=>	select phrase with cat x and hop your head to the right of the selected head	x==>

Exercise.

It will be good to do an exercise just to help it all sink in. (To do this simple exercise, you should first read the notes above on both head movement and affix hopping, and download the parser files from the website.)

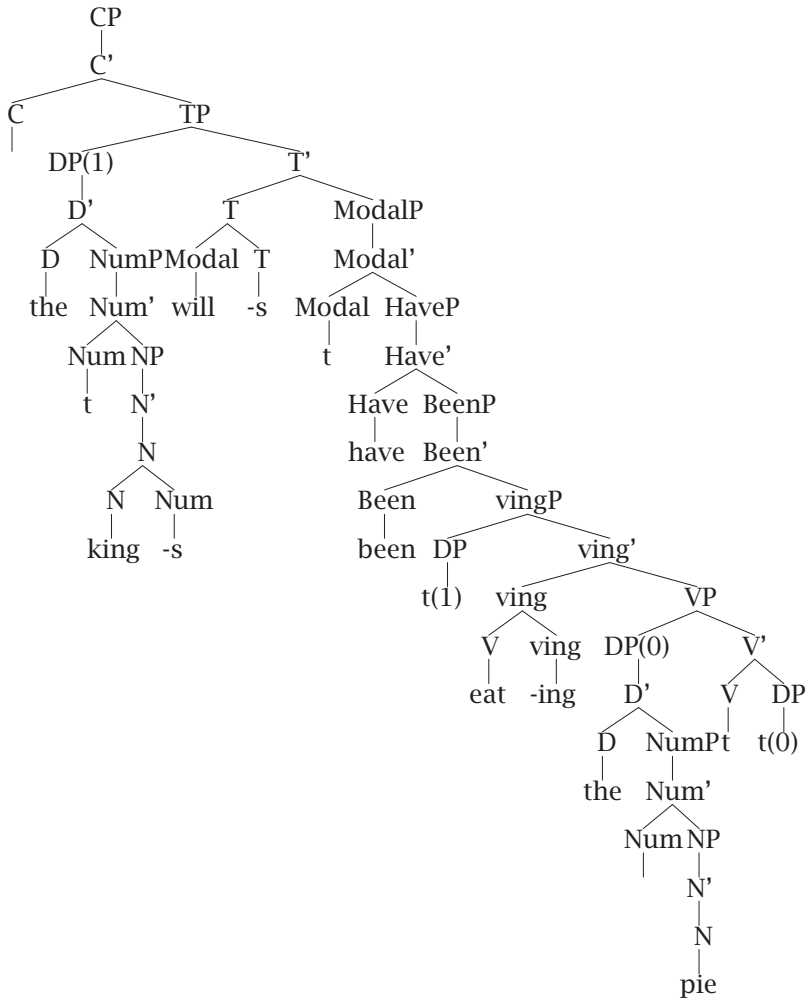
The reasoning that supports the idea that English tense affixes "hop" down to the verb also supports the idea that plural marking "hops" down to the noun. That is, we see that English allows modifiers before a noun:

the foolish king will eat the cake

It is natural to assume that *foolish* attaches as a modifier to the NP *king*. Then what happens when there is an overt indication of number, the plural marked by the English affix *-s*? If the noun moved up to the number position, then we might expect the noun with the number marking to precede the modifier, but that is not what we find:

the two foolish kings will eat the cake
* the two kings foolish will eat the cake

For this reason, linguists have sometimes proposed an analysis that they depict with a tree structure that looks something like this:



(We will see how to add adjectives like *foolish* later, but notice in this tree that (i) D selects Num, (ii) Num selects N, (iii) the unpronounced singular number does not move at all, but (iv) the plural suffix *-s* hops down onto the selected N.)

Modify *gh1.pl* to get this tree, and turn in the result.

10.3 Verb classes and other basics

We have enough machinery in place to handle quite a broad range of syntactic structures. It is worth a brief digression to see how some of the basics get treated in this framework, and this will provide some valuable practice for later.

We have already seen how simple transitive and intransitive verbs get represented in the simplified grammar above. Consider the **transitive verb**

praise::=D +k V

Here we see that it selects a DP and then moves it to assign (accusative) case, forming a VP. Then this VP is selected by v and the head V is left adjoined to the head v, and then the subject (the "external argument") of the verb is selected. In contrast, an **intransitive verb** has a lexical entry like this:

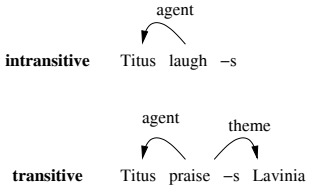
laugh::V

This verb selects no object and assigns no case, but it combines with v to get its subject in the usual way.

Notice that some verbs like *eat* can occur in both transitive and intransitive forms, so verbs like this have two lexical entries:

eat::V eat::=D +k V

Considering what each V and its associated v selects, we can see that they are the semantic arguments. So the familiar semantic relations are being mirrored by selection steps in the derivation:



Throughout this section, we will aim to have derivations that mirror semantic relations in this way.

10.3.1 CP-selecting verbs and nouns

It is easy to add verbs that select categories other than DP. For example, some verbs select full clauses as their complements. It is commonly observed that matrix clauses have an empty complementizer while embedded clauses can begin with *that*, and verbs vary in the kinds of clauses they allow:

(15) * That Titus laughs

(16) Titus knows that Lavinia laughs

(17) Titus knows which king Lavinia praises

(18) Titus thinks that Lavinia laughs

(19) * Titus thinks which king Lavinia praises

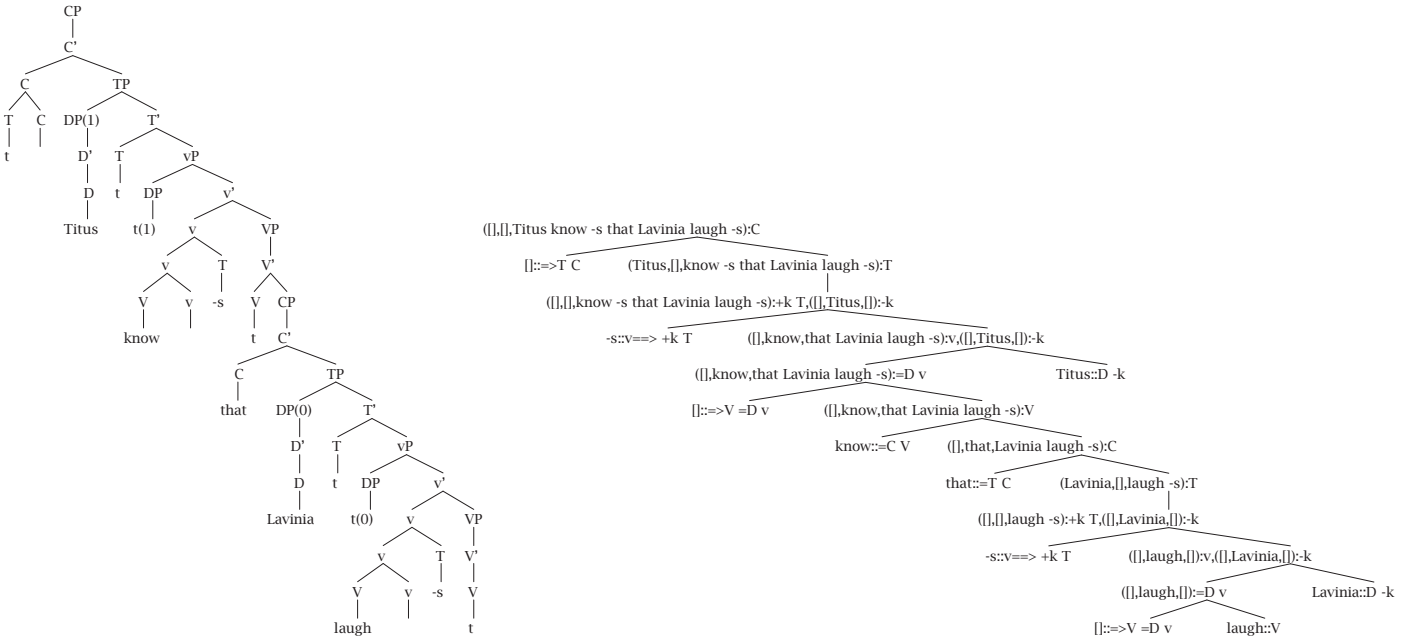
(20) * Titus wonders that Lavinia laughs

(21) Titus wonders which king Lavinia praises

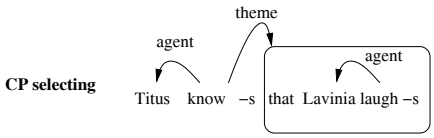
Verbs like *know* can also occur in transitive and intransitive constructions. We can get these distinctions with lexical entries like this:

that::=T Ce	ϵ::=T Ce		
ϵ::=T +wh Cwh	ϵ::=>T +wh Cwh		
know::=Ce V	know::=Cwh V	know::=D +k V	know::V
doubt::=Ce V	doubt::=Cwh V	doubt::V	
think::=Ce V		think::V	
	wonder::=Cwh V	wonder::V	

With these lexical entries we obtain derivations like this (showing a conventional depiction on the left and the actual derivation tree on the right):



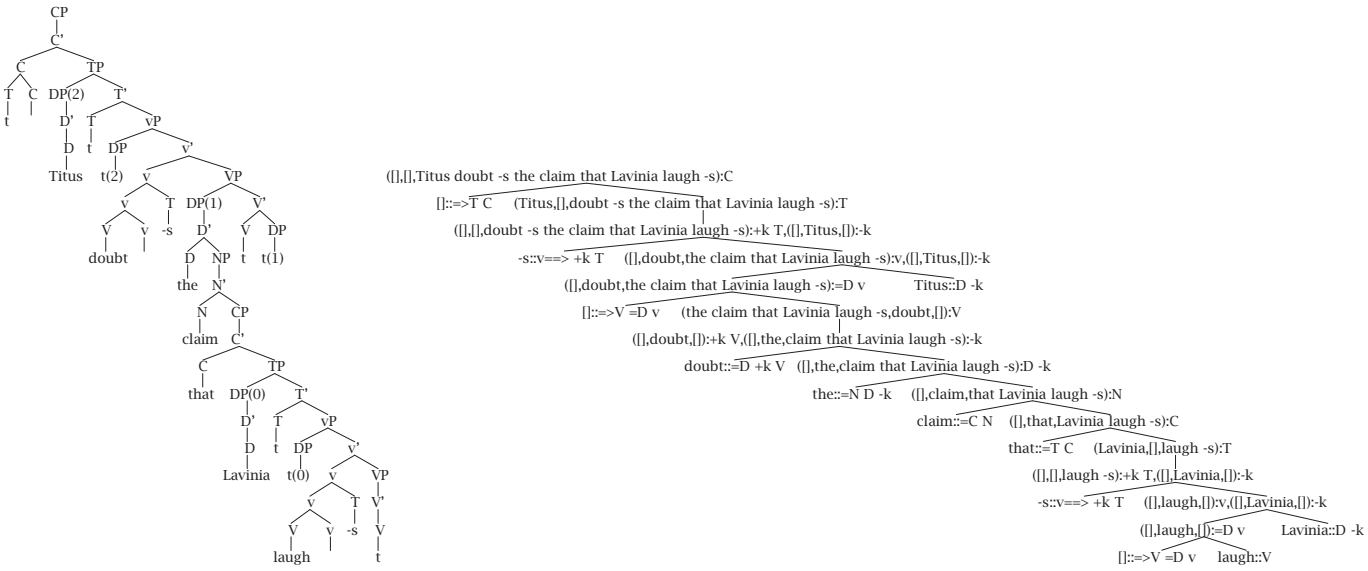
Semantically, the picture corresponds to the derivation as desired:



We can also add nouns that select clausal complements:

claim::=C N proposition::=C N

With these lexical entries we get trees like this:



10.3.2 TP-selecting raising verbs

The selection relation corresponds to the semantic relation of taking an argument. In some sentences with more than one verb, we find that not all the verbs take the same number of arguments. We notice for example that auxiliaries select VPs but do not take their own subjects or objects. A more interesting situation arises with the so-called "raising" verbs, which select clausal complements but do not take their own subjects or objects. In this case, since the main clause tense must license case, a lower subject can move to the higher clause.

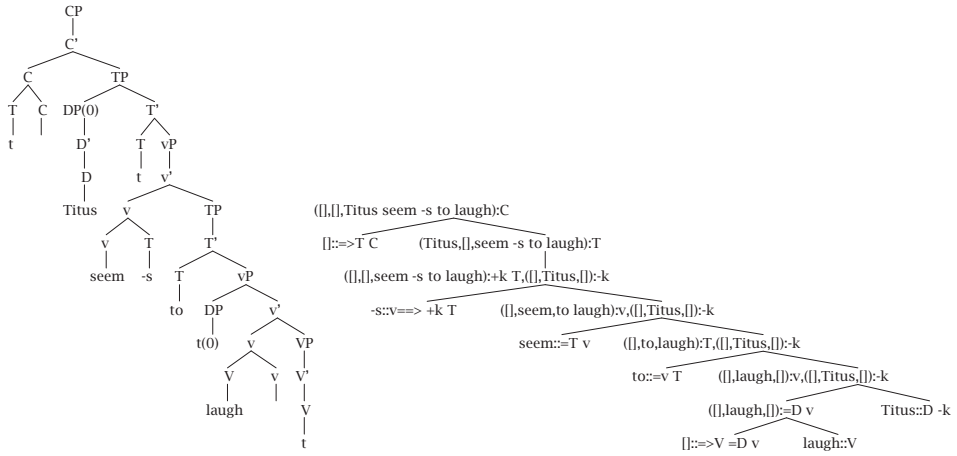
A simple version of this idea is implemented by the following lexical item for the raising verb *seem*

seem::=T v

and by the following lexical items for the infinitival *to*:

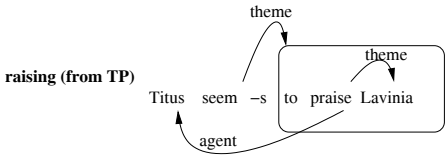
to::=v T to::=Have T to::=Be T

With these lexical entries, we get derivations like this:

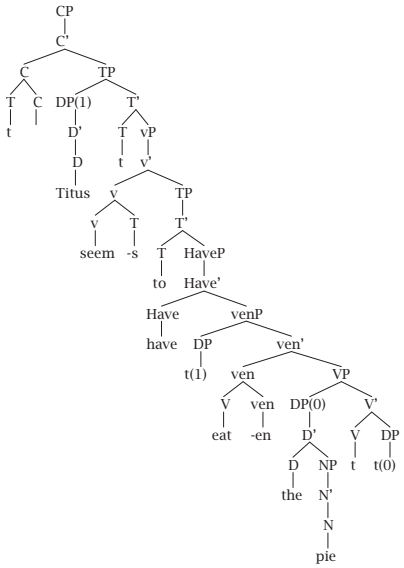


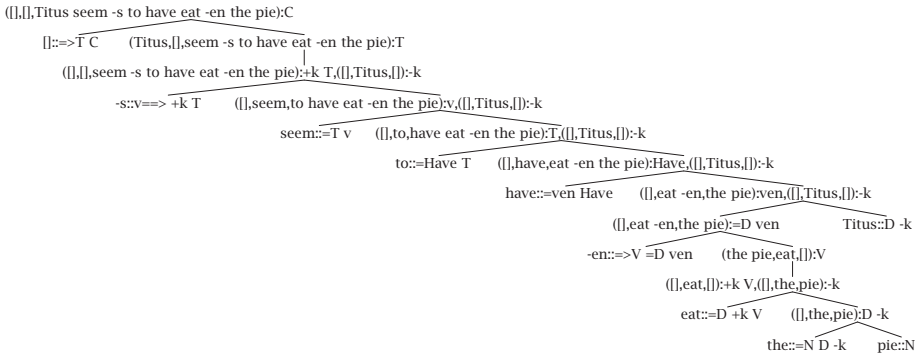
Notice that the subject of *laugh* cannot get case in the infinitival clause, so it moves to the higher clause. In this kind of construction, the main clause subject is not selected by the main clause verb!

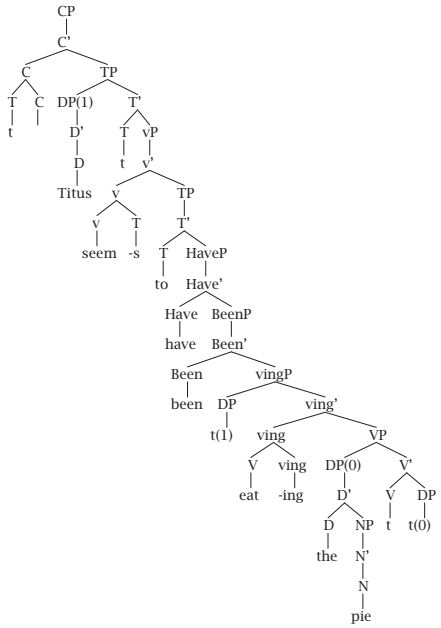
Semantically, the picture corresponds to the derivation as desired:



Notice that the infinitival *to* can occur with *have, be* or a main verb, but not with a modal:





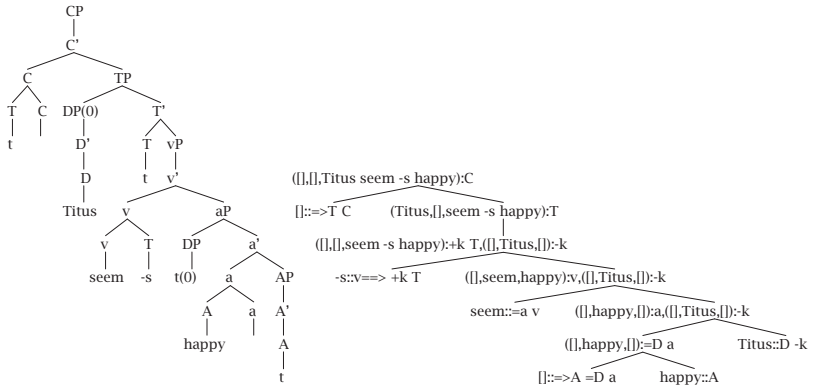


10.3.3 AP-selecting raising verbs

A similar pattern of semantic relations occurs in constructions like this:

Titus seems happy

In this example, Titus is not the 'agent' of seeming, but rather the 'experiencer' of the happiness, so again it is natural to assume that *Titus* is the subject of *happy*, raising to the main clause for case. We can assume that adjective phrase structure is similar to verb phrase structure, with the possibility of subjects and complements, to get constructions like this:



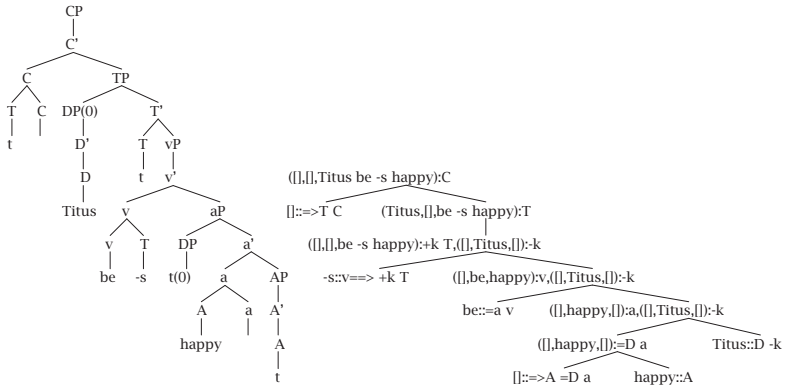
We obtain this derivation with these lexical items:

ϵ::=>A =D a. black::A	white::A
happy::A	unhappy::A
seem::=a v	

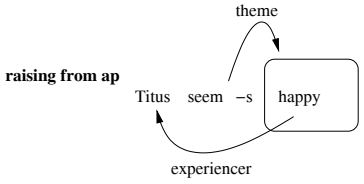
The verb *be* needs a similar lexical entry

be::=a v

to allow for structures like this:



Semantically, the picture corresponds to the derivation as desired:



10.3.4 AP small clause selecting verbs, raising to object

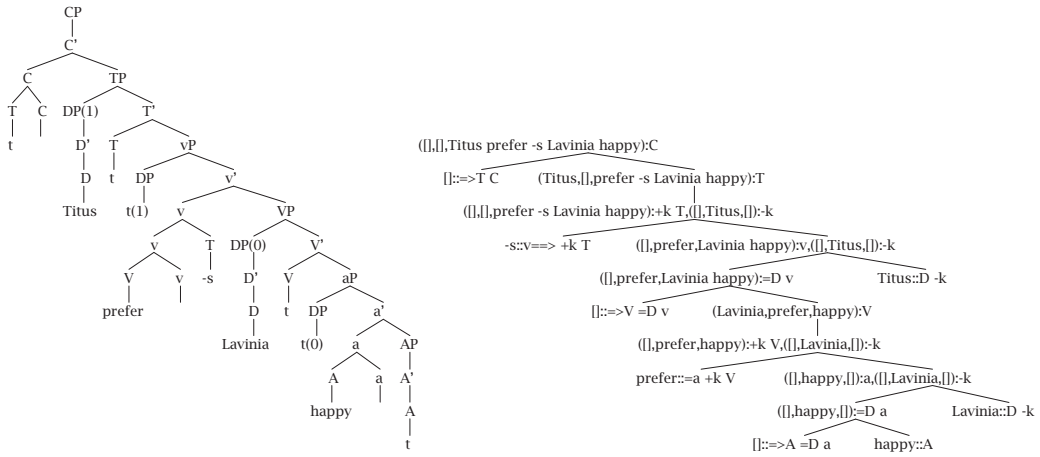
We get some confirmation for the analyses above from so-called "small clause" constructions like:

Titus considers Lavinia happy
He prefers his coffee black
He prefers his shirts white

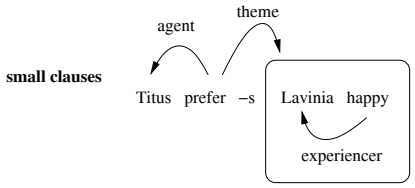
The trick is to allow for the embedded object to get case. One hypothesis is that this object gets case from the governing verb. A simple version of this idea is implemented by the following lexical items:

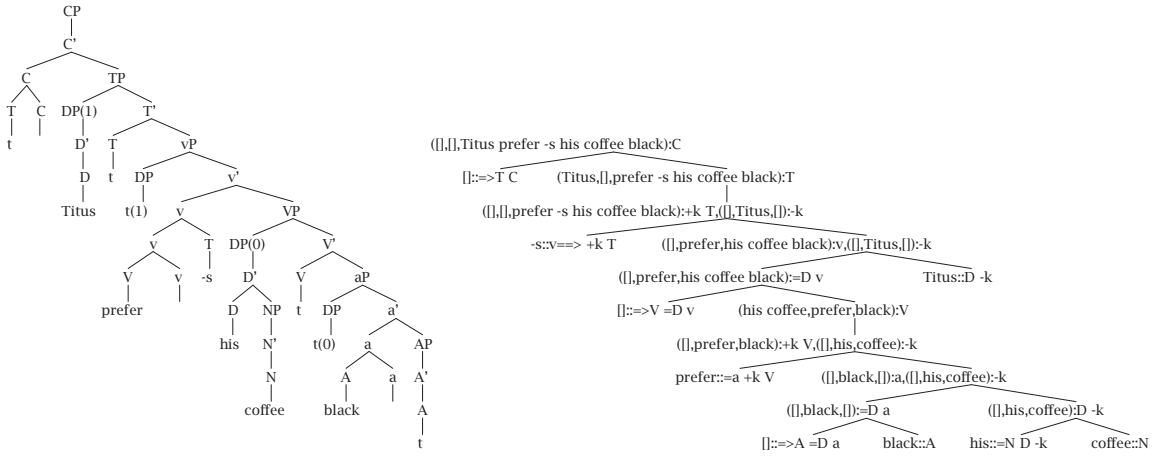
prefer::=a +k V	prefer::=T +k V
consider::=a +k V	consider::=T +k V

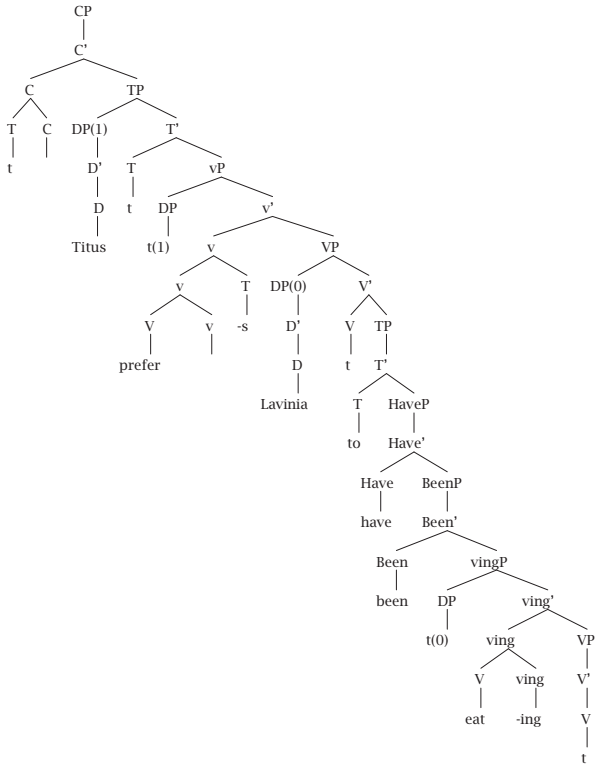
With these lexical items, we get derivations like this:



Semantically, the picture corresponds to the derivation as desired:







10.3.5 PP-selecting verbs, adjectives and nouns

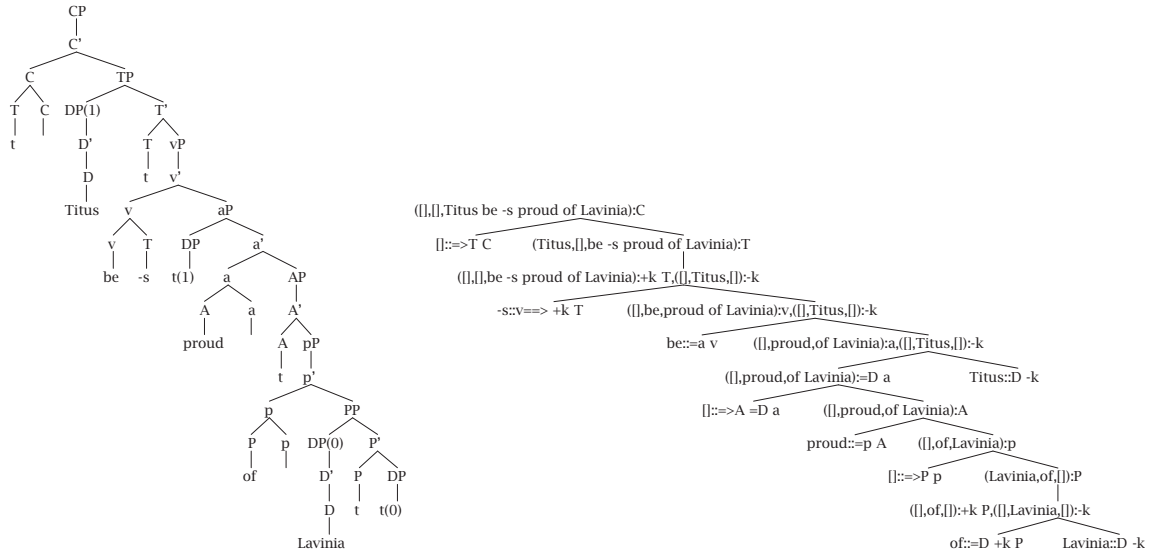
We have seen adjective phrases with subjects, so we should at least take a quick look at adjective phrases with complements. We first consider examples like this:

Titus is proud of Lavinia
Titus is proud about it

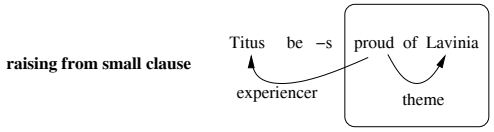
We adopt lexical items which make prepositional items similar to verb phrases, with a “little” p and a “big” P:

proud::=p A	proud::A	proud::=T a
ϵ::=>P p		
of::=D +k P	about::=D +k P	

With these lexical items we get derivations like this:



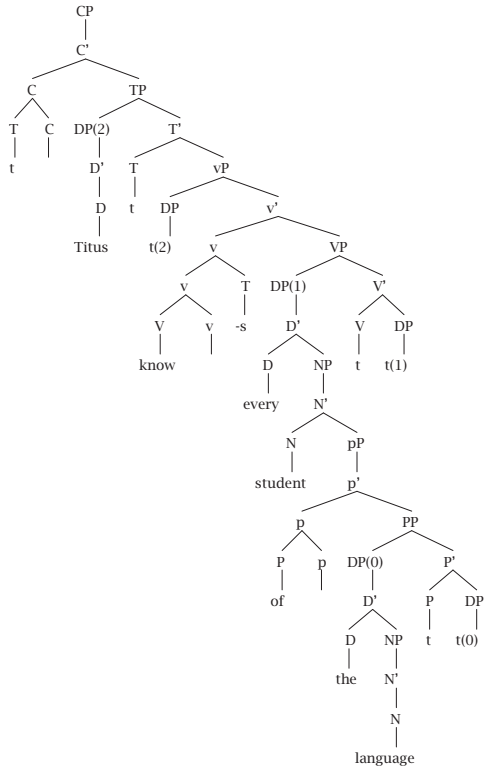
Semantically, the picture corresponds to the derivation as desired:



Similarly, we allow certain nouns to have PP complements, when they specify the object of an action or some other similarly constitutive relation:

student::=p N	student::N
citizen::=p N	citizen::N

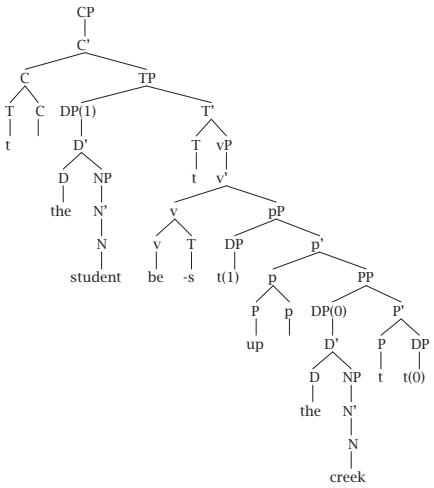
to get constructions like this:

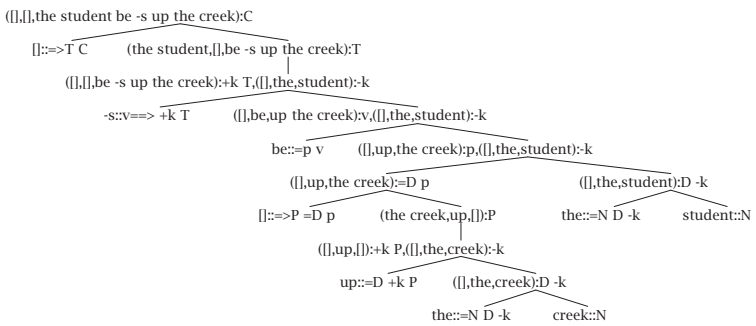


If we add lexical items like the following:

be::=p v	seem::=p v
ϵ::=>P =D p	up::=D +k P
creek::N	

then we get derivations like this:





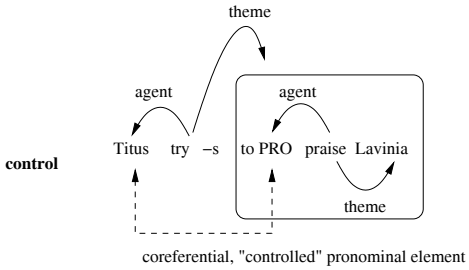
10.3.6 Control verbs

There is another pattern of semantic relations that is actually more common that the raising verb pattern: namely, when a main clause has a verb selecting the main subject, and the embedded clause has no pronounced subject, with the embedded subject understood to be the same as the main clause subject:

Titus wants to eat
Titus tries to eat

One proposal for these constructions is that the embedded subjects in these sentences is an empty (i.e. unpronounced) pronoun which must be "controlled" by the subject in the sense of being coreferential.[47]

The idea is that we have a semantic pattern here like this:

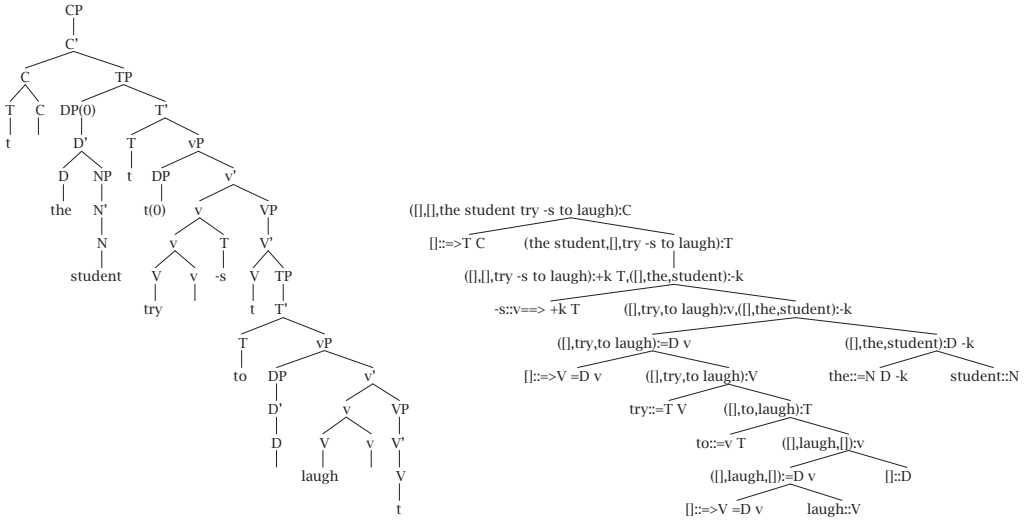


We <u>almost</u> succeed in getting a simple version of this proposal with just the following lexical items:

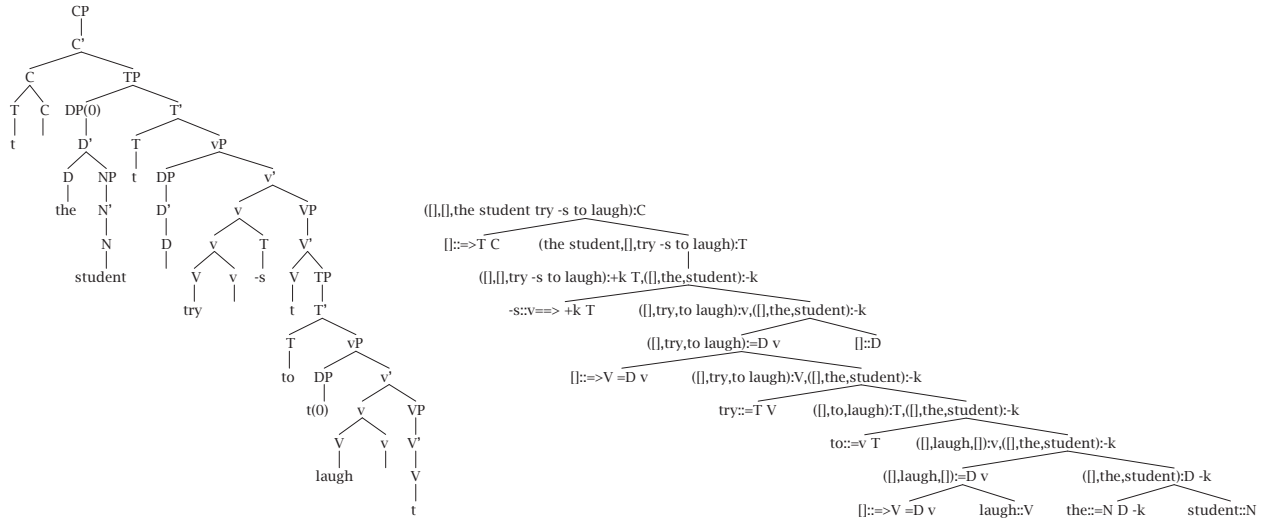
try::=T V want::=T V want::=T +k V
ϵ::D

Notice that the features of *try* are rather like a control verb's features, except that it does not assign case to the embedded object. Since the embedded object cannot get case from the infinitival either, we need to use the empty determiner provided here because this lexical item does not need case.

The problem with this simple proposal is that the empty D is allowed to appear in either of two positions. The first of the following trees is the one we want, but the lexical items allow the second one too:



[47] For historical reasons, these verbs are sometimes also called "equi verbs."



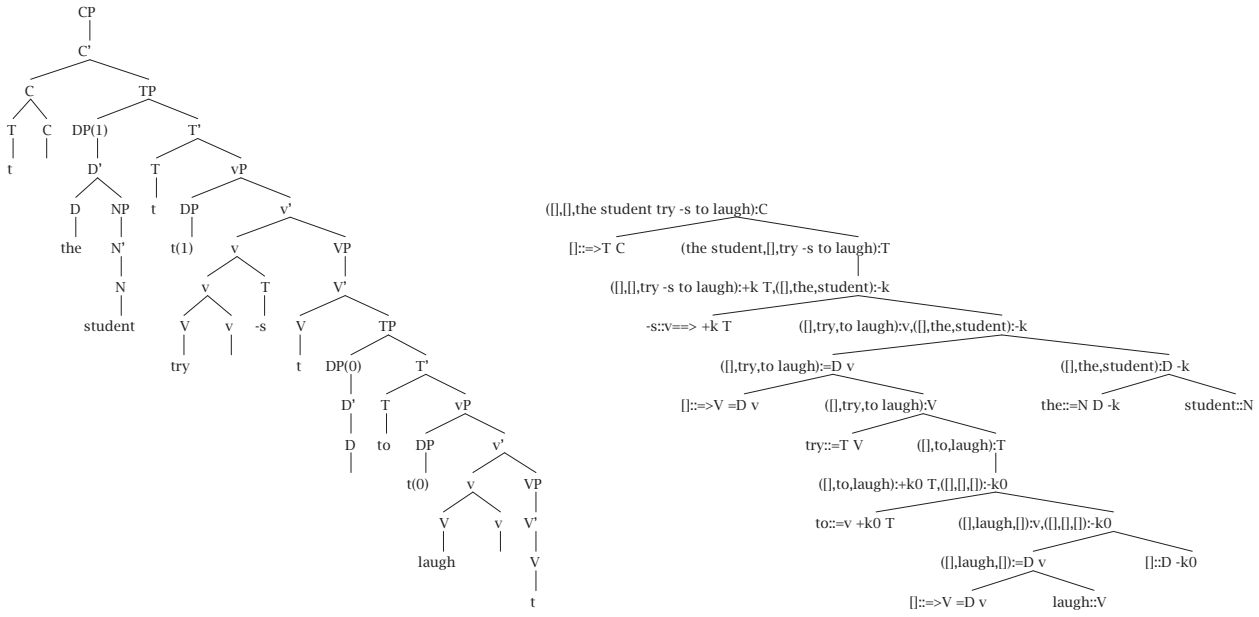
This second derivation is kind of wierd – it does not correspond to the semantic relations we wanted. How can we rule it out?

One idea is that this empty pronoun (sometimes called PRO) actually requires some kind of feature checking relation with the infinitive tense. Sometimes the relevant feature is called "null case" (Chomsky and Lasnik, 1993; Watanabe, 1993; Martin, 1996). (In fact, the proper account of control constructions is still controversial – cf., for example, Hornstein, 1999.)

A simple version of this proposal is to use a new feature k0 for "null case," in lexical items like these:

ϵ:: D –k0
to::=v +k0 T to::=Have +k0 T to::=Be +k0 T

With these we derive just one analysis for *the student try -s to laugh*:



Notice how this corresponds to the semantic relations diagrammed on the previous page.

10.4 Modifiers as adjuncts

One thing we have not treated yet is the adjunction of modifiers. We allow PP complements of N, but traditional transformational grammar allows PPs to adjoin on the right of an NP to yield expressions like

student [from Paris]
student [from Paris] [in the classroom]
student [from Paris] [in the classroom] [by the blackboard].

Adjective phrases can also modify a NP, typically adjoining to the left in English:

[Norwegian] student
[young] [Norwegian] student
[very enthusiastic] [young] [Norwegian] student.

And of course both can occur:

[very enthusiastic] [young] [Norwegian] student [from Paris] [in the classroom] [by the blackboard].

Unlike selection, this process seems optional in all cases, and there does not seem to be any fixed bounds on the number of possible modifiers, so it is widely (but by no means universally) thought that the mechanisms and structures of modifier attachment are fundamentally unlike complement attachment. Adopting that idea for the moment, let's introduce a new mechanism for adjunction.

To indicate that APs can left adjoin to NP, and PPs and CPs (relative clauses) can right adjoin to NP, let's use the notation:

a»N N«p N«Cwh

(Notice that in this notation the "arrows" point to the head, to the thing modified.) Similarly for verb modifiers, as in *Titus loudly laughs* or *Titus laughs loudly* or *Titus laughs in the castle*:

Adv»v v«Adv v«P

For adjective modifiers like *very* or *extremely*, in the category deg(ree), as in *Titus is very happy*:

Deg»a

Adverbs can modify prepositions, as in *Titus is completely up the creek*:

Adv»P

The category num can be modified by qu(antity) expressions like many,few,little,1,2,3,...as in *the 1 place to go is the cemetery*, *the little water in the canteen was not enough*, *the many activities include hiking and swimming*:

Qu » Num

Determiners can be modified on the left by *only, even* which we give the category emph(atic), and on the right by CPs (appositive relative clauses as in *Titus, who is the king, laughs*):

Emph»D D«Cwh

Constructions similar to this were mentioned earlier in exercise 3 on page 55.

In the simple grammar we are constructing here, we would also like to allow or simple DP adjuncts of DP. These are the appositives, as in *Titus, the king, laughs*. The problem with these adjunction constructions is that they contain two DPs, each of which has a case feature to be checked, even though the sentence has just one case checker, namely the finite tense *-s*. This can be handled if we allow an element with the features D –k to combine with another element having those same features, to yield one (not two) elements with those features. We represent this as follows:

D –k«D –k.

It is not hard to extend our grammar to use adjunction possibilities specified in this format.

In the framework that allows head movement, we need rules like this:

$$\frac{s_s, s_h, s_c \cdot f\gamma, \alpha_1, \ldots, \alpha_k \qquad t_s, t_h, t_c \cdot g\eta\nu, \iota_1, \ldots, \iota_l}{s_s s_h s_c t_s, t_h, t_c : g\eta\nu, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l}\ \text{left-adjoin1: if } f\gamma\text{»}g\eta$$

$$\frac{s_s, s_h, s_c \cdot f\gamma, \alpha_1, \ldots, \alpha_k \qquad t_s, t_h, t_c \cdot g\eta\nu, \iota_1, \ldots, \iota_l}{s_s, s_h, s_c t_s t_h t_c : g\eta\nu, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l}\ \text{right-adjoin1: if } g\eta\text{«}f\gamma$$

And we have two other rules for the situations where the modifier is moving, so that its string components are not concatenated with anything yet. For non-empty δ:

$$\frac{s_s, s_h, s_c \cdot f\gamma\delta, \alpha_1, \ldots, \alpha_k \qquad t_s, t_h, t_c \cdot g\eta\nu, \iota_1, \ldots, \iota_l}{t_s, t_h, t_c : g\eta\nu, s_s, s_h, s_c :: \delta, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l}\ \text{left-adjoin2: if } f\gamma\text{»}g\eta$$

$$\frac{s_s, s_h, s_c \cdot f\gamma\delta, \alpha_1, \ldots, \alpha_k \qquad t_s, t_h, t_c \cdot g\eta\nu, \iota_1, \ldots, \iota_l}{t_s, t_h, t_c : g\eta\nu, s_s, s_h, s_c :: \delta, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l}\ \text{right-adjoin1: if } g\eta\text{«}f\gamma$$

Notice that the domains of left-adjoin1 and left-adjoin2 are disjoint, so their union is a function which we can call left-adjoin. And similarly for right-adjoin. And notice that we have ordered the arguments to all these functions so that the modifier appears as the first argument, even when it is adjoined to the right, in analogy with the merge rules which always have the selector first.

EXAMPLES

10.5 Summary and implementation

It is important to see that the rather complex range of constructions surveyed in the previous sections §§10.3,10.4, are all derived from a remarkably simple grammar. Here is the whole thing:

```
% File    : gh5.pl
% Author : E Stabler
% Updated: Feb 2002

% complementizers
[]::['=T','C'].          []::[=>'T','C'].      []::[=>'T',+wh,'C'].  []::['=T',+wh,'C'].
[that]::['=T','Ce'].     []::['=T','Ce'].          % embedded clause
[]::['=T',+wh,'Cwh'].    []::[=>'T',+wh,'Cwh'].  % embedded wh-clause

% finite tense
['-s']::[v==>,+k,'T'].  % for affix hopping
['-s']::[=>'Modal',+k,'T'].  ['-s']::[=>'Have',+k,'T'].   ['-s']::[=>'Be',+k,'T'].      ['-s']::[=v,+k,'T'].

% simple nouns
[queen]::['N'].              [pie]::['N'].            [human]::['N'].       [car]::['N'].    ['Goth']::['N'].
[coffee]::['N'].             [shirt]::['N'].          [language]::['N'].    [king]::['N'].

% determiners
[the]::['=Num','D',-k].          [every]::['=Num','D',-k].     [a]::['=Num','D',-k].    [an]::['=Num','D',-k].
[some]::['=Num','D',-k].         [some]::['D',-k].

% number marking (singular, plural)
[]::['=N','Num'].                ['-s']::['N'==>,'Num'].

% names as lexical DPs
['Titus']::['D',-k].   ['Lavinia']::['D',-k].  ['Tamara']::['D',-k].  ['Saturninus']::['D',-k].
['Rome']::['D',-k].    ['Sunday']::['D',-k].

% pronouns as lexical determiners
[she]::['D',-k].  [he]::['D',-k].   [it]::['D',-k].   ['I']::['D',-k]. [you]::['D',-k]. [they]::['D',-k].  % nom
[her]::['D',-k].  [him]::['D',-k].                    [me]::['D',-k].  [us]::['D',-k].   [them]::['D',-k].  % acc
[my]::['=Num','D',-k]. [your]::['=Num','D',-k].
[her]::['=Num','D',-k]. [his]::['=Num','D',-k].  [its]::['=Num','D',-k].  % gen

% wh determiners
[which]::['=Num','D',-k,-wh].  [which]::['D',-k,-wh].
[what]::['=Num','D',-k,-wh].   [what]::['D',-k,-wh].

% auxiliary verbs
[will]::['=Have','Modal'].   [will]::['=Be','Modal'].    [will]::[=v,'Modal'].
[have]::['=Been','Have'].    [have]::[=ven,'Have'].
[be]::[=ving,'Be'].          [been]::[=ving,'Been'].

% little v
[]::[=>'V',='D',v].     ['-en']::[=>'V',='D',ven].  ['-ing']::[=>'V',='D',ving].
['-en']::[=>'V',ven].       ['-ing']::[=>'V',ving].

% DP-selecting (transitive) verbs - select an object, and take a subject too (via v)
[praise]::['=D',+k,'V'].   [sing]::['=D',+k,'V'].  [eat]::['=D',+k,'V'].  [have]::['=D',+k,'V'].

% intransitive verbs - select no object, but take a subject
[laugh]::['V'].          [sing]::['V'].       [charge]::['V'].    [eat]::['V'].

% CP-selecting verbs
[know]::['=Ce','V'].   [know]::['=Cwh','V'].   [know]::['=D',+k,'V'].   [know]::['V'].
[doubt]::['=Ce','V'].  [doubt]::['=Cwh','V'].  [doubt]::['=D',+k,'V'].  [doubt]::['V'].
[think]::['=Ce','V'].                                                   [think]::['V'].
```

```
                        [wonder]::[='Cwh','V'].                    [wonder]::['V'].
% CP-selecting nouns
[claim]::['Ce','N'].  [proposition]::['Ce','N'].  [claim]::['N'].  [proposition]::['N'].

% raising verbs - select only propositional complement, no object or subject
[seem]::['T',v].

% infinitival tense
[to]::[=v,'T'].  [to]::['Have','T'].  [to]::['Be','T'].  % nb does not select modals

% little a
[]::[=>'A',='D',a].

% simple adjectives
[black]::['A'].  [white]::['A'].  [human]::['A'].  [mortal]::['A'].
[happy]::['A'].  [unhappy]::['A'].

% verbs with AP complements: predicative be, seem
[be]::[=a,'Be'].  [seem]::[=a,v].

% adjectives with complements
[proud]::[=p,'A'].  [proud]::['A'].  [proud]::['T',a].

% little p (no subject?)
[]::[=>'P',p].

% prepositions with no subject
[of]::['D',+k,'P'].  [about]::['D',+k,'P'].  [on]::['D',+k,'P'].

% verbs with AP,TP complements: small clause selectors as raising to object
[prefer]::[=a,+k,'V'].  [prefer]::['T',+k,'V'].  [prefer]::['D',+k,'V'].
[consider]::[=a,+k,'V'].  [consider]::['T',+k,'V'].  [consider]::['D',+k,'V'].

% nouns with PP complements
[student]::[=p,'N'].  [student]::['N'].
[citizen]::[=p,'N'].  [citizen]::['N'].

% more verbs with PP complements
[be]::[=p,v].  [seem]::[=p,v].  []::[=>'P',='D',p].  [up]::['D',+k,'P'].  [creek]::['N'].

% control verbs
[try]::['T','V'].  [want]::['T','V'].  [want]::['T',+k,'V'].

% verbs with causative alternation: using little v that does not select subject
[break]::['D',+k,'V'].
% one idea, but this intrans use of the transitivizer v can cause trouble:
%[break]::['D','V'].  []::[='V',v].  % so, better:
[break]::['D',v].

% simple idea about PRO that does not work: []::['D'].
% second idea: "null case" feature k0
[]::['D',-k0].
[to]::[=v,+k0,'T'].  [to]::['Have',+k0,'T'].  [to]::['Be',+k0,'T'].  % nb does not select modals

% modifiers
['N']<<[p].  [v]<<[p].  [v]<<['Adv'].  ['D',-k]<<['D',-k].  %['N']<<['A']. % for testing only
['A']>>['N'].  ['Adv']>>[v].  [deg]>>['A'].  [deg]>>['Adv'].  ['Adv']>>['P'].  [emph]>>['D',-k].  [qu]>>['Num'].

[completely]::['Adv'].  [happily]::['Adv'].  [very]::[deg].  [only]::[emph].  [3]::[qu].

startCategory('C').
```

10.5.1 Representing the derivations

Consider the context free grammar $G1 = \langle \Sigma, N, \rightarrow \rangle$, where

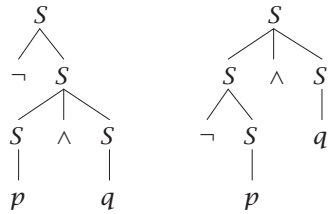
Σ $= \{p, q, r, \neg, \vee, \wedge\}$,

N $= \{S\}$, and

$\rightarrow$ has the following 6 pairs in it:

$$\begin{array}{lll} S \rightarrow p & S \rightarrow q & S \rightarrow r \\ S \rightarrow \neg S & S \rightarrow S \vee S & S \rightarrow S \wedge S \end{array}$$

This grammar is ambiguous since we have two derivation trees for $\neg p \wedge q$:



Here we see that the yield $\neg p \wedge q$ does not determine the derivation.

One way to eliminate the ambiguity is with parentheses. Another way is to use Polish notation. Consider the context free grammar $G2 = \langle \Sigma, N, \rightarrow \rangle$, where

Σ $= \{p, q, r, \neg, \vee, \wedge\}$,

N $= \{S\}$, and

$\rightarrow$ has the following 6 pairs in it:

$$\begin{array}{lll} S \rightarrow p & S \rightarrow q & S \rightarrow r \\ S \rightarrow \neg S & S \rightarrow \vee\ S\ S & S \rightarrow \wedge\ S\ S \end{array}$$

With this grammar, we have just one derivation tree for $\wedge \neg pq$, and just one for $\neg \wedge pq$:

Consider the minimalist grammar $G2 = \langle \Sigma, N, Lex, \mathcal{F} \rangle$, where

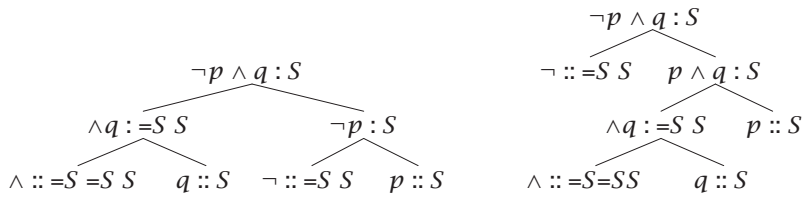
Σ $= \{p, q, r, \neg, \vee, \wedge\}$,

N $= \{S\}$, and

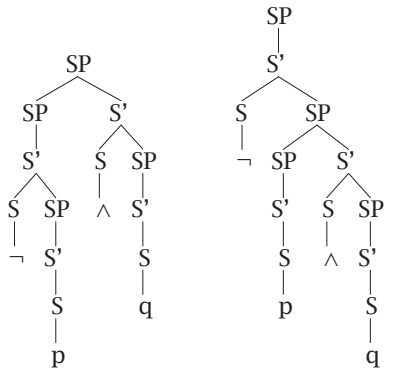
Lex has the following 6 lexical items built from Σ and N:

$$\begin{array}{lll} p :: S & q :: S & r :: S \\ \neg :: {=}S\ S & \vee :: {=}S\ {=}S\ S & \wedge :: {=}S\ {=}S\ S \end{array}$$

This grammar has ambiguous expressions, since we have the following two different two derivations of $\neg p \wedge q$:



These correspond to trees that we might depict with X-bar structure in the following way:



While these examples show that $\mathcal{G}2$ has ambiguous expressions, they do not show that $\mathcal{G}2$ has ambiguous yields. Notice that the yields of the two simple derivation trees shown above (not the X-bar structures, but the derivation trees) are not the same. The two yields are, respectively,

$$\wedge :: {=}S{=}SS \quad q :: S \quad \neg :: {=}SS \quad p :: S$$

$$\neg :: {=}SS \quad \wedge :: {=}S{=}SS \quad q :: S \quad p :: S$$

In fact, not only this grammar, but every minimalist grammar is unambiguous in this sense (Hale and Stabler, 2001). Each sequence of lexical items has at most one derivation. These sequences are, in effect, Polish notation for the sentence, one that completely determines the whole derivation. Notice that if we leave out the features from the lexical sequences above, we have exactly the standard Polish notation:

$$\wedge q \neg p$$

$$\neg : \wedge q p$$

This fact is exploited in the implementation, where `mgh.pl` computes the derivations (if any) and represents them as sequences of lexical items (numbered in decimal notation), and then `lhp.pl` converts those sequences to the tree representations for viewing by humans.

The addition of adjunction requires special treatment, because this operation is not triggered by features. What we do is to insert » or « into the lexical sequence (immediately before the modifier and modifiee lexical sequences) to maintain a sequence representation that unambiguously specifies derivations. (No addition is required for coordinators, since they are always binary operators that apply to two constituents which are "completed" in the sense that their first features are basic categories.)

Exercises: Two more easy exercises just to make sure that you understand how the grammar works. Plus one extra credit problem.

1. The grammar gh5.pl in §10.5 on page 222 allows wh-movement to form questions, but it does not allow topicalization, which we see in examples like this:

   Lavinia, Titus praise -s
   The king, Titus want -s to praise

   One idea is that the lexicon includes in addition to DPs like *Lavinia*, a -topic version of this DP, which moves to a +topic specifier of CP. Extend grammar gh4.pl to get these topicalized constructions in this way.

2. We did not consider verbs like *put* which require two arguments:

   the cook put -s the pie in the oven
   * the cook put -s the pie
   * the cook put -s in the oven

   One common idea is that while transitive verbs have two parts, v and V, verbs like *put* have three parts which we could call v and V and VV, where VV selects the PP, V selects the object, and v selects the subject. Extend grammar gh4.pl in this way so that it gets *the cook put -s the pie in the oven*. Make sure that your extended grammar does NOT get *the cook put -s the pie*, or *the cook put -s in the oven*.

Extra credit: As described in §10.5.1, the parser `mghp.pl` represents each derivation by the sequence of lexical items that appears as the yield of that derivation.

In the earlier exercise on page 25, we provided a simple way to represent a sequence of integers in a binary prefix code. Modify `mghp.pl` so that

i. before showing the yield of the derivation as a list of decimal numbers, it prints the number of bits in the ascii representation of the input (= which we can estimate as the number of characters $\times 7$), and

ii. after showing the yield of the derivation as a list of decimal numbers, it outputs the binary prefix code for that same sequence, and then

iii. on a new line prints the number of bits in the binary prefix code representation.

10.6 Some remaining issues

10.6.1 Locality

When phrasal movement was defined in §9.1 on page 170, it will be recalled that we only allowed the operation to apply to a structure with a +f head and exactly 1 -f constituent.[48] We mentioned that this restriction is a simple, strong version of a kind of "shortest move constraint" in the sense that each -f constituent must move to the first available +f position. If there are two -f constituents in any structure, this requirement cannot be met. This is also a kind of "relativized minimality" condition in the sense that the domains of movement are relativized by the inventory of categories (Rizzi, 1990). A -wh constituent cannot get by any other -wh constituent, but it can, for example, get by a -k constituent.

Notice that while this constraint allows a wh element to move to the front of a relative clause,

the man who$_i$ you like t_i visited us yesterday

it properly blocks moving another wh-element out, e.g. forming a question by questioning the subject *you*:

* who$_j$ did the man who$_i$ t_j like t_i visited us yesterday * the man who$_i$ I read a statement which$_j$ t_j was about t_i is sick

When this impossiblity of extracting out of a complex phrase like *the man who you like* was observed by Ross (1967), he observed that extraction out of complex determiner phrases is quite generally blocked, even when there are (apparently) no other movements of the same kind (and not only in English). For example, the following should not be accepted:

* who$_i$ did the man with t_i visit us
* the hat which$_i$ I believed the claim that Otto was wearing t_i is red

How can we block these? The SMC is apparently too weak, and needs to be strengthened. (We will see below that the SMC is also too strong.)

Freezing

Wexler, Stepanov, et al.

10.6.2 Multiple movements and resumptive pronouns

In other respects, the SMC restriction is too strong. In English we have cases like (22a), though it is marginal for many speakers (Fodor, 1978; Pesetsky, 1985):

(22) a. ?? Which violins$_1$ did you say which sonatas$_2$ were played t_2 on t_1
b. * Which violins$_1$ did you say which sonatas$_2$ were played t_1 on t_2

The example above is particularly troubling, and resembles (23a) famously observed by Huang (1982):

(23) a. ? [Which problem]$_i$ do you wonder how$_j$ to solve t_i t_j?
b. * How$_j$ do you wonder [which problem]$_i$ to solve t_i t_j?

It seems that in English, extraction of two wh-elements is possible (at least marginally) if an argument wh-phrase moves across an adjunct wh-phrase, but it is notably worse if the adjunct phrase extracts across an argument phrase. We could allow the first example if wh-Adverbs have a different feature than wh-DPs, but then we would allow the second example too. If there is really an argument-adjunct asymmetry here, it would apparently require some kind of fundamental change in the nature of our SMC.

Developing insights from Obenauer (1983), Cinque (1990), Baltin (1992) and others, Rizzi (2000) argues that what is really happening here is that certain wh-elements, like the wh-DP in a above, can be related to their traces across another wh-element when they are "referential" in a certain sense. This moves the restriction on movement relations closer to "binding theory," which will be discussed in §11. (Similarly semantic accounts have been offered by many linguists.

[48] In our implementation, we actually do not even build a representation of any constituent which has two -f parts, for any f.

10.6.3 Multiple movements and absorption

Other languages have more liberal wh-extraction than English, and it seems that at least in some of these languages, we are seeing something rather unlike the English movement relations discussed above. See for example, Saah and Goodluck (1995) on Akan; Kiss (1993) on Hungarian; McDaniel (1989) and on German and Romani; McDaniel, Chiu, and Maxfield (1995) on child English. There is interesting ongoing work on these constructions (Richards, 1998; Pesetsky, 2000, for example).

XXX MORE

10.6.4 Coordination

Coordination structures are common across languages, and pose some interesting problems for our grammar. Notice that we could parse:

Titus praise -s the coffee and pie
Titus laugh -s or Lavinia laugh -s
Titus be -s happy and proud

by adding lexical items like these to the grammar `gh4.pl`:

and::=N =N N	and::=C =C C	and::=A =A A
or::=N =N N	or::=C =C C	or::=A =A A

But this approach will not work for

Titus praise -s Lavinia and Tamara.

The reason is that each name needs to have its case checked, but in this sentence there are three names (*Titus, Lavinia, Tamara*) and only two case checkers (*-s, praise*). We need a way to coordinate Lavinia and Tamara that leaves us with just one case element to check. Similar problems face coordinate structures like

Titus and Lavinia will -s laugh
Titus praise -s and criticize -s Lavinia
Who -s Titus praise and criticize
Titus can and will -s laugh
Some and every king will -s laugh

For this and other reasons, it is commonly thought that coordination requires some kind of special mechanism in the grammar, unlike anything we have introduced so far (Citko, 2001; Moltmann, 1992; Munn, 1992). One simple idea is that the grammar includes a special mechanism that is analogous to the adjunction mechanism above, which for any coordinator $x :: coord$ and any phrases $s \cdot \alpha$ and $t \cdot \alpha$, attaching the first argument on the right as complement and later arguments as specifiers. More precisely, we use the following ternary rule:

$$\frac{s_h :: coord \qquad t_s, t_h, t_c \cdot \gamma, \alpha_1, \ldots, \alpha_k \qquad u_s, u_h, u_c \cdot \gamma, \alpha_1, \ldots, \alpha_k}{t_s t_h t_c, s_h, u_s u_h u_c : \gamma, \alpha_1, \ldots, \alpha_k}\ \text{coord1}$$

Allowing γ to be any sequence of features (with the requirement that the coordinated items s and t have this same sequence of features) will have the result that the two case requirements of the names in *Lavinia and Tamara* will be combined into one. The requirement that the moving constituents $\alpha_1, \ldots \alpha_k$ match exactly will give us a version of the "across-the-board" constraint on movements.

XXX MORE COMING

10.6.5 Pied piping

In English, both of the following questions are well-formed:

(24) Who did you talk to?

(25) To whom did you talk?

In the latter question, it appears that the PP moves because it contains a wh-DP.

To allow for this kind of phenomenon, suppose we allow a kind of merge, where the wh-features of a selected item can move to the head of the selector. Surprisingly, this addition alters the character of our formal system rather dramatically, because we lose the following fundamental property:

> In MG derivations (and in derivations involving head movement, adjunction, and coordination) the sequence of features in every derived chain is a suffix of some sequence of features of a lexical item.

11 Semantics, discourse, inference

A logic has three components: a language, a semantics, and an inference relation. As discussed in §1, a computational device may be able to recognize a language and compute the inferences, but it does not even make sense to say that it would compute the semantics. The semantics relates expressions to things in the world, and those things are only relevant to a computation to the extent that they are represented. For example, when the bank computes the balance in your account, the actual dollars do not matter to the computation; all that matters is the representations that are in the bank's computer. The interpretation function that maps the numbers to your dollars is not computed. So typically when "semantics" is discussed in models of language processing, what is really discussed is the computation of representations for reasoning. The semantics is relevant when we are thinking about what the reasoning is about, and more fundamentally, when we are deciding whether the state changes in a machine should be regarded as reasoning at all.

Standard logics are designed to have no structural ambiguity, but as we have seen, human language allows extensive ambiguity. (In fact, S6.6.3 shows that the number of different derivations cannot be bounded by any polynomial function of the number of morphemes in the input.) The different derivations often correspond to different semantic values, and so linguists have adopted the strategy of interpreting the derivations (or sometimes, the derived structures). But it is not the interpretation that matters in the computational model; rather it is the syntactic analysis itself that matters.

With this model of human language use, if we call the representation of percieved sounds PF (for 'phonetic' or 'phonological form') and the representation of a completed syntactic LF (for 'logical form'), the basic picture of the task of the grammar is to define the LF-PF relation. The simplest idea, and the hypothesis adopted here, is that LF simply is the syntactic analysis. We find closely related views in passages like these:

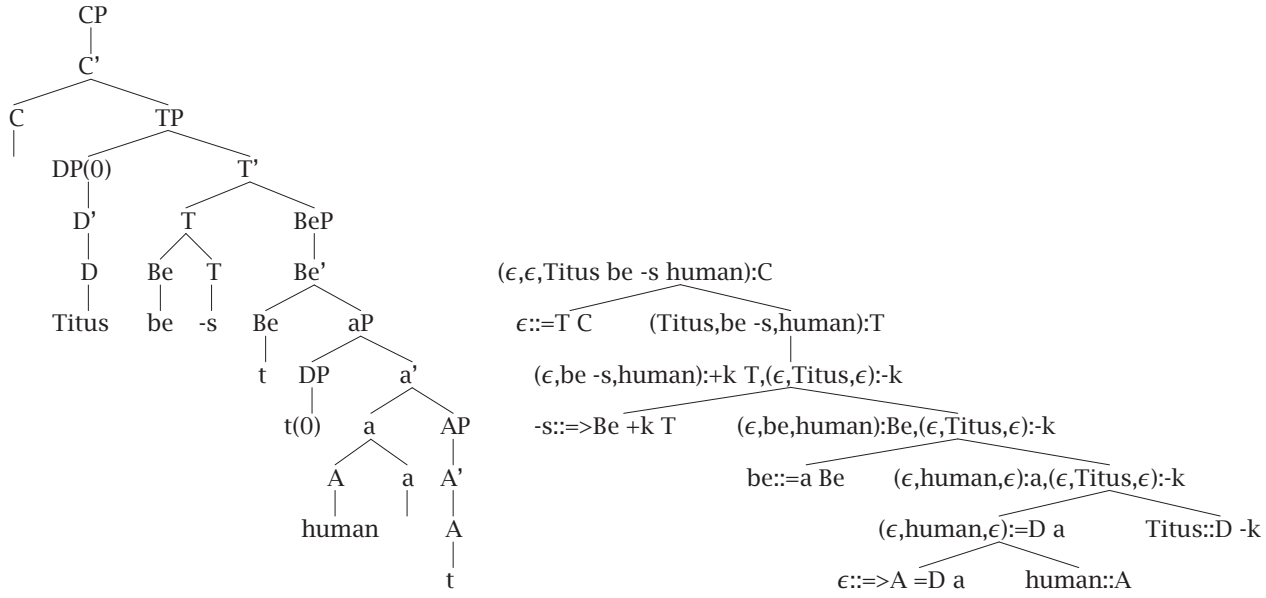
> PF and LF constitute the 'interface' between language and other cognitive systems, yielding direct representations of sound, on the one hand, and meaning on the other as language and other systems interact, including perceptual and production systems, conceptual and pragmatic systems. (Chomsky, 1986, p68)

> The output of the sentence comprehension system…provides a domain for such further transformations as logical and inductive inferences, comparison with information in memory, comparison with information available from other perceptual channels, etc...[These] extra-linguistic transformations are defined directly over the grammatical form of the sentence, roughly, over its syntactic structural description (which, of course, includes a specification of its lexical items). (Fodor et al., 1980)

> …the picture of meaning to be developed here is inspired by Wittgenstein's idea that the meaning of a word is constituted from its use – from the regularities governing our deployment of the sentences in which it appears…understanding a sentence consists, by definition, in nothing over and above understanding its constituents and appreciating how they are combined with one another. Thus the meaning of the sentence does not have to be *worked out* on the basis of what is known about how it is constructed; for that knowledge by itself constitutes the sentence's meaning. If this is so, then compositionality is a trivial consequence of what we mean by "understanding" in connection with complex sentences. (Horwich, 1998, pp3,9)

In these passages, the idea is that reasoning is defined "directly" over the syntactic analyses of the perceived language. Understanding an expression is nothing more than having the ability to obtain a structural analysis over basic elements whose meanings are understood.

It might seem that this makes the account of LF very simple. After all, we already have our syntactic analyses. For example, the grammar from the previous chapter, *gh5.pl* provides an analysis of *Titus be -s human*:



And we noticed in §10.5.1 that the whole derivation is unambiguously identified (not by the input string but) by the sequence of lexical items at the leaves of the derivation tree:

$$\epsilon\text{::=T C} \quad \text{-s::=>Be +k T} \quad \text{be::=a Be} \quad \epsilon\text{::=>A =D a} \quad \text{human::A} \quad \text{Titus::D -k}$$

This sequence is a kind of Polish notation, with the functions preceding their arguments.

Since we will be making reference to the function-argument relations in the syntactic analyses, it will be helpful to enrich the lexical sequence with parentheses. The parentheses are redundant, but they make the analyses more readable. When no confusion will result, we will also sometimes write just the string part of the lexical items, leaving the lexical items out, except when the lexical item is empty, in which case we sometimes use the category label of the element and leave everything else out. With these conventions, the representation above can be represented this way:

C(-s(be(a(human(Titus)))))

Notice that in this style of representation, we still have all the lexical items in the order that they appear in the derivation tree; we have just abbreviated them and added parentheses to make these sequences more readable for humans.

So now we just need to specify the inference relations over these analyses. For example, we should be able to recognize the following inference (using our new convenient notation for derivations):

$$\frac{\texttt{C(-s(be(a(human(Titus)))))} \qquad \texttt{C(-s(be(a(mortal(every(Num(human)))))))}}{\texttt{C(-s(be(a(mortal(Titus)))))}}$$

Simplifying for the moment by leaving out the empty categories, *be*, number and tense, what we have here is:

$$\frac{\texttt{human(Titus)} \qquad \texttt{mortal(every(human))}}{\texttt{mortal(Titus)}}$$

Clearly this is just one example of an infinite relation. The relation should include, for example, the derivations corresponding to strings like the following, and infinitely many others:

$$\frac{\texttt{reads(some(Norwegian(student)))}}{\texttt{reads(some(student))}} \qquad \frac{\texttt{(quickly(reads))(some(student))}}{\texttt{reads(some(student))}} \qquad \frac{\texttt{read(10(students))}}{\texttt{reads(some(student))}} \qquad \frac{\texttt{laughing(juan)}}{\texttt{(laughing}\vee\texttt{crying)(juan)}}$$

The assumption we make here is that these inferences are linguistic, in the sense that someone who does not recognize entailment relations like these cannot be said to understand English.

It is important to reflect on what this view must amount to. A competent language user must not only be able to perform the syntactic analysis, but also must have the inference rules that are defined over these analyses. This is an additional and significant requirement on the adequacy of our theory, one that is only sometimes made explicit:

> For example, trivially we judge pretheoretically that 2b below is true whenever 2a is.
>
> 2a. John is a linguist and Mary is a biologist.
>
> b. John is a linguist.
>
> Thus, given that 2a,b lie in the fragment of English we intend to represent, it follows that our system would be descriptively inadequate if we could not show that our representation for 2a formally entailed our representation of 2b. (Keenan and Faltz, 1985, p2)

Spelling out the idea that syntax defines the structures to which inference applies, we see that syntax is much more than just a theory of word order. It is, in effect, a theory about how word order can be a reflection of the semantic structures that we reason with. When you learn a word, you form a hypothesis not only about its positions in word strings, but also about its role in inference. This perhaps surprising hypothesis will be adopted here, and some evidence for it will be presented.

So we have the following views so far:

- semantic values and entailment relations are defined over syntactic derivations
- linguistic theory should explain the recognition of entailment relations that hold in virtue of meaning

Now, especially if the computational model needs the inference relations (corresponding to entailment) but does not really need the semantic valuations, as noted at the beginning of this section, this project may sound easy. We have the syntactic derivations, so all we need to do is to specify the inference relation, and consider how it is computed. Unfortunately, things get complicated in some surprising ways when we set out to do this. Three problems come up right away:

First: *Certain collections of expressions have similar inferential roles, but this classification of elements according to semantic type does not correspond to our classification of syntactic types.*

Second: *Semantic values are fixed in part by context.*

Third: *Since the syntax is now doing more than defining word order, we may want to modify and extend it for purely semantic reasons.*

We will develop some simple ideas first, and then return to discuss these harder problems in §16. We will encounter these points as we develop our perspective.

So to begin with the simplest ideas, we will postpone these important complications: we will ignore pronouns and contextual sensitivity generally; we will ignore tense, empty categories and movement. Even with these simplifications, we can hope to achieve a perspective on inference which typically concealed by approaches that translate syntactic structures into some kind of standard first (or second) order logic. In particular:

- While the semantics for first order languages obscures the Fregean idea that quantifiers are properties of properties (or relations among properties, the approach here is firmly based on this insight.
- Unlike the unary quantifiers of first order languages – e.g. $(\forall X)\phi$ – the quantifiers of natural languages are predominantly binary or "sortal" – e.g. every(ϕ, ψ). The approach adopted here allows binary quantifiers.
- While standard logic allows coordination of truth-value-denoting expressions, to treat human language we want to be able to handle coordinations of almost every category. That is not just

$$\text{human(Socrates)} \wedge \text{human(Plato)}$$

but also things like

$$\text{human}(\text{Socrates} \wedge \text{Plato})$$
$$(\text{human} \wedge \text{Greek})(\text{Socrates})$$
$$((\text{snub-nosed} \wedge \text{Greek})(\text{human}))(\text{Socrates})$$
$$(((\text{to} \wedge \text{from})(\text{Athens}))(\text{walked}))(\text{Socrates})$$

- Standard logical inference is deep, uses few inference rules, and depends on few premises, while typical human reasoning seems rather shallow, with possibly a large number of inference rules and multiple supports for each premise. – We discuss this in §16.5.1 below.
- Standard logical inference seems well designed for monotonicity-based inferences, and negative-polarity items of various kinds (*any, ever, yet, a red cent, give a damn, one bit, budge an inch*) provide a visible syntactic reflex of this. For example:

  i. every publisher of any book will get his money

  ii. * every publisher of Plato will get any money

  iii. no publisher of Plato will get any money

  We see in these sentences that the contexts in which *any* can appear with this meaning depend on the quantifier in some way. Roughly, *any* can appear only in monotone decreasing contexts – where this notion is explained below, a notion that is relevant for a very powerful inference step. We will see that "the second argument of *every*" is increasing, but "the second argument of *no*" is decreasing.

12 Review: first semantic categories

12.1 Things

Let's assume that we are talking about a certain domain, a certain collection of things. In a trivial case, we might be discussing just John and Mary, and so our **domain of things, or entities** is:

$$E = \{j, m\}.$$

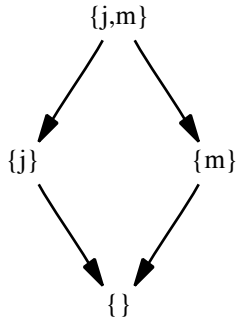
A simple idea is that names like *John* refer to elements of the universe, but Montague and Keenan and many others have argued against this idea. So we will also reject that idea and assume that no linguistic expressions refer directly to elements of E.

12.2 Properties of things

The denotations of **unary predicates** will be properties, which we will identify "extensionally," as the sets of things that have the properties. When E is the set above, there are only 4 different **properties of things**,

$$\wp(E) = \{\emptyset, \{j\}, \{m\}, \{j, m\}\}.$$

We can reveal some important relations among these by displaying them with with arcs indicating subset relations among them as follows:

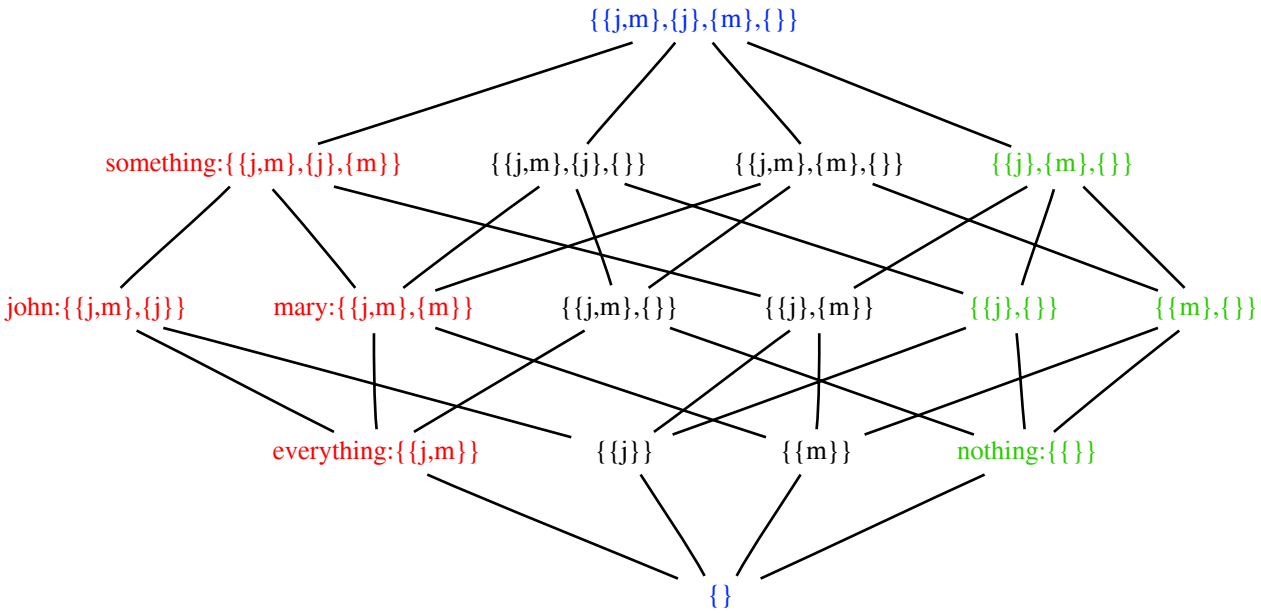


So if only John sings, we will interpret *sings* as the property $\{j\}$,

$$[\![\text{sings}]\!] = \{j\}.$$

12.3 Unary quantifiers, properties of properties of things

Now we turn to **unary quantifiers** like *something, everything, nothing, John, Mary,...*. These are will **properties of properties**, which we will identify with sets of sets of properties, $\wp(\wp(E))$. When E is the set above, there are 16 different unary quantifiers, namely,



Notice that the English words that denote some of the unary quantifiers are shown here. Notice in particular that we are treating names like *John* as the set of properties that John has.

If you are looking at this in color, we have used the colors red and blue to indicate the 6 quantifiers Q that are **increasing** in the sense that if $p \in Q$ and $r \supseteq q$ then $r \in Q$. That is, they are closed under the superset relation:

$$\{\} \quad \{\{j,m\}\} \quad \{\{j,m\},\{j\}\} \quad \{\{j,m\},\{m\}\} \quad \{\{j,m\},\{j\},\{m\}\} \quad \{\{j,m\},\{j\},\{m\},\{\}\}.$$

(If you don't have color, you should mark these yourself, to see where they are.) Notice that, on this view, names like *John* denote increasing quantifiers, as do *something* and *everything*.

And if you are looking at this in color, we have used the colors green and blue to indicate the 6 quantifiers Q that are **decreasing** in the sense that if $p \in Q$ and $r \subseteq q$ then $r \in Q$. That is, they are closed under the subset relation:

$$\{\} \quad \{\{\}\} \quad \{\{j\},\{\}\} \quad \{\{m\},\{\}\} \quad \{\{j\},\{m\},\{\}\} \quad \{\{j,m\},\{j\},\{m\},\{\}\}.$$

(If you don't have color, you should mark these yourself, to see where they are.) Notice that *nothing* denotes a decreasing quantifier.

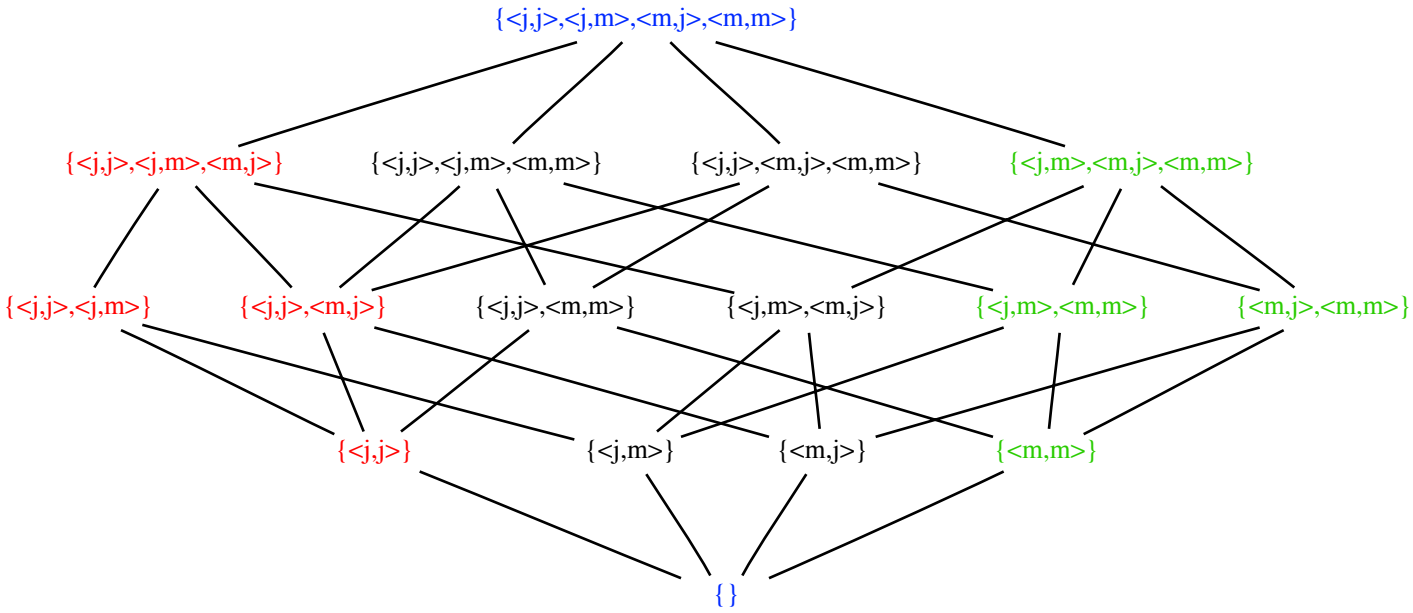
The color blue indicates the 2 quantifiers that are both increasing and decreasing, namely the top and bottom:

$$\{\} \qquad \{\{j,m\},\{j\},\{m\},\{\}\}.$$

The first of these could be denoted by the expression *something and nothing*, and the second by the expression *something or nothing*.

12.4 Binary relations among things

The denotations of **binary predicates** will be **relations among things**, which we will identify "extensionally," as the sets of pairs of things. When E is the set above, there are 16 different binary relations, namely,



So if only John loves Mary and Mary loves John, and no other love is happening, we will interpret *loves* as the property $\{\langle j,m\rangle,\langle m,j\rangle\}$,

$$[\![\text{loves}]\!] = \{\langle j,m\rangle,\langle m,j\rangle\}.$$

And the property of "being the very same thing as"

$$[\![\text{is}]\!] = \{\langle j,j\rangle,\langle m,m\rangle\}.$$

It is a good exercise to think about how each of these properties could be named, e.g.

$$[\![\text{loves and doesn't love}]\!] = \{\},$$
$$[\![\text{loves or doesn't love}]\!] = \{\langle j,j\rangle,\langle j,m\rangle,\langle m,j\rangle,\langle m,m\rangle\}.$$

Notice that some of the binary predicates are increasing, some are decreasing, and some are neither.

12.5 Binary relations among properties of things

Now we turn to **binary quantifiers**. Most English quantifiers are binary. In fact, *everything*, *everyone*, *something*, *someone*, *nothing*, *noone*, are obviously complexes built from the binary quantifiers *every, some, no* and a noun *thing, one* that specifies what "sort" of thing we are talking about. A binary quantifier is a **binary relation among properties of things**. Unfortunately, there are too many too diagram easily, because in a universe of n things, there are 2^n properties of things, and so $2^n \times 2^n = 2^{2n}$ pairs of properties, and $2^{2^{2n}}$ sets of pairs of properties of things. So in a universe of 2 things, there are 4 properties, 16 pairs of properties, and 65536 sets of pairs of properties. We can consider some examples though:

$$
\begin{array}{rll}
[\![\text{every}]\!] & = \{\langle p,q\rangle |\ p \subseteq q\} & \\
[\![\text{some}]\!] & = \{\langle p,q\rangle |\ (p \cap q) \neq \emptyset\} & \\
[\![\text{no}]\!] & = \{\langle p,q\rangle |\ (p \cap q) = \emptyset\} & \\
[\![\text{exactly N}]\!] & = \{\langle p,q\rangle |\ |p \cap q| = N\} & \text{for any } N \in \mathbb{N} \\
[\![\text{at least N}]\!] & = \{\langle p,q\rangle |\ |p \cap q| \geq N\} & \text{for any } N \in \mathbb{N} \\
[\![\text{at most N}]\!] & = \{\langle p,q\rangle |\ |p \cap q| \leq N\} & \text{for any } N \in \mathbb{N} \\
[\![\text{all but N}]\!] & = \{\langle p,q\rangle |\ |p - q| = N\} & \text{for any } N \in \mathbb{N} \\
[\![\text{between N and M}]\!] & = \{\langle p,q\rangle |\ N \leq |p \cap q| \leq M\} & \text{for any } N, M \in \mathbb{N} \\
[\![\text{most}]\!] & = \{\langle p,q\rangle |\ |p - q| > |p \cap b|\} & \\
[\![\text{the N}]\!] & = \{\langle p,q\rangle |\ |p - q| = 0 \text{ and } |p \cap q| = N\} & \text{for any } N \in \mathbb{N}
\end{array}
$$

For any binary quantifier Q we use $\uparrow Q$ to indicate that Q is (monotone) **increasing in its first argument**, which means that whenever $\langle p,q\rangle \in Q$ and $r \supseteq p$ then $\langle r,q\rangle \in Q$. Examples are *some* and *at least N*.

For any binary quantifier Q we use $Q\uparrow$ to indicate that Q Q is (monotone) **increasing in its second argument** iff whenever $\langle p,q\rangle \in Q$ and $r \supseteq q$ then $\langle p,r\rangle \in Q$. Examples are *every, most, at least N, the, infinitely many,...*.

For any binary quantifier Q we use $\downarrow Q$ to indicate that Q is (monotone) **decreasing in its first argument**, which means that whenever $\langle p,q\rangle \in Q$ and $r \subseteq p$ then $\langle r,q\rangle \in Q$. Examples are *every, no, all, at most N,...*

For any binary quantifier Q we use $Q\downarrow$ to indicate that Q Q is (monotone) **decreasing in its second argument** iff whenever $\langle p,q\rangle \in Q$ and $r \subseteq q$ then $\langle p,r\rangle \in Q$. Examples are *no, few, fewer than N, at most N,...*.

Since *every* is decreasing in its first argument and increasing in its second argument, we sometimes write $\downarrow$every$\uparrow$. Similarly, $\downarrow$no$\downarrow$, and $\uparrow$some$\downarrow$.

13 Correction: quantifiers as functionals

14 A first inference relation

We will define our inferences over derivations, where these are represented by the lexical items in those derivations, in order. Recall that this means that a sentence like

Every student sings

is represented by lexical items like this (ignoring empty categories, tense, movements, for the moment):

`sings every student`.

If we parenthesize the pairs combined by merge, we have:

`(sings (every student))`.

The predicate of the sentence selects the subject DP, and the D inside the subject selects the noun. Thinking of the quantifier as a relation between the properties **[**student**]** and **[**sing**]**, we see that the predicate **[**sing**]** is, in effect, the second argument of the quantifier.

sentence:	every	student	sings
	Q	A	B
derivation:	sings	every	student
	B	Q	A

And this sentence is true iff $\langle A, B\rangle \in Q$.

14.1 Monotonicity inferences for subject-predicate

It is now easy to represent sound patterns of inference for different kinds of quantifiers.

$$\frac{\texttt{B(Q(A))} \qquad \texttt{C(every(B))}}{\texttt{C(Q(A))}}\,[Q\uparrow] \quad \text{(for any Q}\uparrow\text{: all, most, the, at least N, infinitely many,...)}$$

$$\frac{\texttt{B(Q(A))} \qquad \texttt{B(every(C))}}{\texttt{C(Q(A))}}\,[Q\downarrow] \quad \text{(for any Q}\downarrow\text{: no, few, fewer than N, at most N,...)}$$

$$\frac{\texttt{B(Q(A))} \qquad \texttt{C(every(A))}}{\texttt{B(Q(C))}}\,[\uparrow Q] \quad \text{(for any }\uparrow\text{Q: some, at least N, ...)}$$

$$\frac{\texttt{B(Q(A))} \qquad \texttt{A(every(C))}}{\texttt{B(Q(C))}}\,[\downarrow Q] \quad \text{(for any }\downarrow\text{Q: no, every, all, at most N, at most finitely many,...)}$$

Example: Aristotle noticed that "Darii syllogisms" like the following are sound:

Some birds are swans All swans are white

Therefore, some birds are white

We can recognize this now as one instance of the $Q\uparrow$ rule:

$$\frac{\texttt{birds(some(swans))} \qquad \texttt{white(every(swan))}}{\texttt{white(some(birds))}}\,[Q\uparrow]$$

The second premise says that the step from the property ⟦bird⟧ to ⟦white⟧ is an "increase," and since we know *some* is increasing in its second argument, the step from the first premise to the conclusion always preserves truth.

Example: We can also understand the simplest traditional example of a sound inference:

Socrates is a man

All men are mortal

Therefore, Socrates is mortal

Remember we are interpreting *Socrates* as denoting a quantifier! It is the quantifier that maps a property to true just in case Socrates has that property. Let's call this quantifier `socrates`. Then, since `socrates`$\uparrow$ we have just another instance of the $Q\uparrow$ rule:

$$\frac{\texttt{socrates(man)} \qquad \texttt{mortal(every(man))}}{\texttt{socrates(mortal)}}\,[Q\uparrow]$$

Example: Aristotle noticed that "Barbara syllogisms" like the following are sound:

All birds are egg-layers

All seagulls are birds

Therefore, all seagulls are egg-layers

Since the second premise tells us that the step from ⟦birds⟧ to ⟦seagulls⟧ is a decrease and $\downarrow$all, we can recognize this now as an instance of the $\downarrow Q$ rule:

$$\frac{\texttt{egg-layer(all(bird))} \qquad \texttt{bird(every(seagull))}}{\texttt{egg-layer(all(seagull))}}\,[\downarrow Q]$$

14.2 More Boolean inferences

A first step toward reasoning with *and, or* and *not* or *non-* can be given with the following inference rules. In the first place, we have the following useful axioms, for all properties A, B:

$$(\mathsf{A}\ (\mathsf{every}\ (\mathsf{A} \wedge \mathsf{B}))) \qquad (\mathsf{A}\ (\mathsf{every}\ (\mathsf{B} \wedge \mathsf{A}))) \qquad (\mathsf{A} \vee \mathsf{B})(\mathsf{every}\ \mathsf{A}) \qquad (\mathsf{B} \vee \mathsf{A})(\mathsf{every}\ \mathsf{A})$$

These axioms just say that the step from ⟦A⟧ to ⟦A or B⟧ is an increase, and the step from ⟦A⟧ to ⟦A and B⟧ is an decrease. In other words, every A is either A or B, and everything that is A and B is A. We also have rules like this:

$$\frac{(\mathsf{B}\ (\mathsf{every}\ \mathsf{A}))}{(\mathsf{non{-}A}\ (\mathsf{every}\ \mathsf{non{-}B}))} \qquad \frac{(\mathsf{B}\ (\mathsf{no}\ \mathsf{A}))}{\mathsf{not}(\mathsf{B}\ (\mathsf{some}\ \mathsf{A}))} \qquad \frac{\mathsf{not}(\mathsf{B}\ (\mathsf{someA}))}{(\mathsf{B}\ (\mathsf{no}\ \mathsf{A}))} \qquad \frac{(\mathsf{non{-}B}\ (\mathsf{every}\ \mathsf{A}))}{(\mathsf{B}\ (\mathsf{noA}))} \qquad \frac{(\mathsf{B}\ (\mathsf{no}\ \mathsf{A}))}{(\mathsf{non{-}B}\ (\mathsf{every}\ \mathsf{A}))}$$

Example: Many adjectives are "intersective" in the sense that [A N] signifies ($[\![A]\!] \wedge [\![N]\!]$). For example, *Greek student* signifies (`Greek` $\wedge$ `student`). Allowing ourselves this treatment of intersective adjectives, we have

Every student sings
Therefore, every Greek student sings

We can recognize this now as one instance of the $Q\uparrow$ rule:

$$\frac{\texttt{(sings (every student))} \qquad \texttt{(student(every (Greek}\wedge\texttt{student)))}}{\texttt{sings(every (Greek}\wedge\texttt{student))}} \; [\downarrow Q]$$

The second premise says that the step from the property ⟦student⟧ to ⟦Greek student⟧ is a "decrease," and since we know $\downarrow$*every*, the step from the first premise to the conclusion preserves truth. Notice that this does not work with other quantifiers!

every student sings ⇒ every Greek student sings
every Greek student sings ⇏ every student sings
some student sings ⇏ some Greek student sings
some Greek student sings⇒ some student sings
exactly 1 student sings ⇏ exactly 1 Greek student sings
exactly 1 Greek student sings ⇏ exactly 1 student sings

15 Exercises

1. Consider the inferences below, and list the English quantifiers that make them always true. Try to name at least 2 different quantifiers for each inference pattern:

   a. $\frac{B(Q(A))}{(A \wedge B)(Q(A))}$ [*conservativity*] (*for any conservative quantifier Q*)

   What English quantifiers are conservative?

   b. $\frac{B(Q(A)) \qquad C(Q(B))}{C(Q(A))}$ [*transitivity*] (*for any transitive quantifier Q*)

   What English quantifiers are transitive?

   c. $\frac{B(Q(A))}{A(Q(B))}$ [*symmetry*] (*for any symmetric quantifier Q*)

   What English quantifiers are symmetric?

   d. $\frac{}{A(Q(A))}$ [*reflexivity*] (*for any reflexive quantifier Q*)

   What English quantifiers are reflexive?

   e. $\frac{B(Q(A))}{B(Q(B))}$ [*weak reflexivity*] (*for any weakly reflexive quantifier Q*)

   What English quantifiers are weakly reflexive?

2. Following the examples in the previous sections, do any of our rules cover the following "Celarent syllogism"? (If not, what rule is missing?)

   No mammals are birds
   All whales are mammals
   Therefore, no whales are birds

   (I think we did this one in a rush at the end of class? So I am not sure we did it right, but it's not too hard)

3. Following the examples in the previous sections, do any of our rules cover the following "Ferio syllogism"? (If not, what rule is missing?)

   No student is a toddler
   Some skaters are students
   Therefore, some skaters are not toddlers

15.1 Monotonicity inferences for transitive sentences

Transitive sentences like *every student loves some teacher* contain two quantifiers *every, some*, two unary predicates (nouns) *student, teacher*, and a binary predicate *loves*. Ignoring quantifier raising and other movements, a simple idea is that the sentence is true iff $\langle [\![\text{student}]\!], [\![\text{lovessometeacher}]\!] \rangle \in [\![\text{every}]\!]$, where [loves some teacher] is the property of loving some teacher. This is slightly tricky, but is explained well in (Keenan, 1989), for example:

sentence:	every	student	loves	some	teacher
	Q_1	A	R	Q_2	B
derivation:	loves	some	teacher	every	student
	R	(Q_2	A)	(Q_1	B)

And this sentence is true iff $\langle A, \{a|\ \langle B, \{b|\ \langle a, b\rangle \in R\}\rangle \in Q_2\rangle \in Q_1$.

Example: Before doing any new work on transitive sentences, notice that if you keep the object fixed, then our subject-predicate monotonicity rules can apply. Consider

Every student reads some book

Therefore, every Greek student reads some book

As before, this is an instance of the $Q\uparrow$ rule:

$$\frac{\texttt{(reads some book)(every student))} \qquad \texttt{student(every(Greek}\wedge\texttt{student))}}{\texttt{(reads some book)(every (Greek}\wedge\texttt{student))}} \quad [\downarrow Q]$$

The thing we are missing is how to do monotonicity reasoning with the object quantifier.

We noticed in §?? that some binary relations R are increasing, and some are decreasing.

15.2 Monotonicity inference: A more general and concise formulation

Adapted from (Fyodorov, Winter, and Francez, 2003; Bernardi, 2002; Sanchez-Valencia, 1991; Purdy, 1991).

1. **Label the derivation** (i.e. the lexical sequence) as follows:
   i. bracket all merged pairs
   ii. label all quantifiers with their monotonicities in the standard way:

   e.g. ↓every↑, ↑some↑, ↓no↓, ↑not-all↓, ~most↑, ~exactly 5~

   iii. for $\circ, \diamond$ in $\{\uparrow, \downarrow, \sim\}$, label $(\circ Q \diamond\ A)$ as $(\circ Q \diamond\ A^{\circ})$

   $$(\downarrow \text{every}\uparrow\ \text{student}^{\downarrow})$$

   iv. for $\circ, \diamond$ in $\{\uparrow, \downarrow, \sim\}$, label $(P(\circ Q \diamond\ A^{\circ})$ as $(P^{\diamond}(\circ Q \diamond\ A^{\circ})$

   $$(\text{sing}^{\uparrow}\ (\downarrow \text{every}\uparrow\ \text{student}^{\downarrow}))$$
   $$(\text{praise}^{\uparrow}\ (\downarrow \text{every}\uparrow\ \text{student}^{\downarrow}))$$
   $$((\text{praise}^{\uparrow}\ (\downarrow \text{every}\uparrow\ \text{student}^{\downarrow})^{\uparrow}(\uparrow \text{some}\uparrow\ \text{teacher}^{\uparrow})))$$

   v. label outermost parentheses ↑ (or if no parentheses at all, label single element ↑)
2. Each constituent A **has the superscripts** of the constituents containing it and its own.
   Letting $\uparrow = 1$, $\downarrow = -1$, $\sim = 0$, the **polarity of A** is the polarity p=the product of its superscripts.
3. Then for any expression with constitutent A with non-zero polarity p, we have the rule mon:

$$\frac{(\ldots A \ldots) \qquad A \leq^{p} B}{(\ldots B \ldots)}$$

That is, you can increase any positive constituent, and decrease any negative one.

Example: What inferences do we have of the form:

$$\frac{\text{Every student carefully reads some big book}}{\text{Therefore,...}}$$

We parse and label the premise:

$$((\texttt{carefully reads})^{\uparrow}\ (\texttt{some}\ (\texttt{big book})^{\uparrow}))^{\uparrow}(\texttt{every student}^{\downarrow}))^{\uparrow}$$

In this expression, we see these polarities

+1 (carefully reads)
+1 (big book)
+1 (carefully reads some big book)
-1 student

So we have the following inferences

$$\frac{\texttt{(carefully reads some big book)(every student)} \qquad \texttt{(Greek}\wedge\texttt{student)}\leq\texttt{student}}{\texttt{(carefully reads some big book)(every (Greek}\wedge\texttt{student))}}\ [mon]$$

$$\frac{\texttt{(carefully reads some big book)(every student)} \qquad \texttt{(carefully reads)}\leq\texttt{reads}}{\texttt{(reads some big book)(every student)}}\ [mon]$$

$$\frac{\texttt{(carefully reads some big book)(every student)} \qquad \texttt{(big book)}\leq\texttt{book}}{\texttt{(carefully reads some book)(every student)}}\ [mon]$$

Example: What inferences do we have of the form:

$$\frac{\text{No student reads every big book}}{\text{Therefore,...}}$$

We parse and label the premise:

$$(\texttt{reads}^{\uparrow}\ (\texttt{every}\ (\texttt{big book})^{\downarrow}))^{\downarrow}(\texttt{no student}^{\downarrow}))^{\uparrow}$$

In this expression, we see

-1 (reads)
+1 (big book)
-1 (reads some big book)
-1 student

So we have the following inferences

$$\frac{\texttt{(reads every big book)(no student)} \qquad \texttt{(Greek}\wedge\texttt{student)}\leq\texttt{student}}{\texttt{(reads every big book)(no (Greek}\wedge\texttt{student))}}\ [mon]$$

$$\frac{\texttt{(reads every big book)(no student)} \qquad \texttt{(carefully reads)}\leq\texttt{reads}}{\texttt{(carefully reads every big book)(no student)}}\ [mon]$$

$$\frac{\texttt{(reads every big book)(no student)} \qquad \texttt{(big book)}\leq\texttt{book}}{\texttt{(reads every book)(no student)}}\ [mon]$$

This logic is "natural" compared to the more standard modus-ponens-based first order logics, but it is not too hard to find examples that reveal that it is still "inhumanly logical."

Example: We have not had time to discuss relative clauses. Let's just suppose for the moment that they are adjoined intersective modifiers:

everyone who reads Hamlet = (every (person $\wedge$ reads Hamlet))

And let's interpret *lover* as *one who loves some person*. Then we are in a position to see that the following inference is sound:

$$\frac{\text{Everyone loves a lover} \qquad \text{Romeo loves Juliet}}{\text{Therefore, Bush loves Bin Laden}}$$

We haven't dealt with pronouns, but if we just treat *my baby* as a name and replace *me* and *I* by a name (e.g. yours) then we can establish the soundness of:

$$\frac{\text{Everyone loves my baby} \qquad \text{My baby don't love nobody but me}}{\text{Therefore, I am my baby}}$$

(Fyodorov, Winter, and Francez, 2003) provides a decision method for the logic that has the quantifiers, disjunction and conjunction. The idea is simply this: given finitely many axioms, since each expression has only finitely many constituents, we can exhaustively explore all possible proofs to find whether any are complete in the sense of deriving from the axioms.

An implementation of this method for MGs is not quite ready, but Fyodorov has an implementation with a web-interface at

`http://www.cs.technion.ac.il/~yaroslav/oc/`

16 Harder problems

16.1 Semantic categories

Two expressions can be regarded as having the same semantic category if they make similar contributions to the semantic properties of the expressions they occur in.

> …we need to find room for a conception of something from which an expressions inferential properties may be regarded as flowing.…Just such a conception is provided for by what I shall call an interpretational semantics. A semantic theory of that type specifies, for each kind of semantic expression, an entity – a set, a truth value, a function from sets to truth values, or whatever, which may appropriately be assigned to members of that kind upon an arbitrary interpretation of the language. We can regard the specification of the kind of assignment as a specification of the underlying real essence which a word has in common with many other words, and of which the validity of certain inferences involving it is a consequence.…we aim at the sort of illumination that can come from an economical axiomatization of the behaviour of groups of expressions. Then we can say: '*This* is the kind of expression *e* is, and that is why these inferences are valid.' (Evans, 1976, pp61,63)

The simplest idea would be that this task uses the very same categories that we have needed to distinguish in syntax, but this is not what we find.[49] For example, as noted by Geach (1962), while it is natural to regard *a cat* as referring to a particular cat in a typical utterance of (1a), this is not natural in the case of (1b):

(1) a. a cat scratched me yesterday
    b. Jemima is a cat

Similarly, to use an example from Williams (1983), we mean a particular tree in typical utterances of (2a), this is not natural in the case of (2b):

(2) a. I planted a tree me yesterday
    b. Every acorn grows into a tree

In the examples above, we plausibly have the same syntactic element playing quite different semantic roles. If that is correct, then we also discover different kinds of syntactic elements playing the same semantic role. That is, in the previous examples, it seems that the indefinite noun phrases are playing the same syntactic role as would be played by an adjective phrase like *feline* or *old*:

(3) a. Jemima is feline
    b. Every acorn grows old

Adjective phrases and determiner phrases have different distributions in the language, but they can play the same roles in certain contexts.

There are also cases where different elements of the same syntactic category play different semantic roles. To take just one familiar case, discussed by Montague (1969), and many others: the syntactic roles of the adjectives *fake* and *Norwegian* may be the same, but they have very different roles in the way they specify what we are talking about:

(4) a. Every Norwegian student is a student and Norwegian
    b. Fake diamonds typically are not diamonds. Often they are glass, and not fake glass.

[49] Evans makes note of this too, saying "A logically perfect language would have a one-to-one correspondence between its semantic and syntactic categories. I see no reason to suppose that natural languages are logically perfect, at any level. There can be a breakdown of the one-to-one correspondence in either direction. We have find it necessary to subdivide a syntactically unitary category…And equally, we may find it convenient to make assignments of the same kind to expressions of different syntactic categories. (Evans, 1976, pp71-72)" But he does not seriously raise the question of why human languages would all fail to be "logically perfect" in this sense.

All these examples suggest that syntactic and semantic categories do not correspond in human languages. Why would this be true? The matter remains obscure and poses another criterion of adequacy on our theories: while parsing specifies how syntactic elements are combined into a syntactic complexes, the semantics needs to specify how the semantic elements are combined to determine the semantic values of those same complexes.

Given this state of affairs, and given that we would like to compute inference relations, it seems that our representation of lexical knowledge needs to be augmented with some indication of semantic category. In previous chapters, the lexicons contained only phonological/orthographic and syntactic information:

phonological/orthographic form::syntactic features

For our new concerns, we can elaborate our lexical entries as follows:

phonological/orthographic form::syntactic features::semantic features.

A first, basic account of what some of the semantic features is usually given roughly as follows.

1. First, we let names denote in some set of individuals e, the set of all the things you could talk about. We can add this information to our names like this:

   Titus::D -k::e

2. Simple intransitive predicates and adjectives can be taken as representing properties, and we can begin by thinking of these as sets, as the function from individuals to truth values, $e \to t$, that maps an individual to true iff it has the property.

   laugh::V::$e \to t$ happy::A::$e \to t$

   Simple transitive verbs and adjectives take two arguments:

   praise::=D +k V::$e \to e \to t$ proud::=p A::$e \to e \to t$

3. For the moment, we can dodge the issue of providing an adquequate account of tense T by simply interpreting each lexical item in this as the identity function. We will use id to refer to the identity function, so as not to confuse it with the symbol we are using for selection.

   -s::=v T::id

   We will do the same thing for all elements in in the functional categories Num, Be, Have, C, a, v, and p.

   Then we can interpret simple intransitive sentences. First, writing + for the semantic combinations we want to make

   $[\![\texttt{C(-s(be(a(mortal(Titus)))))}]\!] = id + (id + (id + (id + (e \to t : \text{mortal} + (e : \text{Titus})))))$.

   Now suppose that we let the semantic combinations be forward or backward application[50] In this case, forward application suffices:

   $$\begin{aligned}[\![\texttt{C(-s(be(a(mortal(Titus)))))}]\!] &= id + id + id + id + e \to t : \text{mortal}(e : \text{Titus}))))) \\ &= e \to t : \text{mortal}(e : \text{Titus}) \\ &= t : \text{mortal}(\text{Titus})\end{aligned}$$

4. While a sentence like *Titus be -s mortal* entails that something is mortal, a sentence like *no king be -s mortal* obviously does not.

   In general, the entailment holds when the subject of the intransitive has type e, but may not hold when it is a quantifier, which we will say is a function from properties to truth values, a function of type $(e \to t) \to t$.

   To get this result, we will say that a determiner has type $(e \to t) \to (e \to t) \to t$. Then the determination of semantic values begins as follows:

   $$\begin{aligned}&[\![\texttt{C(-s(be(a(mortal(no(Num(king)))))))}]\!] \\ &\quad = id + id + id + id + e \to t : \text{mortal} + (e \to t) \to (e \to t) \to t : \text{no} + (id + ((e \to t) : \text{king})) \\ &\quad = e \to t : \text{mortal} + (e \to t) \to (e \to t) \to t : \text{no}((e \to t) : \text{king})) \\ &\quad = t : \text{no}(\text{king})(\text{mortal})\end{aligned}$$

[50] An alternative is to use forward application and Curry's lifting combinator $\mathbf{C}_*$ which is now more often called **T** for "type raising" (Steedman, 2000; Smullyan, 1985; Curry and Feys, 1958; Rosser, 1935).

16.2 Contextual influences

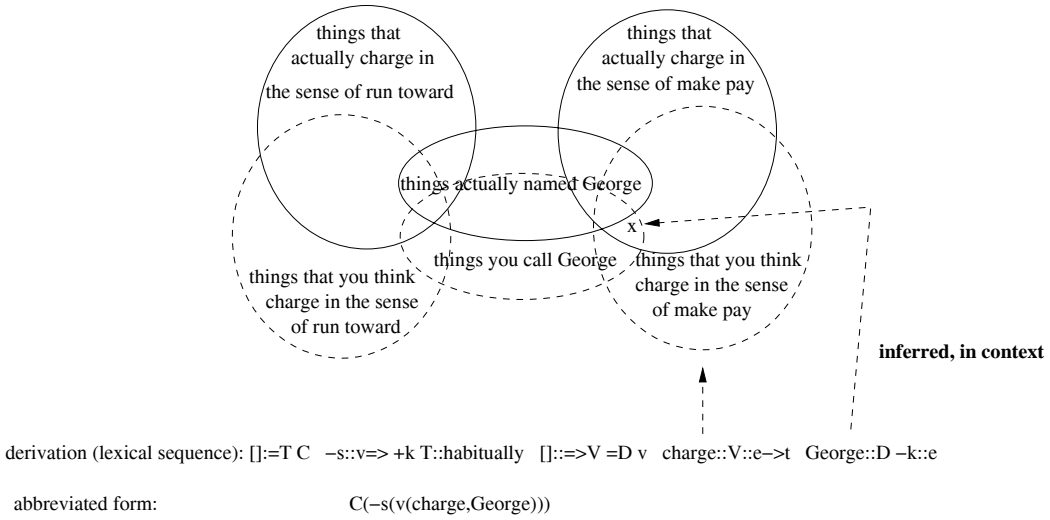
This is obvious with the various pronouns and tense markings found in human languages. A standard way to deal with this is found in Tarski's famous and readable treatment of expressions like $p(X)$ in first order logic (Tarski, 1935), and now in almost every introductory logic text. The basic idea is to think of each (declarative) sentence itself as having a value which is a function from contexts to truth values.

We will postpone the discussion of pronouns and tenses, but we can extend the basic idea of distinguishing the linguistic and non-linguistic components of the problem to a broad range of contextual influences. For example, if in a typical situation I say *George charge -s*, I may expect you to understand a particular individual by *George*, and I may expect you to figure out something about what I mean by *charge*. For example, George could be a horse, and by *charge* I may mean *run toward you*. Or George could be a grocer, and I may mean that he makes you pay for your food. This kind of making sense of what an utterance means is clearly something that goes on when we understand fluent language, but it is regarded as non-linguistic because the process apparently involves general, non-linguistic knowledge about the situation. Similarly deciding what the prepositional phrases attach to in sentences like the following clearly depends on your knowledge of what I am likely to be saying, what kinds of nouns name common currencies, etc.

I bought the lock with the keys I bought the lock with my last dollar
I drove down the street in my car I drove down the street with the new asphalt

The decisions we make here are presumably non-linguistic too.

Suppose that we have augmented our lexical items for names with an indication that they denote a member of the set e of individuals, and that we have augmented the intransitive verb $e \rightarrow t$ to indicate that it can be regarded as a property, a function that tells you of each thing $x \in e$ whether it's true that x has the property. Then the particular circumstances of utterance are typically relied upon to further clarify the matter, so that we know which individual is intended by *George*, and which sense of charge is intended by *charge*. We could depict the general situation like this, indicating the inferences made by the listener with dashed lines:



For example, the listener could make a mistake about which sense of *charge* is intended, and the listener could also be under some misconceptions about which things, exactly, really do charge in either sense. Either of these things could make the listeners judgement about the truth of the sentence incorrect. For our purposes, the important question is: What is the nature of the inferential processes here?

It has been generally recognized since the early studies of language in the tradition of analytic philosophy, and since the earliest developments in modern formal semantics, that the problem of determining the intended reading of a sentence, like the problem of determining the intended reference of a name or determiner phrase is (i) non-linguistic, potentially involving virtually any aspect of the listener's knowledge, and (ii)

non-demonstrative and defeasible, that is, prone to error and correction. In fact these inferences are widely regarded as fundamentally beyond the analytical tools available now. See, e.g., Chomsky (1968, p6), Chomsky (1975, p138), Partee (1975, p80), Kamp (1984, p39), Fodor (1983), Putnam (1986, p222), and many others.[51] Putnam argues that

> …deciding – at least in hard cases – whether two terms have the same meaning or whether they have the same reference or whether they *could* have the same reference may involve deciding what is and what is not a good scientific explanation.

For the moment, then, we will not consider these parts of the problem, but see the further discussion of these matters in §16.5.1 and in §**??** below.

Fortunately, we do not need the actual semantic values of expressions in order to recognize many entailment relations among them, just as we do not need to actually interpret the sentences of propositional logic or of prolog to prove theorems with them.

16.3 Meaning postulates

To prove the theorems we want, theorems of the sort shown on 232, we often need to know more than just the type of semantic object we are dealing with. To take a simple example, we have allowed coordinators to combine sentences that express truth values, so both *and* and *or* presumably have the type $t \rightarrow t \rightarrow t$, but they are importantly different. To capture the difference, we need some additional information that is specific to each of these lexical items.[52] Among the important inferences that the language user should be able to make are these, which a logician will find familiar. For all structures Δ, Γ:

$$\frac{C(\Delta) \quad C(\Gamma)}{C(and, C(\Delta), C(\Gamma))} \qquad \frac{C(\Delta)}{C(or, C(\Delta), C(\Gamma))} \qquad \frac{C(\Gamma)}{C(or, C(\Delta), C(\Gamma))}$$

$$\frac{C(\text{and}, C(\Delta), C(\Gamma)}{C(\Delta)} \qquad \frac{C(\text{and}, C(\Delta), C(\Gamma)}{C(\Gamma)}$$

For the quantifiers, we can can generalize Aristotle's syllogistic approach to a more general reasoning with monotonicity properties in intransitive sentences. Here we just leave out the functional categories, and use

[51] Philosophers, trying to make sense of how language could be learned and how the claims we make in language are related to reality have worried about the two error-prone steps in the picture above: the assessment of what the speaker intends and then the assessment of what things actually are charging in the intended sense. Are these possible sources of error always present? Suppose that instead of interpreting what someone else said, you are interpreting what you said to yourself, or what you just wrote in your diary, or some such thing. In the typical case, this would reduce the uncertainty about what was intended. (But does it remove it? What about self-deception, memory lapses, etc?) And suppose that instead of talking about something abstract like charging (in any sense), we are talking about something more concrete and directly observable. Then we could perhaps reduce the uncertainty about the actual extensions of our predicates. (But again, can the uncertainty be removed? Even in claims about your own sensations, this is far from clear. And furthermore, even if the uncertainty were reduced for certain perceptual reports that you make to yourself, it is very unclear how the more interesting things you know about could perch on these perceptual reports for their foundation.) The search for a secure empirical foundation upon which human knowledge could rest is often associated with the "positivist tradition" in philosophy of science, in the work of Carnap, Reichenbach and others in the mid 1900's. These attempts are now generally regarded as unsuccessful (Quine, 1951b; Fodor, 1998, for example), but some closely related views are still defended (Boghossian, 1996; Peacocke, 1993, for example).

[52] Logicians and philosophers have sometimes assumed that the rules for quantifiers and for the propositional operators would not need to be given in this lexically specific fashion, but that they might be "structural" in the sense that the validity of the inferences would follow from their semantic type alone. In our grammar, though, we have not found any motivation for distinguishing semantic types for each of the coordinators. This kind of proposal was anticipated and discussed in the philosophical literature, for example in the following passage:

> …with the exception of inferences involving substitution of sentences with the same truth value, none of the standard inferences involving sentential connectives is structurally valid. Briefly, the sentences 'P and Q' and 'P or Q' have the same structure; the former's entailing P is due to the special variation the word 'and' plays upon a theme it has in common with 'or'. Quantifiers are more complicated but they too can be seen as falling into a single semantic category…(Evans, 1976, pp64-66)

A, B for the predicates and *Q* for the determiner in sentences like *every student laugh -s*, *Q A B*, which gets a syntactic analysis of the form *B(Q(A))* since *Q* selects *A* and then *B* selects *Q*. We capture many entailment relations among sentences of this form with schemes like the following, depending on the determiners *Q*.[53]

$$\frac{\texttt{B(Q(A))} \qquad \texttt{C(every(B))}}{\texttt{C(Q(A))}} \quad [Q\uparrow] \quad (\textit{for any right monotone increasing } Q\textit{: all, most, the, at least } N\textit{, infinitely many},\ldots)$$

$$\frac{\texttt{B(Q(A))} \qquad \texttt{B(every(C))}}{\texttt{C(Q(A))}} \quad [Q\downarrow] \quad (\textit{for any right monotone decreasing } Q\textit{: no, few, fewer than } N\textit{, at most } N,\ldots)$$

$$\frac{\texttt{B(Q(A))} \qquad \texttt{C(every(A))}}{\texttt{B(Q(C))}} \quad [\uparrow Q] \quad (\textit{for any left monotone increasing } Q\textit{: some, at least } N, \ldots)$$

$$\frac{\texttt{B(Q(A))} \qquad \texttt{A(every(C))}}{\texttt{B(Q(C))}} \quad [\downarrow Q] \quad (\textit{for any left monotone decreasing } Q\textit{: no, every, all, at most } N\textit{, at most finitely many},\ldots)$$

There is an absolutely fundamental insight here: substitution of a "greater" constituent is sound in a increasing context, and substitution of a "lesser" constituent is sound in an decreasing context. It is worth spelling out this notion of "context" more carefully.

A competent language user learns more specific information about each verb too, some of which can be encoded in schemes roughly like this:

$$\frac{\texttt{v(praise(Obj),Subj)}}{\texttt{v(think,Subj)}} \qquad \frac{\texttt{v(prefer(Obj),Subj)}}{\texttt{v(think,Subj)}} \qquad \frac{\texttt{v(doubt(Obj),Subj)}}{\texttt{v(think,Subj)}} \qquad \frac{\texttt{v(wonder(Obj),Subj)}}{\texttt{v(think,Subj)}}$$

$$\frac{\texttt{v(eat(Obj),Subj)}}{\texttt{v(eat,Subj)}} \qquad \frac{\texttt{v(eat,Subj)}}{\texttt{v(eat(some(thing)),Subj)}}$$

[53] Since ↑every↓, the "Barbara" syllogism is an instance of the rule $[Q\uparrow]$. Since ↓no↓, the "Celarent" syllogism is an instance of the rule $[\downarrow Q]$.

16.4 Scope inversion

That idea has been developed to apply to the data above in recent work by Beghelli and Stowell (1996) and Szabolcsi (1996).

This strategy may seem odd, but Szabolcsi (1996) notes that we may be slightly reassured by the observation that in some languages, we seem to find overt counterparts of the "covert movements" we are proposing in English. For example, Hungarian exhibits scope ambiguities, but there are certain constructions with "fronted" constituents that are scopally unambiguous:

(5) a. Sok ember mindenkit felhívott
many man everyone-ACC up-called
'Many men phoned everyone'
where `many men` < `everyone`

b. Mindenkit sok ember felhívott
everyone-ACC many man up-called
'Many men phoned everyone'
where `everyone` < `many men`

(6) a. Hatnál több ember hívott fel mindenkit
six-than more man called up everyone-ACC
'More than six men phoned everyone'
where `more than six men` < `everyone`

b. Mindenkit hatnál több ember hívott fel
everyone-ACC six-than more man called up
'More than six men phoned everyone'
where `everyone` < `more than six men`

Certain other languages have scopally unambiguous fronted elements like this, such as KiLega and Palestinian Arabic. Scrambling in Hindi and some Germanic languages seems to depend on the "specificity" of the scrambled element, giving it a "wide scope."

To account for these and many other similar observations, Beghelli and Stowell (1996) and Szabolcsi (1996) propose that determiner phrases occupy different positions in structure according to (inter alia) the type of quantifiers they contain. Furthermore, following the recent tradition in transformational grammar, they assume that every language has structures with special positions for topicalized and focused elements, though languages will differ according to whether the elements in these positions are pronounced there or not.

We can implement this kind of proposal quite easily. First, let's distinguish five categories of determiners:

wh-QPs	(`which, what`)
neg(ative)-QPs	(`no, nobody`)
dist(ributive)-QPs	(`every, each`)
count-QPs	(`few, fewer than five, six,`...)
group-QPs	(optionally, `the, some, a, one, three,` ...)

We assume that these categories can cause just certain kinds of quantified phrases to move.

Both Beghelli and Stowell (1996) and Szabolcsi (1996) propose that the clause structure be elaborated with new functional heads: not because those heads are ever overt, but just in order to provide specifier positions for the various kinds of quantifier phrases. In our framework, multiple specifiers are allowed and so we do not need the extra heads. Furthermore, introducing extra heads between T and v would disrupt the affix-hopping analysis of English proposed in §10.2.1, since affix hopping is not recursive in the way that verb raising is: one affix hop cannot feed another. Also, Beghelli and Stowell (1996, p81) propose that Dist can license any number

of -dist elements, either by putting them in multiple specifiers or by some kind of "absorption." This would require some revisions discussed in §10.6.3, so for the moment we will restrict our attention to sentences with just one quantifier of each kind.

With a multiple specifier approach, we simply augment the licensing capabilities of our functional categories as follows:

C	licenses wh and group
T	licenses k and count
Dist	licenses dist
Share	licenses group
Neg	licenses neg

We can just add these assumptions to the grammar by *first*, modifying our entries for C and T with the new options:

ϵ::=T +group C	ϵ::=>T +group C	ϵ::=>T +wh +group C	ϵ::=T +wh +group C
ϵ:: =v +k T	ϵ:: =v +k +count T	ϵ:: =Dist +k T	ϵ:: =Dist +k +count T
ϵ:: =Share +k T	ϵ:: =Share +k +count T	ϵ:: =Neg +k T	ϵ:: =Neg +k +count T
ϵ:: =v +k T	ϵ:: =v +k +count T		

And *second*, we add the entries for the new projections:

ϵ:: =Share +dist Dist	ϵ:: =Neg +dist Dist	ϵ:: =v +dist Dist
	ϵ:: =Neg +group Share	ϵ:: =v +group Share
		ϵ:: =v +neg Neg

Finally, we modify our entries for the determiners as follows:

which:: =N D -k -wh	what:: =N D -k -wh		
no:: =N D -k -neg	every:: =N D -k -dist	few:: =N D -k -count	
the:: =N D -k -group	some:: =N D -k -group	one:: =N D -k -group	two:: =N D -k -group
the:: =N D -k	some:: =N D -k	one:: =N D -k	two:: =N D -k

With these additions we get derivations like this:

XXX

16.4.1 Binding and control

16.4.2 Discourse representation theory

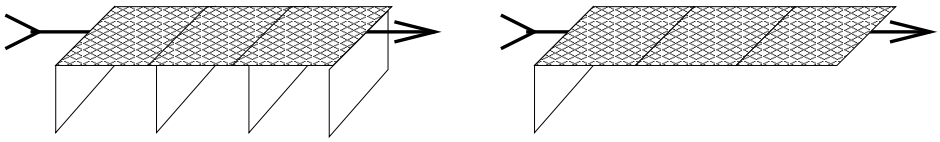
16.5 Inference

16.5.1 Reasoning is shallow

It is often pointed out that commonsense reasoning is "robust" and tends to be shallow and in need of support from multiple sources, while scientific and logical inference is "delicate" and relies on long chains of reasoning with very few points of support. Minsky (1988, pp193,189) puts the matter this way:

That theory is worthless. It isn't even wrong! – WOLFGANG PAULI

> As scientists we like to make our theories as delicate and fragile as possible. We like to arrange things so that if the slightest thing goes wrong, everything will collapse at once!…
>
> Here's one way to contrast logical reasoning and ordinary thinking. Both build chainlike connections between ideas…



Commonsense Reasoning

Mathematical Logic

> Logic demands just one support for every link, a single, flawless deduction. Common sense asks, at every step, if all of what we've found so far is in accord with everyday experience. No sensible person ever trusts a long, thin chain of reasoning. In real life, when we listen to an argument, we do not merely check each separate step; we look to see if what has been described so far seems plausible. We look for other evidence beyond the reasons in that argument.

Of course, shallow thinking can often be wrong too! In fact, it seems that in language understanding, there are many cases where we seem to make superficial assumptions even when we know they are false. For example, Kamp (1984, p39) tells the following story which I think is now out of date.

> We are much assisted in our making of such guesses [about the referents of pronouns] by the spectrum of our social prejudices. Sometimes, however, these may lead us astray, and embarassingly so, as in the following riddle which advocates of Women's Lib have on occasion used to expose members of the chauvanistic rearguard: In a head-on collision both father and son are critically wounded. They are rushed into a hospital where the chief surgeon performs an emergency operation on the son. But it is too late and the boy dies on the operating table. When an assistant asks the surgeon, 'Could you have a look at the other victim?', the surgeon replies 'I could not bear it. I have already lost my son.' Someone who has the built-in conception that the chief surgeons are men will find it substantially more difficult to make sense of this story than those who hold no such view.

What is interesting in the present context is that this story was puzzling in 1984 even for people who knew perfectly well that many surgeons were women, because the stereotypical surgeon was still a man. That is, superficial reasoning can relies on stereotypes that are false, and to be clear to your audience it is important to state things in a way that anticipates and avoids confusions that may be caused by them. The role of superficial assumptions has been explored in studies of conceptual "prototypes" and human language processing (Lynch, Coley, and Medin, 2000; Dahlgren, 1988; Smith and Medin, 1981; Rosch, 1978)

As mentioned in §16.5.1, it could be that the reasoning is actually not as unbounded as it seems, because it must be shallow. For example, It is historical and literary knowledge that Shakespeare was a great poet, but the knowledge of the many common Shakespearean word sequences is linguistic and perfectly familiar to most speakers. If we start thinking of familiar phrasing as a linguistic matter, this could actually take us quite far into what would have been regarded as world knowledge.[54]

[54]This kind of linguistic knowledge is often tapped by cluse for crossword puzzles. Although solving crossword puzzles from clues involves many domains of human knowledge, it draws particularly on *how that knowledge is conventionally represented in language*,

16.5.2 A simple reasoning model: iterative deepening

Depth-first reasoning, pursuing one line of reasoning to the end (i.e. to success, to failure in which case we backtrack, or to nontermination) is not a reasonable model of the kind of superficial reasoning that goes on in commonsense understanding of entailment relations among sentences. Really, it is not clear what kind of model could even come close to explaining human-like performance, but we can do better than depth-first.

One idea that has been used in theorem-proving and game-playing applications is "iterative deepening" (Korf, 1985; Stickel, 1992). This strategy searches for a shallow proof first (e.g. a proof with depth = 0), and then if one is not found at that depth, increases the depth bound and tries again. Cutting this search off at a reasonably shallow level will have the consequence that the difficult theorems will not be found, but all the easy ones will be.

Since in our application, the set of premises and inference schemes (the meaning postulates) may be very large, and since we will typically be wanting to see whether some particular proposition can be proven, the most natural strategy is "backward chaining:" we match the statement we want to prove with a conclusion of some inference scheme, and see if there is a way of stepping back to premises that are accepted, where the number of steps taken in this way is bounded by the iterative deepening method.

Exercises: I did not get the implementation of the reasoning system together quickly enough to give an exercise with it yet, so this week there are just two extra credit questions. (That means: you can take a vacation on this homework if you want.)

Extra credit: One of the ways to support Minsky's idea on page **??** that our commonsense reasoning is shallow is to notice that some arguments taking only a few steps are nevertheless totally unnoticed.

One example that I heard from Richard Cartwright is this one. He noticed that if you take it literally, the song lyric

everyone loves my baby, but my baby doesn't love anyone but me.

implies that I am my baby. The reasoning is simple:

i. To say 'my baby doesn't love anyone but me' means that for all X, if my baby loves X, then X=me!

ii. If everyone loves my baby, then my baby loves my baby.

iii. By i and ii, I have to be my baby

Another example is this one, from George Boolos. It is commonly said that

1. everyone loves a lover

and everyone knows that

2. Romeo loves Juliet.

So here is the exercise:

a. It follows from 1 and 2 that Bush loves Bin-Laden. Explain why.

b. Why don't people notice this fact? (Do you think Minsky is right that we don't notice because we just do not do this kind of logical reasoning, even for one or two steps? Just give me your best guess about this, in a sentence or two.)

and so theories about crossword solving overlap with language modeling methods to a rather surprising degree! (Keim et al., 1999, for example).

Extra credit: Jurafsky and Martin (1999, p503) present the following four semantic representations for the sentence *I have a car*:

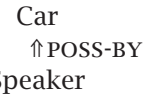
First order predicate calculus:

$$\exists x, y\ Having(x) \wedge Haver(Speaker, x) \wedge HadThing(y, x) \wedge Car(y)$$

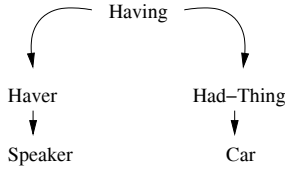
Frame-based representation:

```
Having
    Haver:   Speaker
    HadThing:   Car
```

Conceptual dependency diagram:

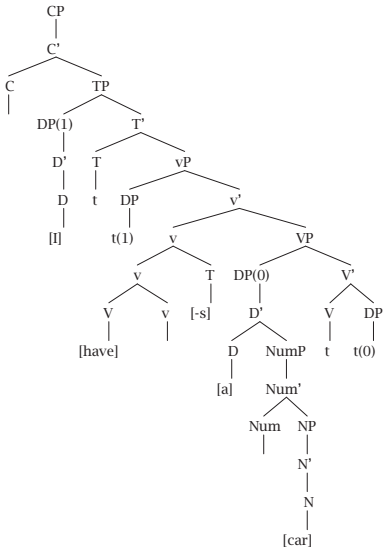


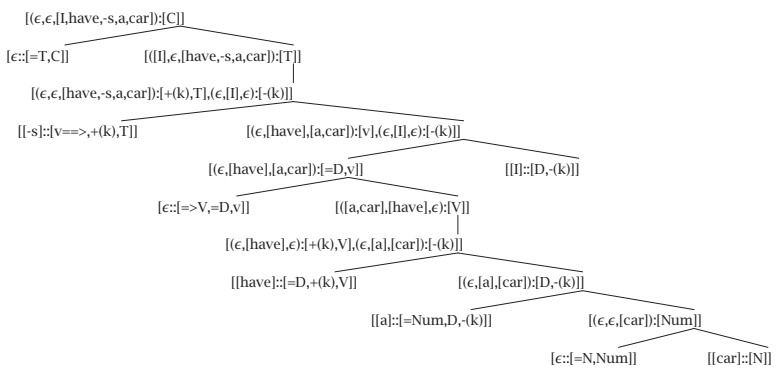
Semantic network:



They say (p502)"…there are a number of significant differences among these four approaches to representation…" but then later they say of the latter three representations (p539) "It is now widely accepted that meanings represented in these approaches can be translated into equivalent statements in [first order predicate calculus] with relative ease."

Contrasting with all of these representations, our representation of "I have -s a car" is just its derivation:





which we can abbreviate unambiguously by the lexical sequence

ϵ::=T C -s::v=> +k T ϵ::=>V =D v have::=D +k V a::=Num D -k ϵ::=N Num car::N I::D -k

or, for convenience, even more briefly with something like:

C(-s(v(have(a(Num(car))),I))) or even have(a(car),I).

These structures cannot generally be translated into the first order predicate calculus, since they can have non-first-order quantifiers like *most*, modal operators like tense, etc.

Another respect in which the syntactic structures differ from the ones considered by Jurafsky and Martin is that their structures refer to "havers" and "had things". That idea is similar to the proposal that we should be able to recognize the subject of *have* as the "agent" and the object is the "theme."

In transformational grammar, it is often proposed that these semantic, "thematic" roles of the arguments of a predicate should be identifiable from the structure. A strong form of this idea was proposed by Baker (1988) for example, in something like the following form:

Uniformity of Theta Assignment Hypothesis (UTAH): identical thematic relationships between items are represented by identical structural relationships between those items in the positions where they are selected (before movement).

So for example, we might propose

(H) the specifier of v is always the agent of the V that v selects.

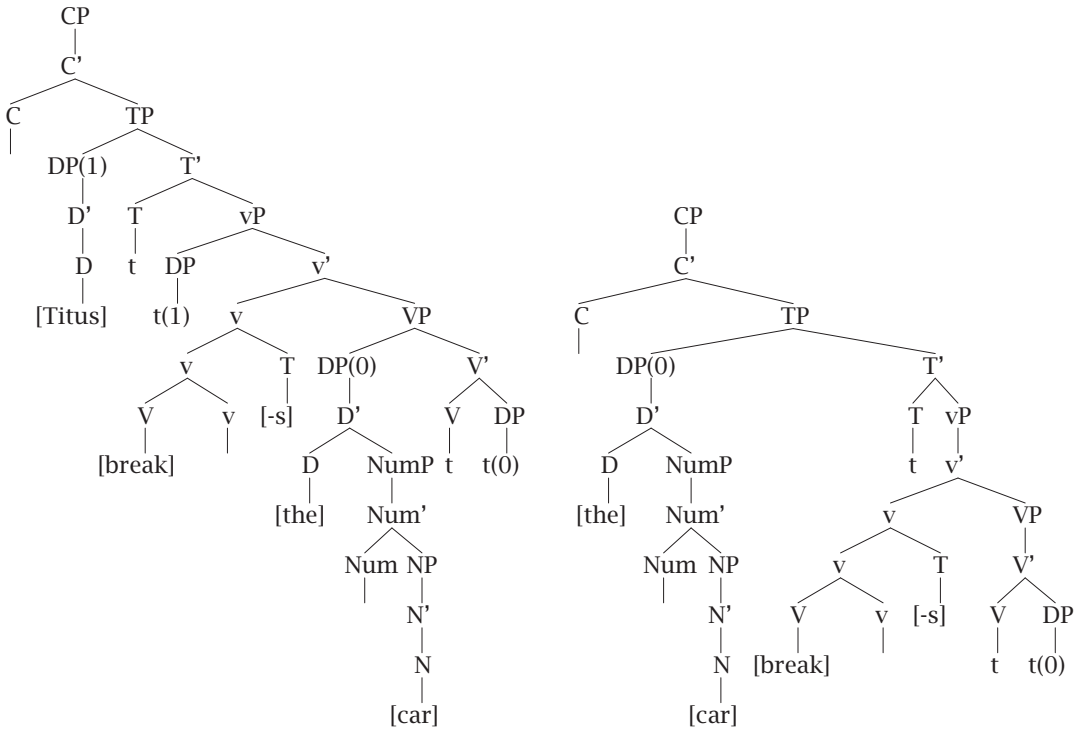
(Actually, the proposal will need to be a little more complex that this, but let's start with this simple idea).

A potential problem for this simple proposal comes with verbs that exhibit what is sometimes called the "causative alternation":

a. i. Titus break -s the vase
   ii. The vase break -s
b. i. Titus open -s the window
   ii. The window open -s
c. i. The wind clear -s the sky
   ii. The sky clear -s

In the a examples, the subject is the agent and the object is the theme, as usual, but the b examples cause a problem for H and UTAH, because there, it seems, the subject is the theme.

This problem for H and UTAH can be avoided if we assume that the single argument forms are not simple intransitives like *laugh*, but are a different class of verb, where the verb selects just an object, not a subject. One way to have this happen is to provide lexical entries that will generate trees like this:



Notice that *the car* is the object of *break* in both trees. What minimal modifications to *gh4.pl* allow these trees to be generated? (You should just need the lexical entries for these two forms of *break*, plus one other thing.)

Notice that with UTAH, we have all the advantages of the four representations Jurafsky and Martin show, but without any of the work of computing some new graphical or logical representations!

17 Morphology, phonology, orthography

17.1 Morphology subsumed

In common usage, "word" refers to some kind of linguistic unit. We have a rough, common sense idea of what a word is, but it would not be a big surprise if this notion did not correspond exactly to what we need for a scientific account of language.

(1) The commonsense notion of "word" comes close to the idea of a **morpheme** by which we will mean the simplest meaningful units of language, the "semantic atoms."

A different idea is that words are **syntactic atoms**. Syntactic atoms and semantic atoms are most clearly different in the case of idioms.

I actually think that common usage of the term "morpheme" in linguistics is closer to the notion of "syntactic atom," as has been argued, for example, by Di Sciullo and Williams (1987).

(2) A distinction is often drawn between elements which can occur independently, *free morphemes*, and those that can only appear attached to or inside of another element, *bound morphemes* or *affixes*. Affixes that are attached at the end of a word are called *suffixes*; at the beginning of the word, *prefixes*, inside the word, *infixes*; at the beginning and end *circumfixes*. This looks like a phonological fact.

(3) What we ordinarily call "words" can have more than one syntactic and semantic atom in them. For example, English can express the idea that we are talking about a plurality of objects by adding the sound [s] or [z] at the end of certain words:

book	book-s
table	table-s
friend	friend-s

The variation in pronunciation here looks like a phonological fact, but the fact that this is a mark of pluralization, one that apples to nouns (including demonstratives, etc.), looks syntactic and semantic.

(4) The same suffix can mark a different distinction too, as we see in the 3rd singular present marking on regular verbs. English can modify the way in which a verb describes the timing of an action by adding affixes:

He dance -s	present tense (meaning habitually, or at least sometimes)
He danc -ed	past tense
He be -s danc -ing	present *am* progressive *-ing* (meaning he is dancing now)

In English, only verbs can have the past tense or progressive affixes. That is, if a word has a past or progressive affix, it is a verb. Again, the reverse does not always hold. Although even the most irregular verbs of English have *-ing* forms (*being, having, doing*), some verbs sound very odd in progressive constructions:

?He is liking you a lot

And again, it is important to notice that there are some other *-ing* affixes, such as the one that lets a verb phrase become a subject or object of a sentence:

Dancing is unusual

Clearly, in this last example, the *-ing* does not mean that the dancing going on now, as we speak, is unusual.

(5) In sum, to a significant extent, morphology and syntax are sensitive to the same category distinctions.

Some derivational suffixes can combine only to roots (Fabb, 1988):

-an -ian	changes N to N	librari-an, Darwin-ian
	changes N to A	reptil-ian
-age	changes V to N	steer-age
	changes N to N	orphan-age
-al	changes V to N	betray-al
-ant	changes V to N	defend-ant
	changes V to A	defi-ant
-ance	changes V to N	annoy-ance
-ate	changes N to V	origin-ate
-ed	changes N to A	money-ed
-ful	changes N to A	peace-ful
	changes V to A	forget-ful
-hood	changes N to N	neighbor-hood
-ify	changes N to V	class-ify
	changes A to V	intens-ify
-ish	changes N to A	boy-ish
-ism	changes N to N	Reagan-ism
-ist	changes N to N	art-ist
-ive	changes V to A	restrict-ive
-ize	changes N to V	symbol-ize
-ly	changes A to A	dead-ly
-ly	changes N to A	ghost-ly
-ment	changes V to N	establish-ment
-ory	changes V to A	advis-ory
-ous	changes N to A	spac-eous
-y	changes A to N	honest-y
-y	changes V to N	assembl-y
-y	changes N to N	robber-y
-y	changes N to A	snow-y, ic-y, wit-ty, slim-y

Some suffixes can combine with a root, or a root+affix

-ary	changes N-ion to N	revolut-ion-ary
-ary	changes N-ion to A	revolut-ion-ary, legend-ary
-er	changes N-ion to N	vacat-ion-er, prison-er
-ic	changes N-ist to A	modern-ist-ic, metall-ic
-(at)ory	changes V-ify to A	class-ifi-catory, advis-ory

Some suffixes combine with a specific range of suffixed items

-al	changes N to A allows -ion, -ment, -or	natur-al
-ion	changes V to N allows -ize, -ify, -ate	rebell-ion
-ity	changes A to N allows -ive, -ic, -al, -an, -ous, -able	profan-ity
-ism	changes A to N allows -ive, -ic, -al, -an	modern-ism
-ist	changes A to N allows -ive, -ic, -al, -an	formal-ist
-ize	changes A to V allows -ive, -ic, -al, -an	special-ize

(6) This coincidence between syntax and morphology extends to "subcategorization" as well. In the class of verbs, we can see that at least some of the subcategories of verbs with distinctive behaviors correspond to subcategories that allow particular kinds of affixes. For example, we observed on the table on page **??** that *-ify* and *-ize* combine with N or A to form V: *class-ify, intens-ify, special-ize, modern-ize, formal-ize, union-ize*, but now we can notice something more: the verbs they form are all transitive:

i. a. The agency class-ified the documents
   b. *The agency class-ified

ii. a. The activists union-ized the teachers
   b. *The activists union-ized (no good if you mean they unionized the teachers)

iii. a. The war intens-ified the poverty
   b. *The war intens-ified (no good if you mean it intensified the poverty)

(7) Another suffix *-able* combines with many transitive verbs but not with most verbs that only select an object:

i. a. Titus manages the project (transitive verb)
   b. This project is manag-able
ii. a. Titus classified the document (transitive verb)
   b. This document is classifi-able
iii. a. The sun shines (intransitive verb)
   b. *The sun is shin-able
iv. a. Titus snores (intransitive verb)
   b. *He is snorable
v. a. The train arrived ("unaccusative" verb)
   b. * The train is arriv-able

(8) In English morphology, it is commonly observed that the right hand element of a complex determines its properties. This is well evidenced by various kinds of compounds:

[$_V$ [$_N$ bar] [$_V$ tend]]

[$_N$ [$_N$ apple] [$_N$ pie]]

[$_A$ [$_N$ jet] [$_A$ black]]

[$_{Npl}$ [$_{Nsg}$ part] [$_{Npl}$ suppliers]

[$_{Nsg}$ [$_{Npl}$ parts] [$_{Nsg}$ supplier]

[$_N$ [$_N$ [$_N$ rocket] [$_N$ motor]] [$_N$ chamber]]

And it plausible extends to affixes as well:

[$_{Num}$ [$_N$ bar] [$_{Num}$ -s]]

[$_N$ [$_N$ sports] [$_N$ bar]]

[$_{Num}$ [$_N$ [$_N$ sports] [$_N$ bar]] [$_{Num}$ -s]]

This English-specific regularity in English compounds is often described as follows:

a. In English, the rightmost element of a compound is the **head**.
b. A compound word has the category and features of its head.

This is sometimes called the **right hand head rule**.

(9) Notice that in the complex *bar tend*, the bar is the object of the tending, the theme. So one way to derive this structure is with lexical items like this:

tend::=>N V bar::N

If English incorporation mainly adjoins to the left (and we have independent evidence that it does) then the right hand head rule is predicted for these structures by our analysis of left adjoining incorporation. Extending this analysis to noun compounding would require an addition to our grammar, since the relation between the elements is not generally argument-selection, but is often some kind of modification. To subsume these cases, we would need to allow left adjoining incorporation of adjuncts. Incorporation of adjuncts has been argued for in other languages. See, e.g. Mithun (1984), Launey (1981, pp167-169), Shibatani (1990), Spencer (1993). This kind of incorporation seems unusual, though its apparent "unusualness" may be due at least in part to the fact that incorporation of the object of a prepositional phrase is not possible (Baker, 1988, pp86-87).

17.2 A simple phonology, orthography

Phonological analysis of an acoustic input, and orthographic analysis of an written input, will commonly yield more than one possible analysis of the input to be parsed. In fact, the relation the input and the morpheme sequence to be parsed will typically be many-many: the definite articles *a,an* will get mapped to the same syntactic article, and an input element like *read* will get mapped to the bare verb, the bare noun, the verb + present, and the verb + past.

Sometimes it is assumed that the set of possible analyses can be represented with a regular grammar or finite state machine. Let's explore this idea first, before considering reasons for thinking that it cannot be right.

(10) For any set S, let $S^\epsilon = (S \cup \{\epsilon\})$. Then as usual, a **finite state machine**(FSM) $A = \langle Q, \Sigma, \delta, I, F\rangle$ where

Q is a finite set of states ($\neq \emptyset$);

Σ_1 is a finite set of input symbols ($\neq \emptyset$);

$\delta \subseteq Q \times \Sigma^\epsilon \times Q$,

$I \subseteq Q$, the initial states;

$F \subseteq Q$, the final states.

(11) Intuitively, a **finite transducer** is an acceptor where the transitions between states are labeled by pairs. Formally, we let the pairs come from different alphabets: $T = \langle Q, \Sigma_1, \Sigma_2, \delta, I, F\rangle$ where

Q is a finite set of states ($\neq \emptyset$);

Σ_1 is a finite set of input symbols ($\neq \emptyset$);

Σ_2 is a finite set of output symbols ($\neq \emptyset$);

$\delta \subseteq Q \times \Sigma_1^\epsilon \times \Sigma_2^\epsilon \times Q$,

$I \subseteq Q$, the initial states;

$F \subseteq Q$, the final states.

(12) And as usual, we assume that for any state q and any transition function δ, $\langle q, \epsilon, \epsilon, q\rangle \in \delta$.

(13) For any transducers $T = \langle Q, \Sigma_1, \Sigma_2, \delta_1, I, F\rangle$ and $T' = \langle Q', \Sigma_1', \Sigma_2', \delta_2, I', F'\rangle$, define the **composition** $T \circ T' = \langle Q \times Q', \Sigma_1, \Sigma_2', \delta, I \times I', F \times F'\rangle$ where $\delta = \{\langle\langle q_i, q_i'\rangle, a, b, \langle q_j, q_j'\rangle\rangle |$ for some $c \in (\Sigma_2^\epsilon \cap \Sigma_1'^\epsilon)$, $\langle q_i, a, c, q_j\rangle \in \delta_1$ and $\langle q_i', c, b, q_j'\rangle \in \delta_2\}$ (Kaplan and Kay, 1994, for example).

(14) And finally, for any transducer $T = \langle Q, \Sigma_1, \Sigma_2, \delta, I, F\rangle$ let its **second projection** $2(T)$ be the FSM $A = \langle Q, \Sigma_1, \delta', I, F\rangle$, where $\delta' = \{\langle q_i, a, q_j\rangle |$ for some $b \in \Sigma_2^\epsilon$, $\langle q_i, a, b, q_j\rangle \in \delta\}$.

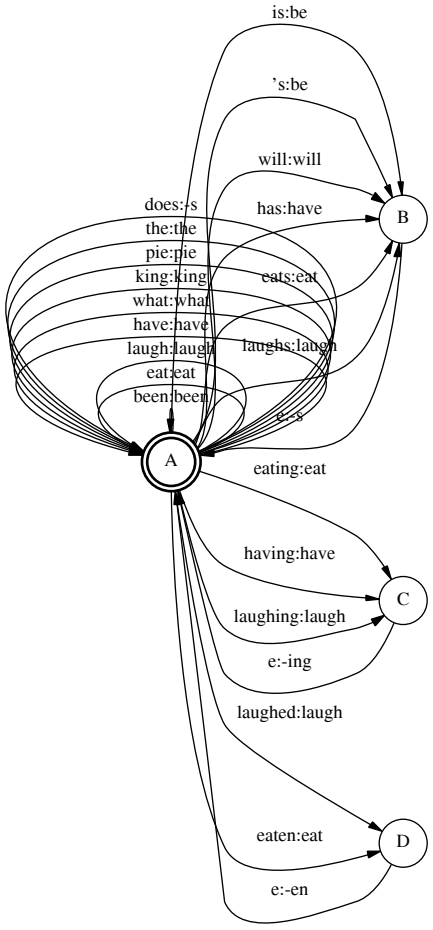
(15) Now for any input $s \in V^*$ where $s = w_1 w_2 \ldots w_n$ for some $n \geq 0$, let $string(s)$ be the transducer $\langle\{0, 1, \ldots, n\}, \Sigma, \Sigma, \delta_0, \{0\}, \{n\}\rangle$, where $\delta = \{\langle i-1, w_i, w_i, i\rangle | \ 0 \leq i\}$.

(16) Let a (finite state) **orthography** be a transducer $M = \langle Q, V, \Sigma, \delta, I, F\rangle$ such that for any $s \in V^*$, $2(string(s) \circ M)$ represents the sequences of syntactic atoms to be parsed with a grammar whose vocabulary is Σ. For any morphology M, let the function $input_M$ from V^* to Σ^* be such that for any $s \in V^*$, $input(s) = 2(string(s) \circ M)$.

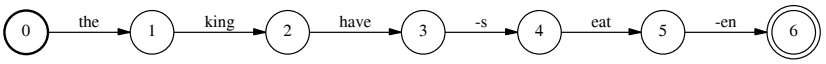
17.2.1 A first example

(17) Let $M0$ be the 4-state transducer $\langle\{A,B,C,D\},V,\Sigma,\delta,\{A\},\{A\}\rangle$ where δ is the set containing the following 4-tuples:

⟨A,the,the,A⟩	⟨A,has,have,B⟩	⟨A,eaten,eat,C⟩	⟨A,eating,eat,D⟩
⟨A,king,king,A⟩	⟨A,is,be,B⟩	⟨A,laughed,laugh,C⟩	⟨A,laughing,laugh,D⟩
⟨A,pie,pie,A⟩	⟨A,eats,eat,B⟩	⟨C,ϵ,-en,A⟩	⟨D,ϵ,-ing,A⟩
⟨A,which,which,A⟩	⟨A,laughs,laugh,B⟩		
⟨A,eat,eat,A⟩	⟨A,will,will,B⟩		
⟨A,laugh,laugh,A⟩	⟨B,ϵ,-s,A⟩		
⟨A,does,-s,A⟩			



(18) With this morphology, $input_{M0}$(`the king has eaten`) is the FSM depicted below, a machine that accepts only `the king have -s eat -en`:



(19) Notice that the last triple in the left column above provides a simple kind of do-support, so that $input_{M0}$(`what does the king eat`) is the FSM that accepts only: `what -s the king eat`. This is like example (12) from §10.2.1, and we see here the beginning of one of the traditional accounts of this

construction.

(20) With the morphology in *or4.pl* and the grammar gh4.pl, we can parse:

```
showParse(['Titus',laughs]).        showParse(['Titus',will,laugh]).
showParse(['Titus',eats,a,pie]).    showParse([is,'Titus',laughing]).
showParse([does,'Titus',laugh]).    showParse([what,does,'Titus',eat]).
```

(21) Obviously, more complex morphologies (and phonologies) can be represented by FSMs (Ellison, 1994; Eisner, 1997), but they will all have domains and ranges that are regular languages.

17.3 Better models of the interface

The previous section shows how to translate from input text to written forms of the morphemes, whose syntactic features are then looked up. We will not develop this idea here, but it is clear that it makes more sense to translate from the input text directly to the syntactic features. In other words,

represent the lexicon as a finite state machine: input → feature sequences

This would allow us to remove some of the redundancy. In particular, whenever two feature sequences have a common suffix, that suffix could be shared. However, this model has some other, more serious shortcomings.

17.3.1 Reduplication

In some languages, plurality or other meanings are sometimes expressed not by any particular phonetic string, but by reduplication, as mentioned earlier on pages 24, 182 above. It is easy to show that the language accepted by any finite transducer is only a regular language, and hence one that cannot recognize the crossing relations apparently found in reduplication.

17.3.2 Morphology without morphemes

Reduplication is only one of various kinds of morphemic alterations which do not involve simple affixation of material with specific phonetic content. Morphemic content can be expressed by word internal changes in vowel quality, for example, or by prosodic cues. The idea that utterances are sequences of phonetically given morphemes is not tenable (Anderson, 1992, for example). Rather, a range of morphological processes are available, and the languages of the world make different selections from them. That means that having just left and right adjunction as options in head movement is probably inadequate: we should allow various kinds of expressions of the sequences of elements that we analyze in syntax.

17.3.3 Probabilistic models, and recognizing new words

When we hear new words, we often make assumptions about how they would combine with affixes without hesitation. This suggests that some kind of similarity metric is at work. The relevant metric is by no means clear yet, but a wide range of proposals are subsumed by imagining that there is some "edit distance" that language learners use in identifying related lexical items. The basic idea is this: given some ways of changing a string (e.g. by adding material to either end of the string, by changing some of the elements of the string, by copying all or part of the string, etc.), a relation between pairs of strings is given by the number of operations required to map one to the other. If these operations are weighted, then more and less likely relations can be specified, and this metric can be adjusted based on what has already been learned (Ristad and Yianilos, 1996). This approach is subsumed by the more general perspective in which the similarity of two sequences is assessed by the length of the shortest program that can produce one from the other (Chater and Vitányi, 2002).

Exercises:

Download *gh4.pl* and *or4.pl* to do these exercises.

1. Modify *gh4.pl* so that *tend* left incorporates the noun *bar*, and modify *or4.pl* in order to successfully parse
   showParse(['Titus',bartends])
   by deriving the sequence *'Titus', bar, tend, -s*. Turn in the modified files.

2. *Extra Credit:* Notice that all of the following are successful:

   showParse(['Titus',eats,a,pie]). showParse(['Titus',eats,an,pie]).
   showParse(['Titus',eats,an,apple]). showParse(['Titus',eats,a,apple]).

   Modify the treatment of *an* in a linguistically appropriate way so that the calls on the right fail, and turn in the modification with a brief explanation of what you did.

18 Some open (mainly) formal questions about language

Quite a few problems that do not look like they should be very difficult remain open, and we have come up against many of them in this class. I expect these problems to be addressed and some of them settled in the next few years. A few of these were decided in the last year, and I show (conjectures that changed from last year).

Empirical difficulty estimate E from 1-10,
where 0 is the difficulty of questions like "How tall is Stabler?"
and 10 is "What language analysis goes on when you read this?" or "How long ago was the big bang?"

Formal difficulty estimate F from 1-10,
where 0 is the difficulty of questions like "Is $a^n b^n$ a regular language?"
and 10 is deciding Poincaré's conjecture or P=NP or the Riemann hypothesis.

E	F		
0	2	Are "minimalist languages" (MLs) closed under intersection with regular languages?	yes
		And are MLs = MCTALs? (Harkema, Michaelis 2001)	yes
0	2	Letting UMLs be languages defined by minimalist grammars where the features of each lexical item are unordered (Chomsky, 1995) UMLs=MLs?	(yes?)
?	2	What dimension (=how many moving components) do "minimalist grammars" for human languages require?	(?)
0	2	What automata recognize exactly the MLs? (Wartena, 1999)	(?)
		Do they allow tabular recognition methods with the correct prefix property? (Harkema 2001)	yes
?	2	How should MGs be extended to handle deletions? suffixaufnahme?	(?)
		rigid MGs can be identified from a certain kind of dependency-structure (Stabler, 2002).	
0	2	Are rigid MGs PAC-learnable from d-structures (or "simple" distributions of them)?	(yes?)
2	0	Can language learners recognize the dependencies encoded in the d-structures?	(?)
0	2	Is the onset constraint of (Prince and Smolensky, 1993, §6), when formulated as a function from inputs to optimal structures as in (Frank and Satta, 1998), a finite transduction?	(yes?)
?	2	Are the contraints of OT phonology "local" in the sense that there is a principled finite bound k such that, when each constraint is formulated as a function from candidates to numbers of violations, it is defined by a finite transducer with k or fewer states?	(yes?)
?	2	Are the contraints of OT phonology "local" in the sense that there is a principled finite bound k such that, when each constraint is formulated as a function from candidates to numbers of violations, it is defined by a finite transducer that is k-Markovian? (i.e. transducer state is determined by last k input symbols)	(yes?)
2	2	Can reduplication be factored out of phonology elegantly, to allow local application of correspondence theory constraints?	(no?)
2	2	What automata are most appropriate for recognizing reduplication?	(?)
		Does it allow tabular recognition methods with the correct prefix property? (Albro 2002)	yes
0	2	Are OT grammars in the sense of (Tesar and Smolensky, 2000) or (Hayes and Boersma, 2001) efficiently PAC-learnable in the sense of (Kearns and Vazirani, 1994) or (Li and Vitányi, 1991)	(no?)
?	?	Why are the most frequently occurring lexical items "grammatical morphemes"?	(?)
?	?	Why are about 37% of word occurrences nouns? (in most discourses, in English, Swedish, Greek, Welsh – Hudson 1994)	(?)

References

Abney, Steven P. 1987. *The English Noun Phrase in its Sentential Aspect*. Ph.D. thesis, Massachusetts Institute of Technology.

Abney, Steven P. 1996a. Statistical methods and linguistics. In Judith Klavans and Philip Resnik, editors, *The Balancing Act*. MIT Press, Cambridge, Massachusetts.

Abney, Steven P. 1996b. Stochastic attribute-value grammars. University of Tübingen. Available at ftp://xxx.lanl.gov/cmp-lg/papers/9610/9610003.

Abney, Steven P. and Mark Johnson. 1989. Memory requirements and local ambiguities of parsing strategies. *Journal of Psycholinguistic Research*, 20:233–249.

Abramson, Harvey and Veronica Dahl. 1989. *Logic Grammars*. Springer-Verlag, NY.

Åfarli, Tor A. 1994. A promotion analysis of restrictive relative clauses. *The Linguistic Review*, 11:81–100.

Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman. 1985. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, Reading, Massachusetts.

Aho, Alfred V. and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling. Volume 1: Parsing*. Prentice-Hall, Englewood Cliffs, New Jersey.

Anderson, Stephen R. 1992. *A-Morphous Morphology*. Cambridge University Press, NY.

Apostol, Tom M. 1969. *Calculus, Volume II*. Wiley, NY.

Baker, Mark. 1988. *Incorporation: a theory of grammatical function changing*. MIT Press, Cambridge, Massachusetts.

Baker, Mark. 1996. *The Polysynthesis Parameter*. Oxford University Press, NY.

Baltin, Mark. 1992. On the characterisation and effects of d-linking: Comments on cinque. In Robert Freidin, editor, *Current Issues in Comparative Grammar*. Kluwer, Dordrecht.

Barker, Chris and Geoffrey K. Pullum. 1990. A theory of command relations. *Linguistics and Philosophy*, 13:1–34.

Barss, Andrew and Howard Lasnik. 1986. A note on anaphora and double objects. *Linguistic Inquiry*, 17:347–354.

Bayer, Samuel and Mark Johnson. 1995. Features and agreement. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 70–76.

Beghelli, Filippo and Tim Stowell. 1996. Distributivity and negation. In Anna Szabolcsi, editor, *Ways of Scope Taking*. Kluwer, Boston.

Berger, Adam L., Stephen A. Della Pietra, and Vincent J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22:39–72.

Bernardi, Raffaela. 2002. *Reasoning with Polarity in Categorial Type Logic*. Ph.D. thesis, University of Utrecht, Utrecht.

Berwick, Robert C. and Amy S. Weinberg. 1984. *The Grammatical Basis of Linguistic Performance: Language Use and Acquisition*. MIT Press, Cambridge, Massachusetts.

Bever, Thomas G. 1970. The cognitive basis for linguistic structures. In J.R. Hayes, editor, *Cognition and the Development of Language*. Wiley, NY.

Bhatt, Rajesh. 1999. Adjectival modifiers and the raising analysis of relative clauses. In *Proceedings of the North Eastern Linguistic Society, NELS 30*. `http://ling.rutgers.edu/nels30/`.

Billot, Sylvie and Bernard Lang. 1989. The structure of shared forests in ambiguous parsing. In *Proceedings of the 1989 Meeting of the Association for Computational Linguistics*.

Blackburn, Simon and Keith Simmons. 1999. *Truth*. Oxford University Press, Oxford.

Boeder, Winfried. 1995. Suffixaufname in Kartvelian. In Frans Plank, editor, *Double Case: Agreement by Suffixaufnahme*. Oxford University Press, NY.

Boghossian, Paul. 1996. Analyticity reconsidered. *Noûs*, 30:360–391.

Boolos, George. 1979. *The Unprovability of Consistency*. Cambridge University Press, NY.

Boolos, George and Richard Jeffrey. 1980. *Computability and Logic*. Cambridge University Press, NY.

Boullier, Pierre. 1998. Proposal for a natural language processing syntactic backbone. Technical Report 3242, Projet Atoll, INRIA, Rocquencourt.

Brent, Michael R. and Timothy A. Cartwright. 1996. Lexical categorization: Fitting template grammars by incremental MDL optimization. In Laurent Micla and Colin de la Higuera, editors, *Grammatical Inference: Learning Syntax from Sentences*. Springer, NY, pages 84–94.

Bresnan, Joan. 1982. Control and complementation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, Massachusetts.

Bresnan, Joan, Ronald M. Kaplan, Stanley Peters, and Annie Zaenen. 1982. Cross-serial dependencies in Dutch. *Linguistic Inquiry*, 13(4):613–635.

Bretscher, Otto. 1997. *Linear Algebra with Applications*. Prentice-Hall, Upper Saddle River, New Jersey.

Brosgol, Benjamin Michael. 1974. *Deterministic Translation Grammars*. Ph.D. thesis, Harvard University.

Buell, Leston. 2000. Swahili relative clauses. UCLA M.A. thesis.

Burzio, Luigi. 1986. *Italian Syntax: A Government-Binding Approach*. Reidel, Boston.

Carnap, Rudolf. 1956. Empiricism, semantics and ontology. In *Meaning and Necessity*. University of Chicago Press, Chicago.

Charniak, Eugene. 1993. *Statistical Language Learning*. MIT Press, Cambridge, Massachusetts.

Charniak, Eugene, Sharon Goldwater, and Mark Johnson. 1998. Edge-based best-first chart parsing. In *Proceedings of the Workshop on Very Large Corpora*.

Chater, Nick and Paul Vitányi. 2002. The generalized universal law of generalization. *Journal of Mathematical Psychology*.

Chen, Stanley and Joshua Goodman. 1998. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Harvard University, Cambridge, Massachusetts.

Chi, Zhiyi. 1999. Statistical properties of probabilistic context free grammars. *Computational Linguistics*, 25:130–160.

Chomsky, Noam. 1957. *Syntactic Structures*. Mouton, The Hague.

Chomsky, Noam. 1963. Formal properties of grammars. In R. Duncan Luce, Robert R. Bush, and Eugene Galanter, editors, *Handbook of Mathematical Psychology, Volume II*. Wiley, NY, pages 323–418.

Chomsky, Noam. 1968. *Language and Mind*. Harcourt Brace Javonovich, NY.

Chomsky, Noam. 1975. *Reflections on Language*. Pantheon, NY.

Chomsky, Noam. 1981. *Lectures on Government and Binding*. Foris, Dordrecht.

Chomsky, Noam. 1986. *Knowledge of Language*. Praeger, NY.

Chomsky, Noam. 1993. A minimalist program for linguistic theory. In Kenneth Hale and Samuel Jay Keyser, editors, *The View from Building 20*. MIT Press, Cambridge, Massachusetts.

Chomsky, Noam. 1995. *The Minimalist Program*. MIT Press, Cambridge, Massachusetts.

Chomsky, Noam and Howard Lasnik. 1993. Principles and parameters theory. In J. Jacobs, A. von Stechow, W. Sternfeld, and T. Vennemann, editors, *Syntax: An international handbook of contemporary research*. de Gruyter, Berlin. Reprinted in Noam Chomsky, *The Minimalist Program*. MIT Press, 1995.

Church, Kenneth and Ramesh Patil. 1982. How to put the block in the box on the table. *Computational Linguistics*, 8:139–149.

Cinque, Guglielmo. 1990. *Types of A' Dependencies*. MIT Press, Cambridge, Massachusetts.

Cinque, Guglielmo. 1999. *Adverbs and Functional Heads : A Cross-Linguistic Perspective*. Oxford University Press, Oxford.

Citko, Barbara. 2001. On the nature of *merge*. State University of New York, Stony Brook.

Collins, Chris. 1997. *Local Economy*. MIT Press, Cambridge, Massachusetts.

Corcoran, John, William Frank, and Michael Maloney. 1974. String theory. *Journal of Symbolic Logic*, 39:625–637.

Cornell, Thomas L. 1996. A minimalist grammar for the copy language. Technical report, SFB 340 Technical Report #79, University of Tübingen.

Cornell, Thomas L. 1997. A type logical perspective on minimalist derivations. In *Proceedings, Formal Grammar'97*, Aix-en-Provence.

Cornell, Thomas L. 1998a. Derivational and representational views of minimalist transformational grammar. In *Logical Aspects of Computational Linguistics 2*. Springer-Verlag, NY. Forthcoming.

Cornell, Thomas L. 1998b. Island effects in type logical approaches to the minimalist program. In *Proceedings of the Joint Conference on Formal Grammar, Head-Driven Phrase Structure Grammar, and Categorial Grammar, FHCG-98*, pages 279–288, Saarbrücken.

Cornell, Thomas L. and James Rogers. 1999. Model theoretic syntax. In Lisa Lai-Shen Cheng and Rint Sybesma, editors, *The Glot International State of the Article, Book 1*. Holland Academic Graphics,Springer-Verlag, The Hague. Forthcoming.

Crain, Stephen and Mark Steedman. 1985. On not being led up the garden path. In D.R. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Parsing*. Cambridge University Press, NY.

Crocker, Matthew W. 1997. Principle based parsing and logic programming. *Informatica*, 21:263–271.

Curry, Haskell B. and Robert Feys. 1958. *Combinatory Logic, Volume 1*. North Holland, Amsterdam.

Dahlgren, Kathleen. 1988. *Naive Semantics for Natural Language Understanding*. Kluwer, Boston.

Dalrymple, Mary and Ronald M. Kaplan. 2000. Feature indeterminacy and feature resolution. *Language*, 76:759–798.

Damerau, Frederick J. 1971. *Markov Models and Linguistic Theory*. Mouton, The Hague.

Davey, B.A. and H.A. Priestley. 1990. *Introduction to Lattices and Order*. Cambridge University Press, NY.

Davis, Martin and Hilary Putnam. 1960. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215.

de Marcken, Carl. 1996. *Unsupervised language acquisition*. Ph.D. thesis, Massachusetts Institute of Technology.

De Mori, Renato, Michael Galler, and Fabio Brugnara. 1995. Search and learning strategies for improving hidden Markov models. *Computer Speech and Language*, 9:107–121.

Demers, Alan J. 1977. Generalized left corner parsing. In *Conference Report of the 4th Annual Association for Computing Machinery Symposium on Principles of Programming Languages*, pages 170–181.

Demopoulos, William and John L. Bell. 1993. Frege's theory of concepts and objects and the interpretation of second-order logic. *Philosophia Mathematica*, 1:225–245.

Deng, L. and C. Rathinavelu. 1995. A Markov model containing state-conditioned second-order non-stationarity: application to speech recognition. *Computer Speech and Language*, 9:63–86.

Di Sciullo, Anna Maria and Edwin Williams. 1987. *On the definition of word*. MIT Press, Cambridge, Massachusetts.

Dimitrova-Vulchanova, Mila and Giuliana Giusti. 1998. Fragments of Balkan nominal structure. In Artemis Alexiadou and Chris Wilder, editors, *Possessors, Predicates and Movement in the Determiner Phrase*. Amsterdam, Philadelphia.

Drake, Alvin W. 1967. *Fundamentals of Applied Probability Theory*. McGraw-Hill, NY.

Earley, J. 1968. *An Efficient Context-Free Parsing Algorithm*. Ph.D. thesis, Carnegie-Mellon University.

Earley, J. 1970. An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery*, 13:94–102.

Earman, John. 1992. *Bayes or Bust? A Critical Examination of Bayesian Confirmation Theory*. MIT Press, Cambridge, Massachusetts.

Eisner, Jason. 1997. Efficient generation in Primitive Optimality Theory. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*.

Eisner, Jason and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th Annual Meeting, ACL'99*. Association for Computational Linguistics.

Ellison, Mark T. 1994. Phonological derivation in optimality theory. In *Procs. 15th Int. Conf. on Computational Linguistics*, pages 1007–1013. (Also available at the Edinburgh Computational Phonology Archive).

Engelfriet, Joost. 1997. Context free graph grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*. Springer, NY, pages 125–213.

Evans, Gareth. 1976. Semantic structure and logical form. In Gareth Evans and John McDowell, editors, *Truth and Meaning: Essays in Semantics*. Clarendon Press, Oxford. Reprinted in Gareth Evans, *Collected Papers*. Oxford: Clarendon, 1985, pp. 49-75.

Fabb, Nigel. 1988. English suffixation is constrained only by selection restrictions. *Linguistics and Philosophy*, 6:527–539.

Fodor, J.A., M.F. Garrett, E.C.T. Walker, and C.H. Parkes. 1980. Against definitions. *Cognition*, 8:263–367.

Fodor, Janet Dean. 1978. Parsing strategies and constraints on transformations. *Linguistic Inquiry*, 9:427–473.

Fodor, Janet Dean. 1985. Deterministic parsing and subjacency. *Language and Cognitive Processes*, 1:3–42.

Fodor, Jerry A. 1983. *The Modularity of Mind: A Monograph on Faculty Psychology*. MIT Press, Cambridge, Massachusetts.

Fodor, Jerry A. 1998. *In Critical Condition: Polemical Essays on Cognitive Science and the Philosophy of Mind*. MIT Press, Cambridge, Massachusetts.

Fong, Sandiway. 1999. Parallel principle-based parsing. In *Proceedings of the Sixth International Workshop on Natural Language Understanding and Logic Programming*, pages 45–58.

Ford, Marilyn, Joan Bresnan, and Ronald M. Kaplan. 1982. A competence-based theory of syntactic closure. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, Massachusetts.

Forney, G. D. 1973. The Viterbi algorithm. *Proceedings of the IEEE*, 61:268–278.

Frank, Robert and Giorgio Satta. 1998. Optimality theory and the generative complexity of constraint violability. *Computational Linguistics*, 24:307–315.

Frazier, Lyn. 1978. *On Comprehending Sentences: Syntactic Parsing Strategies*. Ph.D. thesis, University of Massachusetts, Amherst.

Frazier, Lyn and Charles Clifton. 1996. *Construal*. MIT Press, Cambridge, Massachusetts.

Frazier, Lyn and Keith Rayner. 1982. Making and correcting errors during sentence comprehension. *Cognitive Psychology*, 14:178–210.

Freidin, Robert. 1978. Cyclicity and the theory of grammar. *Linguistic Inquiry*, 9:519–549.

Fromkin, Victoria, editor. 2000. *Linguistics: An Introduction to Linguistic Theory*. Basil Blackwell, Oxford.

Fyodorov, Yaroslav, Yoad Winter, and Nissim Francez. 2003. Order-based inference in natural logic. *Research on Language and Computation*. Forthcoming.

Gardner, Martin. 1985. *Wheels, Life and other Mathematical Amusements*. Freeman (reprint edition), San Francisco.

Geach, P.T. 1962. *Reference and Generality*. Cornell University Press, Ithaca, New York.

Gecseg, F. and M. Steinby. 1984. *Tree Automata*. Akadémiai Kiadó, Budapest.

Gibson, Edward. 1998. Linguistic complexity: Locality of syntactic dependencies. *Cognition*, 68:1–76.

Girard, Jean-Yves, Yves Lafont, and Paul Taylor. 1989. *Proofs and Types*. Cambridge University Press, NY.

Golding, Andrew R. and Yves Schabes. 1996. Combining trigram-based and feature-based methods for context-sensitive spelling correction. Mitsubishi Electric Research Laboratories Technical Report TR96-03a. Available at ftp://xxx.lanl.gov/cmp-lg/papers/9605/9605037.

Goldman, Jeffrey. 1998. *A digital filter model for text mining*. Ph.D. thesis, University of California, Los Angeles.

Golub, Gene H. and Charles F. Van Loan. 1996. *Matrix Computations: Third Edition*. Johns Hopkins University Press, Baltimore.

Good, I.J. 1953. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40:237–264.

Gorn, Saul. 1969. Explicit definitions and linguistic dominoes. In John Hart and Satoru Takasu, editors, *Systems and Computer Science*. University of Toronto Press, Toronto.

Greibach, Shiela and John E. Hopcroft. 1969. Scattered context grammars. *Journal of Computer and Systems Sciences*, 3.

Grice, H.P. 1975. Logic and conversation. In P. Cole and J.L. Morgan, editors, *Speech Acts*. Academic Press, NY, pages 45–58.

Groenink, Annius. 1997. *Surface without structure: Word order and tractability issues in natural language processing*. Ph.D. thesis, Utrecht University.

Hale, John and Edward P. Stabler. 2001. Representing derivations: unique readability and transparency. Johns Hopkins and UCLA. Publication forthcoming.

Hall, Patrick A. V. and Geoff R. Dowling. 1980. Approximate string matching. *Computing Surveys*, 12:381–402.

Harkema, Henk. 2000. A recognizer for minimalist grammars. In *Sixth International Workshop on Parsing Technologies, IWPT'2000*.

Harris, Theodore Edward. 1955. On chains of infinite order. *Pacific Journal of Mathematics*, 5:707–724.

Hayes, Bruce and Paul Boersma. 2001. Empirical tests of the gradual learning algorithm. *Linguistic Inquiry*, 32:45–86.

Herbrand, Jacques. 1930. *Recherches sur la théorie de la démonstration*. Ph.D. thesis, University of Paris. Chapter 5 of this thesis is reprinted in Jean van Heijenoort (ed.), *From Frege to Gödel: A Source Book in Mathematical Logic, 1879 – 1931. Cambridge, Massachusetts: Harvard University Press*.

Hermes, Hans. 1938. Semiotik, eine theorie der zeichengestalten als grundlage für untersuchungen von formalizierten sprachen. *Forschungen zur Logik und zur Grundlage der exakten Wissenschaften*, 5.

Hirschman, Lynette and John Dowding. 1990. Restriction grammar: A logic grammar. In Patrick Saint-Dizier and Stan Szpakowicz, editors, *Logic and Logic Grammars for Language Processing*. Ellis Horwood, NY, chapter 7, pages 141–167.

Hopcroft, John E. and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, Massachusetts.

Horn, Roger A. and Charles R. Johnson. 1985. *Matrix Analysis*. Cambridge University Press, NY.

Hornstein, Norbert. 1999. Movement and control. *Linguistic Inquiry*, 30:69–96.

Horwich, Paul. 1982. *Probability and Evidence*. Cambridge University Press, NY.

Horwich, Paul. 1998. *Meaning*. Oxford University Press, Oxford.

Huang, Cheng-Teh James. 1982. *Logical Relations in Chinese and the Theory of Grammar*. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Hudson, Richard. 1994. About 37% of word-tokens are nouns. *Language*, 70:331–345.

Huybregts, M.A.C. 1976. Overlapping dependencies in Dutch. Technical report, University of Utrecht. Utrecht Working Papers in Linguistics.

Ingria, Robert. 1990. The limits of unification. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, pages 194–204.

Jacob, Bill. 1995. *Linear Functions and Matrix Theory*. Springer, NY.

Jaynes, E.T. 1957. Information theory and statistical mechanics. *Physics Reviews*, 106:620–630.

Jelinek, Frederick. 1985. Markov source modeling of text generation. In J. K. Skwirzinksi, editor, *The Impact of Processing Techniques on Communications*. Nijhoff, Dordrecht, pages 567–598.

Jelinek, Frederick. 1999. *Statistical Methods for Speech Recognition*. MIT Press, Cambridge, Massachusetts.

Jelinek, Frederick and Robert L. Mercer. 1980. Interpolated estimation of Markov source parameters from sparse data. In E. S. Gelsema and L. N. Kanal, editors, *Pattern Recognition in Practice*. North-Holland, NY, pages 381–397.

Johnson, Mark. 1988. *Attribute Value Logic and The Theory of Grammar*. Number 16 in CSLI Lecture Notes Series. Chicago University Press, Chicago.

Johnson, Mark. 1991. Techniques for deductive parsing. In Charles Grant Brown and Gregers Koch, editors, *Natural Language Understanding and Logic Programming III*. North Holland, pages 27–42.

Johnson, Mark. 1999. Pcfg models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632.

Johnson, Mark, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stephan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting, ACL'99*. Association for Computational Linguistics.

Johnson, Mark and Edward Stabler. 1993. *Topics in Principle Based Parsing*. LSA Summer Institute, Columbus, Ohio.

Joshi, Aravind. 1985. How much context-sensitivity is necessary for characterizing structural descriptions. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Processing: Theoretical, Computational and Psychological Perspectives*. Cambridge University Press, NY, pages 206–250.

Joshi, Aravind K., K. Vijay-Shanker, and David Weir. 1991. The convergence of mildly context sensitive grammar formalisms. In Peter Sells, Stuart Shieber, and Thomas Wasow, editors, *Foundational Issues in Natural Language Processing*. MIT Press, Cambridge, Massachusetts, pages 31–81.

Jurafsky, Daniel and James Martin. 1999. *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics*. Prentice-Hall, Englewood Cliffs, New Jersey.

Kamp, Hans. 1984. A theory of truth and semantic representation. In Geroen Groenendijk, Theo Janssen, and Martin Stokhof, editors, *Formal Methods in the Study of Language*. Foris, Dordrecht.

Kaplan, Ronald and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20:331–378.

Kaplan, Ronald M. and Joan Bresnan. 1982. Lexical-functional grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*. MIT Press, chapter 4, pages 173–281.

Kasami, T. 1965. An efficient recognition and syntax algorithm for context free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.

Kayne, Richard. 1994. *The Antisymmetry of Syntax*. MIT Press, Cambridge, Massachusetts.

Kayne, Richard. 1999. A note on prepositions and complementizers. In *A Celebration*. MIT Press, Cambridge, Massachusetts. Available at `http://mitpress.mit.edu/chomskydisc/Kayne.html`.

Kearns, Michael J. and Umesh V. Vazirani. 1994. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, Massachusetts.

Keenan, Edward L. 1979. On surface form and logical form. *Studies in the Linguistic Sciences*, 8:163–203.

Keenan, Edward L. 1989. Semantic case theory. In R. Bartsch, J. van Benthem, and R. van Emde-Boas, editors, *Semantics and Contextual Expression*. Foris, Dordrecht, pages 33–57. Groningen-Amsterdam Studies in Semantics (GRASS) Volume 11.

Keenan, Edward L. and Leonard M. Faltz. 1985. *Boolean Semantics for Natural Language*. Reidel, Dordrecht.

Keim, Greg A., Noam Shazeer, Michael L. Littman, Sushant Agarwal, Catherine M. Cheves, Joseph Fitzgerald, Jason Grosland, Fan Jiang, Shannon Pollard, , and Karl Weinmeister. 1999. Proverb: The probabilistic cruciverbalist. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-99*. Morgan Kaufmann.

Kenesei, I. 1989. Logikus – e a magyar szórend? *Általános Nyelvézeti Tanulmányok*, 17:105–152.

Kiss, Katalin É. 1993. Wh-movement and specificity. *Linguistic Inquiry*, 11:85–120.

Knill, David C. and Whitman Richards, editors. 1996. *Perception as Bayesian Inference*. Cambridge University Press, NY.

Knuth, Donald E. 1965. On the translation of languages from left to right. *Information and Control*, 8:607–639.

Kolb, Hans-Peter, Uwe Mönnich, and Frank Morawietz. 1999. Regular description of cross-serial dependencies. In *Proceedings of the Meeting on Mathematics of Language, MOL6*.

Koopman, Hilda. 1994. Licensing heads. In David Lightfoot and Norbert Hornstein, editors, *Verb Movement*. Cambridge University Press, NY, pages 261–296.

Koopman, Hilda and Dominique Sportiche. 1991. The position of subjects. *Lingua*, 85:211–258. Reprinted in Dominique Sportiche, *Partitions and Atoms of Clause Structure: Subjects, agreement, case and clitics*. NY: Routledge.

Koopman, Hilda, Dominique Sportiche, and Edward Stabler. 2002. *An Introduction to Syntactic Analysis and Theory*. UCLA manuscript.

Koopman, Hilda and Anna Szabolcsi. 2000a. *Verbal Complexes*. MIT Press, Cambridge, Massachusetts.

Koopman, Hilda and Anna Szabolcsi. 2000b. *Verbal Complexes*. MIT Press, Cambridge, Massachusetts.

Korf, Richard E. 1985. An optimum admissible tree search. *Artificial Intelligence*, 27:97–109.

Kracht, Marcus. 1993. Nearness and syntactic influence spheres. Freie Universitat Berlin.

Kracht, Marcus. 1995. Syntactic codes and grammar refinement. *Journal of Logic, Language and Information*.

Kracht, Marcus. 1998. Adjunction structures and syntactic domains. In Hans-Peter Kolb and Uwe Mönnich, editors, *The Mathematics of Syntactic Structure: Trees and their Logics*. Mouton-de Gruyter, Berlin.

Kraft, L.G. 1949. *A Device for Quantizing, Grouping, and Coding Amplitude Modulated Pulses*. Ph.D. thesis, Cambridge, Massachusetts, Massachusetts Institute of Technology.

Kukich, Karen. 1992. Techniques for automatically correcting words in text. *Association for Computing Machinery Computing Surveys*, 24:377–439.

Kullback, S. 1959. *Information theory in statistics*. Wiley, NY.

Lambek, Joachim. 1958. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170.

Langendoen, D. Terence, Dana McDaniel, and Yedidyah Langsam. 1989. Preposition-phrase attachment in noun phrases. *Journal of Psycholinguistic Research*, 18:533–548.

Larson, Richard K. 1988. On the double object construction. *Linguistic Inquiry*, 19:335–391.

Launey, Michel. 1981. *Introduction à la Langue et à la Littérature Aztèques*. L'Harmattan, Paris.

Lee, Lillian. 1997. Fast context-free parsing requires fast Boolean matrix multiplication. In *Proceedings of the 35th Annual Meeting, ACL'97*. Association for Computational Linguistics.

Lettvin, J.Y., H.R. Maturana, W.S. McCulloch, and W.H. Pitts. 1959. What the frog's eye tells the frog's brain. *Proceedings of the Institute of Radio Engineering*, 47:1940–1959.

Lewis, H.R. and C.H. Papadimitriou. 1981. *Elements of the Theory of Computation*. Prentice-Hall, Englewood Cliffs, New Jersey.

Li, Ming and Paul Vitányi. 1991. Learning concepts under simple distributions. *SIAM Journal of Computing*, 20(5):911–935.

Li, Wentian. 1992. Random texts exhibit Zipf's law-like word frequency distribution. *IEEE Transactions on Information Theory*, 38:1842–1845.

Lloyd, John W. 1987. *Foundations of Logic Programming*. Springer, Berlin.

Lynch, Elizabeth B., John D. Coley, and Douglas L. Medin. 2000. Tall is typical. *Memory and Cognition*, 28:41–50.

Magerman, David M. and Carl Weir. 1992. Efficiency, robustness, and accuracy in picky chart parsing. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*.

Mahajan, Anoop. 2000. Eliminating head movement. In *The 23rd Generative Linguistics in the Old World Colloquium, GLOW '2000, Newsletter #44*, pages 44–45.

Manaster-Ramer, Alexis. 1986. Copying in natural languages, context freeness, and queue grammars. In *Proceedings of the 1986 Meeting of the Association for Computational Linguistics*.

Mandelbrot, Benoit. 1961. On the theory of word frequencies and on related Markovian models of discourse. In Roman Jakobson, editor, *Structure of Language in its Mathematical Aspect, Proceedings of the 12th Symposium in Applied Mathematics*. American Mathematical Society, Providence, Rhode Island, pages 190–219.

Maor, Eli. 1994. *e: The Story of a Number*. Princeton University Press, Princeton.

Marcus, Mitchell. 1980. *A Theory of Syntactic Recognition for Natural Language*. MIT Press, Cambridge, Massachusetts.

Martin, Roger. 1996. *A Minimalist Theory of PRO and Control*. Ph.D. thesis, University of Connecticut, Storrs.

Masek, William J. and Michael S. Paterson. 1980. A faster algorithm for computing string edit distances. *Journal of Computer and System Sciences*, 20:18–31.

Mates, Benson. 1972. *Elementary Logic*. Oxford University Press, Oxford.

McDaniel, Dana. 1989. Partial and multiple wh-movement. *Natural Language and Linguistic Theory*, 7:565–604.

McDaniel, Dana, Bonnie Chiu, and Thomas L. Maxfield. 1995. Parameters for wh-movement types: evidence from child English. *Natural Language and Linguistic Theory*, 13:709–753.

Merlo, Paola. 1995. Modularity and information content classes in principle-based parsing. *Computational Linguistics*, 21:515–542.

Michaelis, Jens. 1998. Derivational minimalism is mildly context-sensitive. In *Proceedings, Logical Aspects of Computational Linguistics, LACL'98*, Grenoble.

Michaelis, Jens and Marcus Kracht. 1997. Semilinearity as a syntactic invariant. In Christian Retoré, editor, *Logical Aspects of Computational Linguistics*, pages 37–40, NY. Springer-Verlag (Lecture Notes in Computer Science 1328).

Michaelis, Jens, Uwe Mönnich, and Frank Morawietz. 2000. Algebraic description of derivational minimalism. In *International Conference on Algebraic Methods in Language Proceesing, AMiLP'2000/TWLT16*, University of Iowa.

Miller, George A. and Noam Chomsky. 1963. Finitary models of language users. In R. Duncan Luce, Robert R. Bush, and Eugene Galanter, editors, *Handbook of Mathematical Psychology, Volume II*. Wiley, NY, pages 419–492.

Minsky, Marvin. 1988. *The Society of Mind*. Simon and Schuster, NY.

Mithun, Marianne. 1984. The evolution of noun incorporation. *Language*, 60:847–893.

Mitton, Roger. 1992. Oxford advanced learner's dictionary of current english: expanded 'computer usable' version. Available from.

Moll, R.N., M.A. Arbib, and A.J. Kfoury. 1988. *An Introduction to Formal Language Theory*. Springer-Verlag, NY.

Moltmann, Frederike. 1992. *Coordination and Comparatives*. Ph.D. thesis, MIT.

Mönnich, Uwe. 1997. Adjunction as substitution. In *Formal Grammar '97, Proceedings of the Conference*.

Montague, Richard. 1969. English as a formal language. In B. Visentini et al., editor, *Linguaggi nella Societá e nella Tecnica*. Edizioni di Communità, Milan. Reprinted in R.H. Thomason, editor, *Formal Philosophy: Selected Papers of Richard Montague*. New Haven: Yale University Press, §6.

Moortgat, Michael. 1996. Categorial type logics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*. Elsevier, Amsterdam.

Morawietz, Frank. 2001. *Two Step Approaches to Natural Language Formalisms*. Ph.D. thesis, University of Tübingen.

Munn, Alan. 1992. A null operator analysis of ATB gaps. *The Linguistic Review*, 9:1–26.

Nederhof, Mark-Jan. 1998. Linear indexed automata and the tabulation of TAG parsing. In *Proceedings TAPD'98*.

Nijholt, Anton. 1980. *Context Free Grammars: Covers, Normal Forms, and Parsing*. Springer-Verlag, NY.

Nozohoor-Farshi, Rahman. 1986. *LRRL(k) grammars: a left to right parsing technique with reduced lookaheads*. Ph.D. thesis, University of Alberta.

Obenauer, Hans-Georg. 1983. Une quantification non canonique: la 'quantification à distance'. *Langue Française*, 48:66–88.

Ojemann, G.A., F. Ojemann, E. Lettich, and M. Berger. 1989. Cortical language localization in left dominant hemisphere: An electrical stimulation mapping investigation in 117 patients. *Journal of Neurosurgery*, 71:316–326.

Papoulis, Athanasios. 1991. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, NY.

Parker, D. Stott. 1995. Schur complements obey Lambek's categorial grammar: another view of Gaussian elimination and LU decomposition. Computer Science Department Technical Report, UCLA.

Partee, Barbara. 1975. Bound variables and other anaphors. In David Waltz, editor, *Theoretical Issues in Natural Language Processing*. Association for Computing Machinery, NY.

Peacocke, Christopher. 1993. How are a priori truths possible? *European Journal of Philosophy*, 1:175–199.

Pearl, Judea. 1988. *Probabilistic Reasoning in Intelligent Systems: Patterns of Plausible Inference*. Morgan Kaufmann, San Francisco.

Pears, David. 1981. The logical independence of elementary propositions. In Irving Block, editor, *Perspectives on the Philosophy of Wittgenstein*. Blackwell, Oxford.

Perline, Richard. 1996. Zipf's law, the central limit theorem, and the random division of the unit interval. *Physical Review E*, 54:220–223.

Pesetsky, David. 1985. *Paths and Categories*. Ph.D. thesis, Massachusetts Institute of Technology.

Pesetsky, David. 1995. *Zero Syntax: Experiencers and Cascades*. MIT Press, Cambridge, Massachusetts.

Pesetsky, David. 2000. *Phrasal movement and its kin*. MIT Press, Cambridge, Massachusetts.

Pollard, Carl. 1984. *Generalized phrase structure grammars, head grammars and natural language*. Ph.D. thesis, Stanford University.

Pollard, Carl and Ivan Sag. 1994. *Head-driven Phrase Structure Grammar*. The University of Chicago Press, Chicago.

Pollard, Carl and Ivan A. Sag. 1987. *Information-based Syntax and Semantics*. Number 13 in CSLI Lecture Notes Series. Chicago University Press, Chicago.

Pollock, Jean-Yves. 1994. Checking theory and bare verbs. In Guglielmo Cinque, Jan Koster, Jean-Yves Pollock, Luigi Rizzi, and Raffaella Zanuttini, editors, *Paths Towards Universal Grammar: Studies in Honor of Richard S. Kayne*. Georgetown University Press, Washington, D.C, pages 293–310.

Prince, Alan and Paul Smolensky. 1993. *Optimality Theory: Constraint Interaction in Generative Grammar*. Forthcoming.

Pritchett, Bradley L. 1992. *Grammatical Competence and Parsing Performance*. University of Chicago Press, Chicago.

Purdy, William C. 1991. A logic for natural language. *Notre Dame Journal of Formal Logic*, 32:409–425.

Putnam, Hilary. 1986. Computational psychology and interpretation theory. In Z.W. Pylyshyn and W. Demopoulos, editors, *Meaning and Cognitive Structure*. Ablex, New Jersey, pages 101–116, 217–224.

Quine, Willard van Orman. 1946. Concatenation as a basis for arithmetic. *Journal of Symbolic Logic*, 11:105–114. Reprinted in Willard V.O. Quine, *Selected Logic Papers*, NY: Random House, 1961.

Quine, Willard van Orman. 1951a. *Mathematical Logic (Revised Edition)*. Harvard University Press, Cambridge, Massachusetts.

Quine, Willard van Orman. 1951b. Two dogmas of empiricism. *Philosophical Review*, 11:105–114. Reprinted in Willard V.O. Quine, *From a Logical Point of View*, NY: Harper & Row, 1953.

Rabin, Michael O. 1969. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35.

Rabiner, Lawrence R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286.

Ratnaparkhi, Adwait. 1998. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. Ph.D. thesis, University of Pennsylvania.

Reichenbach, Hans. 1968. *The Philosophy of Space and Time*. Dover, New York.

Resnik, Philip. 1992. Left-corner parsing and psychological plausibility. In *Proceedings of the 14th International Conference on Computational Linguistics, COLING 92*, pages 191–197.

Richards, Norvin. 1998. The principle of minimal compliance. *Linguistic Inquiry*, 29:599–629.

Ristad, Eric. 1997. Learning string edit distance. In *Fourteenth International Conference on Machine Learning*.

Ristad, Eric and Robert G. Thomas. 1997a. Hierarchical non-emitting Markov models. In *Proceedings of the 35th Annual Meeting, ACL'97*. Association for Computational Linguistics.

Ristad, Eric and Robert G. Thomas. 1997b. Nonuniform Markov models. In *International Conference on Acoustics, Speech, and Signal Processing*.

Ristad, Eric and Peter N. Yianilos. 1996. Learning string edit distance. Princeton University, Department of Computer Science, Research Report CS-TR-532-96.

Rizzi, Luigi. 1990. *Relativized Minimality*. MIT Press, Cambridge, Massachusetts.

Rizzi, Luigi. 2000. Reconstruction, weak island sensitivity, and agreement. Università di Siena.

Robinson, J.A. 1965. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12:23–41.

Rogers, James. 1995. On descriptive complexity, language complexity, and GB. Available at ftp://xxx.lanl.gov/cmp-lg/papers/9505/9505041.

Rogers, James. 1999. *A Descriptive Approach to Language-Theoretic Complexity*. Cambridge University Press, NY.

Rogers, James. 2000. wMSO theories as grammar formalisms. In *Proceedings of the Workshop on Algebraic Methods in Language Processing, AMiLP'2000/TWLT16*, pages 233–250.

Roorda, Dirk. 1991. *Resource-logics: proof-theoretical investigations*. Ph.D. thesis, Universiteit van Amsterdam.

Rosch, Eleanor. 1978. Principles of categorization. In E. Rosch and B.B. Lloyd, editors, *Cognition and categorization*. Erlbaum, Hillsdale, New Jersey.

Rosenfeld, Ronald. 1996. A maximum entropy approach to adaptive statistical language modeling. *Computer, Speech and Language*, 10.

Ross, John R. 1967. *Constraints on Variables in Syntax*. Ph.D. thesis, Massachusetts Institute of Technology.

Rosser, J. Barkley. 1935. A mathematical logic without variables. *Annals of Mathematics*, 36:127–150.

Saah, Kofi K. and Helen Goodluck. 1995. Island effects in parsing and grammar: Evidence from Akan. *The Linguistic Review*, 12:381–409.

Salomaa, Arto. 1973. *Formal Languages*. Academic, NY.

Salton, G. 1988. *Automatic Text Processing*. Addison-Wesley, Menlo Park, California.

Samuelsson, Christer. 1996. Relating Turing's formula and Zipf's law. Available at `http://coli.uni-sb.de/~christer/`.

Sanchez-Valencia, V. 1991. *Studies on Natural Logic and Categorial Grammar*. Ph.D. thesis, University of Amsterdam, Amsterdam.

Satta, Giorgio. 1994. Tree adjoining grammar parsing and boolean matrix multiplication. *Computational Linguistics*, 20:173–232.

Savitch, Walter J., Emmon Bach, William Marsh, and Gila Safran-Naveh, editors. 1987. *The Formal Complexity of Natural Language*. Reidel, Boston.

Sayood, Khalid. 1996. *Introduction to Data Compression*. Morgan Kaufmann, San Francisco.

Schacter, Paul. 1985. Focus and relativization. *Language*, 61:523–568.

Schützenberger, M. P. 1961. A remark on finite transducers. *Information and Control*, 4:185–196.

Seki, Hiroyuki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229.

Shannon, Claude E. 1948. The mathematical theory of communication. *Bell System Technical Journal*, 127:379–423. Reprinted in Claude E. Shannon and Warren Weaver, editors, *The Mathematical Theory of Communication*, Chicago: University of Illinois Press.

Shibatani, Masayoshi. 1990. *The Languages of Japan*. Cambridge University Press, Cambridge.

Shieber, Stuart and Mark Johnson. 1994. Variations on incremental interpretation. *Journal of Psycholinguistic Research*, 22:287–318.

Shieber, Stuart M. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3):333–344.

Shieber, Stuart M. 1992. *Constraint-based Grammar Formalisms*. MIT Press, Cambridge, Massachusetts.

Shieber, Stuart M., Yves Schabes, and Fernando C. N. Pereira. 1993. Principles and implementation of deductive parsing. Technical Report CRCT TR-11-94, Computer Science Department, Harvard University, Cambridge, Massachusetts. Available at `http://arXiv.org/`.

Sikkel, Klaas. 1997. *Parsing Schemata*. Springer, NY.

Sikkel, Klaas and Anton Nijholt. 1997. Parsing of context free languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Volume 2: Linear Modeling*. Springer, NY, pages 61–100.

Smith, Edward E. and Douglas L. Medin. 1981. *Categories and Concepts*. Harvard University Press, Cambridge, Massachusetts.

Smullyan, Raymond M. 1985. *To Mock a Mockingbird*. Knopf, New York.

Spencer, Andrew. 1993. Incorporation in Chukchee. University of Essex.

Sportiche, Dominique. 1994. Adjuncts and adjunctions. Presentation at 24th LSRL, UCLA.

Sportiche, Dominique. 1998a. Movement, agreement and case. In Dominique Sportiche, editor, *Partitions and Atoms of Clause Structure: Subjects, agreement, case and clitics*. Routledge, New York.

Sportiche, Dominique. 1998b. *Partitions and Atoms of Clause Structure : Subjects, Agreement, Case and Clitics*. Routledge, NY.

Sportiche, Dominique. 1999. Reconstruction, constituency and morphology. GLOW, Berlin.

Stabler, Edward P. 1991. Avoid the pedestrian's paradox. In Robert C. Berwick, Steven P. Abney, and Carol Tenny, editors, *Principle-based Parsing: Computation and Psycholinguistics*. Kluwer, Boston, pages 199–238.

Stabler, Edward P. 1992. *The Logical Approach to Syntax: Foundations, specifications and implementations*. MIT Press, Cambridge, Massachusetts.

Stabler, Edward P. 1996. Computing quantifier scope. In Anna Szabolcsi, editor, *Ways of Scope Taking*. Kluwer, Boston.

Stabler, Edward P. 1997. Derivational minimalism. In Christian Retoré, editor, *Logical Aspects of Computational Linguistics*. Springer-Verlag (Lecture Notes in Computer Science 1328), NY, pages 68–95.

Stabler, Edward P. 1998. Acquiring grammars with movement. *Syntax*, 1:72–97.

Stabler, Edward P. 1999. Remnant movement and complexity. In Gosse Bouma, Erhard Hinrichs, Geert-Jan Kruijff, and Dick Oehrle, editors, *Constraints and Resources in Natural Language Syntax and Semantics*. CSLI, Stanford, California, pages 299–326.

Stabler, Edward P. 2002. *Computational Minimalism: Acquiring and parsing languages with movement*. Basil Blackwell, Oxford. Forthcoming.

Steedman, Mark J. 1989. Grammar, interpretation, and processing from the lexicon. In William Marslen-Wilson, editor, *Lexical Representation and Process*. MIT Press, Cambridge, Massachusetts, pages 463–504.

Steedman, Mark J. 2000. *The Syntactic Process*. MIT Press, Cambridge, Massachusetts.

Stickel, Mark E. 1992. A prolog technology theorem prover: a new exposition and implementation in prolog. *Theoretical Computer Science*, 104:109–128.

Stolcke, Andreas. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21:165–201.

Storer, James A. 1988. *Data Compression: Methods and Theory*. Computer Science Press, Rockville, Maryland.

Stowell, Tim. 1981. *Origins of Phrase Structure*. Ph.D. thesis, Massachusetts Institute of Technology.

Szabolcsi, Anna. 1996. Strategies for scope-taking. In Anna Szabolcsi, editor, *Ways of Scope Taking*. Kluwer, Boston.

Szymanski, Thomas G. and John H. Williams. 1976. Noncanonical extensions of bottom-up parsing techniques. *SIAM Journal of Computing*, 5:231–250.

Tarski, Alfred. 1935. Der Wahrheitsbegriff in den formaliserten Sprachen. *Studia Philosophica*, I. Translated by J.H. Woodger as "The Concept of Truth in Formalized Languages", in Alfred Tarski, 1956: *Logic, Semantics and Metamathematics*. Oxford.

Taylor, J. C. 1997. *An Introduction to Measure and Probability Theory*. Springer, NY.

Teahan, W.J. 1998. *Modelling English Text*. Ph.D. thesis, University of Waikato.

Tesar, Bruce and Paul Smolensky. 2000. *Learnability in Optimality Theory*. MIT Press, Cambridge, Massachusetts.

Tomita, Masaru. 1985. *Efficient parsing for natural language: a fast algorithm for practical systems*. Kluwer, Boston.

Valiant, Leslie. 1975. General context free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10:308–315.

Valois, Daniel. 1991. The internal syntax of DP and adjective placement in French and English. In *Proceedings of the North Eastern Linguistic Society, NELS 21*.

van de Koot, Johannes. 1990. *An Essay on Grammar-Parser Relations*. Ph.D. thesis, University of Utrecht.

Vergnaud, Jean-Roger. 1982. *Dépendances et Niveaux de Représentation en Syntaxe*. Ph.D. thesis, Université de Paris VII.

Vijay-Shanker, K., David Weir, and Aravind Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 104–111.

Viterbi, Andrew J. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13:260–269.

Wartena, Christian. 1999. *Storage Structures and Conditions on Movement in Natural Language Syntax*. Ph.D. thesis, Universität Potsdam.

Watanabe, Akira. 1993. *Case Absorption and Wh Agreement*. Kluwer, Dordrecht.

Weaver, Warren. 1949. Recent contributions to the mathematical theory of communication. In Claude E. Shannon and Warren Weaver, editors, *The Mathematical Theory of Communication*. University of Illinois Press, Chicago.

Weir, David. 1988. *Characterizing mildly context-sensitive grammar formalisms*. Ph.D. thesis, University of Pennsylvania, Philadelphia.

Williams, Edwin. 1983. Semantic vs. syntactic categories. *Linguistics and Philosophy*, 6:423–446.

Wittgenstein, Ludwig. 1922. *Tractatus logico-philosophicus*. Routledge and Kegan-Paul, London, 1963. The German text of Ludwig Wittgenstein's Logisch-philosophische Adhandlung, with a translation by D. F. Pears and B. F. McGuinness, and with an introduction by Bertrand Russell.

Wittgenstein, Ludwig. 1958. *Philosophical Investigations*. MacMillan, NY. This edition published in 1970.

Yang, Charles. 1999. Unordered merge and its linearization. *Syntax*, 2:38–64.

Younger, D.H. 1967. Recognition and parsing of context free languages in time $o(n^3)$. *Information and Control*, 10:189–208.

Zipf, George K. 1949. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*. Houghton-Mifflin, Boston.

Zobel, J., A. Moffat, R. Wilkinson, and R. Sacks-Davis. 1995. Efficient retrieval of partial documents. *Information Processing and Management*, 31:361–377.

Index